TOWARDS AUTOMATIC CONFIGURATION OF ACCESS CONTROL

by
Nazia Badar

A dissertation submitted to the
Graduate school-Newark
Rutgers, The State University of New Jersey
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
Graduate Program in Management
Information Technology Major

Written under the direction of
Dr. Vijayalakshmi Atluri
Dr. Jaideep Vaidya
and approved by

_____

_____

_____

_____

Newark, New Jersey
October 2016

DISSERTATION ABSTRACT

TOWARDS AUTOMATIC CONFIGURATION OF ACCESS CONTROL

By Nazia Badar

Dissertation Director: Dr. Vijayalakshmi Atluri and Dr. Jaideep Vaidya

Access control provide means to implement organizational security policies to both of its physical and electronic resources. To date, several access control mechanisms, including Role Based Access Control (RBAC) and Discretionary Access Control (DAC) have been proposed. Regardless of which security mechanism an organization adopts, once the system variables such as policies, roles, and authorizations are defined, continuous configuration management of these systems become necessary in order to ensure that the behavior of implemented system matches with the expected system behavior. In recent years, configuration errors in access control system have emerged as one of the key causes of system failure. Traditional access control system lacks the ability to anticipate potential configuration errors. Therefore, these systems fail to gracefully react to this problem. Configuration errors often occur either in the form of false positive or false negative authorizations. It is not trivial to manually identify such misconfigurations, and moreover, existing methods of analyzing system configuration are not efficient in detecting misconfigurations. Therefore, there is an acute need of better ways for automatic configuration of access control systems. This dissertation aims at developing efficient and automatic methodologies and tools for

access control configuration management that are based on data mining technologies. Specifically, it addresses the following three research issues.

The first research problem is based on using risk estimates for configuration management. There exist a number of situations in which specific user permission assignments based on the security policy cannot be a priori decidable. These may include emergency and disaster management situations where access to critical information is expected because of the need to share, and in some cases, because of the responsibility to provide information. This dissertation has proposed novel methodologies for dynamic computation of risk in such situations where preventing an access to a resource has more deleterious effect than granting it, if the underlying risk is low. Moreover, it has developed a model that facilitates risk-based access control in both DAC and RBAC cases. Also, in case of RBAC, it has developed a method to determine situational role for a user. Computational experiments performed on both synthetic and benchmark real datasets, even in the presence of noise, confirms the viability of the proposed approaches.

The second issue is to investigate the configuration management problems that arise as a result of changes within a system or due to requests from users from collaborating organizations that do not have explicit access to resources. This dissertation has proposed to exploit attribute semantics of users to (semi)automate security configuration and management, and has proposed a methodology to derive credential requirements for roles having permission to access requested object, based on local access control policies using existing access control data. The proposed approach is based on well-known data mining method known as classification. Experimental evaluation shows that the proposed method has outperformed the previously proposed approach to address this problem.

Finally, the third research issue deals with automating the process of identifying and removing misconfigurations in RBAC and DAC. Towards this end, this dissertation has proposed approaches

to automate the process of detection of exceptionally or erroneously granted or denied authorizations in access control data. These approaches are based on using multiple classifiers to identify anomalous assignments. An extensive experimental evaluation has been performed to demonstrate the accuracy and performance of the proposed approaches.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisors, Dr. Vijay Atluri and Dr. Jaideep Vaidya. It was not possible for me to thrive in my research without their support. I also wish to thank my committee members who collaborated with me on this work, Dr. Nabil Adam, Dr. Soon Ae Chun, and Dr. Basit Shafiq .Their insight was invaluable.

I also want to thank my parents and my siblings I owe them all my gratitude for believing in me since the beginning, and for always being there for me. I also want to thank my husband, Asad, who supported me with his patience and encouragement. Finally, I would like to thank my daughter, Emaan - her giggles and little smiles make my world go round.

Thank you all!

To my parents, to whom I owe the greatest debt.
To my siblings, for their true love and for always believing in me.
To my husband, for his encouragement.
To my daughter, who is the most precious gift that I've ever been given.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Securing organizational resources from illegitimate accesses is a prime concern for organizations. In order to ensure the security of resources, businesses invest tremendous cost in terms of both money and time, to implement an effective and efficient access control (AC) system. AC model is the formal representation of access control policies and their functioning. Access control system provide means to execute those policies. In simple words, access control policy is set of rules which defines *who (subject) can access what (object)*, and access control model bridges the gap between the policies and the mechanism to enforce them. To date, several access control mechanisms and their extensions have been proposed in the literature of information security [32]. Mandatory Access Control (MAC), and Discretionary Access control (DAC), are two classical access control mechanisms. However, in recent years, Role Based Access Control (RBAC) [23] has gained unprecedented prominence and has emerged as one of the most robust security model to meet diverse access control requirements.

RBAC is a policy neutral model. In contrast to traditional MAC and DAC based security systems, RBAC offers significant reduction of administrative overheads and an additional advantage of flexibility. Moreover, it provides the ease of administrating subject and object authorizations via *roles*. Unlike traditional systems, where authorizations are granted to users directly, the underlying concept of RBAC is to assign user to a role through which they acquire authorizations which are necessary to complete an assigned job. Enterprises still employing their old access control systems

want to migrate to RBAC.

Top-down approach takes description of business function, processes, and other security information into an account when roles are created. While roles are devised using this approach, difficulty comes from the fact that system engineer usually has little knowledge on the semantic meanings of user responsibilities and business processes within an enterprise. Therefore, he can inadvertently introduce errors in the system configuration through inconsistent assignments. Alternatively, *bottom-up approach* approach a.k.a *role mining* is based on determining roles from existing user-permission authorization data. In bottom-up approach, choice of role mining algorithm is a sensitive aspect and plays a key role in RBAC configuration. Since no role mining method so far is perfect, and that they all come with their own set of benefits and limitations, therefore, it is quite a challenge to choose the role mining method that matches perfectly with the security needs of a particular organization. If not chosen appropriately, it turns entire thing into a self-perpetuating error. Moreover, when ill suited role mining method is adopted, there are high chances that an appropriate set of roles are not constituted. This affects the overall performance of a deployed system negatively, ultimately defeating the sole purpose of implementing access control system. It is therefore necessary to ensure that system configuration is sound and that it guarantees prevention against unauthorized access, whilst enable legitimate subjects to gain unhampered access to their requested resources.

In contrast to RBAC, in DAC, every resource in the system has an owner, and rights to access any resource are explicitly defined by its owner. Unlike MAC, where system administrator solely controls assignments of permissions, DAC is relatively easier to implement, and offers flexibility of delegation of rights i.e. authorized user of a resource can also delegate permission to other users. However, the administrative overhead of maintaining Access control lists (ACLs), especially in large organizations, is unreasonably burdensome. User obtain an access to any resource through

specific permission request, but there are rarely any revocation requests made in this system.

Each access control mechanism and its extension is focused on particular access control sub-problem(s), and offers a unique set of functionalities. Their implementation phase is quite challenging and expensive. In order to realize full benefits of these systems, it is critical to ensure that their configuration is correct. In one of the benchmark study performed on reliability of systems, Jim Gray [28] notes,

"The top priority for improving system availability is to reduce administrative mistakes by making self-configured systems with minimal maintenance and minimal operator interaction. Interfaces that ask the operator for information or ask him to perform some function must be simple, consistent and operator fault-tolerant"

Inconsistent access control policies and poor system configuration leads to problems which ultimately downgrades overall performance of the existing access control system. Moreover, once the system variables such as *policies*, *roles*, and authorizations are defined, continuous monitoring of these systems become vital to ensure that the behavior of implemented system matches with the expected system behavior and that the configuration is up-to-date because an overly compromised or overly restrictive system becomes a major source of security violations and maintaining such system is quite a challenge. Ideally, this maintenance activity should be carried out on regular basis. Unfortunately, however, this issue often gets overlooked.

1.1 Configuration Error Management - Why automated configuration error detection and resolution is necessary?

Configuration management of access control system is an important research issue. It plays a key role in making system *available*. In recent years, system configuration errors have become one of the main causes of system failures by making those systems unavailable and causing severe service

outages and significant downtime. Traditional access control system lacks the ability to anticipate potential configuration errors. Therefore, these systems fail to gracefully react to this problem. Moreover, the cost of resolving configuration errors is often tremendous from the aspects of both money and time. Therefore, many organizations are still reluctant in adopting traditional methods of access control. The graveness of matter, and cost involved in handling configuration errors makes it one of the most challenging problems that needs to be addressed. In recent years, we have witnessed that a number of large internet and cloud-service providers such as Amazon, Google, iCloud, LinkedIn, and Microsoft experienced service outages caused due to misconfiguration in the system [1, 3].

Misconfiguration in access control system occurs when *more than necessary* or *less than necessary* authorizations are assigned to users. These errors manifest themselves in the access control system as either in the form of false positive or false negative authorizations. In businesses of large size and of dynamic nature, security misconfiguration usually occurs as a result of frequent changes in user roles, job responsibilities, addition of more permissions within the system, and when more users are added in to a system. Moreover, old or unnecessary user assignments are not revised by security administrators on regular basis. Configuration errors become even more serious when user accounts are not deleted when they leave an organization. These kind of situations leads to introduction of undesirable inconsistencies in the deployed access control system. Inadequate security allows malicious users to gain unauthorized access and perform unintended actions on sensitive organizational resources. Since configuration spaces are large and complex with several dependencies and correlated parameters, manually identifying errors in the system configuration is not trivial task. Even an experienced, well-trained system administrators may make a mistake in the process. Therefore, there is an acute need of better ways for automatic configuration of access control systems. This dissertation aims at developing efficient and automatic methodologies and

tools for access control configuration management that are based on data mining technologies.

1.2 Problem Statement and Contributions

In recent years, configuration errors in access control system has emerged as one of the key causes of system failure. Recently, one of the main services of Google reported configuration errors as second major issue that contributed towards service-level failure [7]. Although its significance has inspired many research efforts towards developing efficient methods for detecting and fixing misconfigurations, but literature survey shows that still this issue has not yet gained due attention from research community.

Given an access control data, problem of configuration management in access control system is to identify and resolve any inconsistent assignments. Generally, when an access control is implemented in an organization, it is assumed that the access control data has no noise, and that the existing configuration is good to start with. Problem worsens when it is further assumed by administrators that the defined assignments in a system are consistent with the intended access control policy of an organization. As a matter of fact, these assumptions are way too unrealistic. Since access control datasets are no exception, and like any other datasets they may possibly contain large amount of noise. Moreover, access control data is large and complex - having several dependencies and correlations, therefore, it is not trivial to control configuration errors manually. There are some approaches introduced in existing literature to address the problem of configuration management. Those approaches have several limitations which we have discussed later. In this dissertation, we address those limitations and developed efficient methods to facilitate automatic configuration management. Below, we briefly discuss our contributions.

We first consider the situations where flexible decision making regarding the access is required. In many emergency and disaster management situations, access to critical information is expected

because of the need to share, and in some cases, because of the responsibility to provide information. Therefore, the importance of situational semantics cannot be underestimated when access control decisions are made. There is a need for providing access based on the (unforeseen) situation, where simply denying an access may have more deleterious effects than granting access, if the underlying risk is small. Traditional access control operates under the principle that a users request to a specific resource is denied if there does not exist an explicit specification of the permission in the system. These limitations have significantly increased the demand for new access control solutions that provide flexible, yet secure access. In this dissertation, we quantify the risk associated with granting an access based on the technique of classification. The goal is to use risk estimates for making automatic decisions regarding configuration of existing system. We proposed two approaches to address the problem at hand. The first approach, considers only the simple access control matrix model, and evaluates the risk of granting a permission based on the existing user-permission assignments. The second assumes role-based access control, and determines the best situational role that has least risk and allows maximum permissiveness when assigned under uncertainty. We experimentally evaluate both approaches with real and synthetic datasets.

In addition to problem discussed above, configuration management issues also arise in situations where organizations participate in the process of electronic coalition formation. Goal of such collaboration is to enable sharing of resources across the boundaries. Despite of all the compelling benefits that coalition formation offers, it is still risky business for organizations to become a part of such collaborations where participating entities are not long standing partners and they seem to join or leave the collaboration quickly. Participating entities may not want to share their resources with other organizations in an uncontrolled manner. Instead, they want the local access control policy to remain applicable when sharing is done with external entities. In multi-domain environment, fundamental building blocks of access control are not very well defined. Given these

administrative difficulties, and multi domain nature of collaboration, having varying definition of user credentials, roles, and resources several configuration related issues may arise. In this dissertation we present an approach to facilitate automatic enforcement of access control policies when a new user is added to an existing access control system. To evaluate the effectiveness of our approach we performed extensive experiments on both real and synthetic datasets. We compare the performance of our approach to an existing approach that to handles a similar problem. Experimental results show that our approach performs very well. Moreover, our approach is relatively easier to implement.

Finally, this dissertation addresses the problem of automatically detecting and resolving misconfigurations in a deployed access control systems. We consider both RBAC and Permission based authorization system. Our proposed technique is based on combining classifiers to facilitate decision making regarding the inconsistencies in the existing system. Each classifier is considered as an independent expert and its probability output is taken into an account to determine whether a particular user assignment is erroneous or not. The viability of proposed method is assessed by performing experiments on synthetic data sets. In summary, we have made following contributions in this dissertation:

Using Risk Estimates for Configuration Management.

1.a We develop a mechanism for quantifying risk dynamically, in both RBAC and DAC based security systems. Dynamic computation of risk is necessary in situations where preventing an access has more deleterious effect than granting the access.

1.b We propose a model based on risk estimates to support configuration management process.

2. We develop a method for identifying situational role for a new user under uncertainty.

3. We address the challenge of incorporating both conflicting features: *flexibility* and *security* at the same time while access control decisions are made.

Using attribute semantics for Configuration Management.

1. We develop a methodology to derive credential requirements for roles having permission to access requested object, based on local access control policies using existing access control data.
2. We propose a mechanism to facilitate coalition based access control.
3. We prove that the secure sharing of resources is possible among collaborating partners while the local access control policy remains intact.

Detection and Resolution of Misconfiguration in Access Control.
1.a We develop approaches to handle the problem of automatic detection of exceptionally or erroneously granted or denied authorizations in access control data in RBAC *1.b* We develop approaches to handle the problem of automatic detection of exceptionally or erroneously granted or denied authorizations in access control data in permission based authorization system.
2.a We present approaches for automatic resolution of misconfiguration in RBAC.
2.b We present approaches for automatic resolution of misconfiguration in permission based authorization system.

Outline

Remaining dissertation is organized as follows. Chapter 3 presents some important preliminaries which are important to understand our work. Chapter 2 discusses some work related to our work. Later in chapter 4, we present our appraoches to tackle the problem of dynamic computation of risk. These approaches are for both RBAC and permission based access control. Results for experimental evaluation are also given. In chapter 5, approaches to facilitate automatic coalition based access control. Experimental evaluation on both real and synthetic datasets is also included in this chapter. Chapter 6 present approaches to automate the process of misconfiguration detection

and resolution in both RBAC and DAC based security systems. Finally, chapter 7 concludes the dissertation by summarizing our work and future work direction.

CHAPTER 2
RELATED WORK

Development of extensions of access control models is continuously under advancements. While a lot of effort has been spent on developing solutions to make access control systems *flexible* and *secure,* however, not much attention has been devoted to *stability* and *sustainability* aspects of deployed systems or in other words, *optimization* of deployed system. In this section we review literature work from four closely related categories: Risk Based Access Control, Break-glass approaches to access control, coalition based access control, and misconfigurations in access control systems. Additionally, in this section, we discuss some important issues which remained unaddressed in the related work.

2.1 Risk Based Access Control

Limitation of traditional access control models in terms of *rigidity* lead to the development of risk-based decision method for an access control systems. Research in risk based access control is fairly recent. The idea behind risk based access control system is to allow flexible decision making by using a non-static authentication system that allows access if the risk associated with the request is low, but denies the request when the risk associated with the request is high.

Badar et al. [6], presented a classification based approach to facilitate risk based access control. In their approach, they utilized estimates generated from random forest based classifier to quantify risk associated with the requested permission in both DAC and RBAC environments. For DAC,

where there is only user permission authorization data, existing set authorizations of requesting user are evaluated against the requested permission, and if the estimated risk is under pre-specified threshold value, then the access is granted otherwise request to access is denied. For RBAC based environment, where user acquires permission via role and multiple roles may contain requested permission, the risk of each role is calculated first. In their work, risk estimate for any role is not same for all users. Instead, risk for a role is relative to the existing authorizations of requesting user. Requested permission is then assigned to a user via least risk role identified by the given approach.

Sandhu et al. [34] provided a risk adaptive access control(RaDAC) model to adjust security and access control policies according to operational and situational factors. The given model utilizes attributes of different components in RaDAC model for reasoning mechanism. Decision making in the given model is based on attribute risk values however a model lacks mechanism to compute risk values. Molloy et al. [42] presented an architecture for Risk Based Access Control based on Support Vector Machines(SVM) classifiers to compute risk values. Decision outcomes are based on utility and risk measures. The limitation of the given model is that there is no consideration for updating risk model. However, risk is non-static in nature and keeps changing. Nissanke et al. [47] presented an interesting work based on analysing risk of permissions. They consider relative security risk of permission posed by different access operations when performed by different users. This work lacks the consideration for method to quantify risk. It is assumed that risk values are already given in the system. This assumption doesnot hold true in most of the real world situations.

Bertino et al. [46], and Cheng et al. [18] studied the effectiveness of fuzzy approaches to facilitate risk based access control. These approaches suffer limitation due to an assumption that the system already has data about risk values i.e. risk is precomputed. [19] proposed a method to facilitate risk based access control. In their work they determine a situational role for a user in

critical situations. Semantic distance between the actual attributes of a user and attributes required by the situational role. If the distance is acceptable, the situational role is granted to a user.

2.2 Break Glass Approaches to Access Control

Work done in this area was motivated by handling access requests in exceptional situations when overriding of regular access control policies becomes necessary or in other words, those situations in which preventing an access to some requested objects has much more deleterious effects than granting access. Generally, break glass mechanism is integrated into the existing control system as an additional layer which is activated in emergency situation to allow overriding of access control policy rules. Break glass approaches were criticized by community of researchers because of the ”dangerously high” level of access flexibility that these approaches offer.

Ferraiolo et al. [25], [24] contributed approaches to integrate Break glass mechanism within standard Role Based Access Control systems. User may access the requested information if he agrees to the obligations for *breaking the glass* and the actions are recorded and monitored by non-repudiation mechanism. Later, Brucker et al. [13] mainly discusses an architecture to extend a deployed access control model with break glass mechanism. To resolve the problem of conflicting policies, Secure UML based approach is given in their work to facilitate break glass mechanism in Role Based Access Control system. Secure UML is based on generalized Role Based Access Control (RBAC) where components and security policies of RBAC are modeled by using UML state based diagrams. In the given approach, regular policy is exercised under normal circumstances whereas, refined policy is activated in exceptional situations. Distance between regular and refined policies plays important role in decision making process. However, in this paper, it is not discussed that how the distance is actually calculated.

2.3 Coalition Based Access Control Systems

Problem of coalition based access control is a well-studied problem. However, issues which arises as a result of misconfiguration in coalition based access control systems is not studied much. Access control research in the area of dynamic coalitions was first introduced by Philips et al. [48, 49] by providing motivating scenarios in defense and disaster recovery settings. Cohen et al. [21] proposed a model that captures the entities involved in coalition resource sharing and identifies the interrelationships among them. Bharadwaj et al. [10] and Khurana et al. [35] addressed the issue of automating policy negotiation and [66] addressed the issue of building trust. Warner et al. [4, 44, 63], proposed a coalition based access control (CBAC) model that facilitates automatic translation of coalition level policies to the implementation level policies, and vice versa. This primarily employs credential attributes in accomplishing the translation, but does not exploit semantics. Access control via attributes simplifies administration because specific users do not have to be given identities. Instead access rights are determined purely on the basis of attributes and these attributes can apply to many different users, who would present their attributes through the use of credentials. Warner et al . [64], further addressed the problem of determining what credentials should be included in a request was addressed, however, it did not address the problem of what credentials are appropriate to require. Additionally, in [63] a graph pruning method is given to reduce the required credential attribute set. The set of required attributes is reduced using frequency counts and sets of attributes held in common by users assigned to roles. The attribute significance introduced in this paper is a similar concept, but computed differently. Moody et al. [5] proposed a similar role-based access control architecture to ours called OASIS where they mapped users via credentials issues by a third party. However, they did not address how the credential requirements are determined, which is the main focus of our work. Wang et al. [62] presented attribute based access control (ABAC) but they also did not give a mechanism for determining which attributes

should be used. Li et al. [37, 38] address the problem of present a Role-Based Trust Management (RT) framework that addresses the issue of discovering credentials needed to map to a role. However, their algorithm does not address how to select attribute requirements based on existing RBAC policies. [20] presented a prototype system called ”Situation Role-based Privacy Control model” that enables a user to control his privacy.

Krishnan et al. [36], presented an approach Group-Centric Secure Information Sharing (g-SIS), to facilitate secure information sharing and collaboration. In their work, they used the notion of group which represents refer to users and objects from multiple organizations. Basically group represents a set of users from multiple organization sharing the same set of objects available to their respective group. g-SIS based approach is based on first-order linear temporal logic (FOTL). g-SIS works as a complimentary model for infrastructure of security across organizational boundaries when coalitions are formed. User who becomes a member of a particular group can access all objects available to the remaining users of the group. However, their model does not takes in to an account the various authorization semantics. When using the CBAC framework, one can think of two extreme cases: If a resource provider (P) requires all the credentials to be possessed by the resource requester (R) to access its resources, P’s policies are ensured with the highest level of security. However, from the perspective of R it is highly restrictive (or less permissive). At the other extreme, if P does not require any credentials at all for R, then there are no guarantees on the security. However, R has the highest permissiveness to acquire the resource. Many variations in between these extremes may exist. Wang et al. [62] presented attribute based access control (ABAC) but did not give a mechanism for determining which attributes should be used. [4], computes the union of all credential attributes of the users playing a role as the required set to facilitate coalition we assumed that a user would send all possessed credentials with every request. Also, it was assumed that the required credentials would match a union of all credentials that anyone

who had access to the object possessed. There are two obvious drawbacks of the approach. First it required far more credentials than a typical user is likely to have and second it required processing of many irrelevant credentials. under emergencies among entities of heterogeneous nature. Chen et al. [17] also presented a framework based on XACML to facilitate secure sharing of resources. Again, automated discovery of required credentials of role remains unaddressed.

2.4 Misconfigurations in Access Control Systems

Bauer et al. [8] introduced an approach based on association rules to predict if there is any misconfiguration in the implemented access control system. In their work, if the intended request is similar to the implemented policy and the request is denied, then this indicates that there is a misconfiguration in a system. In this paper, the problem of identifying and resolving misconfigurations in access-control system by using data mining technique called *association rule mining*. Directly using *confidence* and *support* metrics to evaluate the correctness of rule can substantially overestimate the benefits of using association rule mining technique in practice, due to their tendency to reward predictions that can be deduced to be redundant. Also, the presented approach is based on using *Access control list*(ACLs) as an input to generate predictive rules. However, the given approach cannot be applied directly to the other fine-grained access-control model, such as extensions of *Role Based Access Control*(RBAC), which are designed for the allowing the access on the basis of principle of least privilege and separation of duty. Performance of their approach suffers a serious limitation when redundant rules are present in the access history patterns. Presence of these redundant patterns shadows importance of access patterns which are not redundant but are correct. Later, Bauer et al. [9] gave a method for pruning redundant rules. In both works, [8] and [9], DAC based systems are considered. Mukkamala et al. [45] presented an approach based on role mining for identifying potential misconfigurations within the RBAC based access control

system. Basically, they use the role mining approach given in Vaidya et al. [59] to detect potential misconfigurations in the deployed system.

Das et al. [22], [51] presented a system for detecting small inconsistencies present in access privileges of users within the same group. Presented system uses misconfiguration detection algorithms to discover small inconsistencies in user permission assignments among similar users within the same organization. However, it is assumed that larger differences in the access control are not necessarily misconfigurations. Though the matrix reduction step in Baaz is related to role mining process but still the functionality in the Role Based Access Control mainly in Hierarchical RBAC environment, where senior roles delegated rights to the junior role, is unclear. Moreover, the choice of reference datasets by security administrators affects the results significantly. For example, if a reference dataset contains fewer reference groups then number of generated potential misconfigurations increases, several of which may be invalid. The reason why it happens is that fewer groups will yield more approximate results. On the other hand, if the reference dataset contains a large number of reference groups then the report will contain fewer misconfiguration candidates because the large number of reference group increases the chances to find exact cover which usually results into detection algorithm to miss some valid misconfigurations. Therefore, significant caution and manual effort is required by the administrator to choose appropriate reference datasets.

Hu et al. [31], proposed model checking based approach to determine whether requested update is achievable or not. This work considers Role based access control environment. Their ”updating algorithm” focusses mainly on the problem of resolving misconfiguration and authors assumes that the misconfigurations are already identified by some other mechanism and only the problem to decide whether a requested change should be made or not is left for the security administrator. Shafiq et al. [58] gave Integer programming based based approaches to reconfigure deployed RBAC system. They mainly consider access control problems that arises due to conflicting security

policies.

CHAPTER 3
PRELIMINARIES

This chapter gives brief review of a early models for access control such as: MAC and DAC access control models followed by the discussion of more recent model Role Based Access Control(RBAC). We also discuss some of the research problems associated with these models, which are not yet adequately addressed.

3.1 Mandatory Access Control

Decision making in MAC is based on the security labels assigned to the subjects and objects within the system. System administrator is solely responsible for assigning security labels. Under MAC, if a subject's security label has clearance to access the requested object then the access is granted, otherwise the access in not allowed. This type of access control was mainly implemented in military and civilian government based environments where confidentiality of information is of prime concern. In order to meet varying security needs, several mandatory access control(MAC) models were introduced in the literature of access control. Some widely adopted ones includes: Biba, Chinese Wall, and Bell-la Padula. These models are aimed at preventing unauthorized access to resources and preserving confidentiality by preventing certain type of user actions such as: no-read ups and no write downs. MAC has both advantages and disadvantages. Rigidity of this model in terms of rights delegation is its main short coming. Also, the assignment of security labels to each users and objects in the system is not feasible especially in large scale organizations. Adding

or deleting a user from a system is difficult under MAC based system.

3.2 Discretionary Access Control

In DAC, access rights to an object are explicitly defined by a resource owner. Discretionary access control(DAC) is also called *Identity based access controlIBAC* where access control decisions are based on the identity and ownership of users and resources. Unlike MAC, where system administrator solely controls assignments of permissions, DAC offers flexibility of delegation of rights i.e. authorized user of a resource can also delegate permission to other users. Permission authorizations are maintained in form of access control matrix which is usually called Access Control List or simply *ACLs*. ACL consists of rows and columns where there is a row for each user and column for each object. Corresponding matrix entry for any user represents authorizations of that user to specific object.

Alternative to early models Mandatory Access Control (MAC) and Discretionary Access Control(DAC), is Role Based Access control RBAC .

3.3 Role Based Access Control

Role Based Access control(RBAC) mechanism has gained unprecedented prominence in the recent years. RBAC is implemented for enforcing access control in databases and operating systems. Unlike traditional models where authorizations are granted to users directly, the underlying concept of RBAC is to assign users to *roles* through which a user gains authorization to execute actions on organizational resources. *Roles* are created to include set of all permissions which are necessary for a user to perform assigned job function. In this way, RBAC doesnot only simplifies the administration of resources and users, but also reduces significantly the overall operational cost [57]. Over the years, several extensions of RBAC have been developed to address varying organizational

needs. However, in most of the models it is assumed that the defined policies in the system are static in nature and the implemented system is error free. To the best of our knowledge that the access control literature still lacks reliable approaches to address problems associated with misconfigurations in existing systems. Before moving on to our detailed discussion of research problems that we intend to address in this thesis, we give brief overview of some important RBAC based boolean matrices:

- $UA \subseteq U \times ROLES$, a many-to-many mapping user-to-role assignment relation.
- PA is defined as $PA \subseteq PRMS \times ROLES$, a many-to-many mapping of permission-to-role assignments.
- $UPA \subseteq U \times PRMS$, a many-to-many mapping of user-to-permission assignments.

where,

- $U, ROLES, OPS$, and OBJ are the set of users, roles, operations, and objects.
- PRMS (the set of permissions) $\subseteq \{(op, obj) | op \in OPS \bigwedge$ $obj \in OBJ\}$.
- $PA \subseteq ROLES \times PRMS$, a many-to-many mapping of role-to-permission assignments.
- $assigned\_users(R) = \{u \in U | (u, R) \in UA\}$, the mapping of role R onto a set of users.
- $assigned\_permissions(R) = \{p \in PRMS | (p, R) \in PA\}$, the mapping of role R onto a set of permissions.
- $RH \subseteq ROLES \times ROLES$, is a partial order on *ROLES* called inheritance relation.

- $assigned\_objects\_per\_permission(r, p) \rightarrow \{o \in O \mid p = (op, o)\}$, the permission-to-object mapping which gives the set of objects associated with permission p for a given role.
- $assigned\_objects(r) \rightarrow \cup\{assigned\_objects\_per\_permission(r, p) | p \in assigned\_permissions(r)\}$, the permission-to-object mapping which gives the set of all objects associated with any of the permissions assigned to role r.
- $RH \subseteq R \times R$ is a partial order on R called the role hierarchy or role dominance relation.

3.4 Attributes

We assume that both users and objects are associated with specific attributes. An *attribute* consists of a name and a value pair, (a_i:v_i) and is referred to by its name, a_i. The set of all attributes defined for an organization is Λ.

3.4.1 User Attributes

User attributes that may be semantically relevant to objects describe what the user is capable of doing, has done or is assigned to do. Some of these attributes may be drawn from *certifiable* credentials [30] possessed by the users indicating, for example, that a user has completed a degree or a training program. Others may be explicitly assigned internally by the organizations to indicate, for example, experience the user has had or their current assignments. The set of all user attributes is denoted as the *user attribute base* (UAB). We use $ua_i = \{(a_i : v_i), (a_j : v_j), \ldots\}$ to denote all the attributes associated with a specific user u_i.

Example 1. Tom, who works at LM, has the following attribute set:$ua_{Tom} = \{$`(hasDegree:bachelors), (performsJob:software), (assignedTo:projectBlue), (hasExpertiseIn:java), (officeLoc:NVC1)`$\}$.

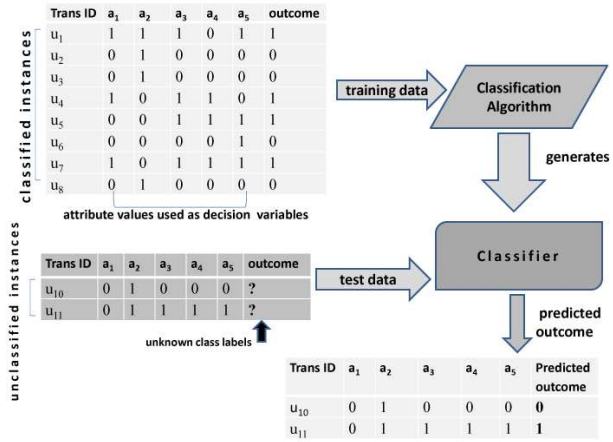


Figure 3.1. The Process of Classification

3.4.2 Object Attributes

Objects attributes are either explicitly defined or automatically derived using text analysis as in [55]. Object attributes might be classified by keywords or content, in terms of their type (e.g., executable, spreadsheet), attributes of their author/owner, or in relationship with each other. Object attributes include object id (o_i), attribute name (a_i) and attribute value (v_i). The set of all object attributes is denoted as the *object attribute base* (OAB). We use $oa_i = \{(a_i : v_i), (a_j : v_j), \ldots\}$ to denote the set of object attributes associated with a particular object o_i.

Example 2. LM has an object "Component 1 software" (*C1*) that has the following attribute set: $oa_{C1} =$

$\{$`(hasContent:java), (hasContent:financial)`

`(hasContent:software), (createdUnderProject:Blue)`$\}$.

3.5 Classification

The problem of classification has long been recognized as one of the most important data mining and machine learning problems. It is applicable in a wide range of science and technology domains. The goal of classification is to build a model based on important structural properties of data with defined class labels and use that model to predict the class label for new or previously unseen data instances. From the perspective of access control problem that we are addressing in this dissertation, we are employing classification method to build a model for predicting role membership status of new users.

Classification model for any role is constructed by using the information about credentials of existing users who have that role, and also the credentials of existing users who don't have that role. Figure 3.1 is an example that shows how we are employing classification method. The input for building the classification model is the existing access control data of the users. For any role, it's membership status for each user within the existing system is already known. In the given example, there are eight users and five attributes. Attributes are also called decision variables. Each row represents a user in figure 3.1. For columns representing attributes, '1' in any cell represents that a user has that attribute and '0' represents that a user doesn't have that attribute. For columns representing roles, '1' in any cell represents that a corresponding user has that role and '0' represents that a user doesn't have that role. New user can become a member of a role if his credentials qualify for a role. This is done by using the classification model for a particular role. For example, in figure 3.1 u10 and u11 are new users and their status for the role membership is unknown. Therefore, attributes of u10 and u11 are checked against the classification model. Role membership status '0' is predicted for u11 which means based on the credentials of u11, he doesn't qualify for the role. However, role membership status '1' is predicted for u11 which means based on the credentials of u11, u11 qualifies for the role.

Some other situations where the classification problem is applicable include: loan granting decisions, credit card fraud detection, network intrusion detection(NIDS), target marketing, medical diagnosis, speech recognition, hand writing recognition, and document classification.In the document classification problem, content within the document is analyzed to classify by topic. Internet search engines and organizational intranets are popular examples of environments where document classification approaches are widely used. Target marketing, is another example of classification problem that strongly depends on underlying classification models used by firms to identify potential customer segment that can be reached for successful promotional activities and business profitability. The purpose of the classification model in this setting is to predict buying decision of potential customers on the basis of their past buying behaviors, interests, age group, ethnicity, etc. Consider an example from the Health care industry. This industry has huge amounts of data. That data can be of great benefit if it is put to an optimal use. For example, a medical history along with other clinical variables about a new patient can be used by health care providers for monitoring health conditions of the patient and for classifying that patient into a known set of diseases or any other related health problems. Predictive models for such classifications are generated by utilizing patient data with known classification outcomes. In a banking application, when a bank receives a new loan request, credentials of a requesting party are evaluated by using predictor variables, such as: education, age, income, and many more predictor variables to decide, whether a loan should be granted or not.

Several approaches for carrying out the task of classification have been proposed in the literature of data mining and machine learning. For example: ID3, C4.5 [54] algorithms are based on decision tree model for classification. SVM [14], on the other hand, is based on linear model. PRISM [15] and Decision Tables [26] uses rule based model for classification. In this dissertation, we are testing our classification based approach using J.48 [54] decision trees, Random

forest [11], and a meta-learner Support Vector Machine [14] algorithms. Below, we briefly discuss each of these classification approaches.

We are mainly using following methods for classification based approach: J.48 [54] decision trees, Random forest [11], and a meta-learner Support Vector Machine [14] algorithms. Below, we briefly discuss each of these approaches.

Decision Trees

A decision tree consists of root node, internal nodes and leaves. Leaf nodes represents decision variables, whereas internal nodes represents decision variables. Number of outgoing edged from each internal node represents number of possible values corresponding attribute can have. The position of attribute's node in a decision tree reflects its importance in reducing entropy. Therefore, an attribute that has minimum entropy is at the root level and successive nodes should be ordered on the basis of their importance in minimizing entropy. Initially a tree is grown to full size by including all attributes and possible relations among attributes, later the full size tree is pruned back to remove any unimportant, redundant or irrelevant branches. The extent of pruning depends on the desired level of complexity. Classical implementation algorithms for decision tree construction are: ID3, C4.5, and C.5 [52], [53]. These algorithms are generally based on divide and conquer strategy where problem of learning classification structure from known set of observation leads to a systematic construction of decision tree. In general, a decision tree is constructed in top-down manner starting from root node and then going all the way down to leaves.

Random Forest

Ensemble based approaches use multiple models to obtain better predictive performance than could be obtained from any of the constituent models. Recently, these approaches have attracted significant interest from data mining and machine learning community. Random Forest [11] is one of these approaches, that has been proved to perform very well in many domains. Random forest is

constructed by ensembles of trees that are built from base learners or decision trees. Each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. The output of a decision forest classifier that gets in input a new observation is the class that is the mode of the classes output by these individual trees. It has been shown that for many data sets, it produces a highly accurate classifier. One of the main strength of this classifier is that it is noticeably consistent even in the presence of noise. This makes it particularly recommended for real world datasets, that usually contain many outliers, missing values, and even other errors.

Support Vector Machines(SVM)

A Support Vector Machine(SVM) [14] based model is constructed by using high dimension feature space.Focus on 2-class classification problem, where there are n number of data points, and each point can belong to any one of the two given classes. In this case, the goal of a classifier is to establish a two-dimensional space by introducing a boundary line (hyperplane), that separates data points of a particular class from data points of another class. Whenever a new point with unknown class label has to be added to this data, its attribute values are compared to the attribute values of existing data points for mapping it to the right side of the hyperplane.

3.6 Feature Selection

Feature selection problem is concerned with finding minimal subset of features that are representative of entire dataset. Two main dimensions for measuring the size of dataset are: number of instances(m), and number of features(n). Usually, dataset that is inherently high-dimensional exhibit the curse of dimensionality properties such as presence of redundancies and/or irrelevant features [39]. Suppose, there is dataset with 200 features. If a classification model has to be constructed from this data set, it is not necessary that all 200 features are necessary for classification.

Some features could be redundant, noisy and, irrelevant. Features are said to be redundant if they are highly correlated with any other features in the same space, whereas they are irrelevant when they do not contain sufficiently discriminative information. Therefore, feature selection process is employed to identify statistically significant, relevant, and minimal subset of features(from entire feature space), subsets which best represents the overall dataset.

In data mining and machine learning literature, several algorithmic implementation of feature selection approaches have been proposed. To name a few, LVF(Las Vegas Filter) algorithm, SFG(Sequential Forward Generation) algorithm, BnB(Branch and Bound) methods [41], Correlation-based Feature Selection(CFSsubsetEvaluation) method [29], are some examples of basic techniques available for feature selection. In this dissertation, we are using CFSsubsetEvaluation technique. For our first approach, purpose of feature selection is to determine a relevant set of permission that represents a role in RBAC based environment. For our second approach, based on non-RBAC access control environment, we are using feature selection to identify related set of permissions in UPA matrix.

3.7 False Positive Assignments(FPs) in Access Control

In a boolean matrix, where 0 in a cell corresponds to *absence*, and 1 corresponds to *presence* of a certain property, a class or a feature, *false positive assignments* are those assignments which are mistakenly recorded as 1's. In RBAC based environment, if any role $r \in ROLES$ is incorrectly present in $Roles(U_x)$ (set of existing roles of user U_x), then such role assignment is called *False Positive Role Assignment(FPRA)*. In non-RBAC based environment, if any permission is present in UPA_{U_x}(existing permission authorizations of U_x), then such assignment is referred as *False Positive Permission Authorization(FPPA)*.

3.8 False Negative Assignments(FNs) in Access Control

In contrast to false positive role assignments, role assignment is referred as false negative role assignment(*FNRA*) if it is added by mistake to existing roles of a user. In non-RBAC based environment, if a user does not have permission which a user should have then such assignment is referred as false negative permission assignment(*FNPA*).

CHAPTER 4
USING RISK ESTIMATES FOR CONFIGURATION MANAGEMENT

Traditional access control operates under the principle that a user's request to a specific resource is honored if there exists an explicit policy specifying that the user is allowed to access that resource. However, there exist a number of situations in which specific user permission assignments based on the security policy cannot be *a priori* decidable. These may include emergency and disaster management situations where access to critical information is expected because of the 'need to share', and in some cases, because of the 'responsibility to provide' information.

Therefore, there are a number of situations where there is a need to permit users to access certain needed resources, who otherwise should not have been allowed. Simply denying an access may have more deleterious effects than granting access, if the underlying risk is small. It is not an easy task- if not impossible - to envision all possible situations beforehand at the time of security policy specification, especially when complex processes are involved.

As an example, consider an access control policy pertaining to the operations of a distillation column in an Industrial system. Under normal operation, the access control policy requires that only the on duty supervisor from a local terminal of the distillation column or from a remote terminal with IP address 165.230.154.45 is authorized to close the steam valve. However, in an emergency or if there is a safety concern, other operators may need to be allowed to close the steam valve even though it is not permitted under the specified security policy. As it can be seen from the above example, the hard coded policy fails to cater to the dynamics of unexpected situations that

arise.

Recently, several break glass approaches [12, 24, 25, 40] have been proposed. Break glass approaches allow overriding of the existing policies in case of emergency. While traditional access control is too restrictive since it does not permit any access that has not been pre-specified in the system, the break glass approaches are too permissive in that they allow all requested accesses based on the situation at hand. Since both these are at two extremes, a fine balance between permissiveness and restrictiveness is desirable.

To accomplish this, in our work, we present approaches to quantify the risk associated with granting an access based on the technique of classification. We propose two approaches for risk-based access control that determine whether or not to honor requests by computing the associated risk. The first approach, considers only the simple access control matrix model, and evaluates the risk of granting a permission based on the existing user-permission assignments (UPA). Essentially, a classification model is built for every permission in UPA that gives the probability of whether that permission is assigned to a user or not. When a user requests a permission which is not included in the UPA, the classification model comprising of the probabilities of the requested permission is used to compute the risk. The second approach assumes role-based access control, and determines the best situational role that has least risk and allows maximum permissiveness when assigned under uncertainty. This will also employ the classification model for risk computation. When the evaluated risk for granting a (non-existing) permission is lower than a given threshold value, the requested access is allowed.

4.1 Risk based Access Control

In this section, we give a more formal description to the approaches for facilitating risk-based access control. The first approach, *risk based permission authorization*, considers a simple access

control matrix for policy specification (only the user permission assignment is given). The second approach *risk based role authorization* assumes that role-based access control is deployed. In both these approaches, we first quantify the risk associated with the requested access, and permit access if it is less than a prespecified threshold value. Depending on the access control model being considered, the access request and authorization semantics of these approaches are different as formally stated below:

1. Risk based permission authorization Given an access request $(U_i, P_j, \{P_{i_1}, \ldots, P_{i_m}\})$, where:

- U_i is the user making the access request;
- P_j is the requested permission; and
- $\{P_{i_1}, \ldots, P_{i_m}\}$ is the set of permissions for which U_i currently has the authorization.

The access request is granted if the computed risk is lower than the threshold value specified by the system for the requested permission.

In case of role-based access control, the access request may be either for a specific role or a permission. If the request is for a role the risk of the role being assigned needs to be computed. Otherwise the risk of all roles having the requested permission needs to be computed and the role with the lowest risk assigned. This is formally specified as below:

2a. Risk based role authorization Given an access request $(U_i, R_x, \{R_{i_1}, \ldots, R_{i_m}\})$, where R_X is the requested role and $\{R_{i_1}, \ldots, R_{i_m}\}$ is the set of roles for which U_i has the authorization. The access request is granted by assigning U_i to role R_x if the risk associated with the (U_i, R_x) assignment is lower than the threshold value specified by the system for the requested role.

	P1	P2	P3	P4	P5
U1	1	1	1	0	1
U2	1	1	1	0	1
U3	0	1	1	0	1
U4	0	1	1	0	1
U5	1	1	1	0	0
U6	1	0	1	0	0
U7	1	0	0	1	1
U8	1	1	1	0	1
U9	1	1	1	0	0
U10	1	0	1	1	1

(a) User Permission Assignment (UPA)

	R1	R2	R3	R4	R5
U1	1	1	0	0	0
U2	1	0	1	0	0
U3	0	0	1	0	0
U4	0	0	1	0	0
U5	1	0	0	0	1
U6	0	0	0	0	1
U7	0	1	0	1	0
U8	1	1	0	0	1
U9	1	0	0	0	0
U10	0	0	0	1	1

(b) User Assignment (UA)

	P1	P2	P3	P4	P5
R1	1	1	1	0	0
R2	1	0	0	0	1
R3	0	1	1	0	1
R4	0	0	0	1	1
R5	1	0	1	0	0

(c) Permission Assignment (PA)

Figure 4.1. Example of UPA, UA and PA matrices

2b. Risk based role authorization. Given an access request $(U_i, P_j, \{R_{i_1}, \ldots, R_{i_m}\})$, where $\{R_{i_1}, \ldots, R_{i_m}\}$ is the set of roles for which U_i has the authorization. The access request is granted by assigning U_i to role R_x such that:

- $(P_j, R_x) \in PA$;
- Risk associated with the (U_i, R_x) assignment is the minimum over assignment to any other role R_y for which $(P_j, R_y) \in PA$; and
- Risk associated with the (U_i, R_x) assignment is lower than the threshold value specified by the system for the requested permission.

The specific approach for each case is discussed below. To illustrate the procedure, we will use the user permission assignment (UPA) matrix depicted in Figure 4.1(a). When role based access control is assumed to be deployed, we will use the user assignment (UA) matrix and permission assignment (PA) matrix depicted in Figure 4.1(b) and Figure 4.1(c), respectively.

4.1.1 Risk based Permission Authorization

Given the UPA matrix, in this approach, we first build the *classification model* (CM) for each permission. This process, of building the classification models, is typically carried out at the start, though the models may be refined (either incrementally or completely recomputed) if the system state changes significantly. Once models are built for all permissions, these models are then stored and utilized for future risk assessment. Suppose that user U_6 (depicted in Figure4.1(a)) requests permission P_2. From Figure 4.1(a), it can be seen that U_6 has been assigned permissions P_1 and P_3. Since all the users who have been assigned permission P_2 in the UPA matrix of Figure 4.1(a) have been assigned either permission P_1 or permission P_3, the risk for granting this access request is relatively low and it may be secure to assign permission P_2 to U_6. We now show how this risk can be quantified.

Essentially, we can treat the permission request as a new instance to be classified for the appropriate model. Thus, if user U_i requests permission P_j, then the classifier model for P_j needs to be used. This classification model makes a decision based on the existing permission set of U_i. Thus, the permissions of U_i form the new instance which is classified by the model, and the classifier returns the probability of that instance belonging to the class of permission P_j or not. Thus, the output of this classification corresponds to the $AssessedRisk$ value. If this value is lower than the threshold, then permission can be granted to the requesting user; otherwise the request is denied.

The procedure consists of the following two steps:

Step 1: *Model building for each-permission* In this step, a classification model is built for each permission within an organization. This step uses the *UPA* matrix which represents the user-to-permission assignments within an organization. For each permission, there is one column in *UPA*. A cell value of '0' represents the absence of the permission, while '1' represents presence of that

permission in the corresponding user's permission set. Essentially, if there are n permissions, n classification models are built - one for each permission. When creating the classification model for each permission, the column for that permission is denoted as the class, while the remaining data are used to train the classifier model. The models built for each permission are then stored for later use. Algorithm 1 gives the formal specifications for this step.

Step2: *Classification of request* User may want to have a permission which was not assigned to him before. For determining whether a user should be given that permission or not, a request is classified on the basis of existing permission set of a user. This is done by first retrieving a classification model for a requested permission and then classifying the request. The output of this classification is the probability that the new instance belongs to the class (probability that the access request should be granted). Thus, the inverse gives the $AssessedRisk$ value.

Algorithm 2 gives the formal specifications for access request classification step.

Algorithm 1 $BuildPermissionClassificationModel()$

Require: User-Permission assignment, UPA
Require: Classification Algorithm, CA
1: $CMlist \leftarrow \phi$
2: **for** each perm P_j **do**
3: Denote the column for P_j in UPA as the class attribute
4: $CM \leftarrow BuildClassifier(CA, UPA)$
5: $CMlist \leftarrow CMlist \cup CM$
6: **end for**
7: Return (CMlist)

Figure 4.2 shows the risk values for the UPA matrix of Figure 4.1(a). The risk values are computed for those permissions that have not been assigned to users in the original policy. For example, the risk value for assigning permission P_2 to user U_6 is 0.24. This low risk value is due to the fact that all the users who have permission P_2 have been assigned either permission P_1, or permission P_3 and U_6 has authorization for both of these permissions. On the other hand, the risk

Algorithm 2 PermissionRisk(U_i,P_j)

Require: $Perms(U_i)$ represents existing permission set of user U_i
Require: Classification Algorithm, CA
Require: δ, the threshold for classification
Require: $CMlist$, the list of classification models built from Algorithm 1
1: $CM \leftarrow$ Retrieve the classifier model for permission P_j from $CMlist$
2: $AssessedRisk \leftarrow CA(CM, Perms(U_i), P_j)$
3: **return** $AssessedRisk$

	P1	P2	P3	P4	P5
U1				1.0	
U2				1.0	
U3	0.323			1.0	
U4	0.323			1.0	
U5				1.0	0.469
U6		0.24		0.8	0.8
U7		1.0	0.308		
U8				1.0	
U9				1.0	0.469
U10		1.0			

Figure 4.2. Assessed risk values for permission authorizations considering the UPA matrix of Figure 4.1(a)

value of permission P_5 for U_6 is 0.8. Intuitively, the reason for this high risk value is that in 5 out of 7 instances of P_5 in Figure 4.1(a), P_5 occurs together with P_2. The risk values in Figure 4.2 are computed using the Random Forest classification model.

4.1.2 Risk Based Authorization of Roles

We now discuss the case where role based access control is used. As discussed above, in this case the user may request a specific role or a specific permission (if so, the user is requesting any role that allows him to access the requested permission). We again use the *UA*, *PA* and *UPA* depicted in Figure 4.1(a) to illustrate the approach.

Note that in either approach, classification models have to be built for each permission. This classification model building follows the same procedure given in Algorithm 1, except that the *UPA*

	R1	R2	R3	R4	R5
U1			0.0	1.0	0.0
U2		0.0		1.0	0.0
U3	0.323	0.323		1.0	0.323
U4	0.323	0.323		1.0	0.323
U5		0.469	0.469	1.0	
U6	0.24	0.8	0.848	0.96	
U7	1.0		1.0		0.308
U8			0.0	1.0	
U9		0.469	0.469	1.0	0.0
U10	1.0	0.0	1.0		

Figure 4.3. Assessed risk values for role authorizations considering the UA and PA matrices of Figure 4.1(b) and (c)

matrix used needs to first be constructed from the UA and PA matrices. Therefore, we assume that the classification models have already been built.

We now examine the case where a specific role is requested. In this case, the risk of the role is computed on the permissions that are present in the role. For a permission that is already owned by the user, the risk is 0 (clearly, there is no risk). For the remaining permissions, we can use the $PermissionRisk$ procedure (Algorithm 2) to determine the risk of assigning that permission to the requesting user. Now, we simply need to compute the aggregated role risk from individual permission risks. This can be done as follows:

Assume role R_x consists of permissions $P_1, \ldots, P_k$. Further assume that the risk of permission P_i for the requesting user is denoted by $risk_{P_i}$ and the risk for role R_x is denoted by $risk_{R_x}$:

$$\begin{aligned} risk_{R_x} &= 1 - \text{Risk of not assigning } R_x \\ &= 1 - \text{Risk of not assigning } (P_1, \ldots, P_k) \\ &= 1 - \prod_{\forall s} (1 - risk_{P_s}) \end{aligned} \tag{4.1}$$

This procedure is formally specified in Algorithm 3.

Figure 4.3 shows the risk values for role authorizations considering the UA and PA matrices of

4.1. Similar to Figure 4.2, the risk values for roles are reported for those roles only that have not been assigned to the user.

In the case when a specific permission is requested, we first determine the roles having that permission. Next we compute the risk of each of these roles using the $RoleRisk$ procedure (Algorithm 3). Finally, we select the role with the lowest risk value. This is formally specified in Algorithm 4.

Algorithm 3 RoleRisk(U_i, R_j)

Require: User-Permission assignment, UPA
Require: Permission to role assignment, PA
Require: Classification Algorithm, CA
Require: $Perms(U_i)$ represents existing permission set of user U_i
Require: R_j represents requested role
Require: $CMlist$, the list of classification models built from Algorithm 1
1: **for** each permission $P_k \in R_j$ **do**
2: **if** $P_k \notin Perms(U_i)$ **then**
3: $risk_{P_k} \leftarrow PermissionRisk(U_i, P_k)$ (Algorithm 2)
4: **else**
5: $risk_{P_k} \leftarrow 0$
6: **end if**
7: **end for**
8: $risk_{R_j} \leftarrow 1 - \prod_{\forall P_k \in R_j}(1 - risk_{P_k})$
9: **return** $risk_{R_j}$

Algorithm 4 RBACPermissionRisk(U_i, P_j)

Require: User-Permission assignment, UPA
Require: Permission to role assignment, PA
Require: Classification Algorithm, CA
Require: $Perms(U_i)$ represents existing permission set of user U_i
Require: P_j represents requested permission
1: $crlslist \leftarrow$ each role R_k in PA having P_j
2: **for** each role R_k in $crlslist$ **do**
3: $rlRisk_k \leftarrow RoleRisk(U_i, R_k)$ (Algorithm 3)
4: **end for**
5: **return** $(\min_k rlRisk_k, R_k)$ {Return the Role with the lowest risk}

	P1	P2	P3	P4	P5
U1				Risk of R4 (1.0)	
U2				Risk of R4 (1.0)	
U3	Risk of R1 **(0.323)** Risk of R2 **(0.323)** Risk of R5 **(0.323)**			Risk of R4 (1.0)	
U4	Risk of R1 **(0.323)** Risk of R2 **(0.323)** Risk of R5 **(0.323)**			Risk of R4 (1.0)	
U5				Risk of R4 (1.0)	Risk of R2 **(0.469)** Risk of R3 (0.469) Risk of R4 (1.0)
U6		Risk of R1 **(0.24)** Risk of R3 (0.848)		Risk of R4 (0.96)	Risk of R2 **(0.8)** Risk of R3 (0.848) Risk of R4 (0.96)
U7		Risk of R1 (1.0) Risk of R3 (1.0)	Risk of R1 (1.0) Risk of R3 (1.0) Risk of R5 **(0.308)**		
U8				Risk of R4 (1.0)	
U9				Risk of R4 (1.0)	Risk of R2 **(0.469)** Risk of R3 (0.469) Risk of R4 (1.0)
U10		Risk of R1 (1.0) Risk of R3 (1.0)			

Figure 4.4. Assessed risk values for user permission assignments assuming the RBAC policy of Figure 4.1(b) and (c)

Figure 4.4 shows the risk values for user permission assignments assuming the RBAC policy with UA and PA matrices depicted in Figure 4.1. As can be seen in Figure 4.1(c), a given permission can be accessed through multiple roles. The risk value in Figure 4.4 is computed for all the roles by which the requested permission can be accessed and the role with the lowest risk value is assigned to the user. For example, U_6 can gain permission P_2 through role R_1 with risk value 0.24 or through role R_3 with risk value 0.848. Since role R_1 has the lowest risk value, it is assigned to U_1 for granting permission P_2.

4.2 Experimental Evaluation

We now present the details of our implementation of the risk-based permission and role authorization approach. The experimental implementation uses Weka [65] to build the classification model and compute the risk for each of the permissions. The risk for a role is computed as per Equation 4.1. Weka, developed at the University of Waikato in New Zealand, is a collection of machine

learning algorithms for data mining tasks implemented in Java. Apart from providing algorithms, it is a general implementation framework, along with support classes and documentation. It is extensible and convenient for prototyping purposes. We have used the Random Forest algorithm for building the UPA classification model. We have evaluated the risk based permission authorization approach with both real and synthetic datasets. The real datasets used are described below [43]:

- *Healthcare dataset*: This dataset comprises of 46 users and 46 permissions. Overall, there are 1486 user permission assignments and the density of the UPA matrix is 0.702.
- *Domino*: This dataset comprises of 79 users and 231 permissions. Overall, there are 730 user permission assignments and the density of the UPA matrix is 0.04.
- *Firewall 1*: This dataset comprises of 365 users and 709 permissions. Overall, there are 31951 user permission assignments and the density of the UPA matrix is 0.123.
- *Firewall 2*: This dataset comprises of 325 users and 590 permissions. Overall, there are 36428 user permission assignments and the density of the UPA matrix is 0.19.
- *APJ*: This dataset comprises of 2044 users and 1146 permissions. Overall, there are 6841 user permission assignments and the density of the UPA matrix is 0.003.
- *EMEA*: This dataset comprises of 35 users and 3046 permissions. Overall, there are 7220 user permission assignments and the density of the UPA matrix is 0.068.
- *Customer*: This dataset comprises of 10961 users and 284 permissions. Overall, there are 45428 user permission assignments and the density of the UPA matrix is 0.015.
- *Americas-small*: This dataset comprises of 3477 users and 1587 permissions. Overall, there are 105206 user permission assignments and the density of the UPA matrix is 0.019.

Table 4.1. Synthetic data sets

	NRoles	NUsers	NPerms	MRolesUsr	MPermsRole
data1	20	100	200	5	10
data2	20	200	200	5	10
data3	20	300	200	5	10
data4	20	500	200	5	10

- *Americas-large*: This dataset comprises of 3485 users and 10127 permissions. Overall, there are 185295 user permission assignments and the density of the UPA matrix is 0.005.

The synthetic datasets were created using the test data generator from Vaidya et al [61]. The test data generator performs as follows: First a set of roles are created. For each role, a random number of permissions up to a certain maximum are chosen to form the role. The maximum number of permissions to be associated with a role is set as a parameter of the algorithm. Next, the users are created. For each user, a random number of roles are chosen. Again, the maximum number of concurrent roles a user can have is set as a parameter of the algorithm. Finally, the user permissions are set according to the roles to which the user has been assigned. Table 4.1 gives the characteristics of the datasets created. Since the effect of large number of users, permissions, and varying densities has already been studied with the real datasets, the synthetic datasets were created with a limited size to enable focused testing of the effect of noise, and of the role risk. As the test data creator algorithm is randomized, 3 datasets for each combination of parameters are created, and the results are averaged.

4.2.1 Risk-based Permission Authorization

Table 4.2 and table 4.3 gives the average risk that was computed for each of the real datasets. In each case, we give the average risk for four discrete cases, as well as the percentage of occurrence for each case:

Table 4.2. Average permission risk for real datasets

Dataset	A0-P0	A0-P1
Healthcare	0.96995 (28.1191%)	0.102286 (1.65406%)
Domino	0.990807 (95.8737%)	0.232318 (0.120555%)
Firewall1	0.999191 (87.6113%)	0.235806 (0.0417335%)
Firewall2	0.999905 (80.9961%)	0.283333 (0.00625815%)
APJ	0.998631 (99.1011%)	0.164787 (0.563963%)
EMEA	0.983113 (92.9369%)	0.269579 (0.289841%)
Customer	0.994464 (98.2509%)	0.290119 (0.289728%)
Americas-small	0.999787 (98.0862%)	0.273659 (0.00723088%)
Americas-large	0.999643 (99.4671%)	0.297221 (0.00785151%)

- Risk for permissions that were actually unassigned to a user, and were correctly predicted by the classifier as not to be assigned (A0-P0).
- Risk for permissions that were actually unassigned to a user, and were incorrectly predicted by the classifier as to be assigned (A0-P1). This case corresponds to incorrect over privilege.
- Risk for permissions that were actually assigned to a user, and were incorrectly predicted by the classifier as not to be assigned (A1-P0). This case correspond to incorrect under privilege.
- Risk for permissions that were actually assigned to a user, and were correctly predicted by the classifier as to be assigned (A1-P1).

It is clear that for all of the real datasets, the average risk for the A0-P0 permissions is very high (ranging from 0.96 – 0.99), while the average risk for the A1-P1 permissions is very low (ranging from 0.0001 – 0.18). The risk for A0-P1 is incorrectly low while the risk for the A1-P0 permissions is incorrectly high. However, the combined percentage of such cases is quite low (less than 4% in all cases). Thus, the results are quite good.

Table 4.3. Average permission risk for real datasets

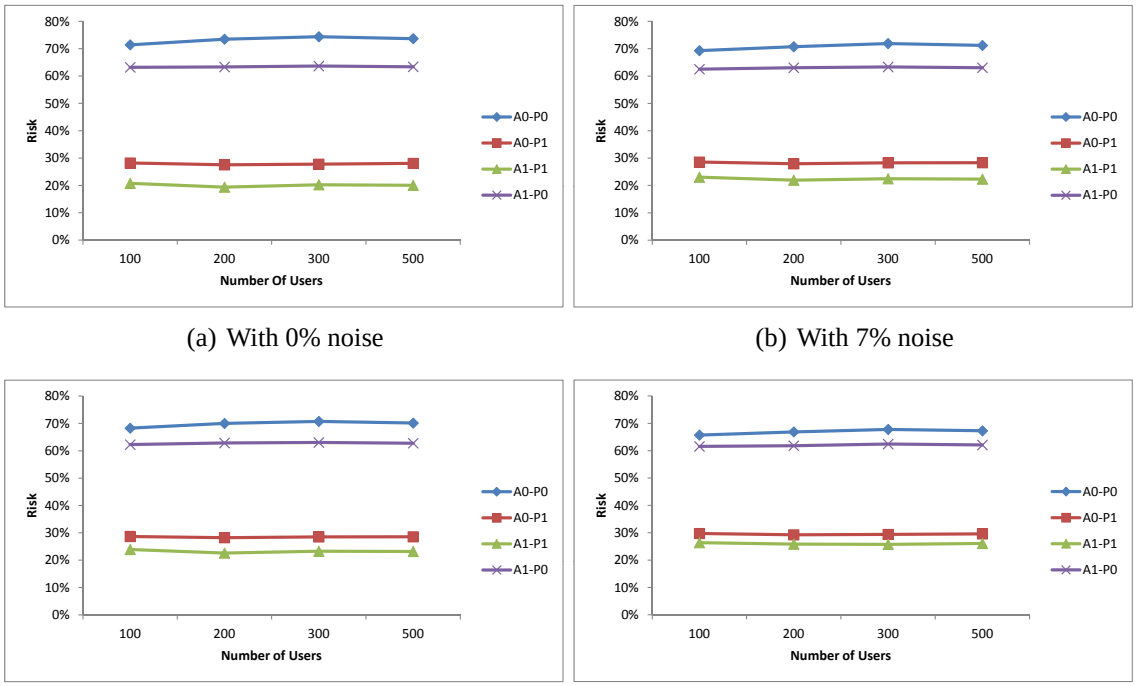
Dataset	A1-P0	A1-P1
Healthcare	0.745429 (1.32325%)	0.00772222 (68.9036%)
Domino	0.889158 (1.21651%)	0.0333792 (2.78919%)
Firewall1	0.925063 (0.233012%)	0.00321427 (12.1139%)
Firewall2	0.66037 (0.0140808%)	0.00019505 (18.9836%)
APJ	0.860487 (0.0511018%)	0.0269798 (0.283824%)
EMEA	0.871483 (3.67414%)	0.150757 (3.09915%)
Customer	0.813936 (0.795715%)	0.184826 (0.66362%)
Americas-small	0.743298 (0.0280355%)	0.00734629 (1.87856%)
Americas-large	0.756415 (0.042754%)	0.0364984 (0.482271%)

Figure 4.5 gives the average risk for the synthetic datasets. The x-axis shows the varying number of users in the four datasets (since the other parameters are the same), while the y-axis shows the average risk. As expected, the risk for the permissions that are actually 0 and are correctly predicted to be 0 is quite high, while the risk for the permissions that are actually 1 and are correctly predicted to be 1 is quite low. It is clear that the increasing number of users does not have any deleterious effect on the results.

We also carried out experiments to evaluate the effect of noise on our approach. To do this, the noise model of [61] was used, and all of the synthetic datasets had noise added to them using randomly (flipping both 0s to 1s and 1s to 0s). Figures 4.5(b)-4.5(d) gives the results with 7%-20% noise added. The main result of the noise is to make the classification results imprecise. The effect can be seen through the lowering risk for the permissions predicted to be 0 and the increasing risk for the permissions predicted to be 1.

4.2.2 Risk-based Role Authorization

To evaluate the risk-based role authorization, we used the same synthetic datasets described in Table 4.1. For each user and each role, the risk was calculated as per Equation 4.1. The risk was then averaged over all of the users and all of the roles. We carried this process out both without

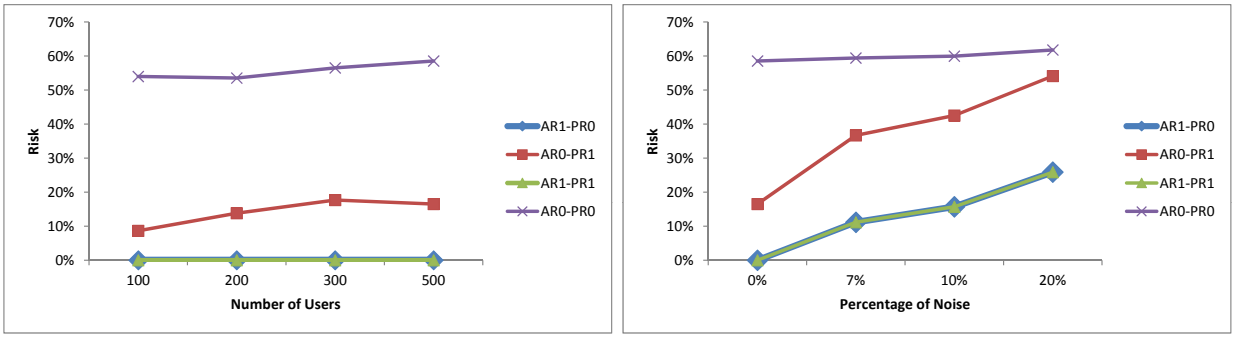


(a) With 0% noise (b) With 7% noise

(c) With 10% noise (d) With 20% noise

Figure 4.5. Average permission risk for synthetic datasets

noise and after the introduction of noise, as described earlier. The results are depicted in Figure 4.6. Figure 4.6(a) shows that the risk of getting an already authorized role is always 0 regardless of whether the role is predicted to be 0 or 1 (AR1-PR0 or AR1-PR1), since all of the permissions associated with that role are already held by the user. However, when the role is not authorized, the risk is quite high if it is predicted to be unauthorized (AR0-PR0) and fairly low if it is predicted to be authorized (AR0-PR1). The results do not vary much based on the number of users. Figure 4.6(b) shows the results when noise is introduced. In the interest of space, we only depict the results in the case of data4 (500 users), though they are the same for all of the other datasets too. It is clear that with the introduction of noise, the risk in all 4 cases increases proportionally to the level of noise. The risk increase is marginal for AR0-PR0 (where it is already high), while being significant in all of the other 3 cases.



(a) Without noise (b) With noise

Figure 4.6. Average role risk in synthetic datasets

CHAPTER 5
USING ATTRIBUTE SEMANTICS FOR CONFIGURATION MANAGEMENT

There is an increasing need for dynamic, efficient and secure sharing of information resources among organizations in a coalition or federation. The goal for electronic collaboration in a coalition is to share specific data and functionality with partners (some of whom are relatively unknown) while ensuring that resources are kept safe from unwanted access by unauthorized individuals. Coalitions support voluntary interactions between members in pursuit of collective goals. They can be geographically dispersed and fluid in membership. The increasing formation of coalitions is driven by a number of applications including emergency and disaster management, peacekeeping, humanitarian operations or simply in support of common business processes, such as supply chain or joint marketing. With the connectivity available to companies and organizations today, sharing of electronic resources can be relatively easily accomplished through virtual networks over the Internet, the Semantic Web, or via cloud services. However, ensuring the security of the coalition is not as easily enabled. In particular, controlling access dynamically in terms of allowing maximum permissiveness to resources needed to meet the coalition goals while protecting resources with appropriate levels of restrictiveness is an unsolved problem. Access control and authentication mechanisms exist, but they are administratively difficult to maintain. This is particularly the case in a federated environment where the entities involved are not long standing partners. Even when the entities involved have longer-term agreements, there are dynamic requirements that come into play. Different resources may be needed at different times, resources may be added, updated or deleted,

and reasons for collaboration may change. In addition, entities may not want to share openly with any member of a partner's organization. Instead, they are more likely to want to control access to objects by external individuals in a way that corresponds to what they do internally for their own users. That is, a resource should not be accessible

5.1 Identifying Required Role Attributes using Classification

Our approach to identify significant attributes for a role membership relies on classification method. Classification model constructed from the existing data is later used for testing credentials of a new user for a role membership by matching them against the classification model of a role. In real life situations, security administrator doesn't have understanding of various aspects of role configuration. Therefore, it is highly probable that she may pick ill-suited role for a new user for granting any requested permission in RBAC security environment. An ill-suited role is a role through which a user may get *lesser than necessary permissions* or *extra permissions than necessary permissions*. When more than necessary permissions are given to a user, security configuration errors are introduced in the deployed system leaving the security loop holes open for attack by malicious users. On the other hand, if less than necessary permissions are given to a user, the task assigned to a user cannot be completed.Our approach based on classification is to ensure that the addition of a new user to an existing system is in compliance with the access control policies that are governing the deployed system.

In RBAC based security system, user acquire a permission to access a resource via role [27]. In our approach to facilitate coalition based access control, we are assuming that for ensuring the enforcement of security policies, RBAC is already deployed in the resource owning organization. For identifying the set of required user credentials for a role, we first build classification model for each role by using user attribute base(UAB), and role assignment data(UA). UAB and UA are

discussed in section 3.4.1 and in section 3.3. Each role's model is computed by using the UAB of all users regardless of their role membership status. Note that our classification based approach is not sensitive to the choice of classification algorithm. Once the classification model for each role is constructed, these models are then stored and are later used for determining whether a new user qualifies for a particular role or not.

Credentials of a new user are tested against the classification model for a role through which a user can acquire a requested authorization. Generally, to prevent misuse of any organizational resources, established local access control policies allows only those users, to access a particular resource who possess attributes comparable to the attributes of a resource. When a new user is added as a result of coalition formation, organization wants similar policies to be exercised while sharing resources with external users takes place. Therefore, it is important to check that external user also possesses required credentials to access a requested resource.

The overall process to add a new user to existing system involves following three steps:

STEP 1. *Building Classification Model for each Role* This step is executed to build separate classifier for each role $r_m \in R$, where R represents set of roles that are locally deployed within the organization. This step can be executed just once, as shown in 5 All models are then stored for later use. Note that if any changes in the configuration of any role are observed, then the model for that role can be updated either incrementally or by rebuilding a model.

STEP 2. *Selecting Roles Having Requested Permission* When an access request is received from a new user to access particular object, or set of objects, this step is executed to identify set of all roles though which a permission can be granted. Note that we are assuming that RBAC is in place, therefore permissions are acquired by a user through role. All those roles through which a user may acquire a permission to access requested object are stored in $candidateRoles$ list, as

shown in Algorithm 6.

STEP 3. *Classifying external user's attributes for Roles* Though there could be multiple roles through which a permission to access requested object can be granted to a new user, but it is important to ensure that the user should be assigned to a role that he qualifies for. To determine a secure role for a new user, his credentials are checked against the model of each role retrieved in $candidateRoles$. If there is any role in the list of $candidateRoles$ for which he qualifies on the basis of his credentials then that role is assigned to new user. However, if he doesn't qualify for any role in a list of $candidateRoles$, access request would be denied .

The details of three steps are given in following subsections.

5.1.1 Building Classification Models to Identify Required Attributes for Roles

This step is carried out just once. Essentially, we need attributes classification model for each role to begin with. Classification model for all roles are generated and stored for future use at this step. Algorithm 5 shows how this step is executed.

Given a role $r_m \in R$, goal of attributes classification model is to identify combination of attributes which are important for categorizing user into the role r_m. Having an RBAC system in place, local user acquire permission to access any object through role. Thus. if $PermO_j$ is requested by a user, to access object O_j, we assume that there would be atleast one role that has permission to access object O_j. Generally, in RBAC based environment, it is possible to have roles with overlapping permissions. Therefore, if credentials of an external user don't match with required credentials for one particular role then they are tested for remaining $candidateroles$. By $candidateroles$, we mean those roles which contains permission to access requested object.

5.1.2 Selecting Candidate Roles

When an access request of an external user is received, all roles through which a requested permission could be assigned are stored in $candidateRoles$ list. However, out of all $candidateRoles$, user is assigned to a role that has a match with attributes of that user. This matching is done at next step. Recall that $candidateRoles$ has only those roles which contains permission requested, therefore if any role from $candidateRoles$ is assigned to a user automatically a user will acquire requested permission through it. Algorithm 6 shows how this selection is done.

5.1.3 Evaluation of Credentials of an External User

Once the set of $candidateRoles$ is identified, attribute set of external user has to be categorized into a role. If a given request is $au_x, Permobj_y$, where au_x represents credentials of an external user u_x and $Permobj_y$ is the $permission$ requested by a user u_x to access obj_y, user u_x's credentials au_x are checked against the attribute classification models of roles containing permission $Permobj_y$. If au_x qualifies for membership into any role that has permission to access obj_y, then that role is assigned to an external user u_x. Essentially, u_x would get $Permobj_y$ via assigned role. However, if au_x don't qualify for any of the roles having $candidateRoles$, then the request is denied.

Algorithm 5 Building Classifiers

Require: CA, represents Classification Algorithm.
Require: UA, represents column of UA having role assignments for role n.
Require: UAB, represents Users Attributes Base.
1: $RARM_{list} \leftarrow \phi$
2: {Building classifier for each Role in UA}
3: **for** each role r in UA **do**
4: {set column r of UA as $classAttribute$}
5: $RARM_r \leftarrow classifierBuilding(CA, UAB, classAttribute)$
6: $RARM_{list} \leftarrow RARM \cup RARM_r$
7: **end for**
8: **return** $RARM_{list}$

Algorithm 6 Selecting Candidate Roles for New User

Require: $Perm(O_j)$, represents requested permission to access object j.
Require: PA, represents permission role assignments.
1: $candidateRoles \leftarrow \phi$
2: {to discover all roles having permission $PermO_j$ to access object O_j}
3: **for** each role rl in PA **do**
4: **if** rl has $PermO_j$ **then**
5: $candidateRoles \leftarrow rl$
6: **end if**
7: **return** $candidateRoles$
8: **end for**

5.2 Identifying Required Attribute Set using Threshold Value

In this section, we present the basic framework and description of approach to which we are comparing our work with. This approach uses object and user attribute semantics for determining user attribute-value pairs or *candidate attributes of users* ($cau^r_{significant}$) that characterize a role. In our work, we refer to this approach as semantics based approach.

5.2.1 Framework for Semantics-based Approach

In this section we present some preliminary concepts that are unique to semantics-based approach.

Concept Hierarchy

Semantics based approach employ concept hierarchies to *link* the different attributes and compare their values. A concept hierarchy is a graphical notation for representing knowledge using interconnected nodes and arcs. In a concept hierarchy, a more specific concept(s) is represented as a descendant(s) of its more general concept(s).

A concept hierarchy, C_i, consists of a partially ordered ($\prec$) set of concepts. Given two concepts $c_1, c_2 \in C_i$ the following four possible relationships are considered between them: $subClass(c_1) = c_2$ for the relationship where c_2 is a more specialized concept; $eq(c_1) = c_2$ where c_1 and c_2 are

equivalent concepts (synonyms); $sup(c_1) = c_2$ where c_2 is the next more general concept than c_1 (i.e., separated by only one link in the concept hierarchy); and $com(c_1) = (c_2)$ where $sup(c_1) = sup(c_2)$. This latter relationship means that c_1 and c_2 are compatible. Note that several other relationships among concepts are defined in [2]. Discussed approach utilizes only the relationships specified above. Note that the concepts can be either the attribute names or their values.

Figure 5.1 shows an example concept hierarchy for the general concept `software`. From the figure, sup(`Financial`) = `Functions` and eq(`Financial`) = `Fiscal` are examples of relationships. Note that when two concepts are synonyms they are enclosed in a box. Finally, com(`Financial`) = `Display` indicating that these two concepts representing have a common parent and thus represent more specific aspects of the same general concept.

Definition 1. We say an attribute a_i is associated with (or belongs to) a concept hierarchy C_j, denoted as $a_i \rightarrow C_j$ if its value v_i is an included concept in C_j.

It is assumed that every attribute in Λ with a non-numerical attribute value must be associated with one or more concept hierarchies. For example, the values for user attributes `performsJob` and `hasExpertiseIn` are concepts in the concept hierarchy for the concept `software`. Likewise, the values for the object attributes `hasContent` or `hasKeyword` are also concepts in the concept hierarchy for the concept `software`. Note that the degree of specialization of a node increases with its distance from the root. For example in Figure 5.1, `Java Code` is more specialized than `Code` which in turn is more specialized than `software`.

Attribute Linking

For the comparison of two attributes and their associated values, semantics based approach need to know if the attributes use the same vocabulary. It is achieved by associating attribute names with

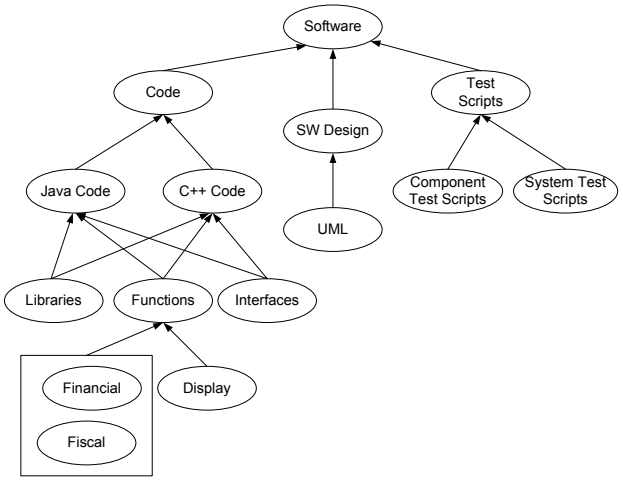


Figure 5.1. Example Concept Hierarchy for Concept `Software`

concept hierarchies where the association indicates the concept hierarchy from which the attribute values are drawn. Attributes are comparable if they are associated with attribute values that are from the same concept hierarchy. Therefore, if any attributes are comparable attributes they are also *linked*.

Semantics based approach requires two types of linking: (1) *referential linking* that links a user attribute name and an object attribute name when associated with the same concept hierarchy; (2) *synonymous linking* that links two user attribute names that are synonyms.

Whether it is referential or synonymous linking, the approach uses $\top(a_i, a_j)$ to represent the linking between two attribute names a_i and a_j. The linking relationship is transitive. That is, if $\top(a_i, a_j)$ and $\top(a_j, a_k)$, then $\top(a_i, a_k)$.

Definition 2. $\top(a_i, a_j)$ is a *referential linking* if a_i is a user attribute and a_j is an object attribute such that $a_i \rightarrow C_i \wedge a_j \rightarrow C_i$.

Definition 3. $\top(a_i, a_j)$ is a *synonymous linking* if both a_i and a_j are user attributes such that $eq(a_i) = a_j$.

Referential linking is used to compare user and object attribute values to extract the necessary attribute-value pairs for a role.

Synonymous linking is used to link a user attribute-value requirement to an attribute-value received from a new user or external user, This is done when when the attribute name in the credential is not exactly that used by the resource owning organization but is a synonym of an attribute name that is used.

5.2.2 Generating Required User Attributes for Roles

This approach assumes that the user should be authorized to access only those objects within the organization which matches semantically with the user attributes. Semantic match *semantic match* is defined as:

Definition 4. [Semantic Match] [User-Object Attribute Semantic Match] There exists a *semantic match* between a user attribute a_i and an object attribute a_j iff $(\top(a_i, a_j)) \wedge ((eq(v_i) = v_j) \vee (subClass(v_j) = v_i))$.

Example 3. For example, the object "Component 1 Software" has the attribute `hasContent` which draws its values from the concept hierarchy for `software` shown in Figure 5.1. A user, Elise, has an attribute `hasExpertiseIn` whose values are also drawn from the concept hierarchy for `software`. Thus, the object attribute and the user attribute are linked. Now suppose the value of the object attribute, `hasContent`, is `functions`. Suppose also that the value of Elise's

`hasExpertiseIn` attribute is `financial`, then there is a semantic match between the two attributes because Elise's attribute value is a subclass of the Component 1 Software's attribute value in the software concept hierarchy. Note that if Elise's attribute value for `hasExpertiseIn` had

also been `functions`, there would have also been a semantic match between the two attributes. However, if Elise's attribute value for `hasExpertiseIn` had been `software`, there would be no semantic match because $subClass$(`function`) $\neq$ `software`. The inclusion of *subClass* is to allow, for example, users with `hasExpertiseIn` in `Java code` to access objects with attributes `Code` or `Software`.

However, there is more to the process than simply performing semantic matches. To determine attribute requirements for a role, the following four step process is executed. This overall process is briefly defined below.

STEP 1. *Discovering User Attributes that are Semantically Related to Object Attributes:*
For each user who is currently a member of the role, consider each of their attribute-value pairs contained in the UAB. This is the *candidate attribute-value pair set* for that user. For each attribute-value pair, determine if there is a semantic match between it and an attribute of any of the objects to which the role has privileges. If there is no semantic match, remove the attribute-value pair from consideration by taking it out of the candidate-value pair set for that user.

STEP 2. *Merging candidate attribute-value pair sets:*
This part of the process involves combining user candidate attribute-value pair sets that are the same or similar. The similarity is determined by finding pairs that have the same attribute names but different attribute values. The values are compared and if there is a relationship as per the concept hierarchy associated with the values, they may be merged.

STEP 3. *Pruning attribute-value pair sets by assessing significance of attribute-value pairs:*
For each candidate attribute value pair set, determine if it is *significant* for that role. An attribute-value pair is significant if it is a characteristic of a high percentage of members of the role and is a characteristic for only a small number of non-members of the role. Only those attribute-value pairs

that are significant for the role are kept for further consideration. The result is a set of candidate attribute-value pair sets that are *significant* to the role.

STEP 4. *Checking Attribute Requirements Across Roles:*
Finally, the candidate attribute-value pair sets of all roles are tested if they obey certain rules when considering the role hierarchy, which involves ensuring that related roles meet certain attribute requirements rules. This is a semi-automated process. Note that new attribute-value pairs may be added during this step.
The details of the four steps are presented in the following subsections.

5.2.3 Semantic Matching of User Attributes to Object Attributes

This step involves extracting and semantically matching user attributes (and their values) for role members to object attributes (and their values) for objects for which the role has some permission. To derive the attribute requirements for membership in role r, we need the set of objects for which permissions are assigned to the role, such objects $assigned\_objects(r)$ and the set of users assigned to the role, $assigned\_users(r)$ as per definition given in section 3.3.

Definition 5. [Candidate Object Attributes] The *candidate object attribute set* for a role r, $coa^r = \{\cup ao_i | o_i \in assigned\_objects(r)\}$.

Essentially, for each of the object $o_i \in assigned\_objects(r)$, all the attributes associated with that objectao_i in OAB are extracted. The union of all the attributes for all the objects in $assigned\_objects(r)$ make up the candidate object attribute set cao^r for role r.

Definition 6. [Candidate User Attributes] The *candidate user attribute set* for user u_i of role r, cau_i^r = $\{a_j | a_j \in au_i \wedge u_i \in assigned\_user(r) \wedge a_j$ has a semantic match to some $o_k \in cao^r\}$.

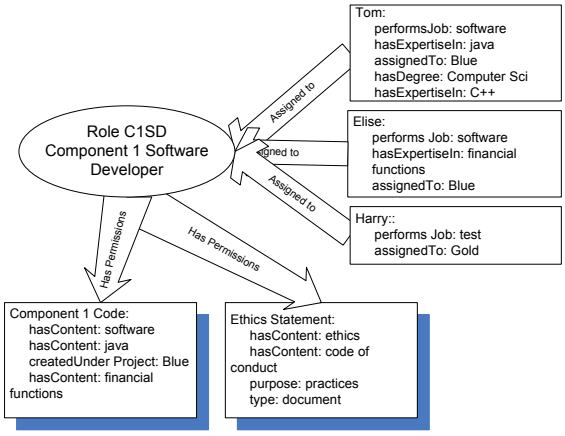


Figure 5.2. Component 1 Software Developer Role, Members, and Permissions

For each of the users $u_i \in assigned\_users(r)$, extract user attribute set, au_i. For each of the attributes $a_i \in au_i$, check whether there is a semantic match (as per Definition 4) between it and any of the object attributes in cao^r. If there is a semantic match, the attribute a_i and the value v_j of the matching object attribute are added to the candidate user attribute requirement set for user u_i under role r, cau_i^r. If a user has no semantic matches to any of the object attributes, that user is flagged since they may be incorrectly assigned to the role. Likewise, if there is an object whose attributes have no semantic matches to any of the users assigned to the role, it is also flagged as an inappropriate object for the role.

Definition 7. [Candidate Role Attributes] The *candidate role attribute set* for role r,

$cau^r = \{\cup cau_i^r | u_i \in assigned\_users(r)\}$

Example 4. Figure 5.2 shows a role, *Component 1 Software Developer*, which is a specialization of the role *Software Developer* (see Figure 5.3). This role has three members, Tom, Elise, and Harry and is assigned permission for all files for *Component 1 code*. The attributes of Tom, Elise, and Harry as well as the `Component 1 Code` object are shown in the figure. When comparing Tom's attributes to those of `Component 1 Code`, there are referential links be-

tween the user attribute performsJob and the object attribute hasContent, the user attribute hasExpertiseIn and the object attribute hasContent, and the user attribute assignedTo and the createdUnderProject object attribute. Therefore, the candidate attribute set when just looking at Tom as the first user assigned to the software developer role would be $cau^{SD} = \{$(performsJob:software), (assignedTo:Blue), (hasExpertiseIn:java)$\}$. Tom's attributes, hasDegree: Computer Science and hasExpertiseIn:C++, were removed from consideration since there is no semantic match between those attributes and attributes of the objects.

For Elise, there is a match between her attribute performsJob and the object attribute hasContent, the user attribute hasExpertiseIn and the object attribute hasContent, the user attribute assignedTo and the createdUnderProject object attribute.

Harry has no attributes which match the attributes of the objects. This should be a concern since it appears that Harry may not belong to the role. Harry's lack of semantically matched attributes should be flagged for further analysis. The empty set is not added as a candidate.

Therefore, the candidate attribute user sets for the *Component 1 Software Developer role (C1SD)* can be expressed as: $cau^{C1SD} = \{\{$(performsJob:software),(assignedTo:Blue), (hasExpertiseIn:java)$\}$; $\{$(performsJob:software), (hasExpertiseIn:financial functions), (assignedTo:Blue)$\}\}$. Note that the ";" represents "or". A set of users in the role have either of the sets of attribute-value pairs.

Now suppose the role *Component 1 Software Developer* also has access to an ethics statement, also shown in the figure. This file has no attributes that are linked to the user attributes of Tom,

Elise or Harry. As such, it does not add or detract from the candidate attribute requirements for the role. However, it can be flagged as a object that may not belong to the role since it does not appear to be semantically related.

5.2.4 Merging Candidate Attribute-Value Pairs

The second step for deriving the user attribute requirements for a role involves merging the sets of unique attribute-value pairs, $as(i)$, contained in cau^r_{all} if there is more than one set of attribute-value pairs. If all the users in the role had exactly the same attribute-value pairs, there would only be one unique set. However, it is very likely that different users have different attribute-value pairs. In Step 1, we have a set of attribute-value pairs of every user who is a member of a role. These sets are reduced to the number of sets such that only members of a role who were assigned for distinct reasons are represented by different sets. This is done through merging of the candidate attribute-value pairs when users have the same attributes but different values for these attributes.

Merger begins by comparing every pair of sets within cau^r_{all}, $as(i)$ and $as(j)$, that have exactly the same attribute names but different attribute values. Let the unequal values be v_i and v_j. If $v_i = sup(v_j)$ or $v_j = sup(v_i)$ then the sets can be merged as one set with the equal pairs being included along with either v_i or v_j respectively. If $com(v_i) = v_j$, and if v_i and v_j represent all the children of $sup(v_i)$, then the value of a_k is replaced with $sup(v_i)$. If v_i and v_j do not represent all the children of $sup(v_i)$, then the value of a_k is replaced with a concatenation of the values v_i and v_j, $(a_k : [v_i, v_j])$ which represents that either v_i or v_j are acceptable values for attribute a_k.

Example 5. An example would be $as(1)$ and $as(3)$ where $as(1) = (a_1 : v_3, a_3 : v_5)$ and $as(3) = (a_1 : v_1, a_3 : v_5)$. Both $as(1)$ and $as(3)$ have the same attributes (a_1 and a_3) but the values of a_1 are different. The merger proceeds pairwise between sets of this type. Let us say that $com(v_3) = v_1$, then $as(1)$ and $as(3)$ can be merged with a new set that includes the multiple

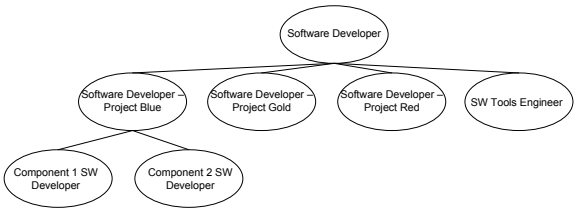


Figure 5.3. Partial Role Hierarchy for LM Systems

values for a_1 that are acceptable. For example, $as(3)$ is removed from cau^r and $as(1)$ is replaced by $as(1) = (a_1 : [v_1, v_3], a_3 : v_5)$.

For example, suppose we have a *Project Manager* role for `Project Blue`, a software project. Some members might be part of that role because they have a college degree and project management experience while others might be part of the role because they have project management certification and have worked on software projects. The attribute and value pairs, `(hasDegree:bachelo` and `(hasExpertiseIn:project management)` are different from `(hasCertificate:Project Management Institute)` and `(hasExpertiseIn:software)` and cannot be combined or merged.

Example 6. For LM, the *Component 1 Software Developer* role, after merger, would have the following attribute sets, cau^{C1SD} = {`(performsJob:software), (assignedTo: Blue), (hasExpertiseIn:[java,financial])`}. The more general *software developer* role (see the partial role hierarchy in Figure 5.3) has the attribute set, cau^{SD} = {`(hasDegree: [bachelors,masters]),(performsJob: software), (assignedTo:[Blue,Gold,Red]), (hasExpertiseIn: [UML,Code])`}.

5.2.5 Pruning the Required Candidate User Attributes by Assessing their Significance

To assess the significance of required candidate user attributes, the significance of each attribute-value set as a whole is considered and each attribute-value pair individually in order to judge what combination of attributes would be the most likely to represent the users who belong in the role and not those who do not have the qualifications of the role. The significance of an attribute-value set $as(i) \in cau^r_{unique}$ can be measured by using the number of users in role r that possess all the attributes and values in $as(i)$ versus the number of users that are not in r but still possess the attributes and values in $as(i)$.

The Attribute-Value Set Significance Factor $\phi_c\ as(i) \in cau^r$, is computed as follows:

$$\phi_c = \frac{(|\psi^r_{as(i)}|/|assigned\_users(r))|}{(|\psi^{\neg r}_{as(i)}|/|(U - assigned\_users(r)|))},$$

where $\psi^r_{as(i)} = \{u_j | u_j \in (assigned\_users(r)) \wedge \forall (a_i : v_i) \in as(i), a_i$ is an attribute of $u_j\}$, and $\psi^{\neg r}_{as(i)} = \{u_j | u_j \in (U - (assigned\_users(r))) \wedge \forall a_i \in as(i) \mid (a_i : v_i)$ is an attribute of $u_j\}$.

Example 7. For the software developer role, let us say that our final merged candidate attribute set is:cau^{SD} = {`(performsJob:software),(assignedTo:[Blue,Gold,Red]), (hasExpertiseI`
LM has 500 employees and 20 are currently assigned to the software developer role. All of those assigned to the role have attributes that match the candidate attribute-value pair set and 4/480 users who are not assigned to the software developer role have the attributes and their required values. The significance factor is thus (20/20)/(4/480) = 120. If we set the threshold value of ϕ_c to 100 (it is a hundred times more likely for a software developer to have the attributes than for someone other than a software developer to have those attributes), then the attribute set is relevant.

Attribute and its Values	ψ_i^r	$\psi_i^{\neg r}$	ϕ_a
`performsJob:software`	20/20	52/480	9.23
`hasExpertiseIn:[UML,Code]`	20/20	55/480	8.73
`assignedTo:[Blue,Gold,Red]`	20/20	130/480	3.69

Table 5.1. The significance factor

5.2.6 Checking Requirements Across Roles

The final step is a post-processing step that evaluates the usability of the required candidate user attributes, $cau^r_{significant}$. This step is to ensure that attribute-value pairs are semantically relevant and significant for the role.

Given a role hierarchy, RH, if two roles, r_i and r_j are such that

r_i is a more specialized role than r_j, then (1) $cau^r_{unique_i} \neq cau^r_{unique_j}$ and (2) the values of attributes held in common in $cau^r_{unique_i}$ and $cau^r_{unique_j}$ must have the relationship $SubClass(v_j) = v_i$ where v_i is an attribute value of a required user attribute for r_i and v_j is an attribute value of a required user attribute for r_j.

The above is a formal way of expressing the following. A member of a more specialized role should have a more specific range of values for an attribute. For example, if an electrical engineering degree is needed to have the role *electrical engineer,* then a stricter requirement that can be imposed for the role *senior electrical engineer* is that a person has a masters degree in electrical engineering.

Example 8. Figure 5.4 shows how the concepts from this step are applied. The more specialized *C1SD* role has more stringent set of required user attributes than that of the `Software Developer - Project Blue` role (i.e., the C1SD Role must have expertise in a specific type of Code - `java` or `financial`). The `SW Developer - Project Blue` role, like-

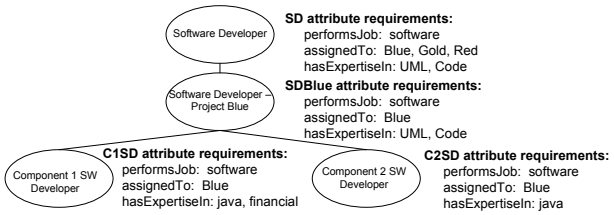


Figure 5.4. Role Attribute Requirement Comparisons

wise, has more stringent required user attributes than the general `SW Developer` role (i.e., the `SDBlue` role requires assignment to Project `Blue`). Thus hierarchically, the attribute requirements are appropriate. However, the two sibling roles, *C1SD* and *C2SD* require reexamination. This is because, the required user attributes for *C2SD* are a subset of those for *C1SD*. Unless it is deemed appropriate that anyone who is in the *C1SD* role with java expertise can also gain access to the *C2SD* role, additional required user attributes must be added for *C2SD*. Let us say that a new attribute is defined to make the *C2SD* the required user attribute set unique. The new required attribute is `hasExpertiseIn:display` because *component 2* includes display functionality. The members of *C2SD* are then rechecked to see if it is appropriate to assign them this attribute and if so the attribute and value are added to the UAB for those users.

5.2.7 Evaluation of Access Requests from a New or External Users

Once required attributes are determined, they can be used to decide whether to grant or disallow access to requested objects. Access decisions are made purely on the basis of the submitted attribute-value pairs.

Example 9. Suppose Lara Werner from HU is sending a request that she would like access to resources of type `UML` that are associated with `Project Blue`. Her request would be presented as follows: ⟨⟨ HU433 ⟩ ⟨ `(objectContains:UML)`, `(project:Blue)` ⟩ ⟨ lwerner, `(title:software engineer), (project:Blue), (expertise:UML)` ⟩⟩.

There are several steps in processing a request:

STEP 1. If the request is made in terms of object attributes, a select query is made to the OAB with the specified attributes and values to determine the objects (if any) whose object attributes match the request. Requested objects, ro, are the set of all objects that meet the request. An object meet the request by having equivalent attribute names to those in the request and associated attribute values which are equal, equivalent, or where the requested value has the subClass relationship with the object value. The roles that have permission to access the object(s) are then identified by locating the objects in $assigned\_objects(r)$.

Example 10. In this example, the object is described by attributes: {`(objectContains:UML),(project:B`
Objects with those attributes are o_{526} and o_{989}. Assuming the only operation allowed on these objects is "read", we consult the permission to role assignment relationship, PA per Definition 3.3, to discover which roles have access to the object(s) that match the request. Both objects o_{526} and o_{989} are assigned to the `SDBlue` role.

STEP 2 Next, the requester's user attributes are examined and compared to the attribute requirements for the role(s), au^r, that have permission to access the requested object(s).

If there are no other roles to test, a denial message is returned to the requester with (optionally) a reason given that the request did not include sufficient credentials.

Example 11. For our example, the role `SDBlue` has the following attribute requirements: {`performsJob:sof`
`assignedTo:Blue, hasExpertiseIn:[Code,UML]`}. Our requester has the attributes
{`title:software engineer,`
`project:Blue, expertise:UML`}.

If the requester does not submit appropriate credentials covering all attribute requirements for access to the requested object, it is assumed that the access is simply denied.

5.3 Experimental Evaluation

In order to compare the performance of both approaches, we conducted an experimental study. This section discusses the details of experimental study. Validation methods that we have employed to study the performance of models are also discussed in this section. Performance evaluation is done by using both real data sets, and the synthetic data sets. Experimental results are evaluated by considering the *F-measure* and *Lift* of the resulting models. We conclude this section with the summary and analysis of results.

The purpose of conducting experimental study was to answer the following questions:

- What is the the overall performance of each of our approaches?
- What is the impact of choosing different threshold values on the performance of our first approach?
- What is the impact of different classification algorithms on the performance of our second approach?
- What is the impact of changing organizational parameters, such as: number of users, number of attributes and number of roles on the performance of our approaches? All experiments were ran on an Intel P-IV machine with 4GB memory and 2GHz dual processor CPU. In order to test the performance of classification based method for facilitating coalition based access control, we are using following three well known classification algorithms:

1. J.48 Decision tree, which is an extension of C4.0 [52],
2. Support Vector Machines(SVMs) [14], and
3. Random Forest [11].

The code for these algorithms are adapted from the Weka machine learning open source repository [65]. Weka is an open source software suite developed at University of Waikato. For SVM

algorithm, we are using libSVM which is a wrapper class of weka [16].

Semantics-based approach is implemented in Java. This approach requires tuning of threshold value for identifying necessary attributes for a role. We ran our experiments with several threshold values. However, in our dissertation, we are including results obtained from models that are built with two threshold values: **0.2** and **0.7**. The reason for not including results of additional models is that we are getting more or less the same range of results.

Validation Methods: To study the quality of produced models, we are using two performance evaluating metrics:

1. *F-measure* and
2. *Lift*.

Brief discussion about these measures is done below.

F-Measure. *F-measure* is a harmonic mean of *precision* and *recall* [50]. Precision corresponds to the number of true positive instances retrieved by the model out of total true instances. Equation 5.1 shows how the precision is computed. Total true instances correspond to the *sum of true positive and false positive instances*. True positive instances are those instances which are correctly classified as belonging to the positive class. On the other hand, false positive instances are those instances which are incorrectly classified as belonging to the positive class.

$$Precision = Rtp/Atp + Afp \tag{5.1}$$

Where,

Rtp = Retrieved number of true positive instances,

Atp = Actual number of true positive instances,

Afp = Actual number of false positive instances.

Recall, on the other hand is the proportion of instances which are actual positives to the number of instances which are correctly identified by the model as positive instances, as given in equation 5.2.

$$Recall = Rtp/Atp + Afn \tag{5.2}$$

Where,

Rtp = Retrieved number of true positive instances,

Atp = Actual number of true positive instances,

Afn = Actual number of false negative instances.

Lift. Lift represents the performance of a particular model in identifying the targeted data instances [65]. It is a ratio between the number of targeted data instances identified when a particular model is employed vs the number of targeted data instances identified when no model is employed. High lift is an indicator of good performance of a model. Lift is computed by first sorting all data instances into descending order of their *probability score*. The *probability score* represents degree of confidence with which an instance is assigned to a particular class. Arranged data is then evenly divided into certain number of chunks(usually 10 chunks). Lift value is then computed for each chunk. Lift value of a chunk is obtained by dividing the number of positive cases within that chunk to the total number of positive cases within the data set. Cumulative lift curve serves as a useful tool for visualizing the performance of a model and is constructed by using *probability scores*. It is generally used to estimate how well the model is at predicting the class than using random predictions alone on the same dataset. On the graph, the x-axis has cumulative number of data instances and y-axis has cumulative number of true positives instances. True positive instances are those instances which are correctly classified by the predictive model.

Real Data sets: We used 4 real data sets. Important characteristics of the real data sets are given in table5.2.

Synthetic Data sets: The synthetic datasets were created using the test data generator from Vaidya et al [60]. The test data generator performs as follows: First a set of roles are created. For each role, a random number of permissions up to a certain maximum are chosen to form the role. The maximum number of permissions to be associated with a role is set as a parameter of the algorithm. Next, the users are created. For each user, a random number of roles are chosen. Again, the maximum number of concurrent roles a user can have is set as a parameter of the algorithm. Finally, the user permissions are set according to the roles to which the user has been assigned. Tables 5.2(a), 5.2(b),5.2(c) gives the characteristics of the datasets created. As the test data creator algorithm is randomized, 5 datasets for each combination of parameters are created, and the results are averaged. Overall, we performed our experiments using 15 synthetic data sets.

Methodology for Creating Synthetic Data Sets. For creating synthetic data sets, first a set of roles are created. For each role, a random number of attributes up to a certain maximum are chosen to form the role. The maximum number of attributes to be associated with a role is set as a parameter to the algorithm. Next, the users are created. For each user, a random number of roles are chosen. Again, the maximum number of concurrent roles a user can have is set as a parameter to the algorithm. Finally, the user attributes are set according to the roles the user has. In some cases, the number of roles randomly chosen is 0 indicating that the user has no roles and therefore no attributes.

Since the test data creator algorithm is randomized, we ran it 5 times on each particular set of parameters to generate the data sets. Both approaches were tested on each of the created data sets. All results reported for a specific parameter set are averaged over the 5 runs.

- *Varying Number of Users with fixed Number of Attributes and Roles*: In the first set of experi-

	numRoles	numUsers	numAttribs
Real Dataset1	121	130	116
Real Dataset2	241	141	494
Real Dataset3	593	358	716
Real Dataset4	158	104	149

Table 5.2. Characteristics of Real Data Sets

ments, we kept the number of attributes and roles constant, while changing the number of users. It is important to note that number of attributes and roles are kept constant to ensure that results of each synthetic dataset within this set of experiment are comparable. Characteristics of data sets for set1 are given in Table 5.2(a).

- *Varying Number of Attributes with fixed Number of Users and Roles*: In the second set of experiments, we kept the number of users and roles constant while varying the number of attributes(and correspondingly, the number of attributes per role). Again, the number of users and roles are kept constant to ensure that results of each synthetic dataset within set2 are comparable. Characteristics of data sets for set2 of experiment are given in Table 5.2(b).

- *Varying Number of Roles with fixed Number of Users and Attributes*: In the third set of experiments, we kept the number of users and number of attributes constant while varying number of roles. Now, the number of users and attributes are kept constant to ensure that results of each synthetic dataset within set3 are comparable. Characteristics of data sets for set3 are given in Table 5.2(c).

Results on real data sets. Data sets are partitioned into training and testing data. Training data has 2/3rd of the data instances and test data consist of remaining 1/3rd of the data instances. Training data is used to build predictive models whereas, test data is used to evaluate the performance of the model. For our first approach, 2 predictive models are built through approach discussed in section 5.1 using two different threshold values: 0.7 and 2.0. For testing the performance of clas-

(a) Parameters for synthetic datasets when varying number of users, keeping everything else constant

	numRoles	numUsers	numAttribs
Synthetic Dataset1	200	300	500
Synthetic Dataset2	200	500	500
Synthetic Dataset3	200	1000	500
Synthetic Dataset4	200	2000	500
Synthetic Dataset5	200	5000	500

(b) Parameters for synthetic datasets when varying number of attributes, keeping everything else constant

	numRoles	numUsers	numAttribs
Synthetic Dataset1	150	1500	100
Synthetic Dataset2	150	1500	250
Synthetic Dataset3	150	1500	500
Synthetic Dataset4	150	1500	1000
Synthetic Dataset5	150	1500	2000

(c) Parameters for synthetic datasets when varying number of roles, keeping everything else constant

	numRoles	numUsers	numAttribs
Synthetic Dataset1	80	1500	500
Synthetic Dataset2	150	1500	500
Synthetic Dataset3	250	1500	500
Synthetic Dataset4	400	1500	500
Synthetic Dataset5	500	1500	500

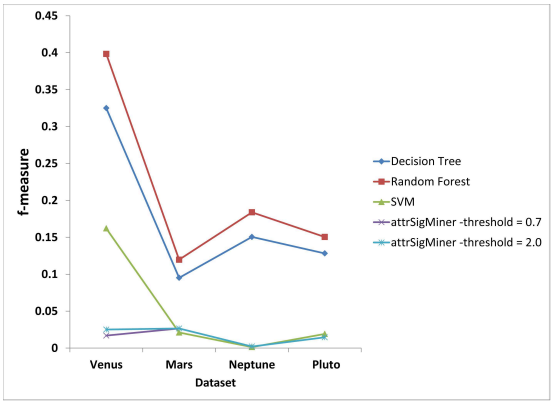
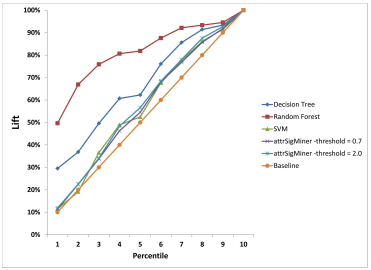


Figure 5.5. Predictive performance in terms of F-measure for real datasets

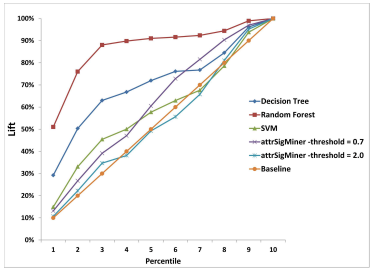
sification based approach, we are using following three classification algorithms: SVMs, Decision Trees and Random Forests, as discussed in section 3.

1) *Performance in terms of F-Measure*: Figure 5.5 shows the performance of obtained models in terms of F-measures. Each model has a separate performance curve on the graph. We observe that the classification based method using Random forest outperforms all other models overall with the minimum F-measure 0.13 and maximum 0.4.

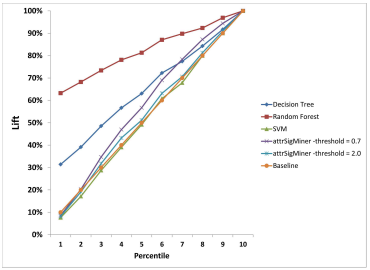
Semantics based method has poor performance on real data sets with no F-measure greater than 0.03. Also, we observe that performance of Semantics based method is actually worst for the larger data sets, whereas it slightly improves for relatively smaller data sets. It is also interesting to note that for real data sets, change in the threshold value used for Semantics based method has little or no impact on the F-measures. On the other hand, for classification based method, choice of classification algorithm has significant effect on the performance. For example, SVM has worst performance. In fact, SVM performs better than semantics based method for just one dataset, but for the remaining three data sets its performance is no better than that of Semantics based method.



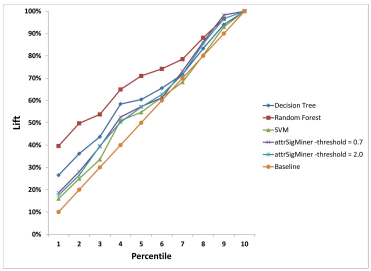
(a) Predictive performance in terms of lift for Real Dataset1



(b) Predictive performance in terms of lift for Real Dataset2



(c) Predictive performance in terms of lift for Real Dataset3

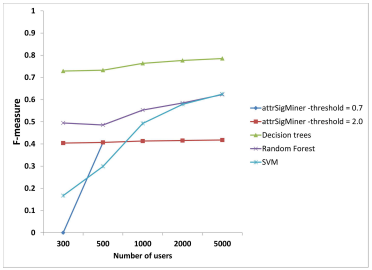


(d) Predictive performance in terms of lift for Real Dataset4

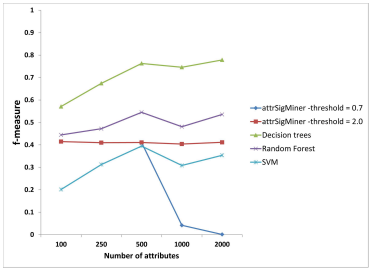
However, the performance of decision trees is close to that of random forests.

2) *Performance in terms of Lift*: Figure 5.6(b), 5.6(c), 5.6(d) show performance of our approaches in terms of lift for real dataset1, real dataset2 and real dataset3, respectively. The area between the baseline curve and the lift curve for any model shows how much a predictive performance can be improved by using that model as opposed to when no model is used. We have separate graph for each real dataset. The lift curves on each graph show that classification based method using Random Forest Algorithm outperforms all other models in terms of lift. Lift curve of a decision tree shows that it is the second best model. The performance of SVM is actually worst among all three classification algorithms that we used.

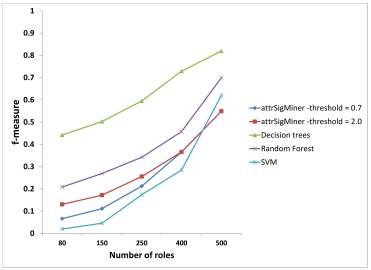
Lift curve for semantics based models shows that these models perform poorly. Their lift curves are also showing that the model that is created with lower threshold value performs slightly better than the one built with higher threshold value.



(e) F-measures, synthetic Data set1



(f) F-measures, synthetic Data set2



(g) F-measures, synthetic Data set3

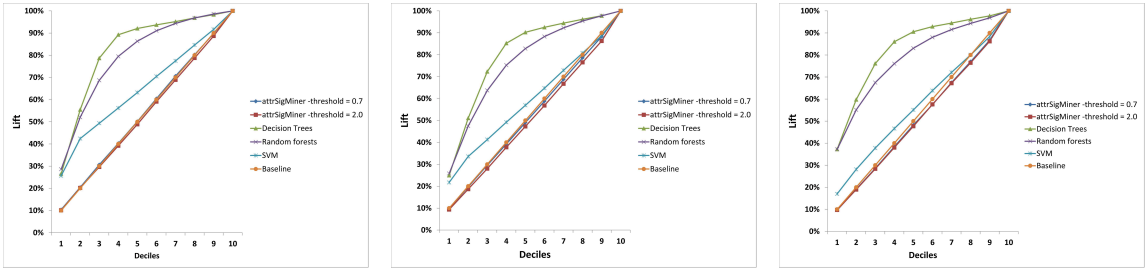
Results on synthetic data sets.

1) *Performance in terms of F-Measure*: Figures 5.6(e), 5.6(f) and 5.6(g) shows performance of models on synthetic data sets in terms of F-measure.

On synthetic data sets, decision tree has overall best performance in terms of F-Measure. Variation in number of users and number of attributes has little or no effect on the performance on Decision tree. However, the change in number of roles affects the performance. The increase in number of roles while keeping other parameters constant improves the performance of Decision tree.

In case of synthetic data sets, Random forest also performs well, and is the second best model in terms of F-measure. Its performance improves with the increase in number of users, while keeping other parameters constant. Its performance also improves with increase in number of attributes while keeping other parameters constant. Finally, increasing the number of roles while keeping everything else constant has a positive impact on the performance of decision tree.

F-measures for semantics based approach are lowest which means it performs poorly on synthetic data sets. Performance of the model built with threshold 0.7 declines significantly when the number of attributes are increased while other parameters are kept constant. Performance of the model built with threshold 2.0 show no change in response to change in number of users. We also observe that the performance of model built with threshold 2.0 is better than that of SVM based model when number of attributes and number of roles are changed, as seen in 5.6(f) and 5.6(g).



(h) Lift curves and baseline for synthetic data 1 (i) Lift curves and baseline for synthetic data 2 (j) Lift curves and baseline for synthetic data 3

Figure 5.6. Performance of models on both real and synthetic data sets

2) *Performance in terms of Lift*: Figures 5.6(h), 5.6(i) and 5.6(j) show performance of models in terms of lift.

Decision tree outperforms all other models in terms of lift. Random forest also performs well and is the second best model. SVM has the worst performance among all the classification algorithms used. Varying parameters has little or no impact on the performance of classification based models.

Lift curves for semantics based approach models show that these models do not perform well. In fact, their lift curve is same as baseline curve which means that results based on these models are no better than results obtained from random predictions.

Summary and analysis of results. Based on the experimental results on both real data sets and synthetic data sets, we demonstrated performance of two methods discussed in this dissertation. The main goal of both approaches is to identify the set of necessary attributes of a user for a role. Results were evaluated in terms of both F-measure and Lift. For the semantic based method, we used two different threshold values to create models based on it. For the classification based method, we used three well known classification algorithms: Support Vector Machines(SVMs), Decision tree, and Random Forests for building classification models. Experimental results indicate that the performance of classification based method is significantly better and consistent as

compared to the performance of semantics based approach. Specifically, Random Forest and Decision tree models are two best performing models to address the problem of coalition based access control. It is interesting to note that decision tree performs better than random forest when synthetic data sets are used, whereas random forest performs better when real data sets are used. Overall the performance of random forest algorithm is quite robust in presence of noise, and decision tree performs better when the data for building classification model is clean. Determining significant attributes through Semantic based approach is quite complex and lengthy process. It involves steps like discovering user attributes that are semantically related to object attributes, forming referential and synonymous linking, and merging candidate attribute-value pair sets. Moreover, the tuning of threshold value is not easy. If the threshold is set to some higher value then very few attributes would be included in the set of necessary attributes for a role. When this identified set of attributes is used to classify new user for a role, false positive errors in the system would increase significantly because many necessary attributes would not be added to a set of significant attributes. Whereas, if the threshold value is set to lower value then many unnecessary attributes would be included in the set of required attributes even when they are not actually significant for a role membership. When this identified set of attributes (which may also include unnecessary attributes) is used to classify new user for a role, false negative errors in the system would increase because many unnecessary attributes would be required from a user to become a member of a role.

Classification based method on the other hand is relatively straightforward and easier to implement. Simply, UAB and UA matrices are used to generate the classification models. No linking of attributes, merging attribute value pairs, or tuning of threshold value are required. More importantly, semantic based approach is rigid in a sense that it returns a fixed set of required attributes for a role. User should have all the required attributes to become a member of a role. On the other hand, Classification model allows flexibility and is based on alternative sets of required attributes.

Moreover, the significance value of each attribute is used to predict the membership status of a new user. If few attributes of lesser significance are missing from new user's attributes set and most of the attributes with higher significance value are present in user attributes set, then the role membership is given through classification method. We believe that this is a realistic way of assessing user credentials for assigning a particular role.

CHAPTER 6

MISCONFIGURATION DETECTION AND RESOLUTION IN ACCESS CONTROL SYSTEMS

6.1 Introduction

Securing organizational resources from any illegitimate access is a prime concern for organizations. Therefore, businesses invest tremendous cost and money to implement an efficient access control (AC) system. AC model is the formal representation of access control policies and their functioning. Access control system provide means to execute those policies. In simple words, access control policy is set of rules which defines *who (subject) can access what (object)*, and access control model bridges the gap between the policies and the mechanism to enforce them. To date, several access control mechanisms and their extensions have been proposed in the literature of information security [32]. Mandatory Access Control (MAC), and Discretionary Access control (DAC), are two classical access control mechanisms. However, in recent years, Role Based Access Control (RBAC) [23] has gained unprecedented prominence and has emerged as one of the most robust security model to meet diverse access control requirements.

RBAC is a policy neutral model. In contrast to traditional MAC and DAC based security systems, RBAC offers significant reduction of administrative overheads and an additional advantage of flexibility. Moreover, it provides the ease of administrating subject and object authorizations via *roles*. Unlike traditional models, where authorizations are granted to user directly, the underlying concept of RBAC is to assign users to a role through which they acquire authorizations which are necessary to complete an assigned job. Enterprises still employing their old access control systems

want to migrate to RBAC. To accomplish the migration, the first phase is to define a good role set. While the role defining problem is seemingly straightforward, it has been recognized as one of the costliest phases in the implementation of RBAC, and poses a great challenge to the system engineers [57]. There are mainly two approaches for identifying set of roles; (i.) *top-down approach*, and (ii.) *bottom-up approach*. Both approaches have their own set of advantages and limitations.

Top-down approach takes description of business function, processes, and other security information into an account while defining roles. While roles are devised using this approach, difficulty comes from the fact that system engineer usually has little knowledge on the semantic meanings of user responsibilities and business processes within an enterprise. Therefore, he can introduce errors in the system configuration through inconsistent assignments. Alternatively, *bottom-up approach* approach a.k.a *role mining* is based on determining roles from existing user-permission authorization data. In bottom-up approach, choice of role mining algorithm is a sensitive aspect and plays a key role in RBAC configuration. Since no role mining method so far is perfect, and that they all come with their own set of benefits and limitations, therefore, it is quite a challenge to choose the role mining method that matches perfectly with the security needs of a particular organization. If not chosen appropriately, it turns entire thing into a self-perpetuating error. Moreover, when ill suited role mining method is adopted, there are high chances that an appropriate set of roles are not constituted. This affects the overall performance of a deployed system negatively, ultimately defeating the sole purpose of implementing access control system.

On the other hand, in DAC, every resource in the system has an owner, and rights to access any resource are explicitly defined by its owner. Unlike MAC, where system administrator solely controls assignments of permissions, DAC is relatively easier to implement, and offers flexibility of delegation of rights i.e. authorized user of a resource can also delegate permission to other users. However, the administrative overhead of maintaining Access control lists (ACLs), especially in

large organizations, is unreasonably burdensome. User obtain an access to any resource through specific permission request, but there are rarely any revocation requests made in this system.

Each access control mechanism and its extension is focused on particular access control sub-problem(s), and offers a unique set of functionalities. Their implementation phase is quite challenging and expensive. But, in order to realize full benefits of these systems, it is critical to ensure that their configuration is correct. Furthermore, once the system variables such as *policies, roles,* and authorizations are defined, continuous monitoring of these systems become vital to ensure that the behavior of implemented system matches with the expected system behavior and that the configuration is up-to-date. Ideally, this maintenance activity should be carried out on regular basis. Unfortunately, however, this issue often gets overlooked.

Given, an access control data, the problem of detecting and resolving misconfigurations refers to the process of identifying those inconsistent assignments in the system which are either mistakenly granted (over-assignment) or denied(under-assignment), and correcting them. This issue has not yet gained due attention from the research community. In most of the related work, it is assumed that the access control data has no noise, existing configuration is good to start with, and defined assignments in the system are consistent with the access control policy of an organization. As a matter of fact, these assumptions are way too unrealistic since access control datasets are no exception and like any other datasets they may possibly contain large amount of noise.

In recent years, configuration errors in access control system has emerged as one of the key causes of system failure. Recently, one of the main services of Google reported configuration errors as second major issue that contributed towards service-level failure [7]. If a system is compromised through erroneous security configurations, organizational resources can be stolen or altered gradually over time, and it can be very expensive (in terms of both money and time) to recover them. Therefore, ensuring that the security settings are configured correctly, and they are audited

frequently is crucial to protect an organizational assets. Traditional access control system lacks an ability to anticipate potential misconfigurations. Therefore, these systems fail to gracefully react to configuration errors. Hence, resulting in to many severe service outages and downtime.

There are many ways in which configuration errors can be introduced in access control system, but human error, and system error are considered as two main causes. Typically, a system administrator(SA) or security officer(SO) is considered responsible for analysis and management of system configurations but their knowledge about access control data is limited as they rely mainly on technical support manuals for making any configuration related decisions. Their job is to ensure that any unauthorized access should be prevented whereas, legitimate access should be granted to the user. But they have little or no understanding of semantics behind the authorizations i.e. why a particular access is given to certain user while some others don't have that access. Therefore, while carrying out the task of assigning or un-assigning authorizations, administrators could make mistakes. For instance, they might add extra permission authorization to a user (over assignment) or they might even sometimes forget to add a legitimate authorizations (under assignment). Moreover, there are situations in which user's role or nature of a job changes and their previously assigned authorization (which are no longer necessary) should be revoked and new set of permissions should be assigned to that user, SA may not perform such tasks efficiently and promptly. Ultimately, importance of configuration management process gets undermined, turning the entire access control system into a completely useless system.

Furthermore, like any other policies, most of the organizational access control policies also evolve over a period of time and the deployed security system must reflect any policy changes that takes place within the organization. It is quite challenging for system administrator to keep the system configurations up-to date in dynamically changing environments. Also, it is extremely error prone for administrators to carry out these tasks manually. In small scale organization, manual

analysis and management is somewhat possible. However, in large scale organizations, where access control data is fairly large in size, it is not feasible to manually perform this task. As a result, either the change remains undefined in the system for a long period of time or re-engineering of a whole new set of rules or a model becomes essential. Frequent re-building of configuration model is not a pragmatic solution because model building and its implementation are highly expensive procedures especially, when changes are significant, and rapid. If left unaddressed, these type of issues can cause serious configuration problems and make sensitive organizational resources vulnerable to serious security threats. Therefore, an acute need of better ways for configuration management of access control systems has arisen. In one of the benchmark study performed on reliability of systems, Jim Gray [28] notes,

"The top priority for improving system availability is to reduce administrative mistakes by making self-configured systems with minimal maintenance and minimal operator interaction. Interfaces that ask the operator for information or ask him to perform some function must be simple, consistent and operator fault-tolerant"

In following sections, we present approaches to automate the process of configuration management mainly in RBAC and DAC based security systems. Our approaches rely mainly on data mining method known as classification. We are integrating classification outcome of multiple classifiers to detect any configuration errors. The main reason for using output of multiple classifiers is to leverage strength of each classifier, and to reduce the impact of classification errors made by an independent classifier.

6.2 Misconfiguration Detection and Resolution

In this section, we give more formal description of the approaches for facilitating automation of misconfiguration detection and resolution process in deployed access control systems. Our first

approach, ***Misconfiguration detection and resolution in role authorization***, considers that role based access control(RBAC) is already in-place within an organization. Therefore, user-to-role authorization (UA) data, and permission-to-role (PA) authorization data is maintained by the organization. Moreover, user-permission authorization matrix (UPA) in this situation can be derived by using both (UA) and (PA) matrices. Our second approach, ***Misconfiguration detection and resolution in permission authorization***, considers simply Access control Lists ($ACLs$) maintained by an organization. ACL mainly contain existing user permission authorizations (UPA) data. For both of our approaches, we further assume that the authorization data is maintained in a form of boolean matrices.For misconfiguration detection, output of multiple classifiers is used. If more than n number of classifiers classify a user's assignment to a particular role or a permission, then the assignment is considered as a $potentialmisconfiguration$. Misconfiguration score is then computed for the potentially misconfigured assignment by using the output of multiple classifiers. On the basis of computed misconfiguration score, a decision regarding the addition or revocation of role or a permission authorizations is done.

6.2.1 Misconfiguration detection and resolution in role authorization

1a. Given a role authorization $(U_i, R_x, \{R_{i_1}, \ldots, R_{i_m}\})$, where R_X is the role that the user does not have authorization, and $\{R_{i_1}, \ldots, R_{i_m}\}$ is the set of roles for which U_i has the authorization. The role authorization is granted and R_x therefore, is added to the existing roles of a user such that:

- U_i is the user within a system who should potentially have an additional role authorization(under-assigned user);
- $(R_x) \notin Roles(U_i)$;

- $(R_x) \in UA$
- Computed misconfiguration score for the (U_i, R_x) assignment is greater than the specified threshold value.

Based on the computed misconfiguration score, role R_x is therefore, added to the existing set of roles (Roles(U_i)) of a user U_i.

1b. Given a role authorization $(U_i, R_j, \{R_{i_1}, \ldots, R_{i_m}\})$, where R_j is an over assignment and $\{R_{i_1}, \ldots, R_{i_m}\}$ is the set of roles for which U_i has the authorization. The role authorization is revoked from the existing roles of a user such that:

- U_i is the user with in a system who has potentially an extra role authorization(over-assigned user);
- $(R_j) \in Roles(U_i)$;
- $(R_j) \in UA$
- Computed misconfiguration score for the (U_i, R_j) assignment is greater than the specified threshold value.

Based on the computed misconfiguration score, role R_j is therefore, removed from the existing set of roles(Roles(U_i)) of a user U_i.

6.2.2 Misconfiguration detection and resolution in permission authorization

2a. Given a permission authorization $(U_i, P_x, \{P_{i_1}, \ldots, P_{i_m}\})$, where:

- U_i is the user with in a system who may potentially need an additional permission(under-assigned user);

- P_x is the permission which is not available to U_i; and
- $\{P_{i_1}, \ldots, P_{i_m}\}$ is the set of permissions for which U_i currently has the authorization.
- Computed misconfiguration score for the (U_i, P_x) assignment is greater than the specified threshold value.

Based on the computed misconfiguration score, permission P_x is added to the existing permission set(Perms(U_i)) of the user U_i.

2b. Given a permission authorization $(U_i, P_j, \{P_{i_1}, \ldots, P_{i_m}\})$, where:

- U_i is the user with in a system who may potentially has an extra permission(over-assigned user);
- P_j is the permission which is should not be available to U_i; and
- $\{P_{i_1}, \ldots, P_{i_m}\}$ is the set of permissions for which U_i currently has the authorization.
- Computed misconfiguration score for the (U_i, P_j) assignment is greater than the specified threshold value.

Based on the computed misconfiguration score, permission P_x is removed from the existing permission set(Perms(U_i)) of the user U_i.

Below we discuss more details of our proposed approaches. In order to illustrate the procedure, we will use the user assignment (UA) matrix, permission assignment (PA) shown in figure (6.1), and figure (6.2), specifically when ($RBAC$) based security environment is assumed, and derived user permission assignment (UPA) matrix, as shown in figure (6.3). For our second approach, $Misconfigurationdetectionandresolutioninpermissionauthorization$, the same UPA matrix which is shown in figure (6.3) is used for the purpose of illustration of procedure.

Misconfiguration Detection and Resolution in Role Authorization

In this case, we assume that ($RBAC$) is deployed within the organization. As mentioned in section 6.2, our approach to detect misconfigurations in a deployed system relies on classification method. Essentially, multiple Classification models are constructed by using access control matrices and their outcome are integrated to make configuration decisions. Once the classifications models are built, they are stored and are later used for testing purpose of the current and new configurations of the access control system. Below we discuss the details of the entire process that we We use UA, and UPA matrices depicted in figure 6.1, 6.3 to illustrate the approach.

Multiple classification models are built for each role which is defined within the deployed access control system. Algorithm 7 shows how they are built. After the model building and storing step, assignment is tested if it is potentially a misconfiguration. Misconfiguration Score is then computed for any assignment that qualifies as a potential misconfiguration. For instance, if assignment $R_n(U_m)$ is under the consideration, where R_n represents a role which a user U_m may or may not have. To check whether the availability or unavailability of this role to a user is correct or not, misconfiguration detection process gets activated. This is a two step process. Firstly, p percent or more classifiers must predict that the existing class value is erroneous. If this condition is satisfied, then the assignment is considered as $potentialmisconfiguration$. Else, it is left unchanged. This step is shown in algorithm 8. Assignment which is identified as $Potentialmisconfigurations$ is further analyzed by computing a misconfiguration score. This score is based on the probability

	R1	R2	R3	R4	R5
U1	0	1	0	1	0
U2	1	0	1	0	0
U3	0	0	0	1	1
U4	0	0	0	1	0
U5	1	0	0	0	0
U6	0	1	1	1	0
U7	0	0	1	0	1
U8	1	0	0	0	1
U9	0	1	0	0	0
U10	0	0	0	0	1

Figure 6.1. UA matrix for 10 users and 5 roles

	P1	P2	P3	P4	P5
R1	1	1	0	0	0
R2	0	0	1	0	0
R3	1	1	1	0	0
R4	0	0	1	1	0
R5	1	0	0	0	1

Figure 6.2. PA matrix for 5 permissions and 5 roles

outcome of multiple classifiers with which each classifier predicts the class attribute of a user. This step is shown in algorithm 9. In this particular case, by class attribute, we mean presence or absence of a role in a given set of roles of a user. If the computed misconfiguration score is greater than the pre-specified threshold value, this assignment is considered as an $actualmisconfiguration$, and for any assignment which is identified as an $actualmisconfiguration$, step for its resolution is performed as shown in algorithm 10.

Misconfiguration resolution step handle false positive cases and false negative cases differently. If a misconfiguration is due to false positive assignment (user is not supposed to have that role but due to an error it is assigned to a user), all the permissions associated with that role are removed from permission set of a user except those which the user acquired via roles. On the other hand, if a misconfiguration is due to false negative assignment (user is supposed to have that role but due to an error it is not assigned to a user), all the permissions associated with that role are added to the permission set of the user. Overall process of misconfiguration detection and resolution has four steps. Description of each step along with illustrative example is given below.

	P1	P2	P3	P4	P5
U1	0	0	1	1	0
U2	1	1	1	0	0
U3	1	0	1	1	1
U4	0	0	1	1	0
U5	1	1	0	0	0
U6	1	1	1	1	0
U7	1	1	1	0	1
U8	1	1	0	0	1
U9	0	0	1	0	0
U10	1	0	0	0	1

Figure 6.3. UPA matrix for 10 users and 5 permissions

Step 1: ***Building n classifiers for each role.***

Our approach to automate misconfiguration detection process utilizes classification outcome of multiple classifiers. At first step, model building process takes place. Mainly at this step x number of classification models are constructed for each role. Algorithm 7 shows how it is done. This step is typically carried out at the start, though the models constructed at this stage may be refined (either incrementally or completely recomputed) if the system state changes significantly. List of classification algorithms is given as an input to construct multiple models. Once they are built for each role, these models are then stored and utilized for future misconfiguration detection.

For building a classification model of a role R_x, access control matrices UPA and UA are used as an input to the classification algorithm. When any instance or data point is tested against the classification model, it returns the predicted *class attribute value*, and the *probability*. *Class attribute* in our case is a $role$, say R_x, and the *probability* outcome is the confidence with which a classifier decides whether a user should have role R_x or not. Since our approach requires output of multiple classifiers, therefore, if user U_i's role assignment R_jx needs to be checked, then the outcome of x number of classifiers for R_j is needed.

Step 2: ***Identifying potential misconfiguration.***

At this step, role assignment of a user, say user U_i, is tested for possibility of erroneous role assignment. Suppose, we want to perform assessment of user U_i's assignment to role R_x. At step 2, this assignment (U_i, R_x) will either be classified as non-erroneous assignment or as a potential misconfiguration. Algorithm 8 shows how this is done.

At this step, classification models constructed at step 1 for role R_x are retrieved. Essentially, we can treat the given role assignment of user as a new instance to be classified. Classification model make a decision based on the existing permission set of U_i. As an output, each model returns the

Algorithm 7 $BuildingAllClassificationModels()$

Require: Matrix ≡ User-Permission assignment matrix, UPA.
Require: Matrix ≡ User-role authorization matrix, UA.
Require: Set of roles defined within the deployed access control system, $ROLES$.
Require: List of Classification Algorithms, CA_{list}.

```
ROLES(CMS_list) ← {}
for each R_x ∈ ROLES do
    CM_list(R_x) ← φ
    for each CA_i ∈ CA_list do
        Denote the column for R_x in UA as the Class Attribute.
        CM_CA_i(R_x) ← buildClassifier(CA_i, UPA)
        CM_list(R_x) ← CM_list(R_x) ∪ CM_CA_i(R_x)
    end for
    ROLES(CMS_list) ← ROLES(CMS_list) ∪ CM_list(R_x)
end for
Return (ROLES(CMS_list))
```

predicted class (in terms of Class Attribute Value), and probability with which user U_i is assigned to that class by a classifier. In our case, *Class Attribute Value* can be either 0 or 1; where, 0 means a user should not have role R_x, and 1 means a user should have role R_x.

At step 2, if $p\%$ or more classifiers predicts that the existing assignment is an error then $R_x(U_i)$ is considered as a potential misconfiguration. For instance, if initial $UA\{R_i, U_i\}$ = 0, $p = 50$, and 3 out of 5 classification models decides that U_i should have R_x (i.e. $UA\{R_i, U_i\}_{predicted} = 1$), then R_x is a potential misconfiguration. Algorithm 8 returns the potentially misconfigured assignment and the misconfiguration status. On the other hand, if less than $p\%$ predicts a class assignment different from the initial assignment, then no changes are needed for such assignment.

Step 3: ***Identifying actual misconfiguration.***

User to Role assignment which is identified as potential misconfiguration at step 2 is tested further to determine whether it is an actual misconfiguration or not. Therefore, misconfiguration score is computed for the user-role assignment identified as potential misconfiguration at previous

Algorithm 8 $potentiallyMisconfiguredRole(U_i, R_x)$

Require: Role assignment of a user, $R_x(U_i)$.
Require: Matrix $\equiv$ User-role authorization matrix, UA.
Require: Permission authorizations of $U_i.Perms(U_i)$.
Require: List of classification models constructed using algorithm 7, $ROLES(CMS_{list})$.
Require: List of classification algorithms, CA_{list}.
Require: Pre-specified threshold percentage of minimum number of classifiers having same class attribute decision, $p\%$.

$CMS(R_x) \leftarrow$ Retrieve classification models for role R_x from $ROLES(CMS_{list})$
$countErrors = 0$
$notErrors = 0$
for each $CA_i \in CA_{list}$ **do**
  $classAttrVal \leftarrow CA_i(CM_{CA_i}(R_x), Perms(U_i), R_x)$
  if $classAttrVal$ = = $UA\{U_i, R_x\}$ **then**
    $noErrors+ = 1$
  else if $classAttrVal$!= $UA\{U_i, R_x\}$ **then**
    $countErrors+ = 1$
  end if
end for
$potentialMisconfigScore = \dfrac{countErrors}{CMS(R_x).count} * 100$
if $potentialMisconfigScore >= p\%$ **then**
  potentialMisconfigStatus = TRUE
  return (potentialMisconfigStatus, $R_x(U_i)$)
else
  return 0
end if

step. If this score is greater than or equal to the predefined threshold, then the assignment is considered as an actual misconfiguration. However, if the computed misconfiguration score for any assignment is less than the predefined threshold value, that assignment does not qualify as a misconfiguration and therefore, no modifications are needed for it. The process of classifying any assignment as an actual misconfiguration is given in algorithm 9.

Algorithm 9 $MisconfiguredRole(R_x, U_i)$

Require: Role assignment identified as a potentially misconfigured assignment in algorithm 8, $R_x(U_i)$.
Require: Matrix $\equiv$ User-role authorization matrix, UA.
Require: Permission authorizations of U_i. $Perms(U_i)$
Require: List of classification models constructed using algorithm 7, $ROLES(CMS_{list})$.
Require: List of classification algorithms, CA_{list}.
Require: θ, pre-specified threshold value.
Require:
$CMS(R_x) \leftarrow$ *Retrieve classification models for role* R_x *from* $ROLES(CMS_{list})$
R_xScore = 0
for each $CA_i \in CA_{list}$ **do**
$R_xPCP \leftarrow CA_i(CM_{CA_i}(R_x), Perms(U_i), R_x)$
$classAttrVal_{predicted} \leftarrow CA_i(CM_{CA_i}(R_x), Perms(U_i), R_x)$
if $classAttrVal_{predicted}$ = = $UA\{U_i, R_j\}$ **then**
R_xScore += $1 - R_xPCP$
else
R_xScore += R_xPCP
end if
end for
$R_xAvgScore = \dfrac{R_xScore}{CMS(R_x).count}$
if $R_xAvgScore \geq \theta$ **then**
Misconfiguration Status = TRUE
return (Misconfiguration Status,$R_x(U_i)$)
else
Leave the assignment unchanged.
end if

Misconfiguration Score is based on the probability with which each classifier decides whether a role should be assigned to a user or not. If a decision made by a classifier is not the same as

that of an actual role assignment status, probability with which a classifier assigns a different class value is counted towards the computation of misconfiguration score. Once the scores from each classifier are obtained, their average is computed to obtain the actual misconfiguration score. If this score is greater than or equal to the predefined threshold value, then the class attribute value for that assignment is considered as configuration error.

Step 4: ***Resolving misconfiguration.***

This is the final step of the process where the detected misconfiguration is resolved from the existing access control system. As discussed earlier, we assume that there are two possible type of configuration errors in access control system; False positive assignment, and false negative assignment. Both cases are handled differently by our approach. Algorithm 10 shows how the resolution of misconfiguration of each of these errors is done. Since, we are considering RBAC, therefore, if a user-role assignment identified by 9 falls under the category of false positive assignment, then all the permissions associated with the identified role are removed from the permission authorizations of a user, except those permissions which a user has acquired through his other roles. Note that the permissions associated with a particular role are obtained from PA matrix. On the other hand, if an identified user-role assignment falls under the category of false negative assignment, all the permissions associated with that role are added to the existing permission authorizations of a user. At the end of this step, permission authorization set and the role status gets updated in the existing system.

Misconfiguration Detection and Resolution in Permission Authorization

We now discuss the case where we assume that the access control is implemented through (DAC)/ACLs. Our approach to detect misconfigurations requires to build multiple classification models and utilize their outcome to classify an assignment for a misconfiguration status. These classifiers are

Algorithm 10 $resolveMisconfiguration(R_x, U_i)$

Require: Matrix $\equiv$ User-role authorization matrix, UA.
Require: Misconfigured role assigment of a user U_i identified in algorithm 9, $R_x(U_i)$
Require: Existing roles of a a user U_i, $Roles(U_i)$.
Require: Existing permissions of a a user U_i, $Perms(U_i)$.
Require: Permissions associated with role R_x. $Perms(R_x)$.

if $R_x(U_i) \in Roles(U_i)$ **then**
  $Perms(U_i)_{updated} \leftarrow Perms(U_i) - Perms(U_i) \cap Perms(R_x)$
  $Roles(U_i)_{updated} = Roles(U_i) - R_x$
else if $R_x(U_i) \notin Roles(U_i)$ **then**
  $Perms(U_i)_{updated} \leftarrow Perms(U_i) \cup Perms(R_x)$
  $Roles(U_i)_{updated} \leftarrow Roles(U_i) \cup R_x$
end if
return ($Perms(U_i)_{updated}$,$Roles(U_i)_{updated}$)

constructed for each permission. UPA matrix is used as an input to the classifier and the column for the permission for which classifiers are constructed is treated as a *class attribute*. Once all the models are constructed for each permission, they are then stored for later use. Resolution of misconfiguration is relatively easy for these systems. If any permission authorization is detected as a misconfiguration, only that particular authorization is changed in the UPA. Below we give the details of a process.

Step 1: ***Building n classifiers for each permission***

At step 1, n classifiers are constructed for each permission. Algorithm 11 shows how this is done. This step requires UPA matrix, and list of classification algorithms as an input. Classifiers for each permission are stored in a list from where they can be retrieved later for testing the assignments. This step is typically carried out at the start, though the models constructed at this stage may be refined (either incrementally or completely recomputed) if the system state changes significantly.

Step 2: ***Identifying potential misconfiguration***

Algorithm 11 $BuildingClassificationModelsPerms()$

Require: Matrix ≡ User-Permission assignment matrix, UPA.
Require: List of Classification Algorithms, CA_{list}.
$Perms(CMS_{list}) \leftarrow \{\}$
for each $P_j \in PERMS$ **do**
$CM_{list}(P_j) \leftarrow \phi$
for each $CA_i \in CA_{list}$ **do**
Denote the column for P_j in UPA as the *Class Attribute*.
$CM_{CA_i}(P_j) \leftarrow buildClassifier(CA_i, UPA)$
$CM_{list}(P_j) \leftarrow CM_{list}(P_j) \cup CM_{CA_i}(P_j)$
end for
$Perms(CMS_{list}) \leftarrow Perms(CMS_{list}) \cup CM_{list}(P_j)$
end for
Return ($Perms(CMS_{list})$)

At step 2, an assignment will either be classified as non-erroneous assignment or as a potential misconfiguration. Algorithm 12 shows this process. Suppose in a given UPA, there is an assignment $UPA\{U_i, P_j\}$ that could possibly be a misconfiguration. At this step, all the classifiers constructed at step 1 for this permission would be retrieved. Basically, we can treat the given permission assignment of user as a new instance to be classified. Each classification model for permission P_j would make a decision based on the existing permission set of U_i. As an output, each model returns the predicted class (in terms of Class Attribute Value), and probability with which user U_i is assigned to that class by a classifier. In our case, *Class Attribute Value* can be either 0 or 1; where, 0 means a user should not have a permission P_j, and 1 means a user should have permission p_j.

If $p\%$ or more classifiers predicts that the existing assignment is an error then $P_j(U_i)$ is considered as a potential misconfiguration. For instance, if initial $UPA\{P_j, U_i\}$ = 0, $p = 50$, and 3 out of 5 classification models decides that U_i should have P_j (i.e. $UA\{P_j, U_i\}_{predicted} = 1$), then P_j is considered as a potential misconfiguration. On the other hand, if less than $p\%$ predicts a class assignment different from the initial assignment, then no changes are needed for such assignment.

Algorithm 12 $potentiallyMisconfiguredPerm(U_i, P_j)$

Require: Permission authorizations of U_i. $Perms(U_i)$.
Require: List of classification models constructed using algorithm 11, $Perms(CMS_{list})$.
Require: List of classification algorithms, CA_{list}.
Require: Pre-specified threshold percentage of minimum number of classifiers having same class attribute decision, $p\%$.

$CMS(P_j) \leftarrow$ Retrieve classification models for permission P_j from $Perms(CMS_{list})$
$countErrors = 0$
$noErrors = 0$
for each $CA_i \in CA_{list}$ **do**
  $classAttrVal \leftarrow CA_i(CM_{CA_i}(P_j), Perms(U_i))$
  if $classAttrVal$ = = $UPA\{U_i, P_j\}$ **then**
    $noErrors+ = 1$
  else if $classAttrVal$!= $UPA\{U_i, P_j\}$ **then**
    $countErrors+ = 1$
  end if
end for
$potentialMisconfigScore = \dfrac{countErrors}{CMS(P_j).count} * 100$
if $potentialMisconfigScore >= p\%$ **then**
  potentialMisconfigStatus = TRUE
  return (potentialMisconfigStatus, $P_j(U_i)$)
else
  return 0
end if

Step 3: ***Identifying actual misconfiguration.***

User-permission assignment which is identified as potential misconfiguration at step 2 is tested further to determine whether it is an actual misconfiguration or not. Therefore, misconfiguration score is computed for an assignment which is identified as potential misconfiguration at previous step. If this score is greater than or equal to the predefined threshold, then the assignment is considered as an actual misconfiguration. However, if the computed misconfiguration score for any assignment is less than the predefined threshold value, that assignment does not qualify as a misconfiguration and therefore, no modifications are needed for it. The formal specification of classifying any assignment as an actual misconfiguration is given in algorithm 13.

Step 4: ***Resolving misconfiguration.***

As discussed earlier, we assume that there are two possible type of configuration errors in access control system; False positive assignment, and false negative assignment. Both cases are handled differently by our approach. Algorithm 14 shows how the resolution of misconfiguration of each of these errors is done. Since, we are considering permission based authorization system, therefore, if a user-permission assignment identified by 13 falls under the category of false positive assignment, then that permission is removed from the permission authorizations of a user and UPA is updated accordingly. On the other hand, if an identified user-permission assignment falls under the category of false negative assignment, that permission is added to the existing permission authorizations of a user, and UPA is updated accordingly.

6.3 Experimental Evaluation

We have performed extensive experimentation for evaluating the performance of our proposed approach to automate the process of misconfiguration detection in the deployed RBAC based se-

Algorithm 13 $MisconfiguredPerm(P_j, U_i)$

Require: Permission authorization identified as a potentially misconfigured assignment in algorithm 12, $P_j(U_i)$.
Require: Permission authorizations of U_i. $Perms(U_i)$
Require: List of classification models constructed using algorithm 11, $Perms(CMS_{list})$.
Require: List of classification algorithms, CA_{list}.
Require: θ, pre-specified threshold value.
Require:
$CMS(P_j) \leftarrow$ Retrieve classification models for permission P_j from $Perms(CMS_{list})$
P_jScore = 0
for each $CA_i \in CA_{list}$ **do**
$P_jPCP \leftarrow CA_i(CM_{CA_i}(P_j), Perms(U_i))$
$classAttrVal_{predicted} \leftarrow CA_i(CM_{CA_i}(P_j), Perms(U_i), P_j)$
if $classAttrVal_{predicted}$ = = $UPA\{U_i, P_j\}$ **then**
P_jScore += $1 - P_jPCP$
else
P_jScore += 0
end if
end for
$$P_jAvgScore = \frac{P_jScore}{CMS(P_j).count}$$
if $P_jAvgScore \geq \theta$ **then**
Misconfiguration Status = TRUE
return ($P_j(U_i)$)
else
return 0
end if

Algorithm 14 $resolveMisconfigurationPerm(P_j, U_i)$

Require: Existing permissions of a a user U_i, $Perms(U_i)$.
Require: permission assignment identified as actual misconfiguration in algorithm 13, $P_j(U_i)$.
if $P_j(U_i) \in Perms(U_i)$ **then**
$Perms(U_i)_{updated} \leftarrow Perms(U_i) - P_j$
else if $P_j(U_i) \notin Perms(U_i)$ **then**
$Perms(U_i)_{updated} \leftarrow Perms(U_i) \cup P_j$
end if
return ($Perms(U_i)_{updated}$)

curity system. All experiments were ran on an Intel P-IV machine with 4GB memory and 2GHz dual processor CPU. In order to test the performance of classification based method for facilitating coalition based access control, we are using following four state-of-the-art algorithms in machine learning:

1. J.48 Decision tree, which is an extension of C4.0 [52],

2. Naive Bayes [**?**],

3. Random Forest [11], and

4. Rotation Forest [**?**].

The code for these algorithms are adapted from the Weka machine learning open source repository [65].The experimental implementation uses Weka for model building purpose, and computing misconfiguration score for authorizations data.

An outcome of classification make each assignment fall under one of the following four categories:

- **True Positive (TP)**. This is a case when actual class attribute value of tested instance is ”1” and the classifier also predicts it as ”1”.

- **True Negative (TN)**. This is a case when actual class attribute value of tested instance is ”0” and the classifier also predicts it as ”0”.

- **False Positive (FP)**. This is a case when actual class attribute value of tested instance is ”0” and the classifier predicts it as ”1”.

- **False Negative (FN)**. This is a case when actual class attribute value of tested instance is ”1” and the classifier predicts it as ”0”.

False Positive and False Negative assignments are considered as an error in the access control data, therefore, we evaluate how well our approach works in detecting these two type of errors.

Validation Methods:

Performance of our approaches is done by using three type of measures: (1) Recall, (2) Precision, and (3) F measure. ***F-Measure***. *F-measure* is a harmonic mean of *precision* and *recall* [50]. Precision corresponds to the number of true positive instances retrieved by the model out of total true instances. Equation 6.1 shows how the precision is computed. Total true instances correspond to the *sum of true positive and false positive instances*. True positive instances are those instances which are correctly classified as belonging to the positive class. On the other hand, false positive instances are those instances which are incorrectly classified as belonging to the positive class.

$$Precision = Rtp/Atp + Afp \tag{6.1}$$

Where,

Rtp = Retrieved number of true positive instances,

Atp = Actual number of true positive instances,

Afp = Actual number of false positive instances.

$$Recall = Rtp/Atp + Afn \tag{6.2}$$

Where,

Rtp = Retrieved number of true positive instances,

Atp = Actual number of true positive instances,

Afn = Actual number of false negative instances.

Recall, on the other hand is the proportion of instances which are actual positives to the number of instances which are correctly identified by the model as positive instances, as given in equation 6.2.

	numRoles	numUsers	numPermissions
Data 1	15	100	80
Data 2	15	250	80
Data 3	15	350	80
Data 4	15	500	80

Table 6.1. Characteristics of Dataset 1. Varying Number of Users, Keeping Everything Else Constant.

Datasets

Availability of real life data for access control is not easy. When an availability of representative data is a problem or when their properties are hard to modify for testing purpose, synthetic data becomes an appealing alternative. Therefore, we use synthetic datasets to study how the performance of our proposed approach is affected by varying organizational, and other relevant parameters. Synthetic datasets were created using the test data generator *Create Test Data* from Vaidya et al [61]. The test data generator performs as follows: First a set of roles are created. For each role, a random number of permissions up to a certain maximum are chosen to form the role. The maximum number of permissions to be associated with a role is set as a parameter of the algorithm. Next, the users are created. For each user, a random number of roles are chosen. Again, the maximum number of concurrent roles a user can have is set as a parameter of the algorithm. Finally, the user permissions are set according to the roles to which the user has been assigned. Table 6.1, 6.2,6.3 gives the characteristics of the datasets created. The effect of large number of users, permissions, and varying densities are studied. As the test data creator algorithm is randomized, 5 datasets for each combination of parameters are created, and the results are averaged. Overall, we performed our experiments using 12 synthetic data sets.

Additional Parameters

For the potential misconfiguration detection step, we use $p = 50\%$ (minimum percentage for

	numRoles	numUsers	numPermissions
Data 1	80	250	50
Data 2	80	250	80
Data 3	80	250	110
Data 4	80	250	150

Table 6.2. Characteristics of Dataset 2. Varying Number of Permissions, Keeping Everything Else Constant

	numRoles	numUsers	numPermissions
Data 1	10	150	50
Data 2	20	150	50
Data 3	25	150	50
Data 4	30	150	50

Table 6.3. Characteristics of Dataset 3. Varying Number of Roles, Keeping Everything Else Constant

number of classifiers agreeing on a particular decision). We also study how the choice of threshold at actual misconfiguration detection step would impact the performance of our approach. Essentially, this threshold value is used to decide regarding the confidence with which each classifier assigns a class attribute value to a user. We use following 3 values for the threshold: 0.5, 0.7, and 0.9. When 0.5 is used as a threshold value, it means even if a classifier's confidence level is 50%, its probability output would still be counted towards computation of misconfiguration score. Whereas, choice of 0.9 for the threshold value means that the probability output of the classifier will not be counted towards the computation of final misconfiguration score unless its confidence level is 90%.

Finally, we also evaluate how the overall performance of our approach is affected by incorporating feature selection method at classification model building step. In other words, we want to test if a model based on feature selection has any advantage, in terms of performance, over the model which is built without using it. We use *Correlation based Feature Selection (CFS)* method [29] for feature selection.

Addition of Noise

Noise in access control data essentially means that the existing boolean matrix is different from the intended boolean matrix. Misconfiguration can be thought of as a noise or an error in a given set of configurations. $RBAC$ data consist of the user-role-assignment matrix (UA),and permission-role-assignment matrix (PA). In PA boolean matrix, 1 signifies that the permission is associated with a corresponding role, and a 0 denotes absence of permission. Similarly, in UA boolean matrix, 1 signifies that a user has a corresponding role, and a 0 denotes absence of that role in the set of roles of a user. In our experiments, we introduce 7%, 12%, 15%, 20% noise into the datasets in order to measure robustness of our approach in identifying the noisy assignments. Percentage of noise refers to flipping of assignment bits. For instance, when 7% noise is introduced, it means 7% bits in UPA are flipped from either 0 to 1 (to test false positive detections) or from 1 to 0 (to test false negative detections). Noise in data is generated by using algorithm given in [**?**].

Results

As discussed earlier, we study the performance of our approach from various aspects. Three main categories of evaluation are based on varying following organizational parameters;

1. Number of users,keeping everything else constant.
2. Number of permissions, keeping everything else constant.
3. Number of roles, keeping everything else constant.

Under each of these categories, we also evaluate how the threshold value, varying level of noise, choice of role mining algorithm, and feature selection based model building affects the overall performance of our approach. Below we discuss results for each category;

1. Effect of changing number of users, keeping everything else constant.

Table 6.1 give parameters that we use for studying the impact of *varying number of users* on performance of our approach.

Models based on feature selection. Figure 6.4(a),6.5(a),6.6(a), 6.4(b),6.5(b),6.6(b) shows performance in terms of F-measure, recall, and precision when threshold is 0.5, figure 6.10(a),6.11(a),6.12(a), 6.10(b), 6.11(b), 6.12(b), when threshold is 0.7, and figure 6.16(a), 6.17(a), 6.18(a), 6.16(b), 6.17(b), 6.18(b) when threshold is set to 0.9. Note that their classification models were based on feature selection technique (CFS).

Models constructed without feature selection. Performance of models for this case, which are constructed without employing feature selection are shown in figures 6.7(a), 6.8(a),6.9(a), 6.7(b), 6.8(b),6.9(b), 6.13(a), 6.14(a), 6.15(a), 6.13(b),6.14(b),6.15(b), 6.19(a), 6.20(a), 6.21(a), 6.19(b), 6.20(b), 6.21(b).

X-axis shows *varying levels of noise*, and y-axis shows the *F measure*. Each Line on a graph represents one set of data. At different threshold values, we observe the following:

At threshold level *0.5*, we observe in figures 6.4(a), and 6.7(a) that the increase in noise level causes fmeasure for false negative to slightly drop. On the other hand, f measure for false positive either improved or remained consistent with the increase in level of noise, as shown in figure 6.4(b), and 6.7(b). When comparing results for CFS-based models and non-CFS based models we observe that our approach works well with both type of models and there is little or no difference in its performance. Overall at threshold level 0.5, both type of models performed consistently in case when the number of users increases.

At threshold level *0.9*, we observe in figures 6.16(a), and 6.19(a) that the increase in noise level causes f measure to slightly drop. Whereas; f measure for false positives remained consistent with the increase in level of noise, as can be seen in figure 6.16(b), and 6.19(b). When comparing results

for CFS-based models and non-CFS based models we observe that f measure for false negatives in data 1 significantly dropped at 15% noise. Although it started improving gradually at next level but not at the rate with which it dropped. No such fluctuation in performance is observed in remaining data. Overall at threshold level 0.9, both type of f measures either improved or remained consistent with the increase in number of users.

Moreover, we observe that the f measures declines in both false positive and false negative cases when threshold value increased from 0.5 to 0.9. This is because the instances which were classified as misconfiguration with the probability less than 0.9, were not picked as errors by the algorithm. Therefore, lower the threshold value is, better would be the performance of our approach.

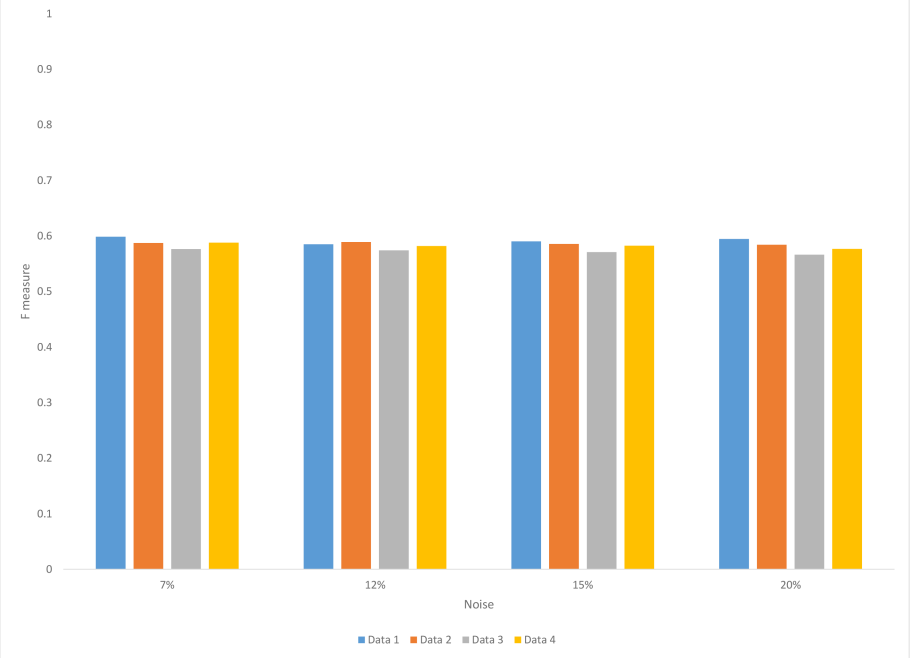
2. Effect of changing number of permissions, keeping everything else constant.

Models based on feature selection. Figure 6.22(a),6.23(a),6.24(a), 6.22(b),6.23(b),6.24(b) shows performance in terms of F-measure, recall, and precision when threshold is 0.5, figure 6.28(a),6.29(a),6.30(a), 6.28(b), 6.29(b), 6.30(b), when threshold is 0.7, and figure 6.34(a), 6.35(a), 6.36(a), 6.34(b), 6.35(b), 6.36(b) when threshold is set to 0.9. Note that their classification models were based on feature selection technique (CFS).

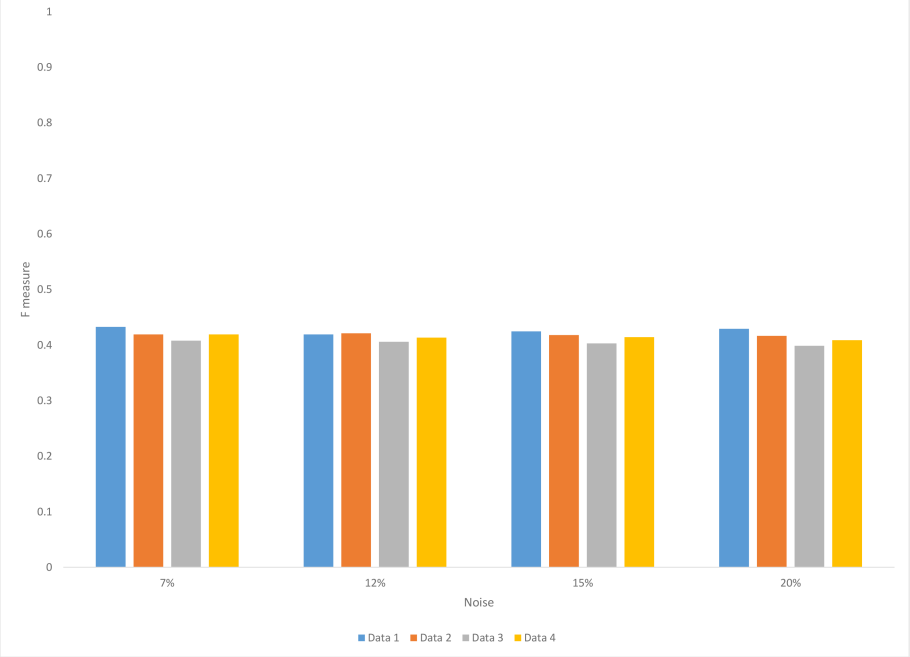
Models constructed without feature selection. Performance of models for this case which are constructed without employing feature selection are shown in figures 6.25(a), 6.26(a),6.27(a), 6.25(b), 6.26(b),6.27(b), 6.31(a), 6.32(a), 6.33(a), 6.31(b),6.32(b),6.33(b), 6.37(a), 6.38(a), 6.39(a), 6.37(b), 6.38(b), 6.39(b).

Table 6.2 give parameters that we use for studying the impact of *varying number of permissions* on performance of our approach. At various noise levels, it can be clearly seen in figure 6.22, 6.23, 6.25, 6.26, 6.28, 6.29, 6.31, 6.32, 6.34, 6.35, 6.38 and 6.37 that its variation does not affect performance significantly. However, by increasing in number of permissions, f measure and recall for false positive detection is affected somewhat negatively as can be seen in figure 6.22(b), 6.23(b), 6.25(b), 6.26(b), 6.34(b),6.35(b), and 6.37(b), 6.38(b). Still the f measure and recall does not decrease significantly. On the other hand, f measure and recall for false negative detection either improved or remained consistent with the increase in number of roles as can be seen in figure 6.22(a), 6.23(a), 6.25(a), 6.26(a) and 6.37(a), 6.38(a). Precision, however in each case is not impressive but still it improves with the increase in number of users.

Performance of our approach in this case remains unaffected for both CFS and non CFS based models. Remaining parameters have not shown any impact on performance either in this case. There is no major fluctuation in performance observed in any case. In general, the performance of our approach improves with the increase in number of permissions.

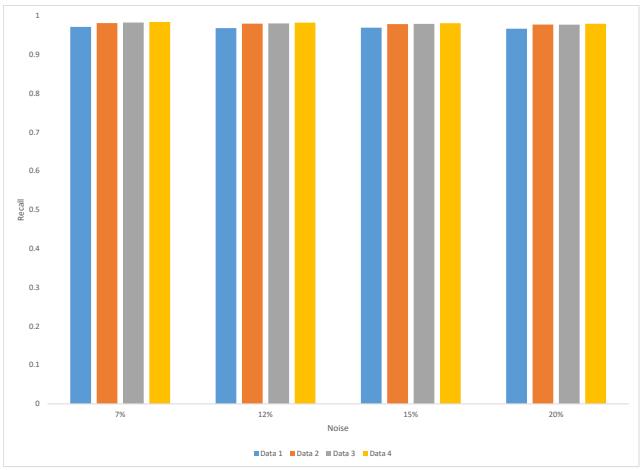


(a) F measure for false negative detections at varying noise levels

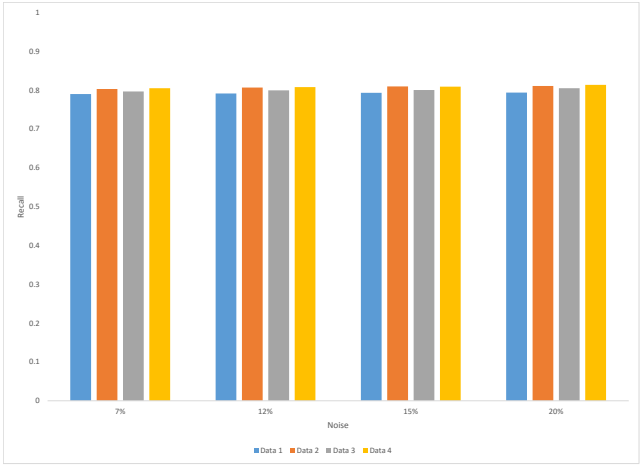


(b) F measure for false positive detections at varying noise levels

Figure 6.4. F measure for DataSet 1, when model construction is CFS based. Threshold = 0.5

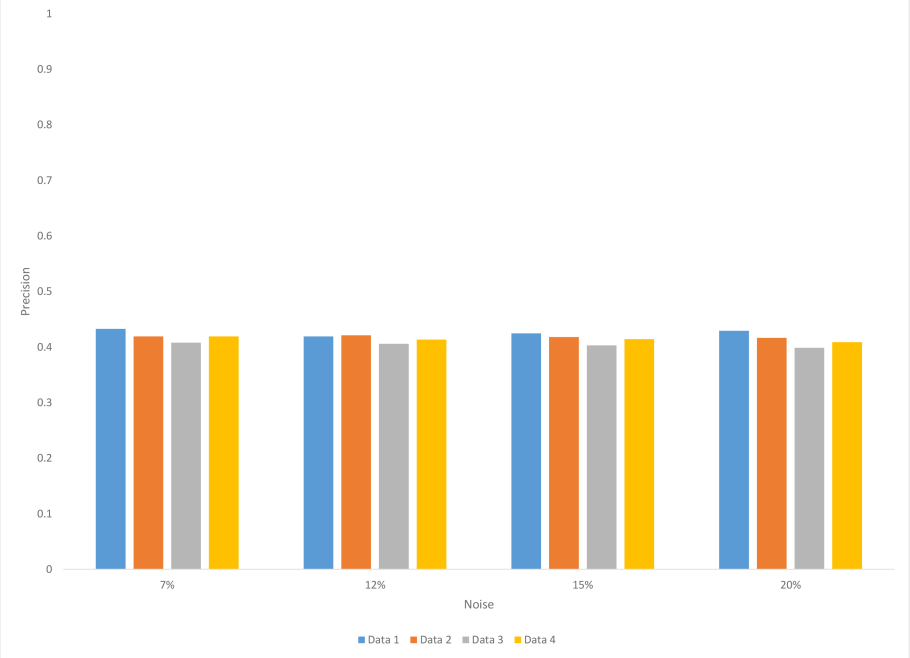


(a) Recall for false negative detections at varying noise levels

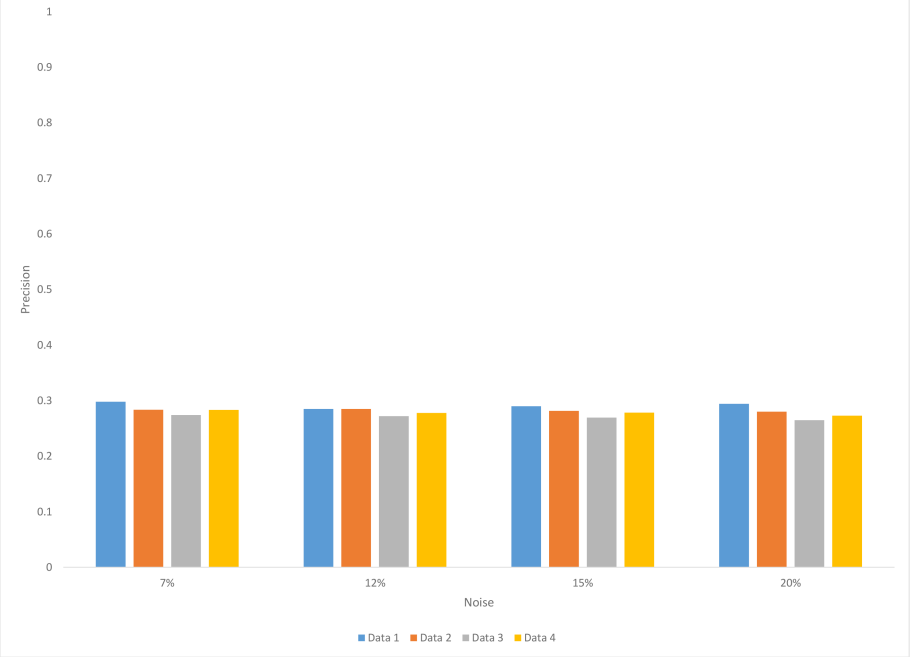


(b) Recall for false positive detections at varying noise levels

Figure 6.5. Recall for DataSet 1, when model construction is CFS based. Threshold = 0.5

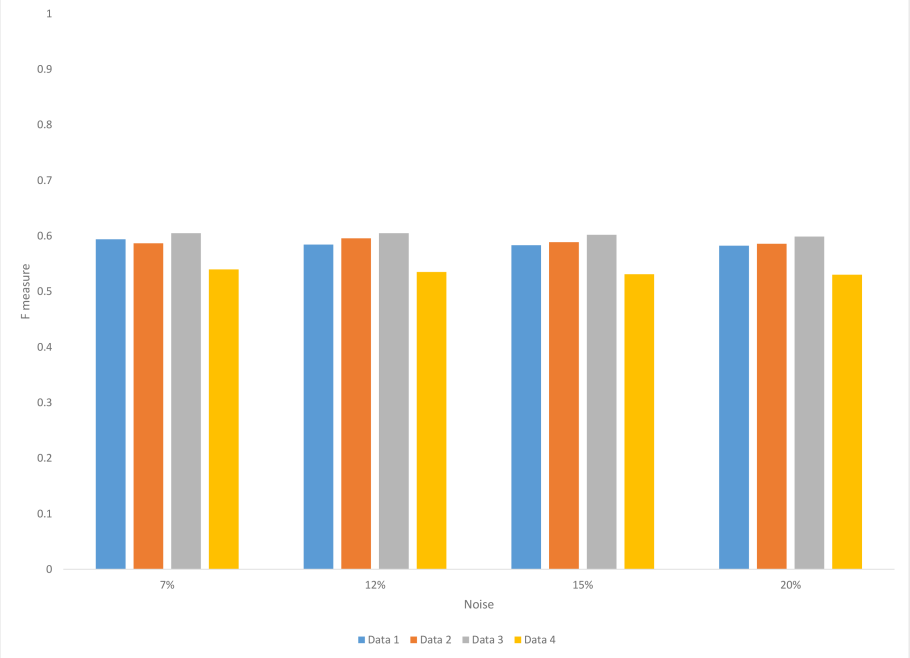


(a) Precision for false negative detections at varying noise levels

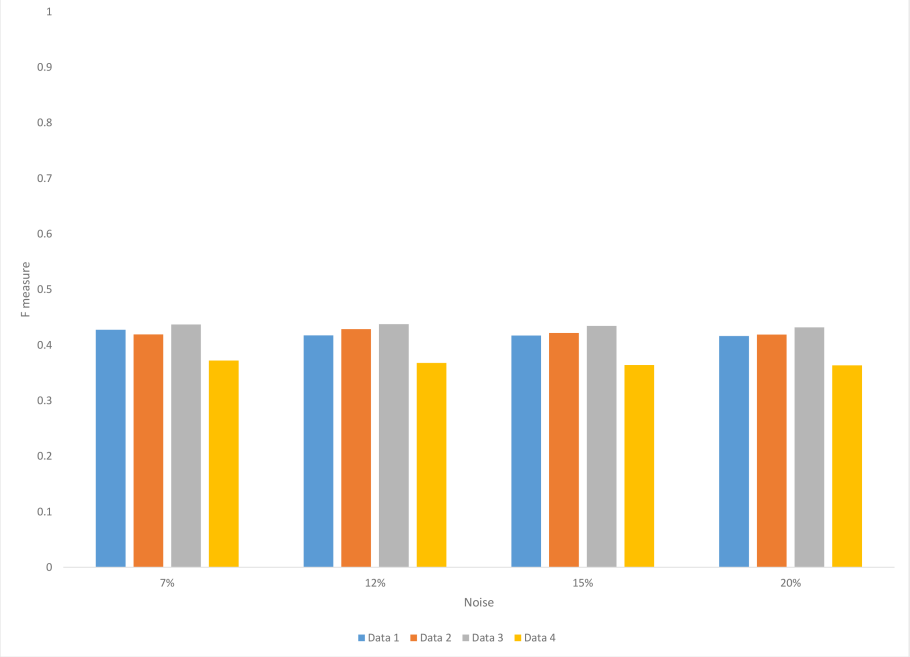


(b) Precision for false positive detections at varying noise levels

Figure 6.6. Precision for DataSet 1, when model construction is CFS based. Threshold = 0.5

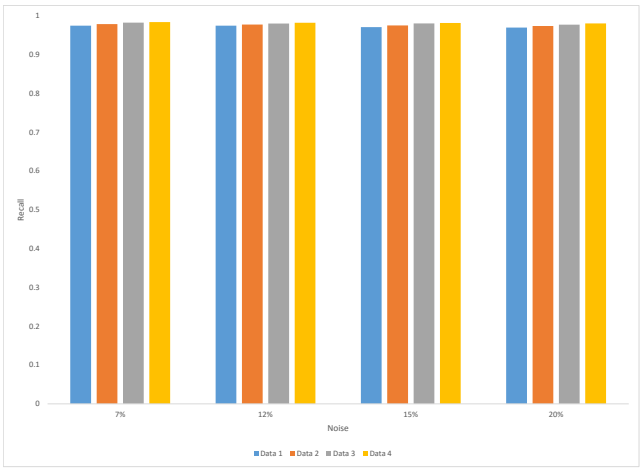


(a) F measure for false negative detections at varying noise levels

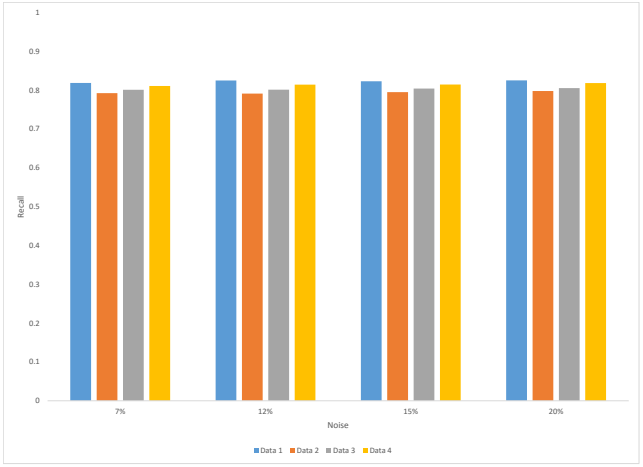


(b) F measure for false positive detections at varying noise levels

Figure 6.7. F measure for DataSet 1, when model construction is not CFS based. Threshold = 0.5

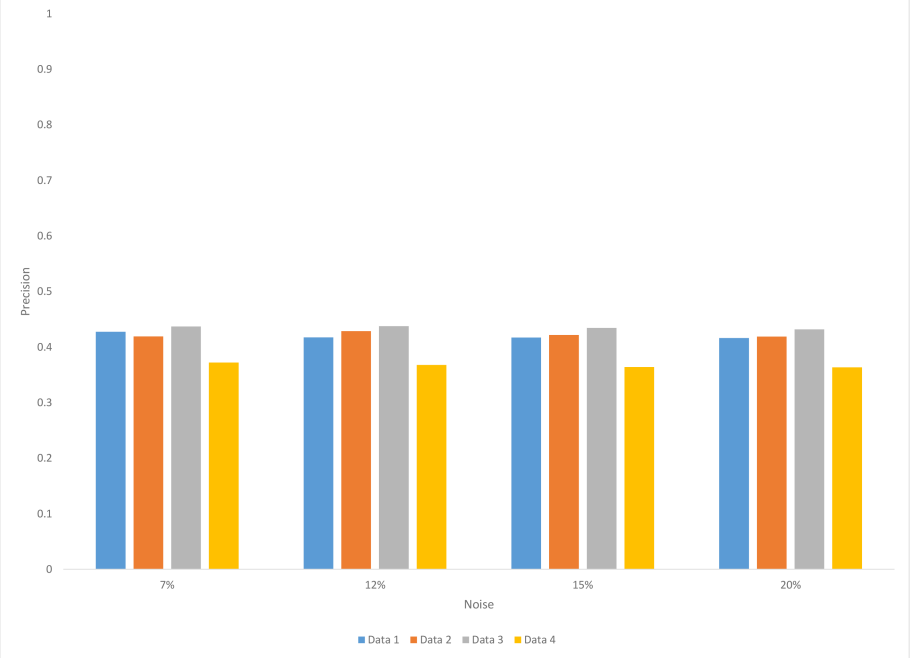


(a) Recall for false negative detections at varying noise levels

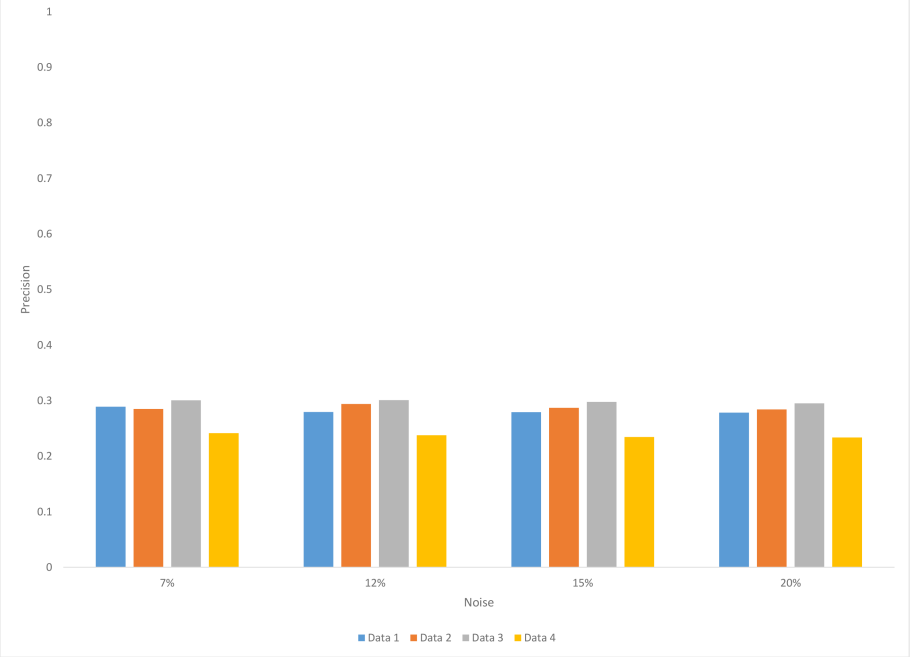


(b) Recall for false positive detections at varying noise levels

Figure 6.8. Recall for DataSet 1, when model construction is not CFS based. Threshold = 0.5

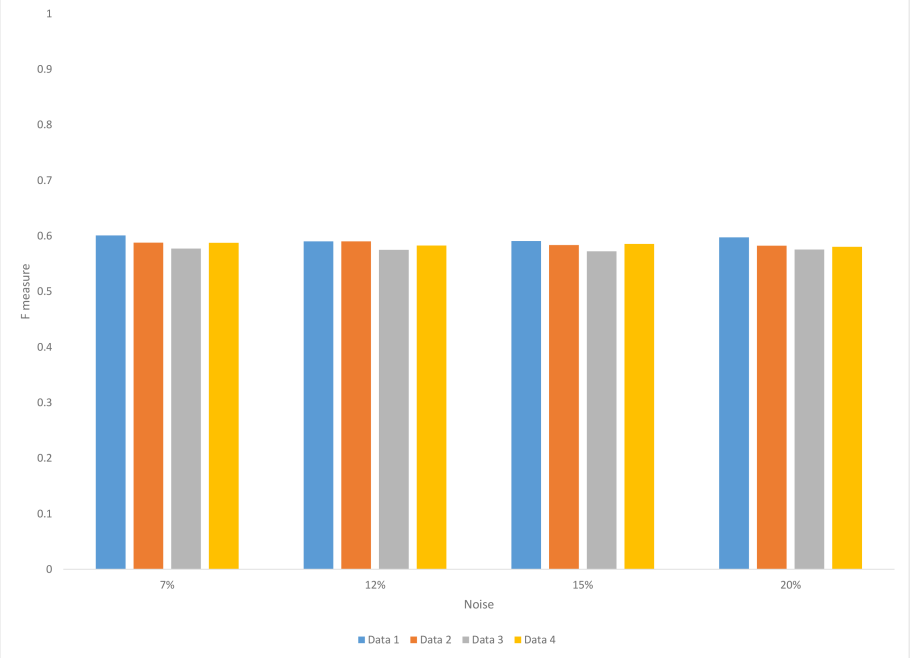


(a) Precision for false negative detections at varying noise levels

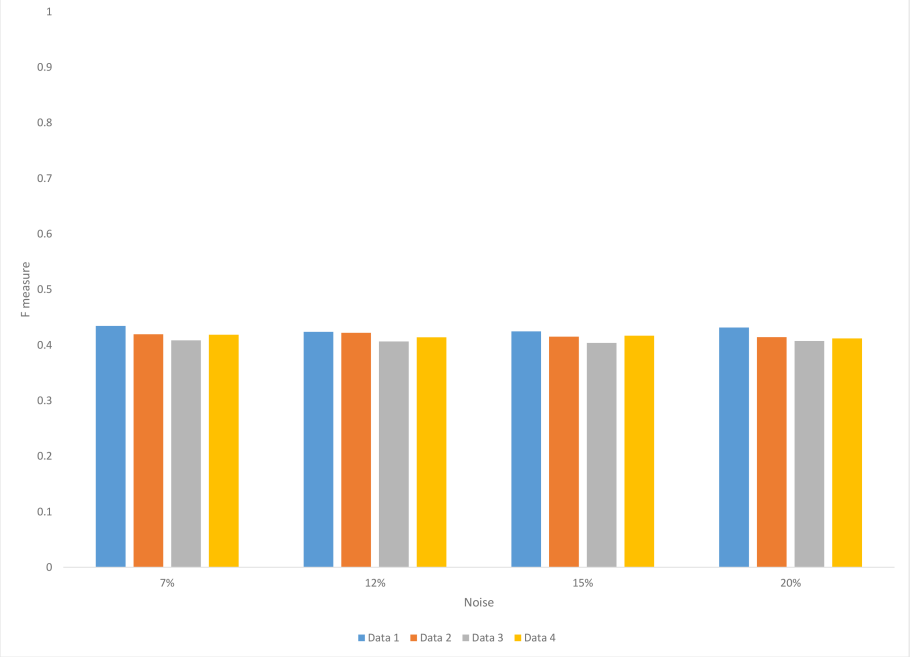


(b) Precision for false positive detections at varying noise levels

Figure 6.9. Precision for DataSet 1, when model construction is not CFS based. Threshold = 0.5

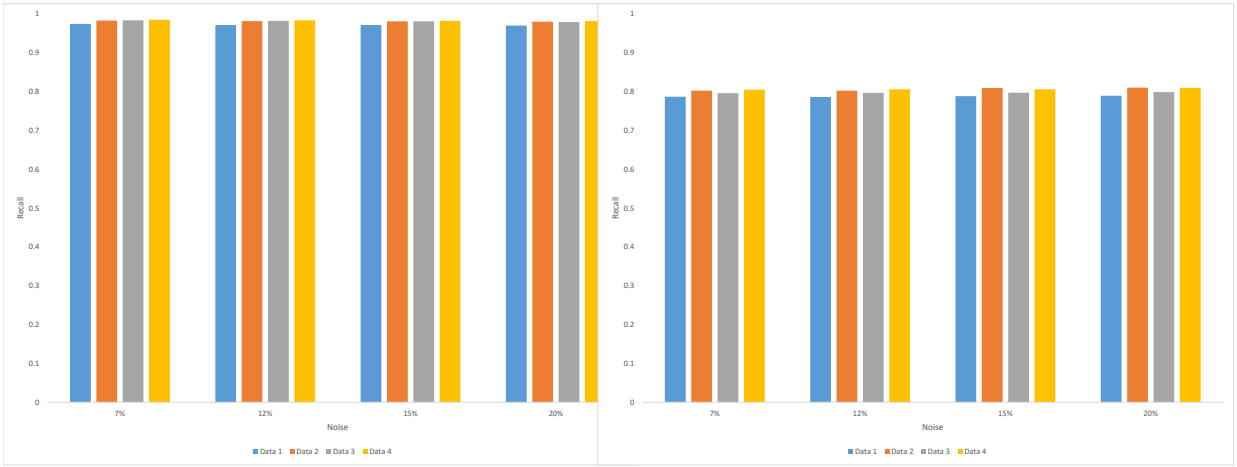


(a) F measure for false negative detections at varying noise levels



(b) F measure for false positive detections at varying noise levels

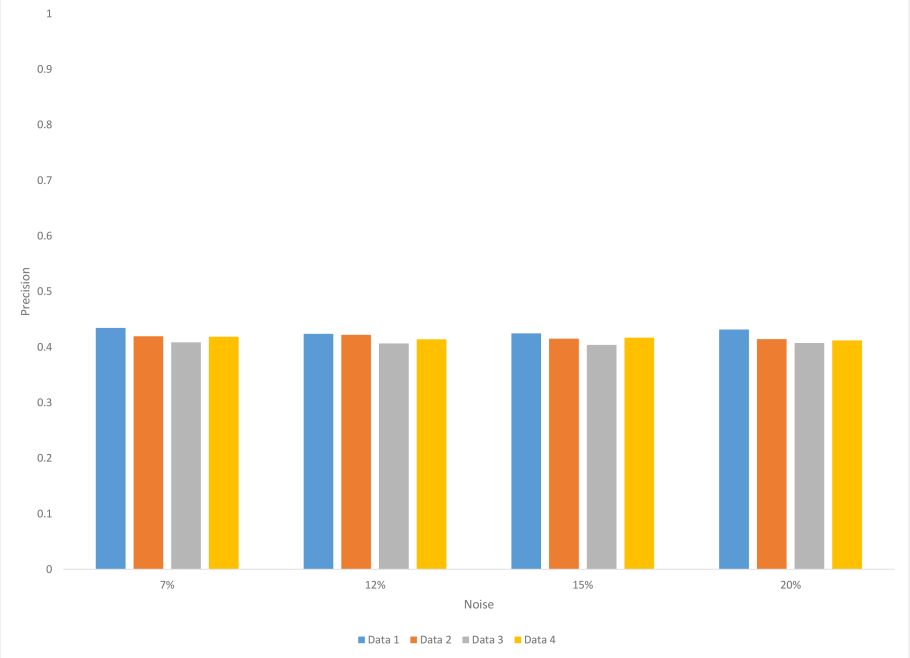
Figure 6.10. F measure for DataSet 1, when model construction is CFS based. Threshold = 0.7



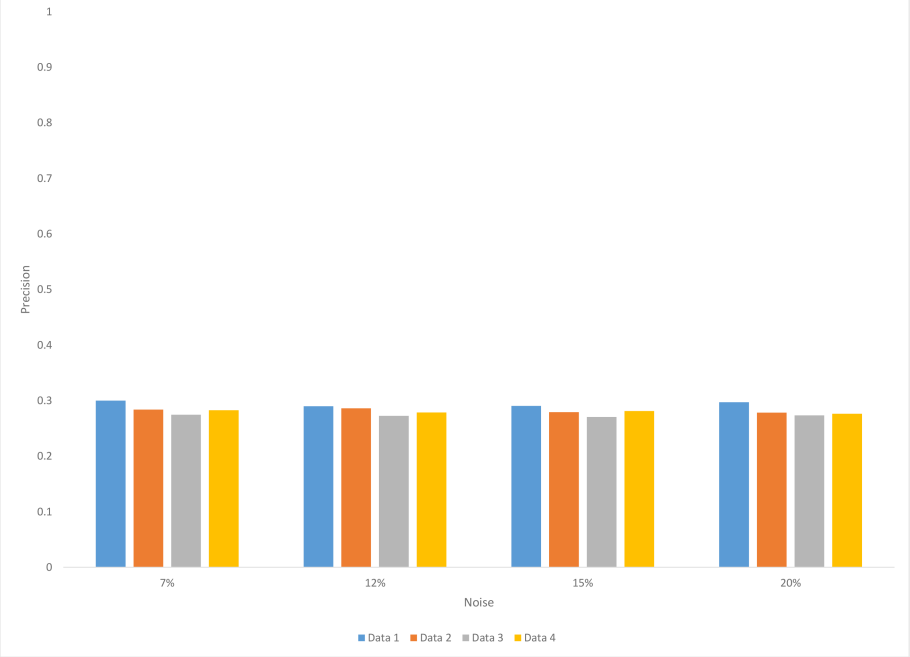
(a) Recall for false negative detections at varying noise levels

(b) Recall for false positive detections at varying noise levels

Figure 6.11. Recall for DataSet 1, when model construction is CFS based. Threshold = 0.7

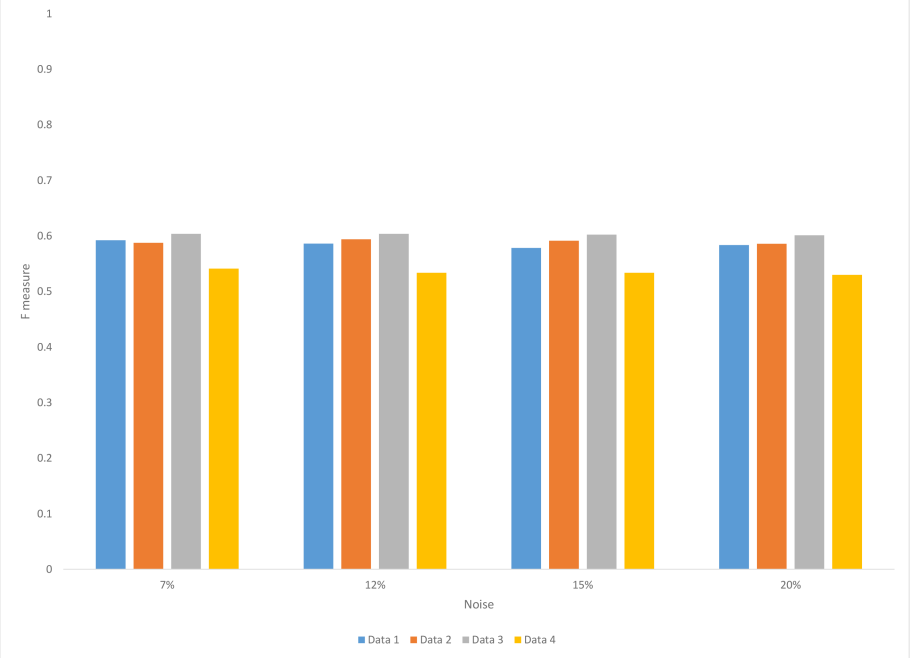


(a) Precision for false negative detections at varying noise levels

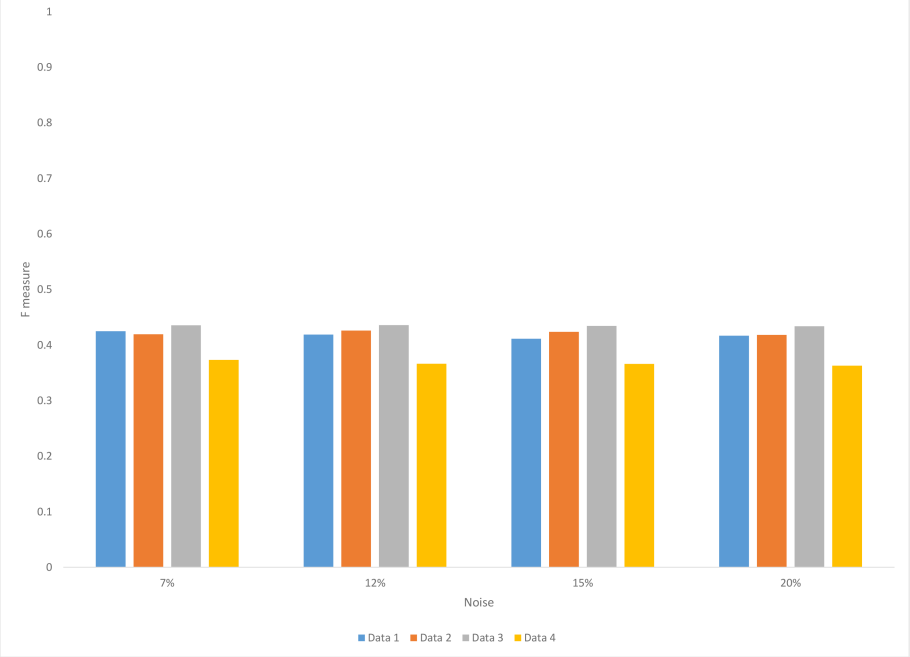


(b) Precision for false positive detections at varying noise levels

Figure 6.12. Precision for DataSet 1, when model construction is CFS based. Threshold = 0.7

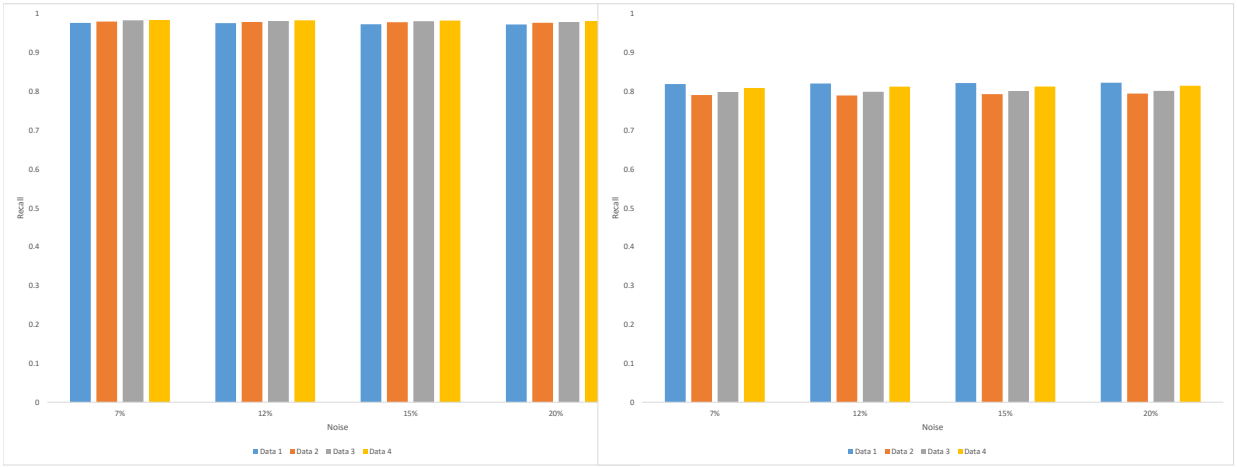


(a) F measure for false negative detections at varying noise levels



(b) F measure for false positive detections at varying noise levels

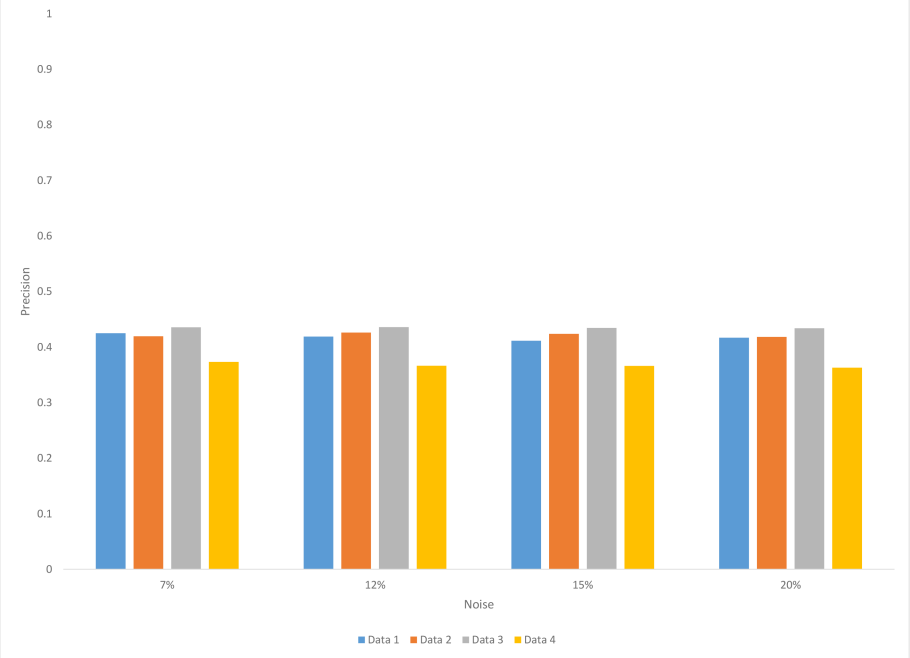
Figure 6.13. F measure for DataSet 1, when model construction is not CFS based. Threshold = 0.7



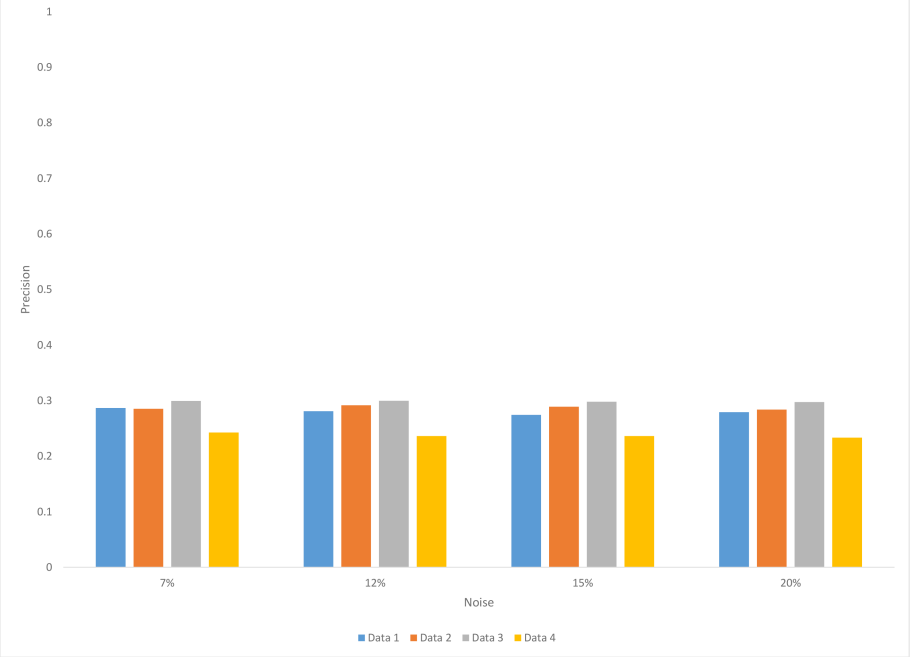
(a) Recall for false negative detections at varying noise levels

(b) Recall for false positive detections at varying noise levels

Figure 6.14. Recall for DataSet 1, when model construction is not CFS based. Threshold = 0.7

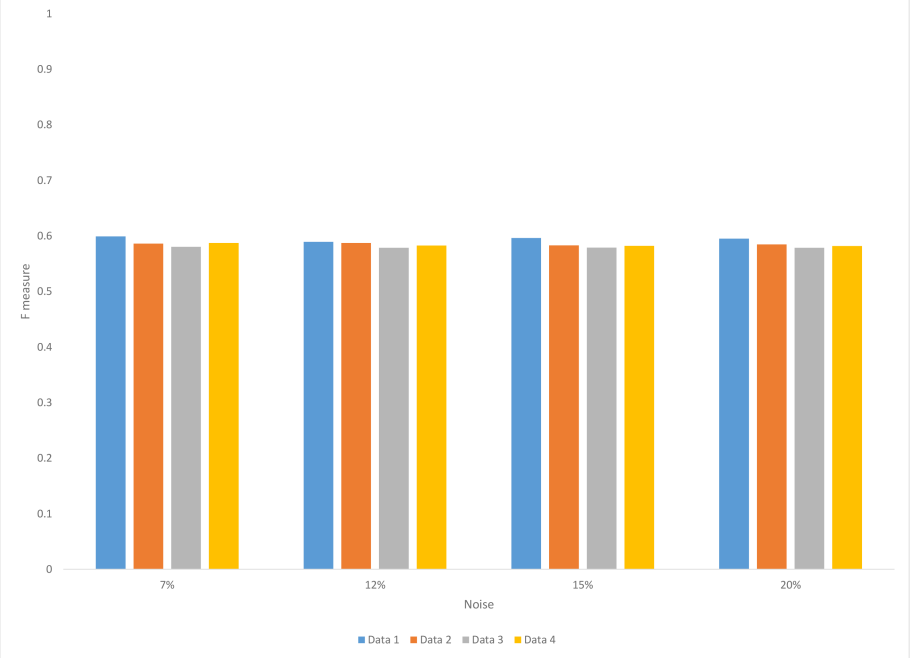


(a) Precision for false negative detections at varying noise levels

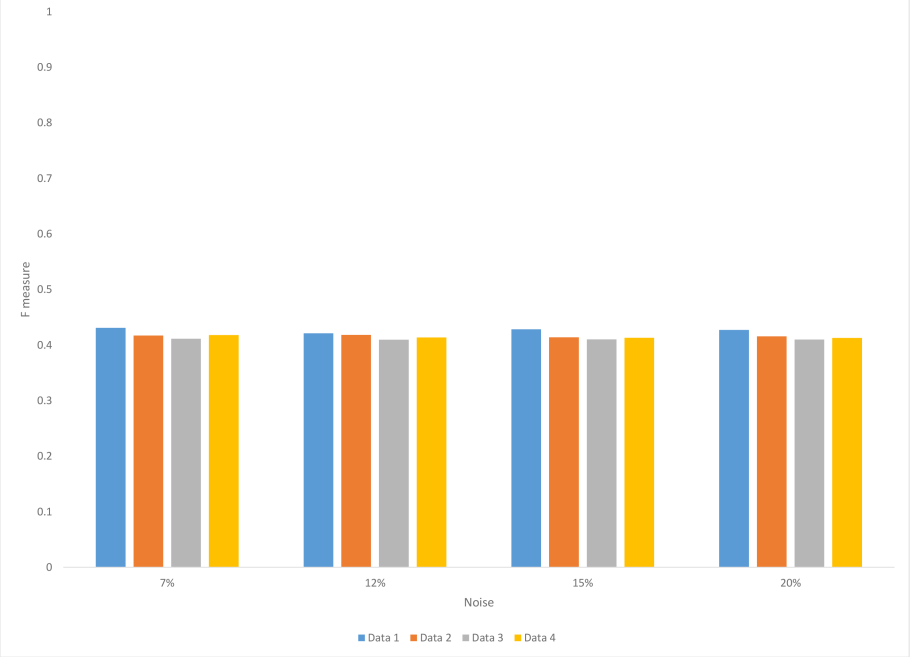


(b) Precision for false positive detections at varying noise levels

Figure 6.15. Precision for DataSet 1, when model construction is not CFS based. Threshold = 0.7

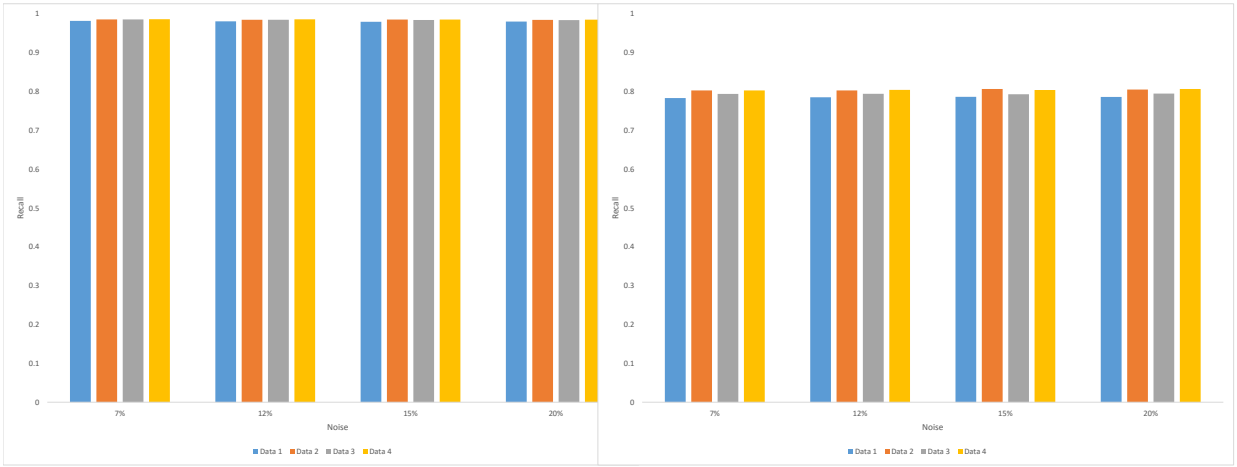


(a) F measure for false negative detections at varying noise levels



(b) F measure for false positive detections at varying noise levels

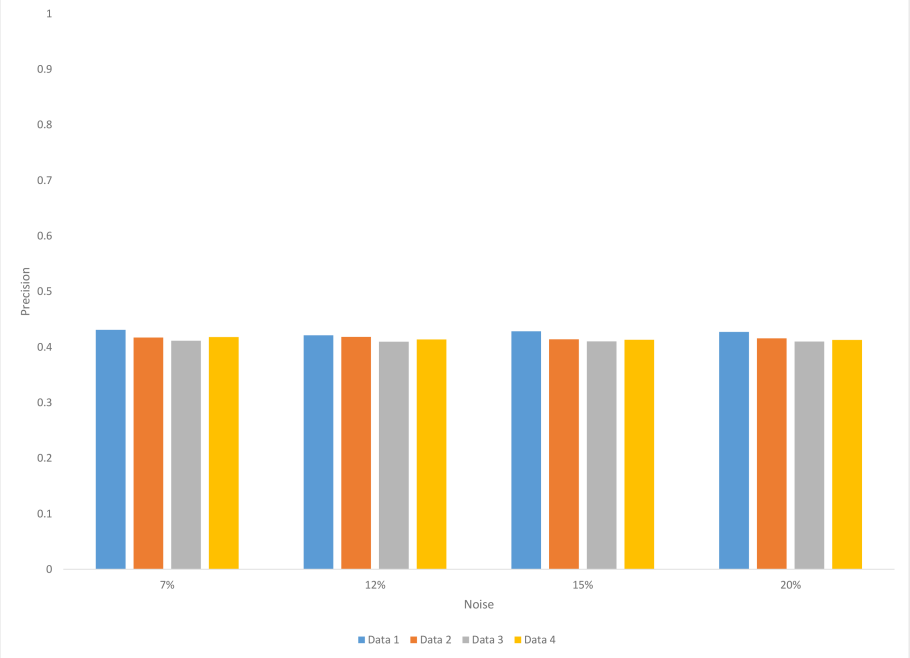
Figure 6.16. F measure for DataSet 1, when model construction is CFS based. Threshold = 0.9



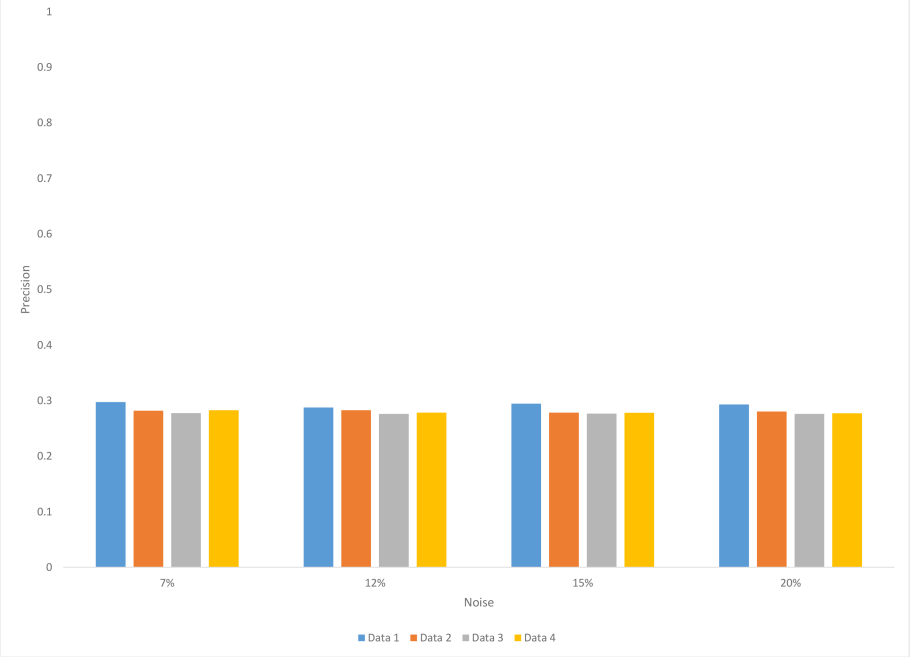
(a) Recall for false negative detections at varying noise levels

(b) Recall for false positive detections at varying noise levels

Figure 6.17. Recall for DataSet 1, when model construction is CFS based. Threshold = 0.9

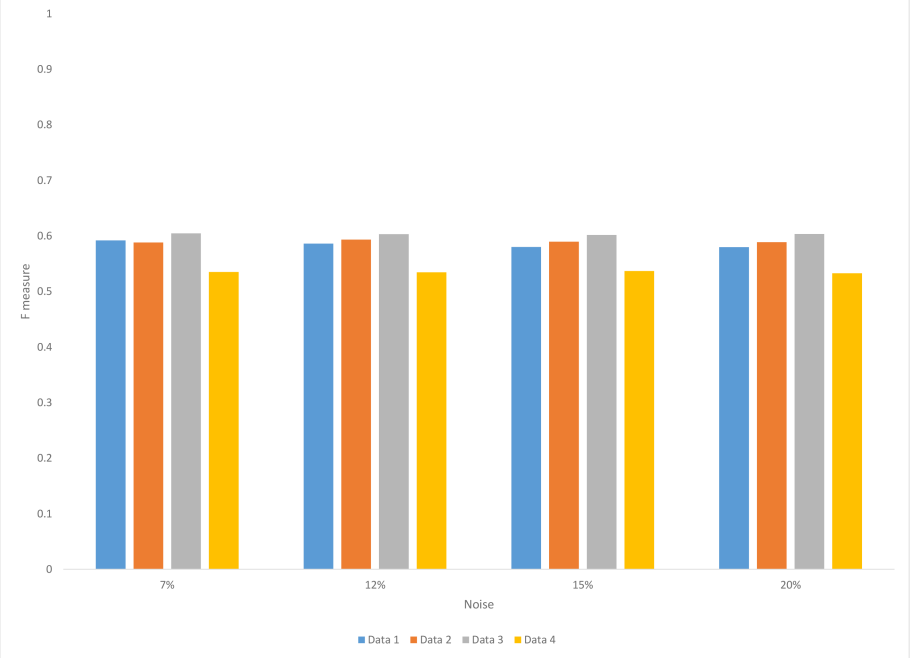


(a) Precision for false negative detections at varying noise levels

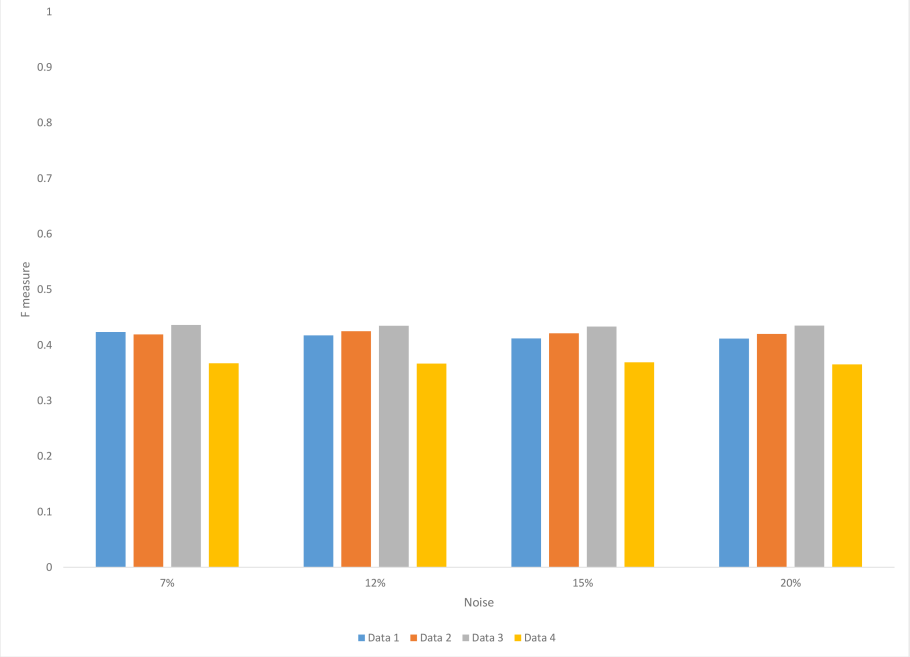


(b) Precision for false positive detections at varying noise levels

Figure 6.18. Precision for DataSet 1, when model construction is CFS based. Threshold = 0.9

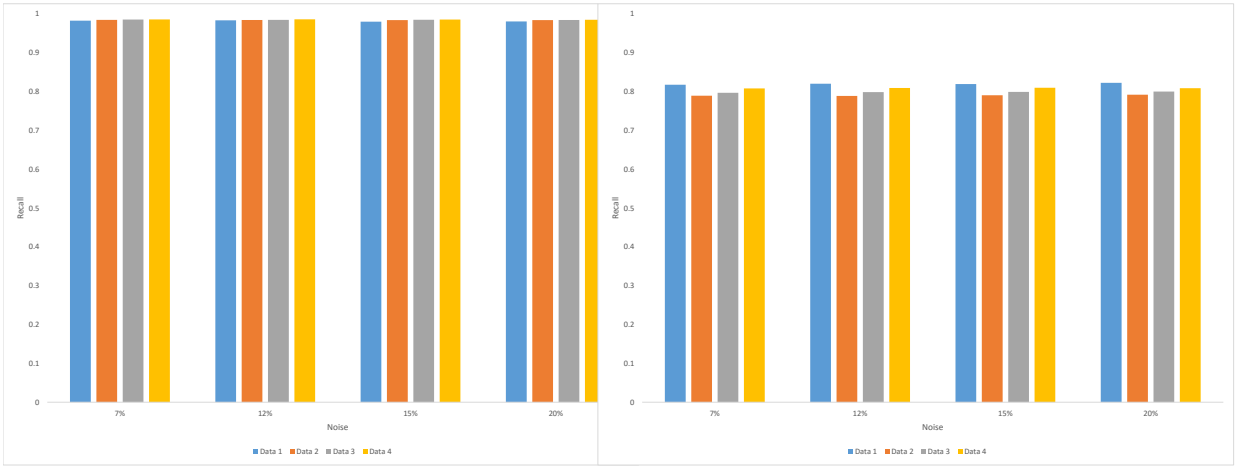


(a) F measure for false negative detections at varying noise levels



(b) F measure for false positive detections at varying noise levels

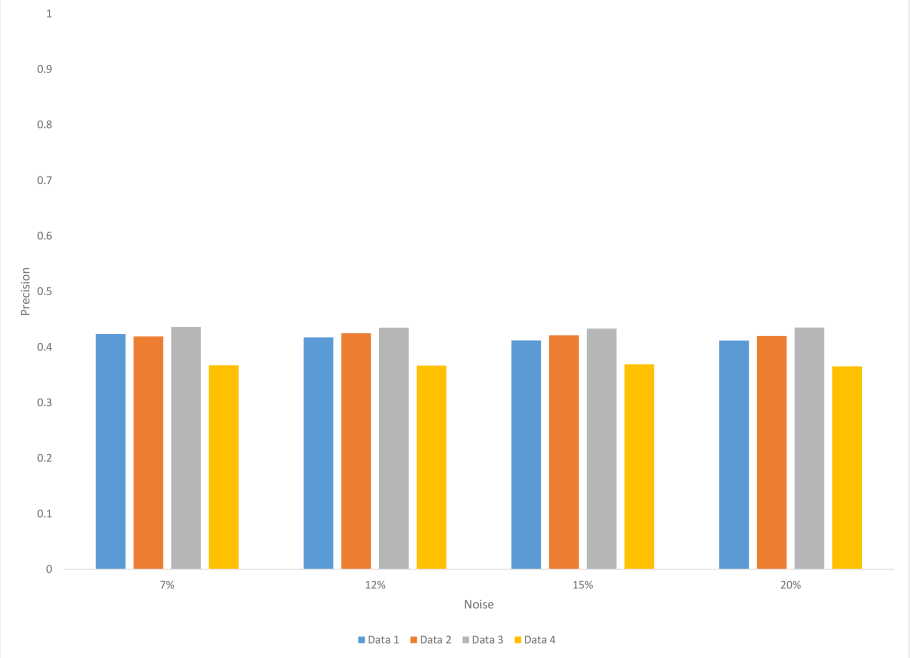
Figure 6.19. F measure for DataSet 1, when model construction is not CFS based. Threshold = 0.9



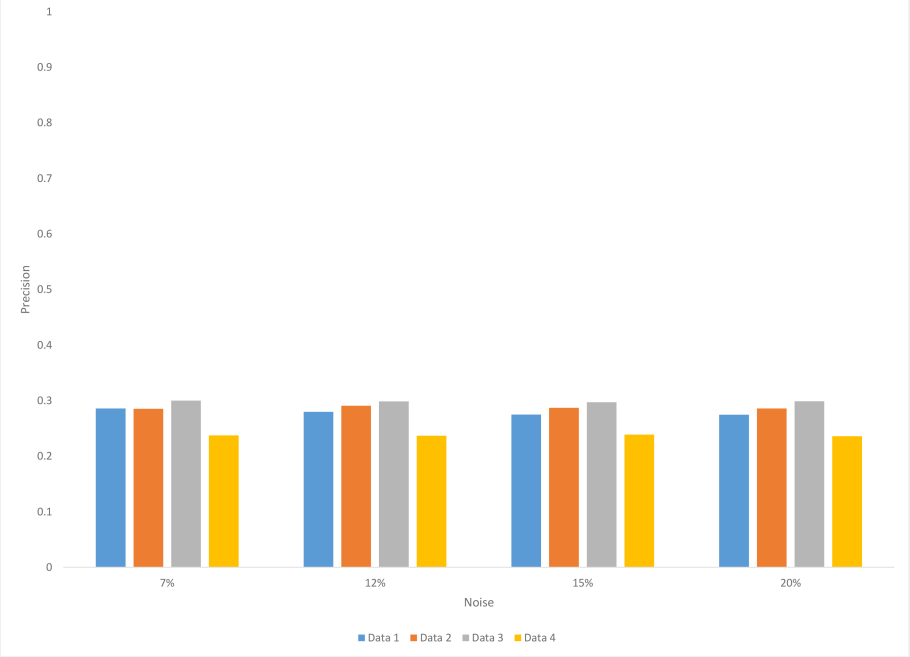
(a) Recall for false negative detections at varying noise levels

(b) Recall for false positive detections at varying noise levels

Figure 6.20. Recall for DataSet 1, when model construction is not CFS based. Threshold = 0.9

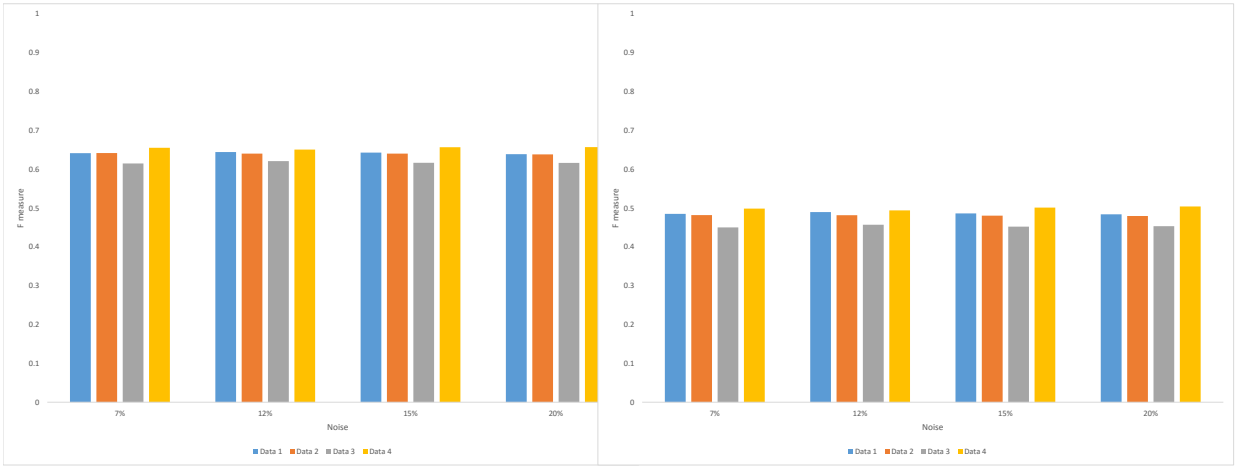


(a) Precision for false negative detections at varying noise levels



(b) Precision for false positive detections at varying noise levels

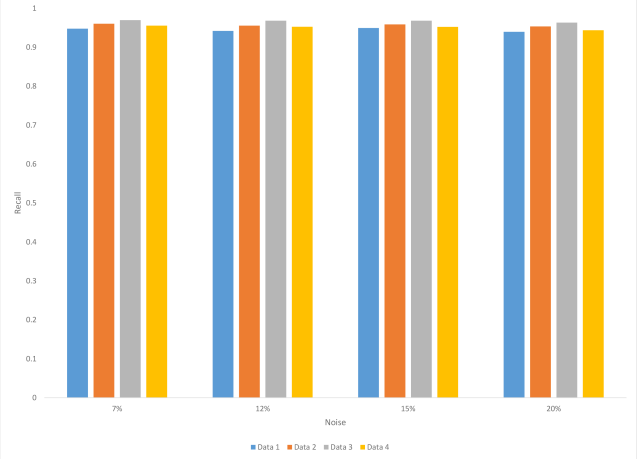
Figure 6.21. Precision for DataSet 1, when model construction is not CFS based. Threshold = 0.9



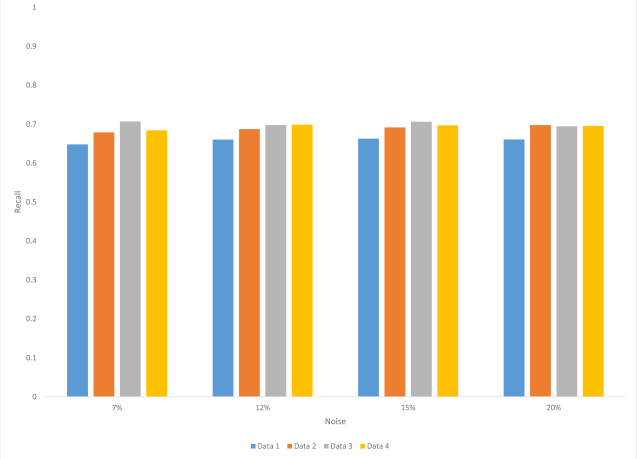
(a) F measure for false negative detections at varying noise levels

(b) F measure for false positive detections at varying noise levels

Figure 6.22. F measure for DataSet 2, when model construction is CFS based. Threshold = 0.5

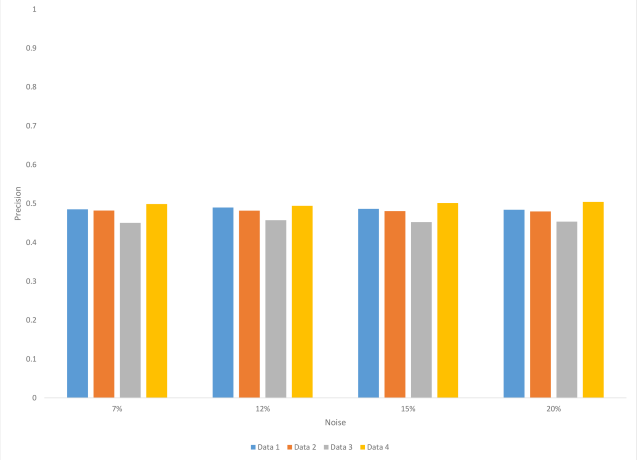


(a) Recall for false negative detections at varying noise levels

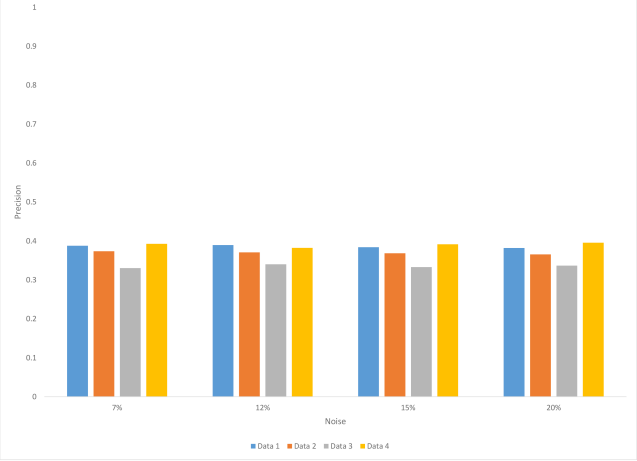


(b) Recall for false positive detections at varying noise levels

Figure 6.23. Recall for DataSet 2, when model construction is CFS based. Threshold = 0.5

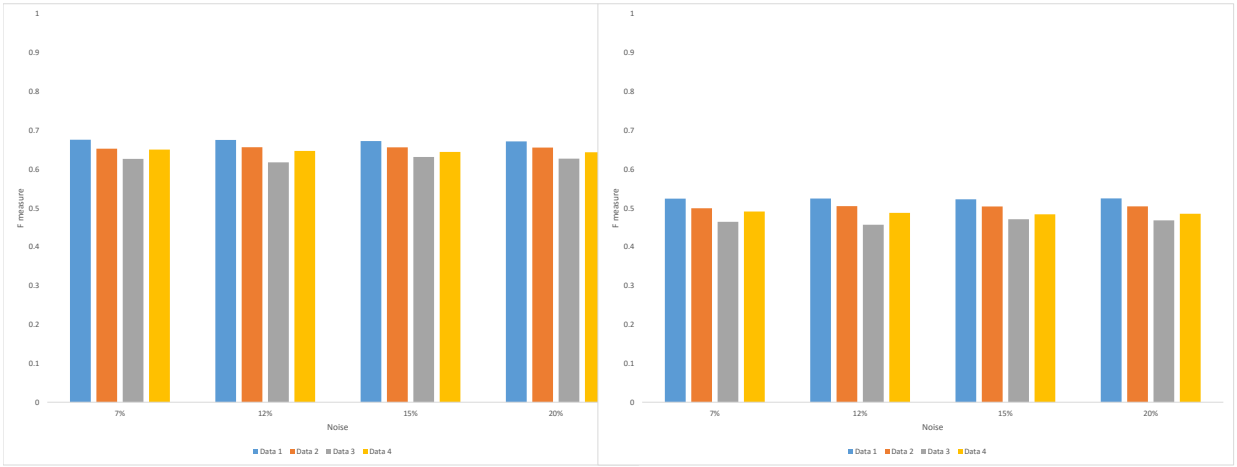


(a) Precision for false negative detections at varying noise levels



(b) Precision for false positive detections at varying noise levels

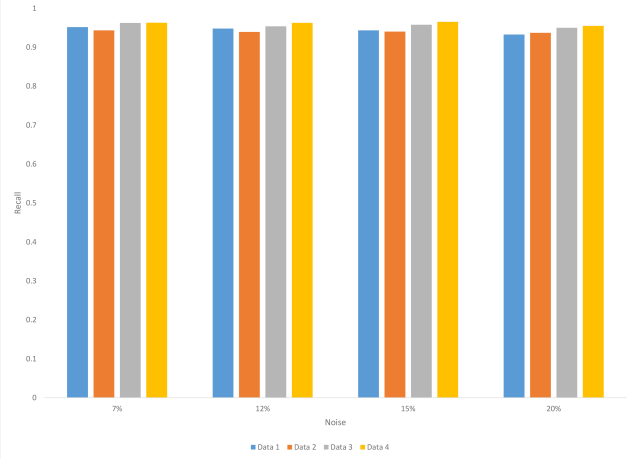
Figure 6.24. Precision for DataSet 2, when model construction is CFS based. Threshold = 0.5



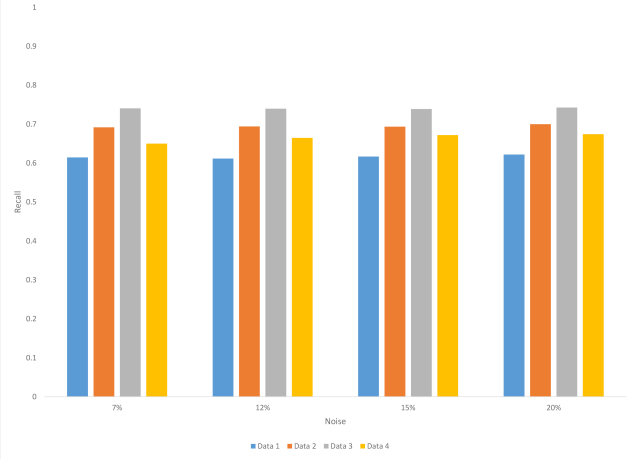
(a) F measure for false negative detections at varying noise levels

(b) F measure for false positive detections at varying noise levels

Figure 6.25. F measure for DataSet 2, when model construction is not CFS based. Threshold = 0.5

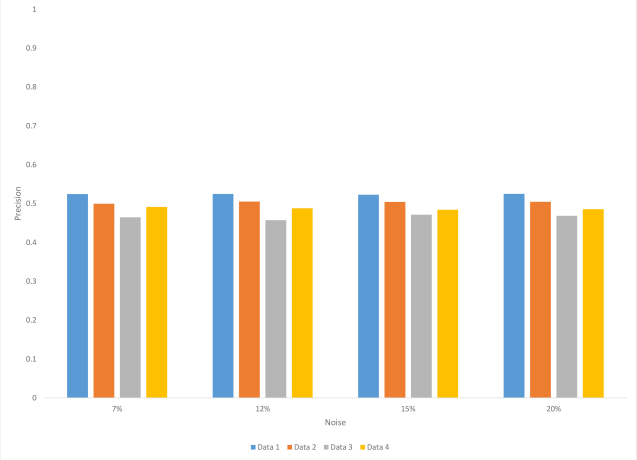


(a) Recall for false negative detections at varying noise levels

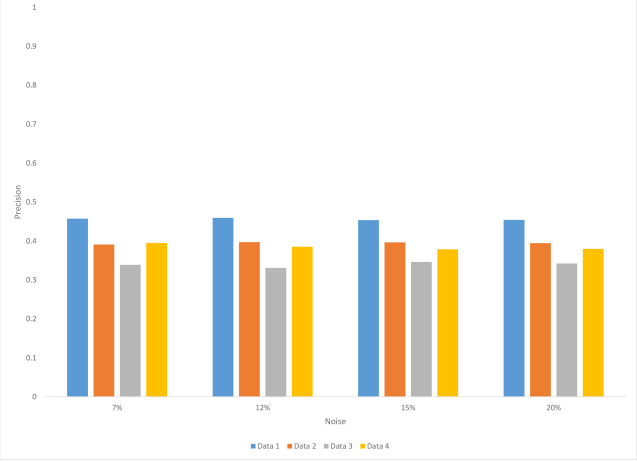


(b) Recall for false positive detections at varying noise levels

Figure 6.26. Recall for DataSet 2, when model construction is not CFS based. Threshold = 0.5

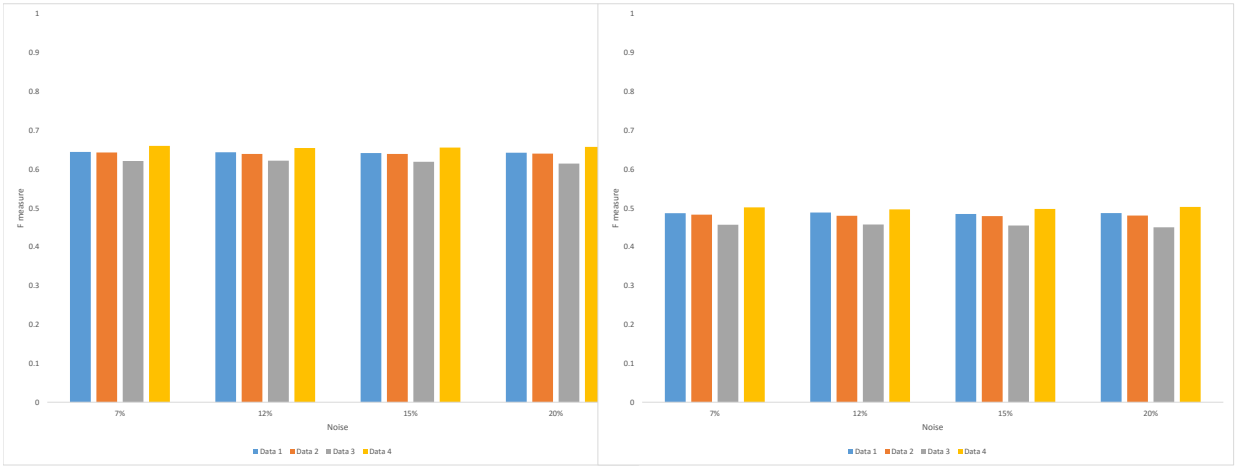


(a) Precision for false negative detections at varying noise levels



(b) Precision for false positive detections at varying noise levels

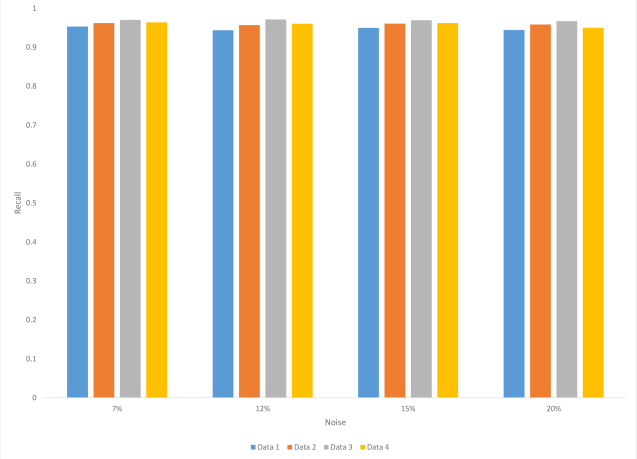
Figure 6.27. Precision for DataSet 2, when model construction is not CFS based. Threshold = 0.5



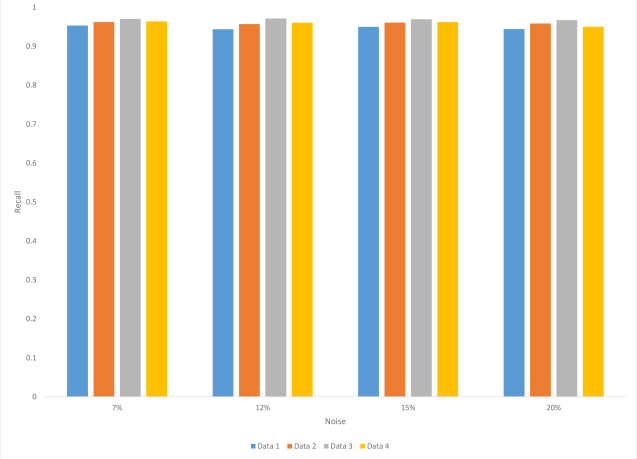
(a) F measure for false negative detections at varying noise levels

(b) F measure for false positive detections at varying noise levels

Figure 6.28. F measure for DataSet 2, when model construction is CFS based. Threshold = 0.7

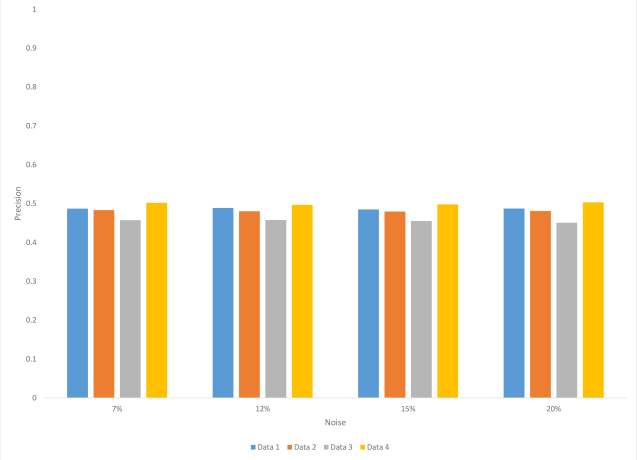


(a) Recall for false negative detections at varying noise levels

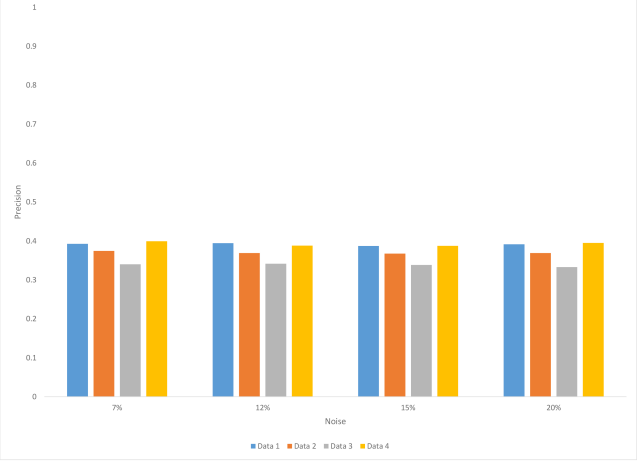


(b) Recall for false positive detections at varying noise levels

Figure 6.29. Recall for DataSet 2, when model construction is CFS based. Threshold = 0.7

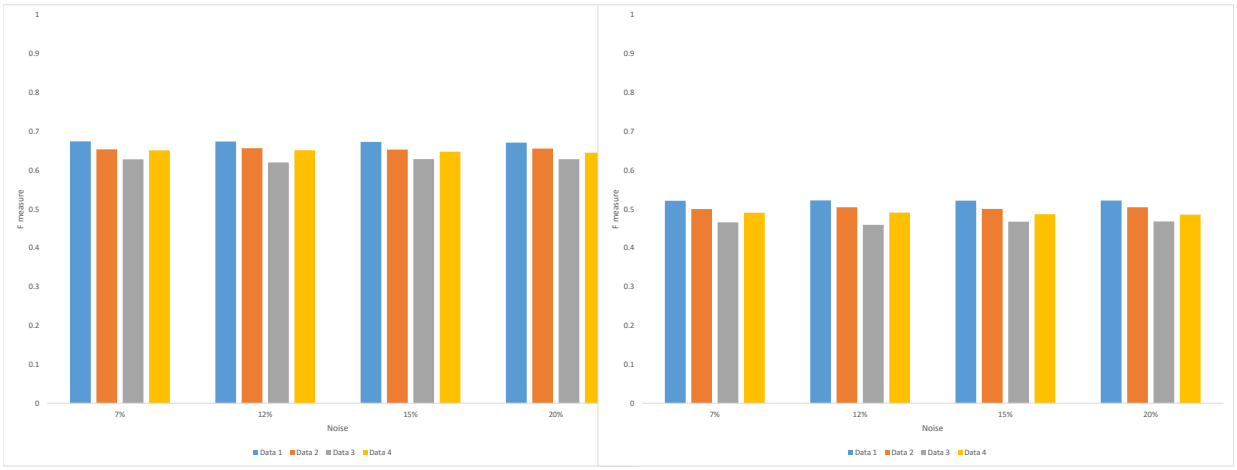


(a) Precision for false negative detections at varying noise levels



(b) Precision for false positive detections at varying noise levels

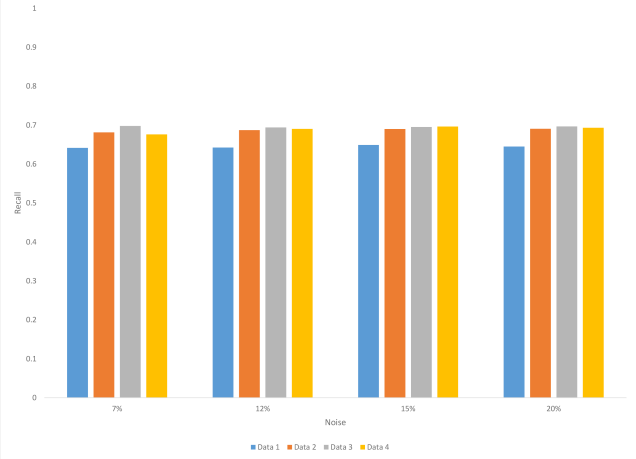
Figure 6.30. Precision for DataSet 2, when model construction is CFS based. Threshold = 0.7



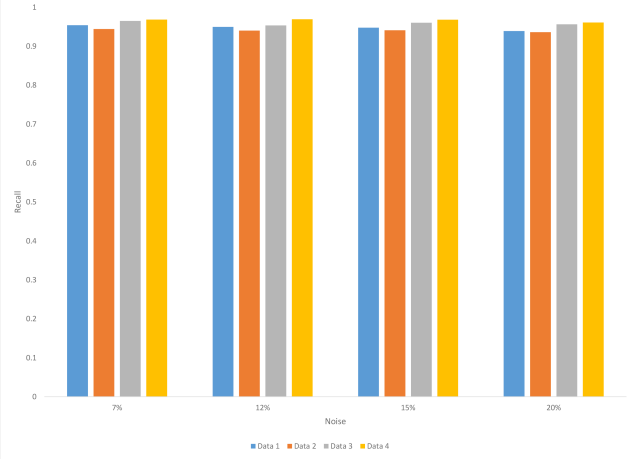
(a) F measure for false negative detections at varying noise levels

(b) F measure for false positive detections at varying noise levels

Figure 6.31. F measure for for DataSet 2, when model construction is not CFS based. Threshold = 0.7

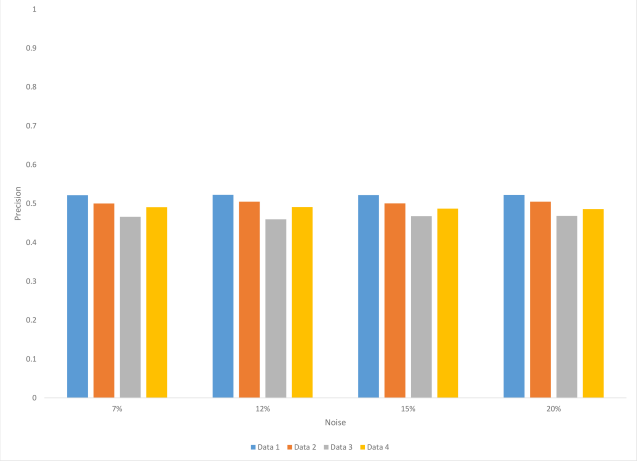


(a) Recall for false negative detections at varying noise levels

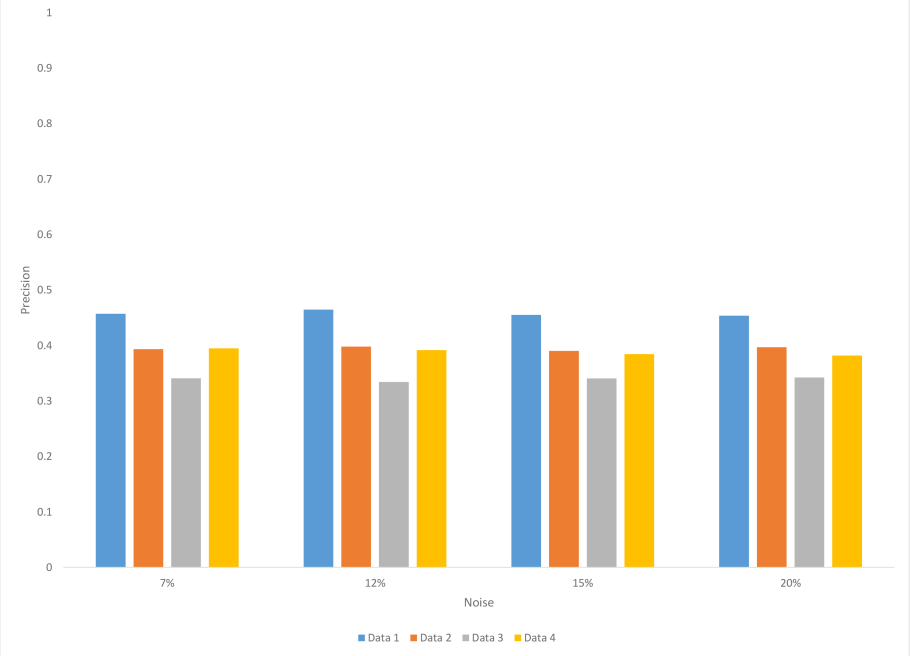


(b) Recall for false positive detections at varying noise levels

Figure 6.32. Recall for for DataSet 2, when model construction is not CFS based. Threshold = 0.7

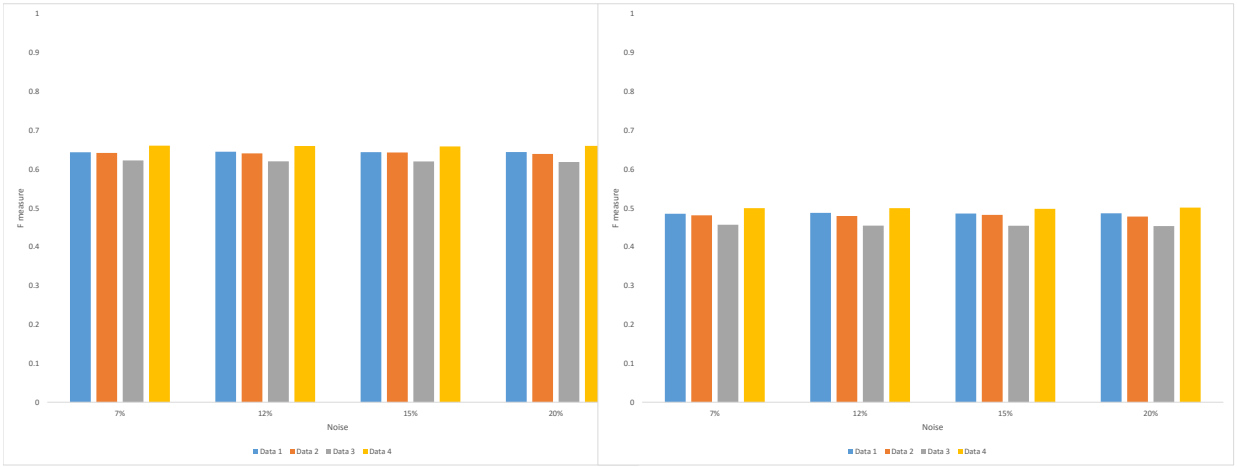


(a) Precision for false negative detections at varying noise levels



(b) Precision for false positive detections at varying noise levels

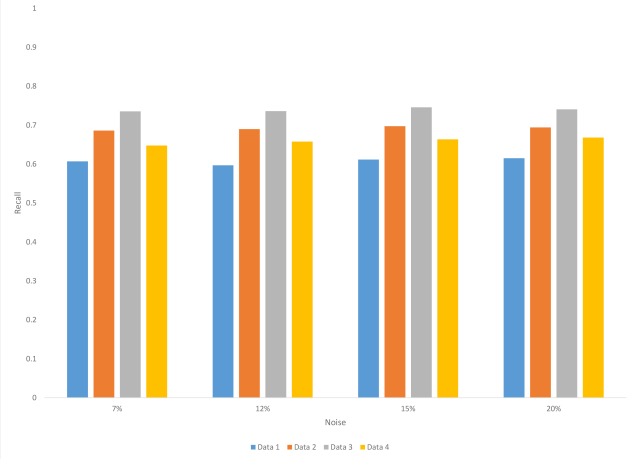
Figure 6.33. Precision for for DataSet 2, when model construction is not CFS based. Threshold = 0.7



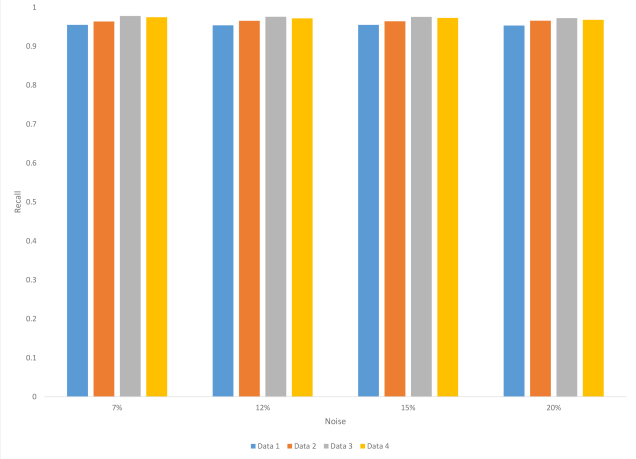
(a) F measure for false negative detections at varying noise levels

(b) F measure for false positive detections at varying noise levels

Figure 6.34. F measure for for DataSet 2, when model construction is CFS based. Threshold = 0.9

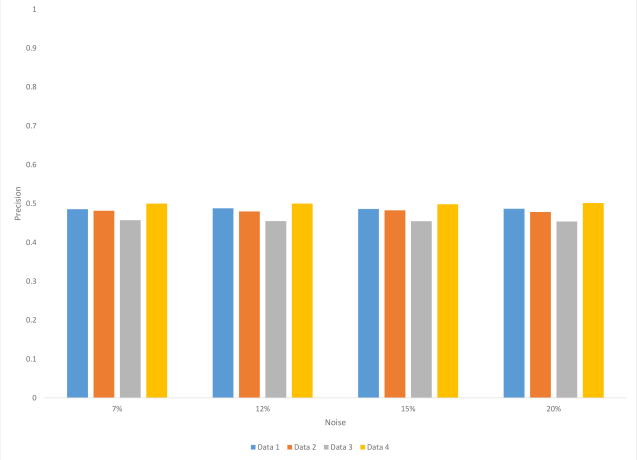


(a) Recall for false negative detections at varying noise levels

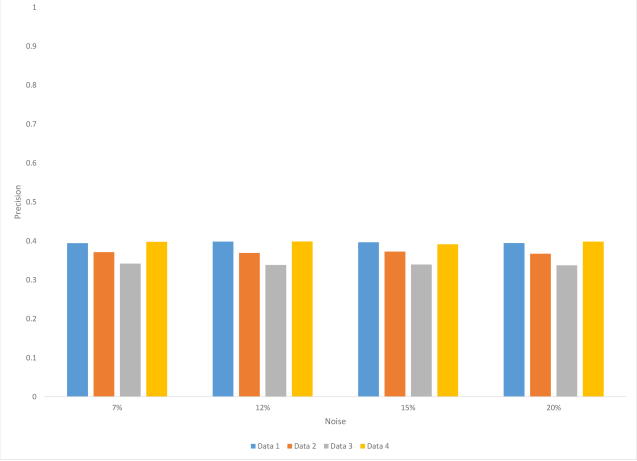


(b) Recall for false positive detections at varying noise levels

Figure 6.35. Recall for for DataSet 2, when model construction is CFS based. Threshold = 0.9

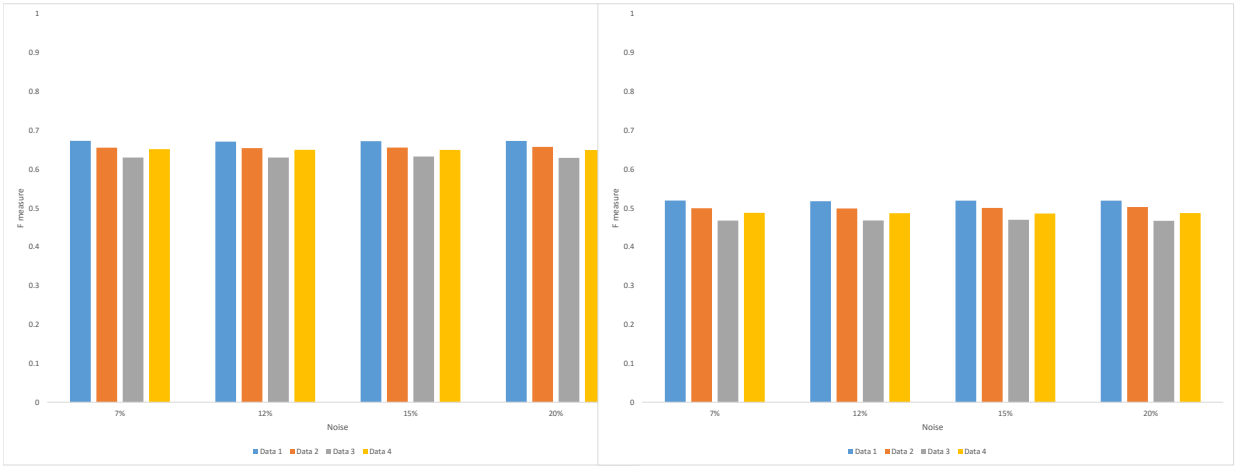


(a) Precision for false negative detections at varying noise levels



(b) Precision for false positive detections at varying noise levels

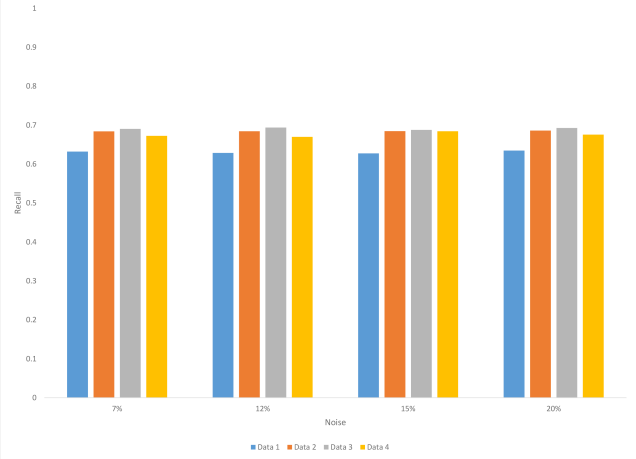
Figure 6.36. Precision for for DataSet 2, when model construction is CFS based. Threshold = 0.9



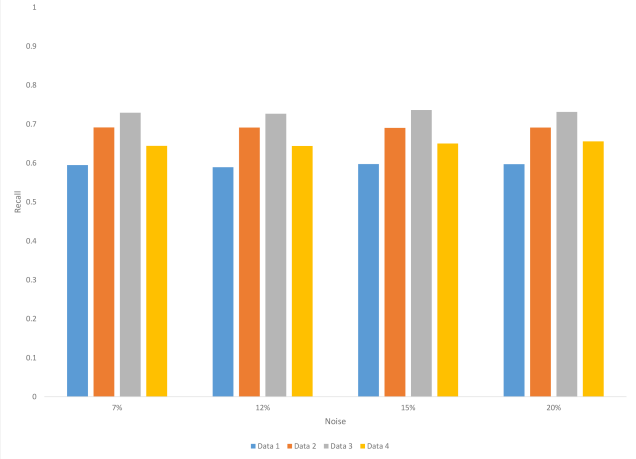
(a) F measure for false negative detections at varying noise levels

(b) F measure for false positive detections at varying noise levels

Figure 6.37. F measure for DataSet 2, when model construction is not CFS based. Threshold = 0.9

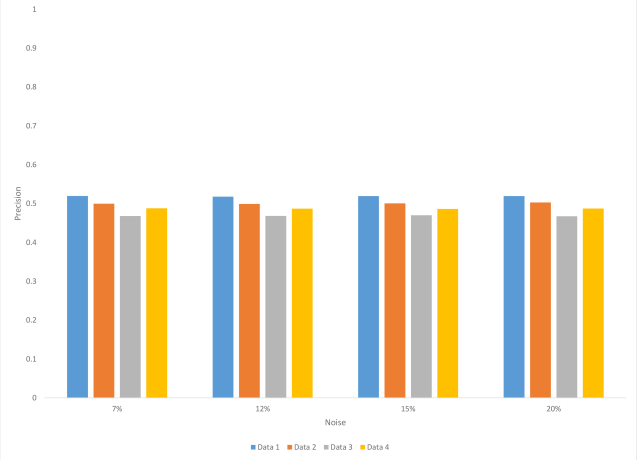


(a) Recall for false negative detections at varying noise levels

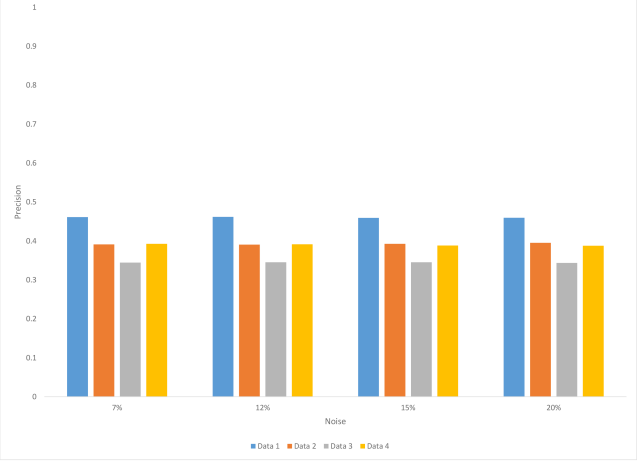


(b) Recall for false positive detections at varying noise levels

Figure 6.38. Recall for DataSet 2, when model construction is not CFS based. Threshold = 0.9



(a) Precision for false negative detections at varying noise levels



(b) Precision for false positive detections at varying noise levels

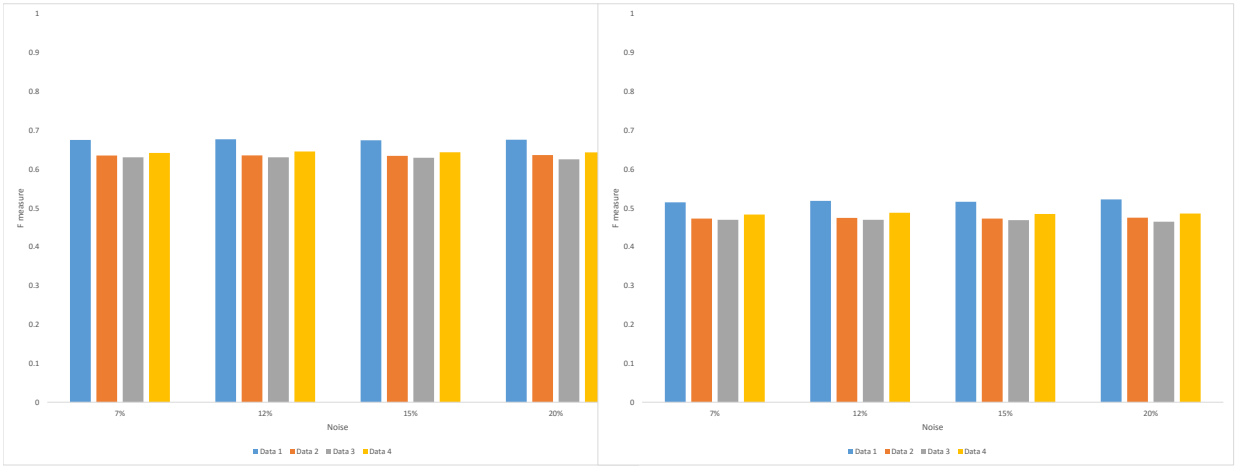
Figure 6.39. Precision for DataSet 2, when model construction is not CFS based. Threshold = 0.9

3. Effect of changing number of roles, keeping everything else constant.

Table 6.3 give parameters that we use for studying the impact of *varying number of roles* on performance of our approach.

Models based on feature selection. Figure 6.40(a),6.41(a),6.42(a), 6.40(b),6.41(b),6.42(b) shows performance in terms of F-measure, recall, and precision when threshold is 0.5, figure 6.46(a),6.47(a),6.48(a), 6.46(b), 6.47(b), 6.48(b), when threshold is 0.7, and figure 6.52(a), 6.53(a), 6.54(a), 6.52(b), 6.53(b), 6.54(b) when threshold is set to 0.9. Note that their classification models were based on feature selection technique (CFS).

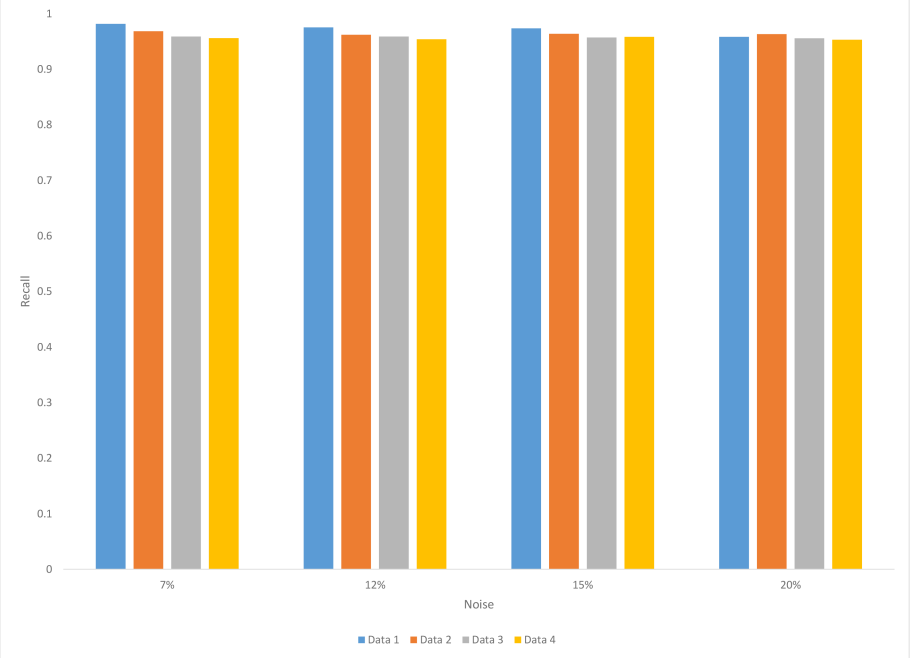
Models constructed without feature selection. Performance of models for this case which are constructed without employing feature selection are shown in figures 6.43(a), 6.44(a),6.45(a), 6.43(b), 6.26(b),6.27(b), 6.31(a), 6.32(a), 6.33(a), 6.49(b),6.50(b),6.51(b), 6.55(a), 6.56(a), 6.57(a), 6.55(b), 6.56(b), 6.57(b). In this case, f measure and recall for false positive detection improves when classification model is based on feature selection method, as can be seen in 6.40(b), 6.41(b), 6.53(b) and 6.52(b). Although with increase in number of roles the rate of detection starts to drop but still the rate goes no lower than 70%, which is quite good, as can be seen in figure 6.40, 6.43, 6.46, 6.49, 6.52 and 6.55.



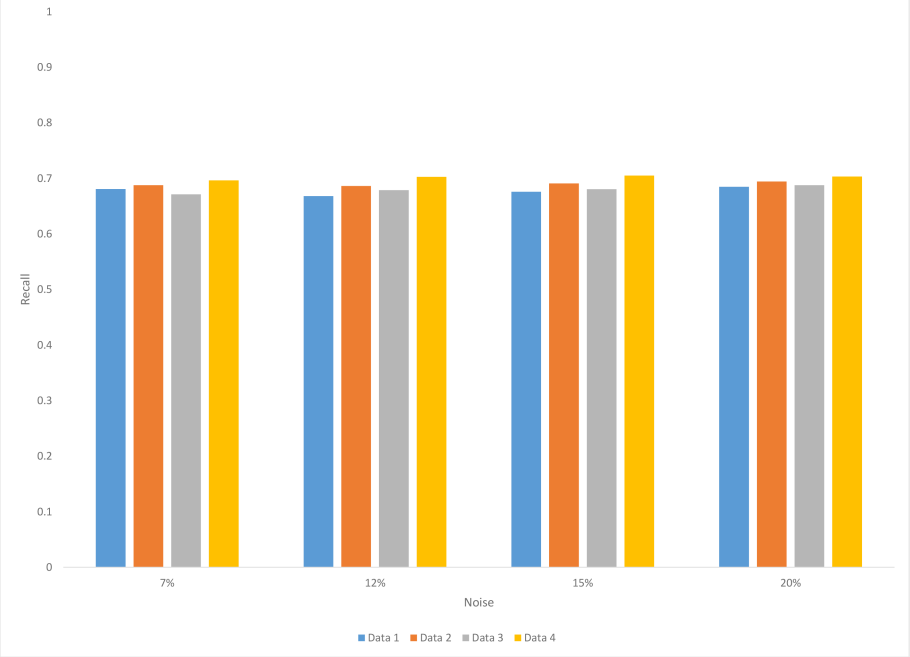
(a) F measure for false negative detections at varying noise levels

(b) F measure for false positive detections at varying noise levels

Figure 6.40. F measure for DataSet 3, when model construction is CFS based. Threshold = 0.5

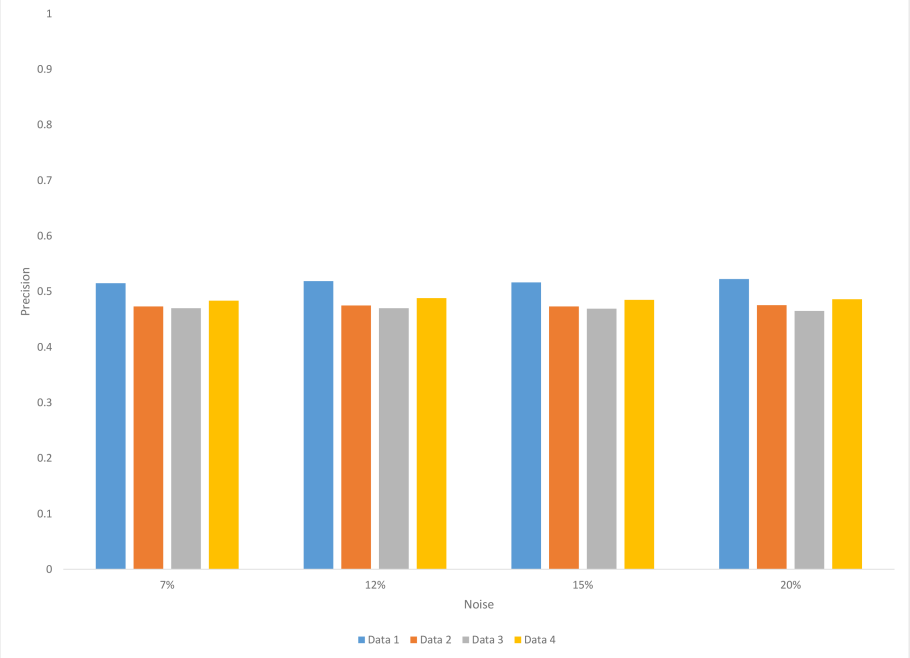


(a) Recall for false negative detections at varying noise levels

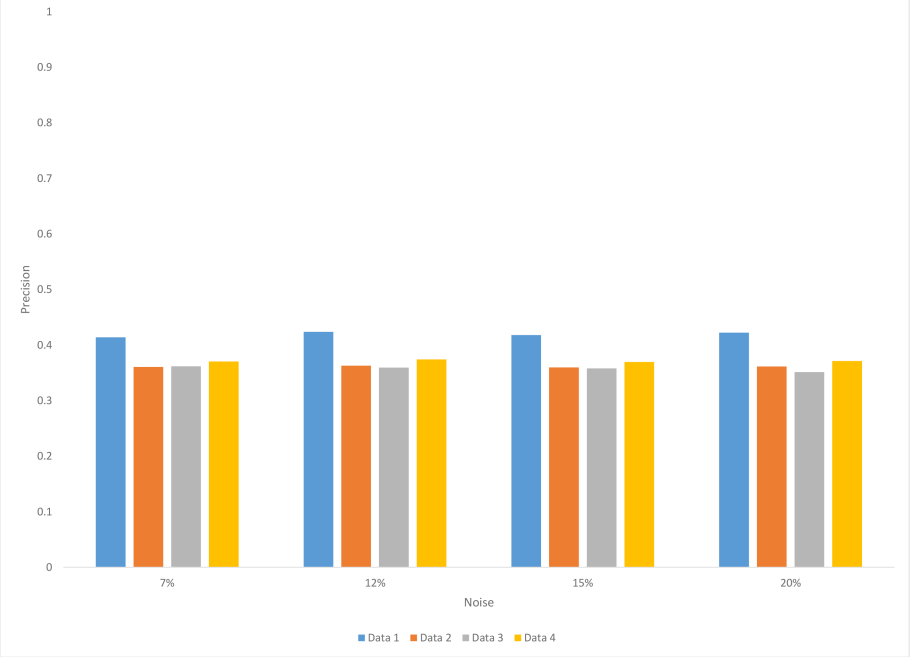


(b) Recall for false positive detections at varying noise levels

Figure 6.41. Recall for DataSet 3, when model construction is CFS based. Threshold = 0.5

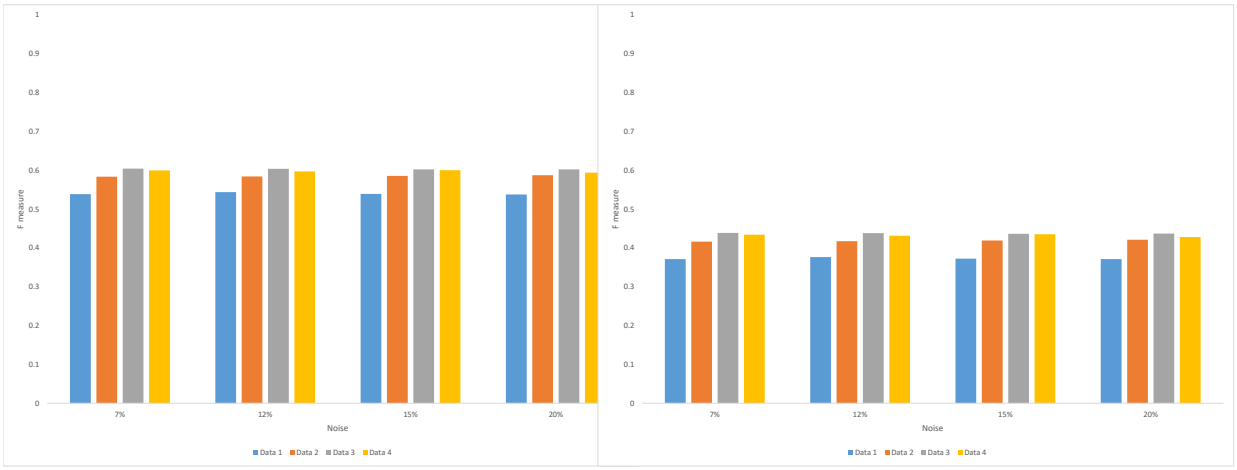


(a) Precision for false negative detections at varying noise levels



(b) Precision for false positive detections at varying noise levels

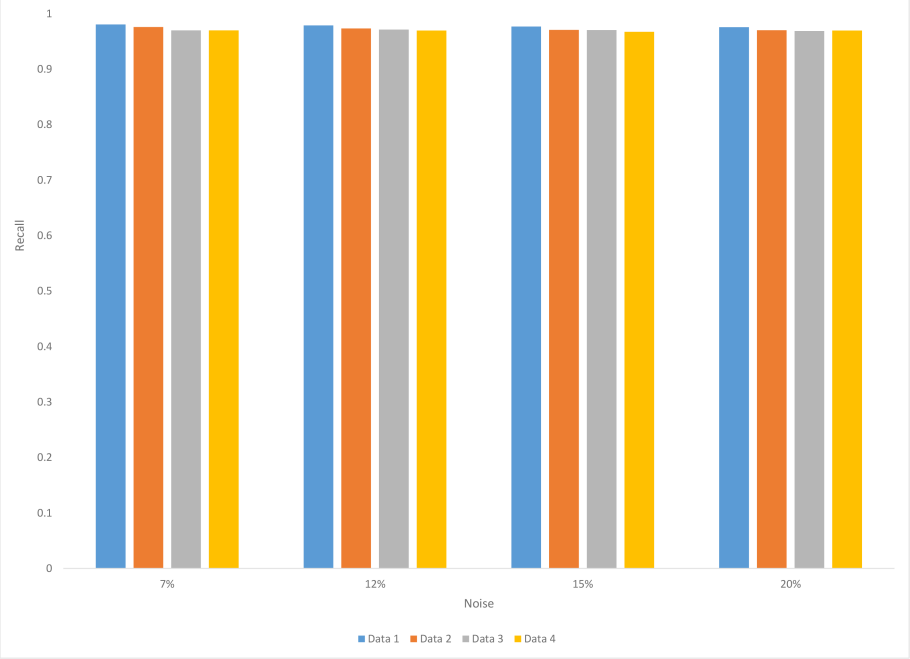
Figure 6.42. Precision for DataSet 3, when model construction is CFS based. Threshold = 0.5



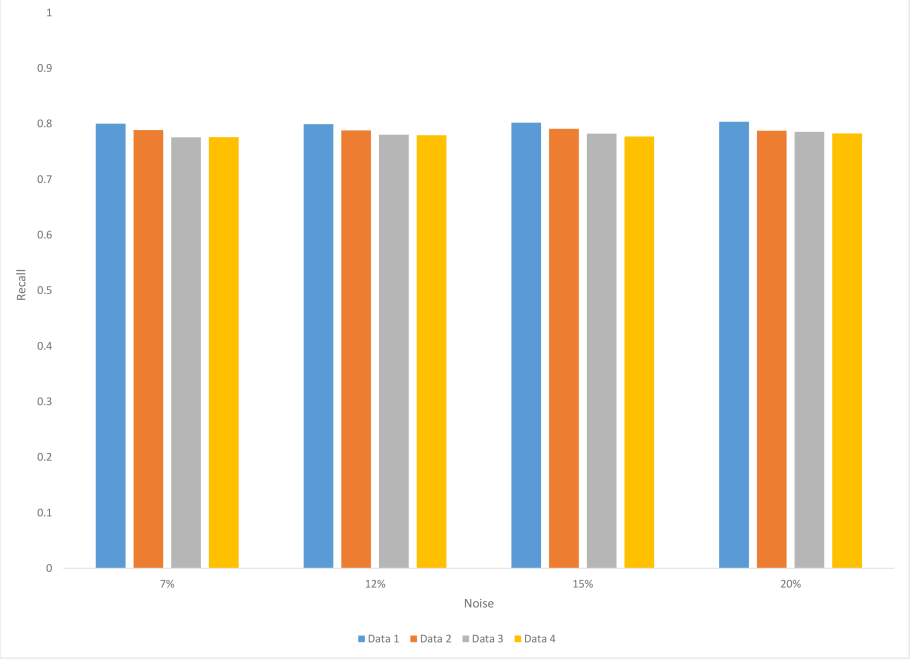
(a) F measure for false negative detections at varying noise levels

(b) F measure for false positive detections at varying noise levels

Figure 6.43. F measure for Predictive performance for DataSet 3, when model construction is not CFS based. Threshold = 0.5

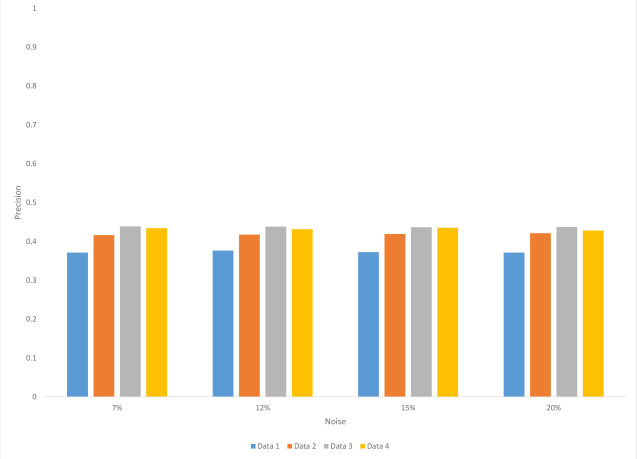


(a) Recall for false negative detections at varying noise levels

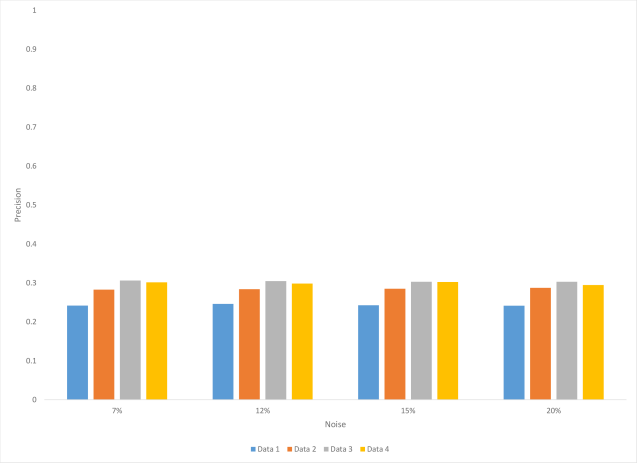


(b) Recall for false positive detections at varying noise levels

Figure 6.44. Recall for Predictive performance for DataSet 3, when model construction is not CFS based. Threshold = 0.5

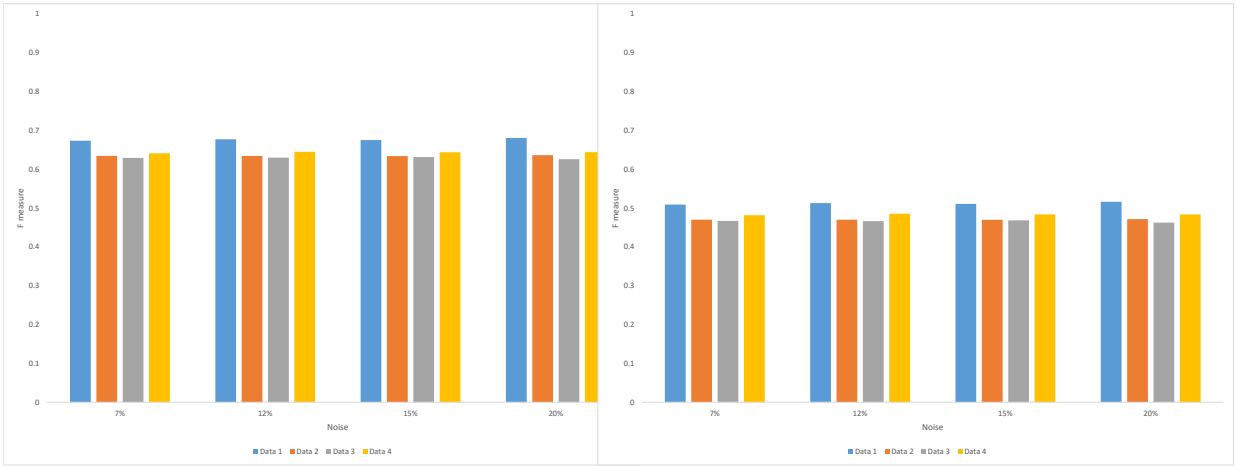


(a) Precision for false negative detections at varying noise levels



(b) Precision for false positive detections at varying noise levels

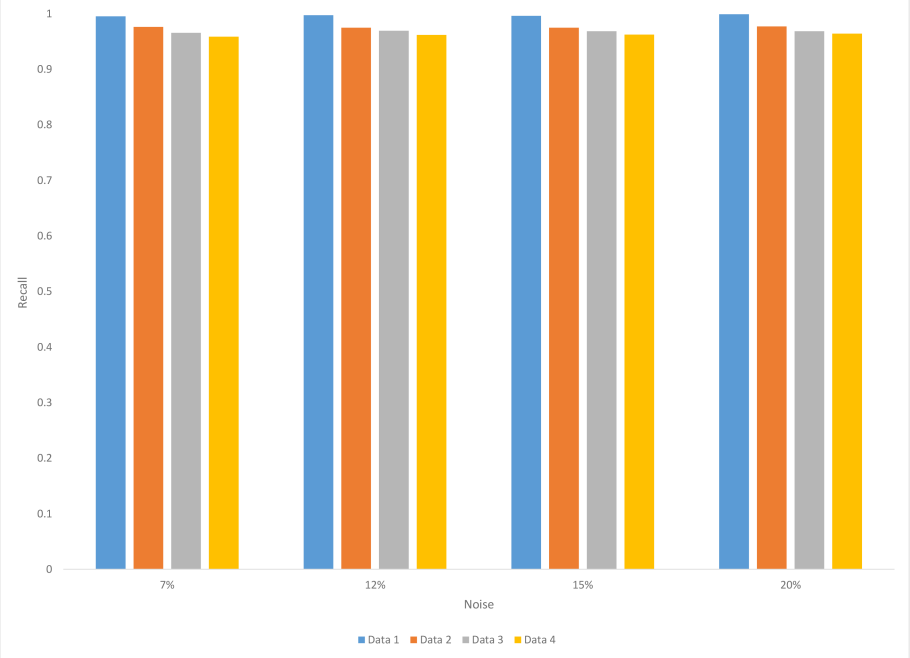
Figure 6.45. Precision for Predictive performance for DataSet 3, when model construction is not CFS based. Threshold = 0.5



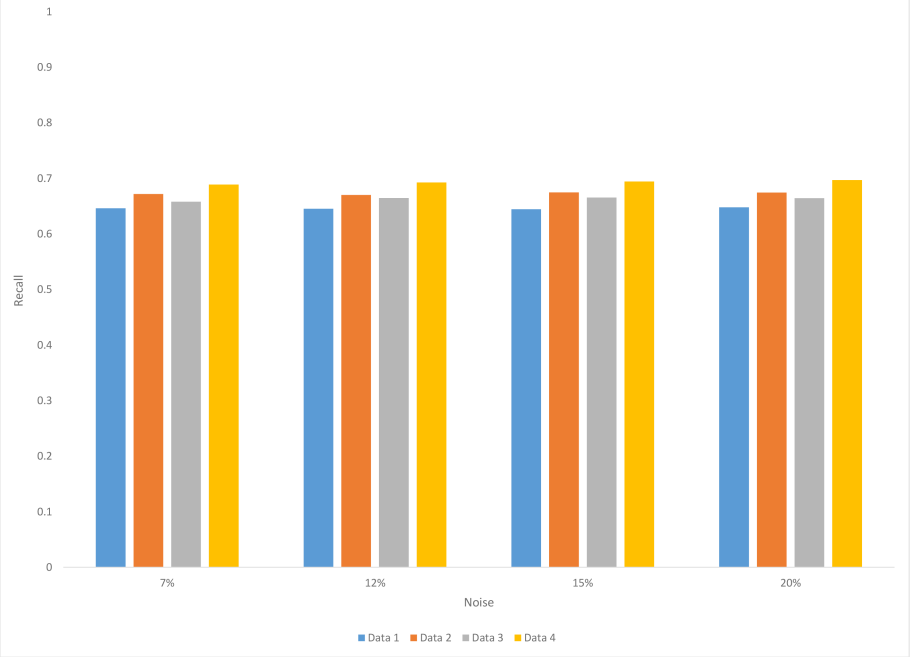
(a) F measure for false negative detections at varying noise levels

(b) F measure for false positive detections at varying noise levels

Figure 6.46. F measure for DataSet 3, when model construction is CFS based. Threshold = 0.5

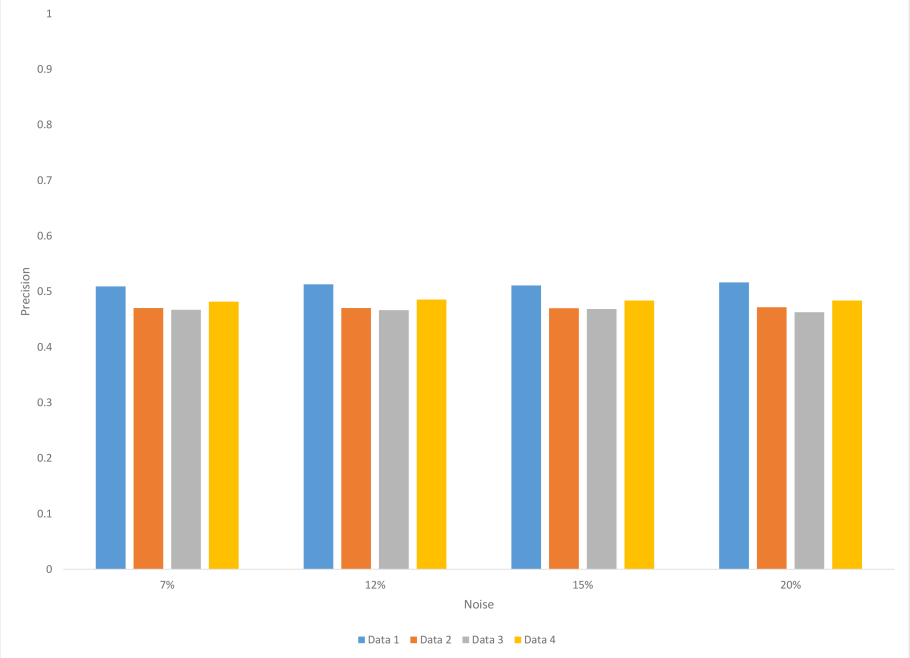


(a) Recall for false negative detections at varying noise levels

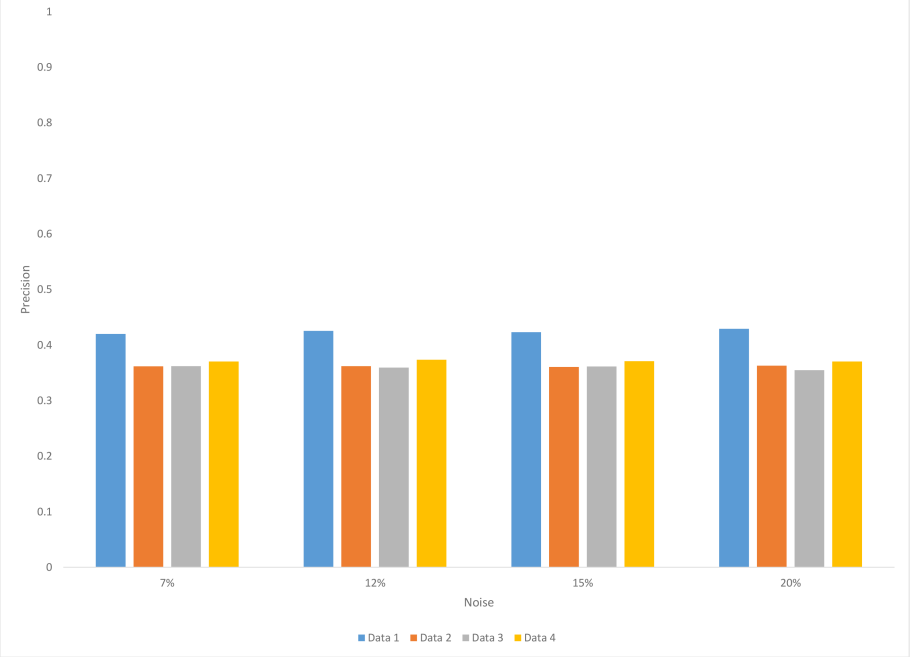


(b) Recall for false positive detections at varying noise levels

Figure 6.47. Recall for DataSet 3, when model construction is CFS based. Threshold = 0.5

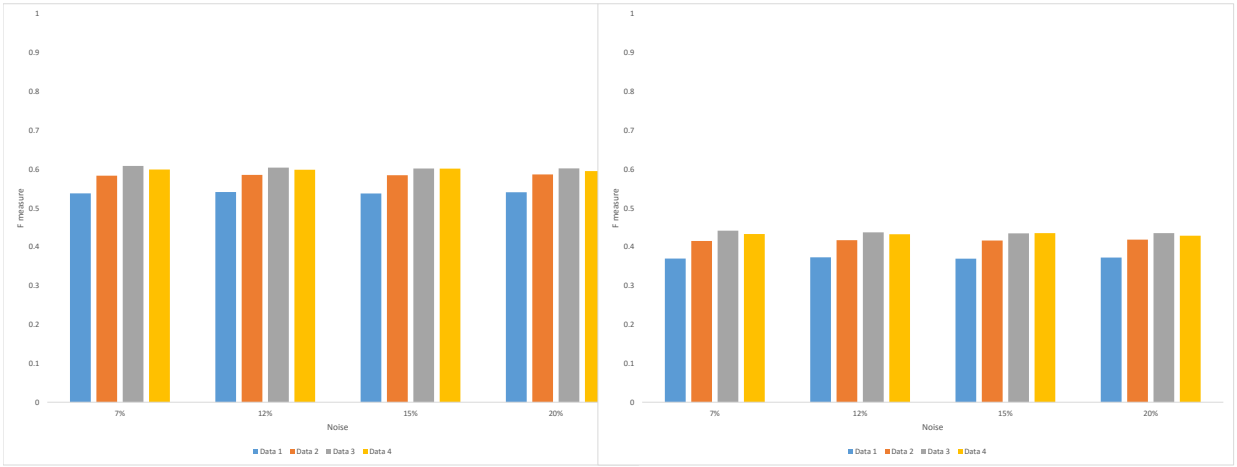


(a) Precision for false negative detections at varying noise levels



(b) Precision for false positive detections at varying noise levels

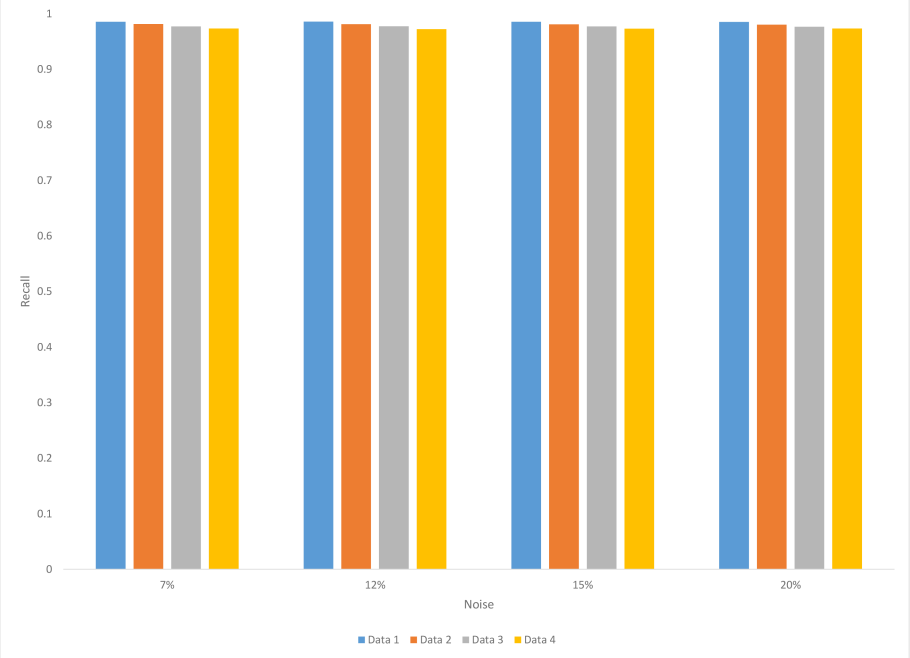
Figure 6.48. Precision for DataSet 3, when model construction is CFS based. Threshold = 0.5



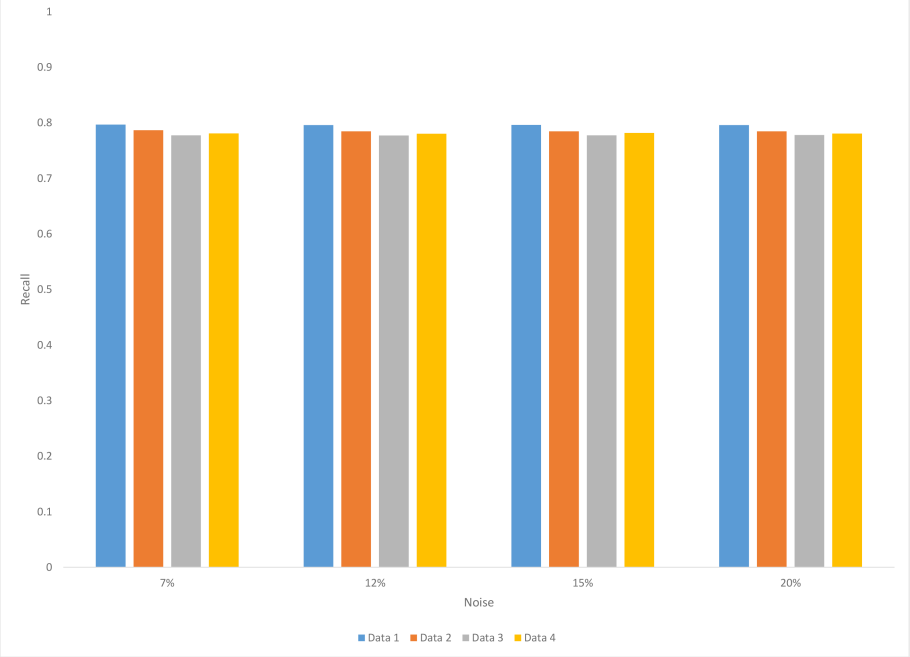
(a) F measure for false negative detections at varying noise levels

(b) F measure for false positive detections at varying noise levels

Figure 6.49. F measure for DataSet 3, when model construction is not CFS based. Threshold = 0.5

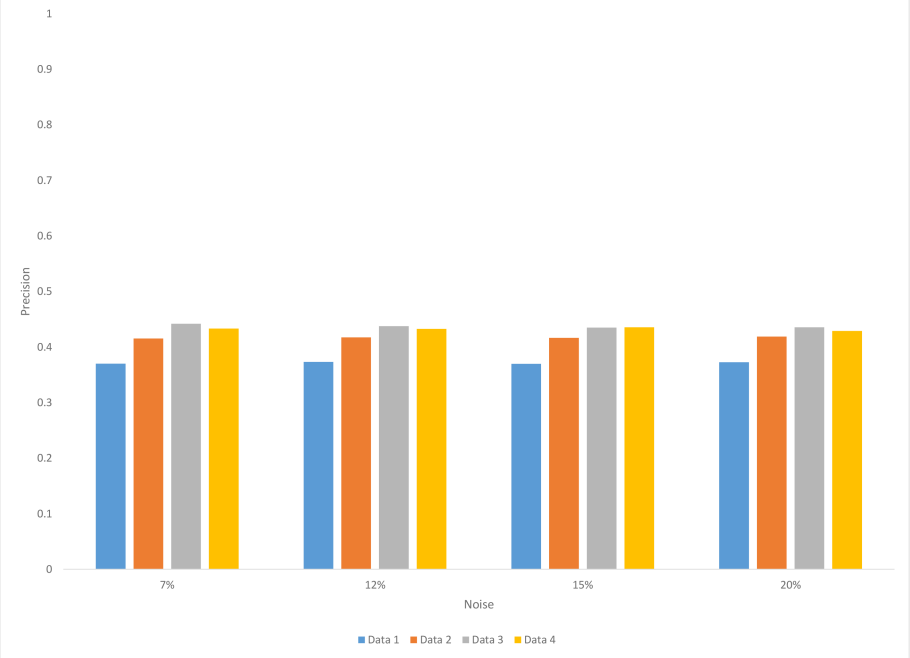


(a) Recall for false negative detections at varying noise levels

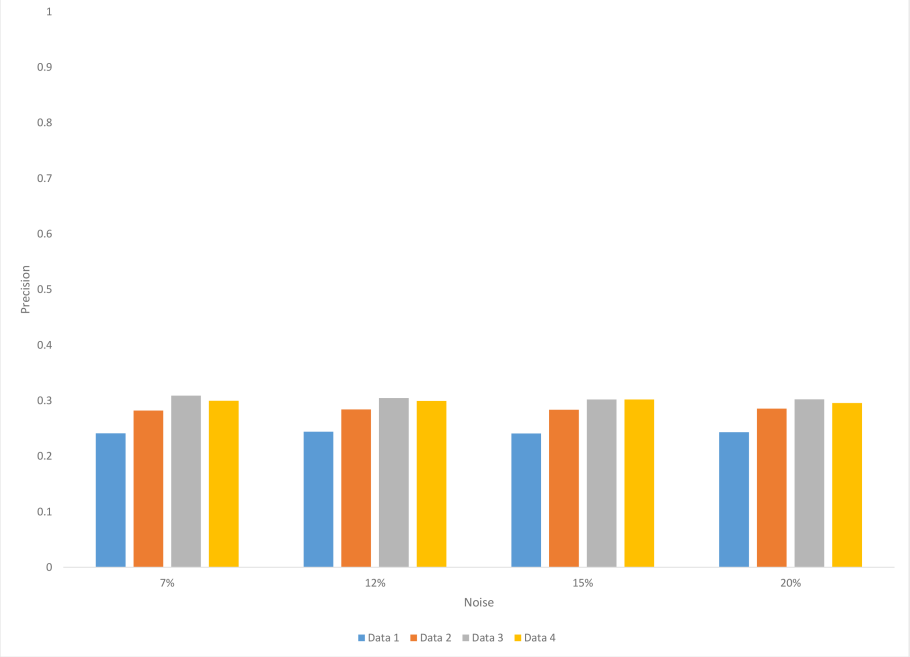


(b) Recall for false positive detections at varying noise levels

Figure 6.50. Recall for DataSet 3, when model construction is not CFS based. Threshold = 0.5

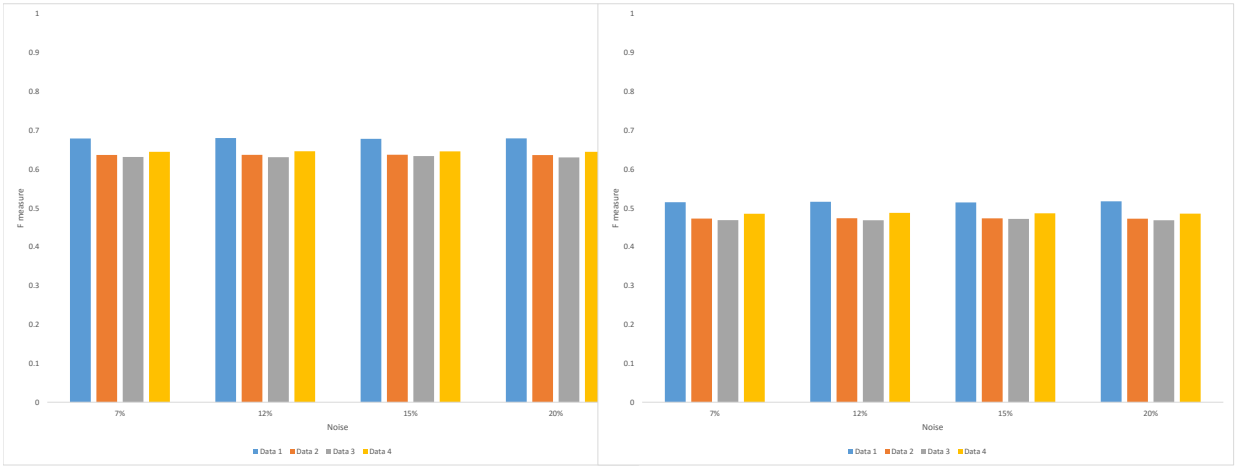


(a) Precision for false negative detections at varying noise levels



(b) Precision for false positive detections at varying noise levels

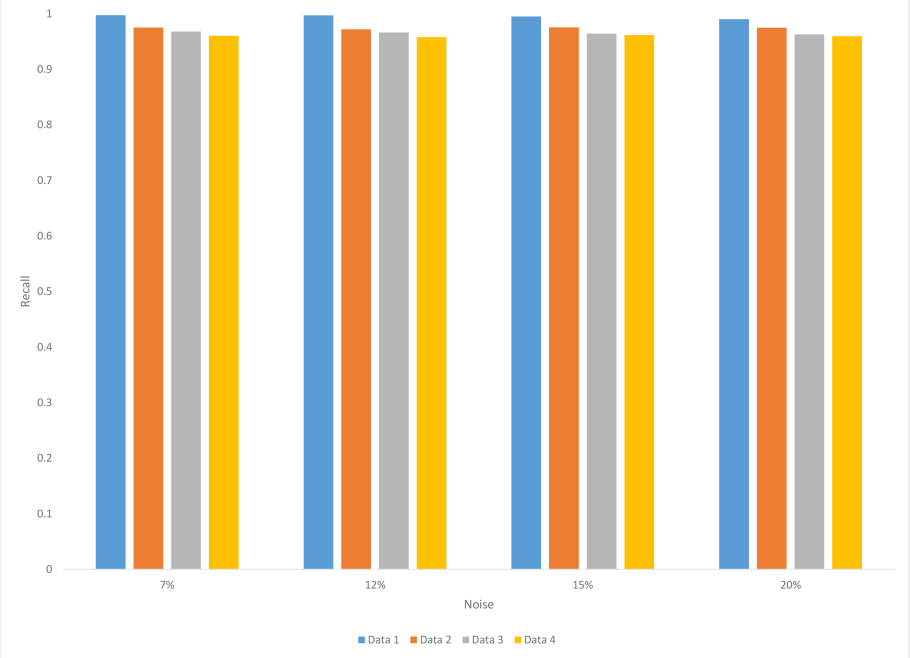
Figure 6.51. Precision for DataSet 3, when model construction is not CFS based. Threshold = 0.5



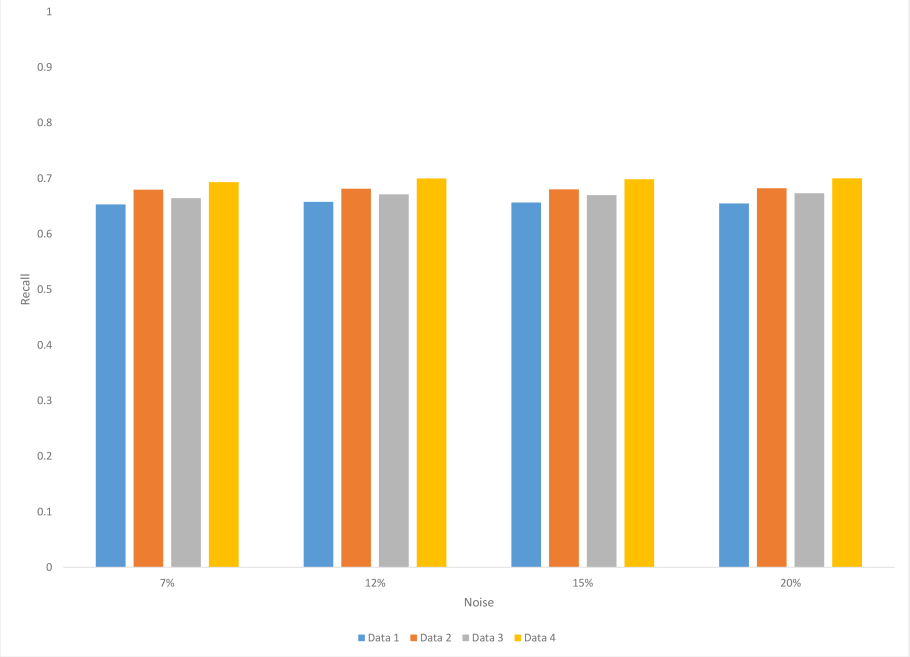
(a) F measure for false negative detections at varying noise levels

(b) F measure for false positive detections at varying noise levels

Figure 6.52. F measure for DataSet 3, when model construction is CFS based. Threshold = 0.9

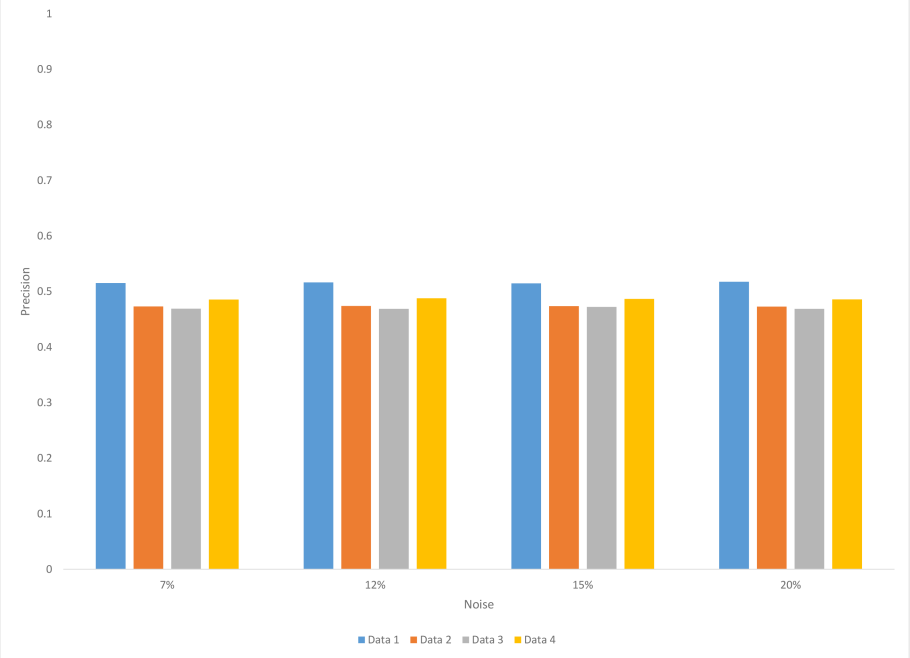


(a) Recall for false negative detections at varying noise levels

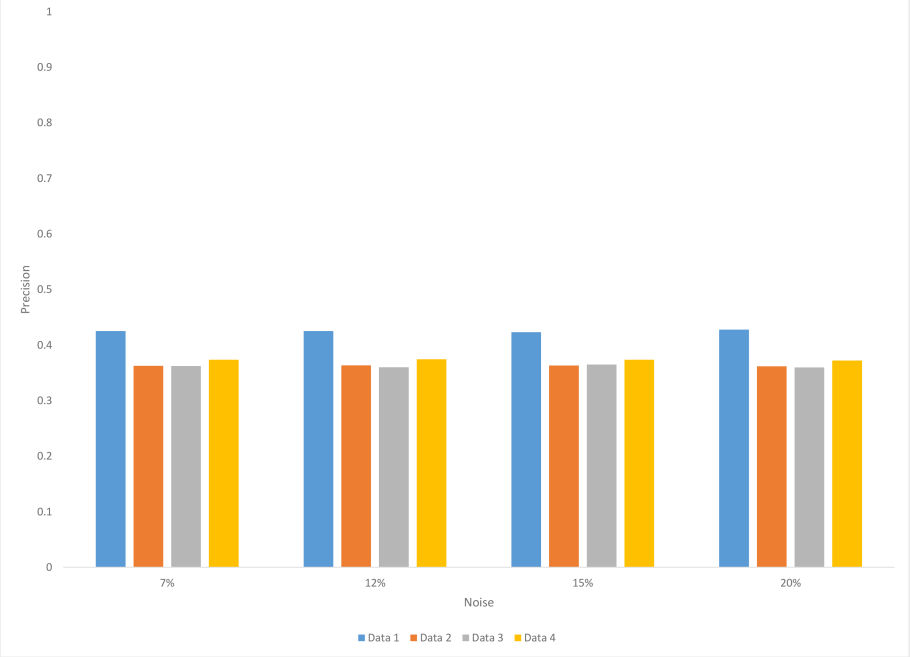


(b) Recall for false positive detections at varying noise levels

Figure 6.53. Recall for DataSet 3, when model construction is CFS based. Threshold = 0.9

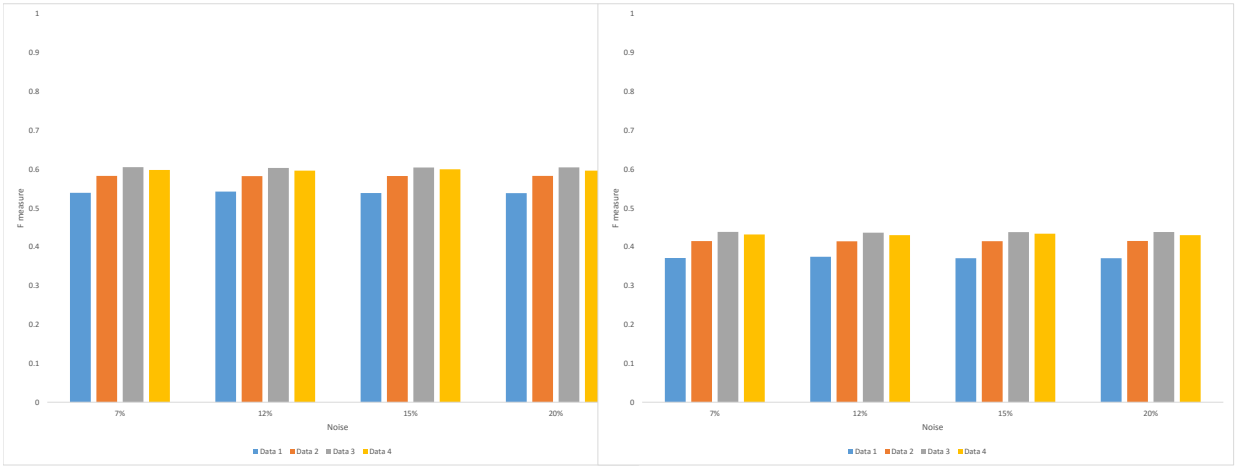


(a) Precision for false negative detections at varying noise levels



(b) Precision for false positive detections at varying noise levels

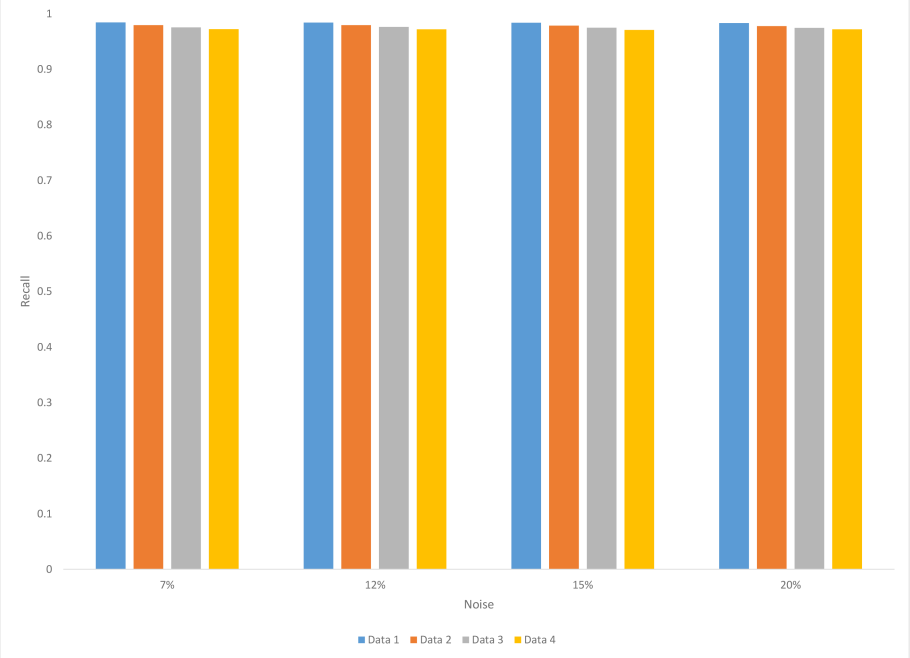
Figure 6.54. Precision for DataSet 3, when model construction is CFS based. Threshold = 0.9



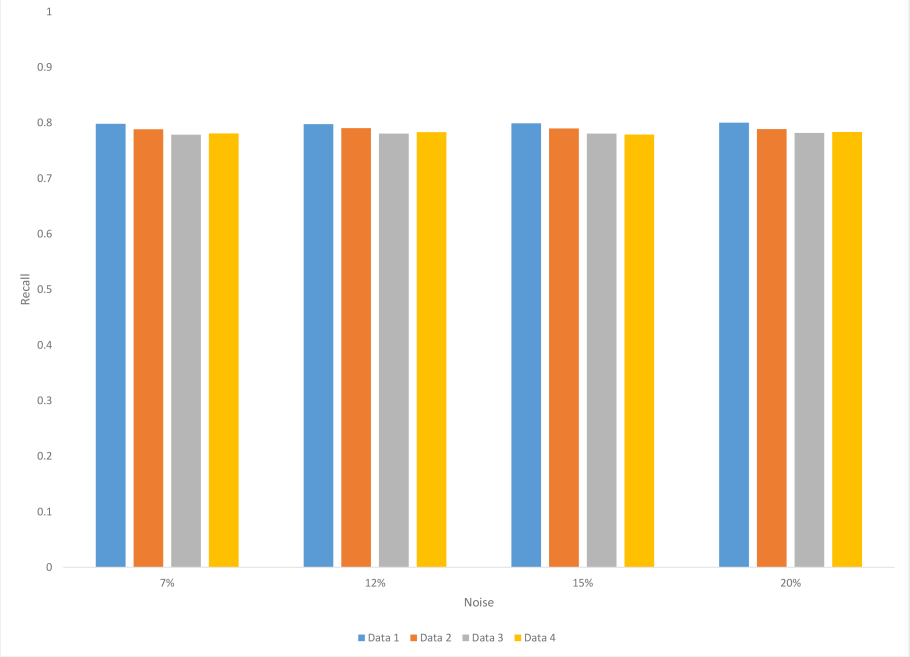
(a) F measure for false negative detections at varying noise levels

(b) F measure for false positive detections at varying noise levels

Figure 6.55. F measure for DataSet 3, when model construction is not CFS based. Threshold = 0.9

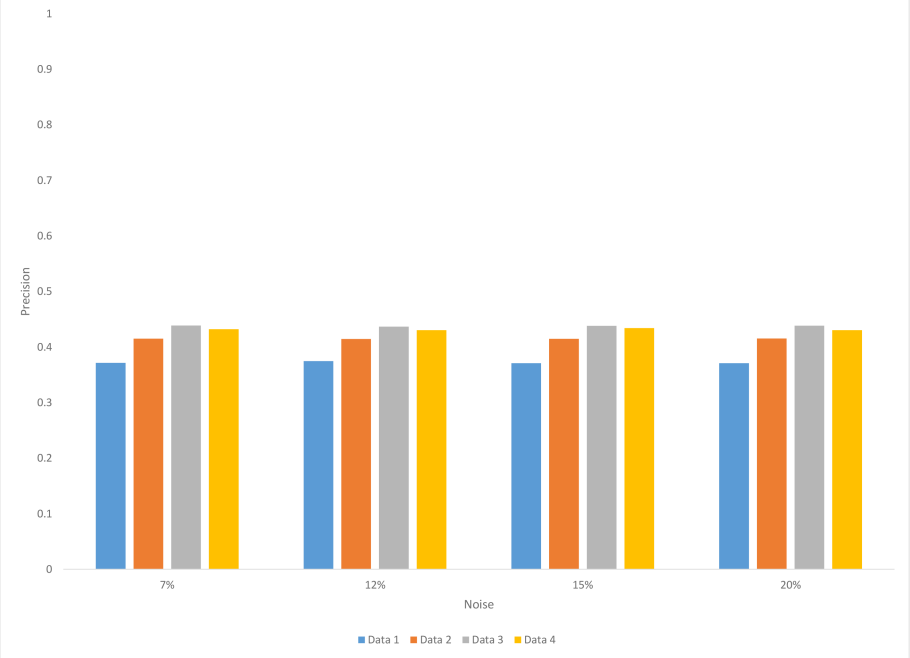


(a) Recall for false negative detections at varying noise levels

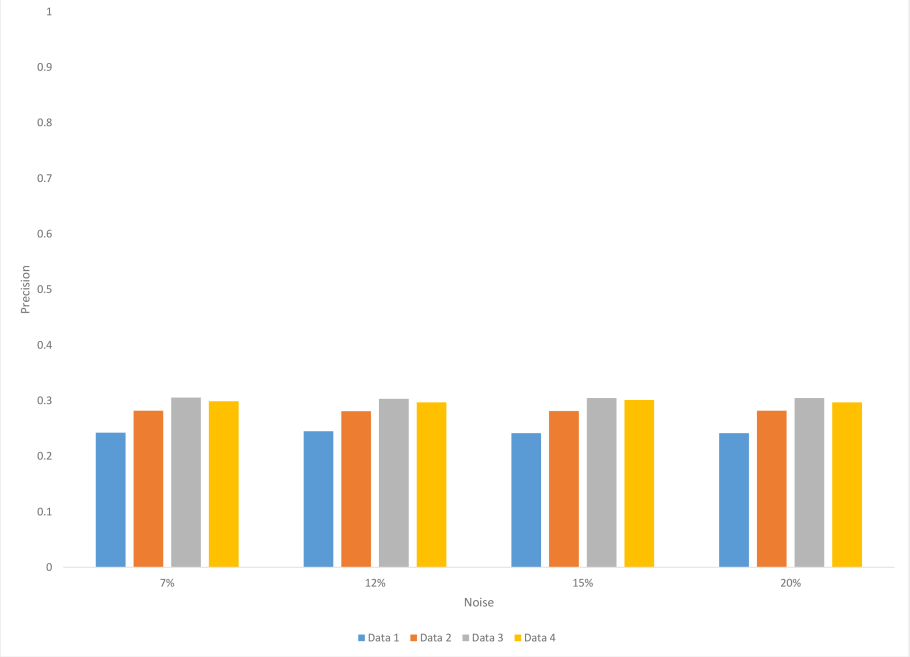


(b) Recall for false positive detections at varying noise levels

Figure 6.56. Recall for DataSet 3, when model construction is not CFS based. Threshold = 0.9



(a) Precision for false negative detections at varying noise levels



(b) Precision for false positive detections at varying noise levels

Figure 6.57. Precision for DataSet 3, when model construction is not CFS based. Threshold = 0.9

CHAPTER 7
SUMMARY OF THE CONTRIBUTIONS AND FUTURE RESEARCH

7.1 Summary of Contributions

This chapter concludes the dissertation by discussion of contributions. We also briefly discuss some future research direction of this work.

In this dissertation, we have addressed the key challenge of automating configuration management process in access control systems. Security configuration is constantly required and is critical in order to optimize the protection of organizational resources and blocking perspective attacks. Since the configuration search space is huge, performing this task manually is not trivial. Moreover, the security and functionality interferes with the process. The difficulty further comes from the fact that a system administrator usually has little knowledge on the semantic meanings of user responsibilities and business processes within an enterprise. Therefore, the likelihood of introducing errors in to a system gets higher. Ultimately, incorrect resolution leads to drastic consequences with potentially many severe service outages and downtime. Traditional access control systems lack an ability to anticipate potential errors in system configuration. Therefore, these systems fail to gracefully react to configuration management issues.

We demonstrated that the automatic mechanism to analyze system configuration can significantly alleviate the task of system administrators. In our effort to systematically analyze the existing configuration of access control system, we have proposed approaches based on data mining techniques. We have also described efficient algorithms to implement our strategy. Below, we

discuss highlights of three research issues that are addressed in this work.

Using Risk Estimates for Configuration Management.

Traditional access control is too restrictive since it does not permit any access that has not been pre-specified in the system, the break glass approaches are too permissive in that they allow all requested accesses based on the situation at hand. Since both of these models are at extreme, a fine balance between permissiveness and restrictiveness is desirable. This dissertation has proposed novel methodologies for dynamic computation of risk in situations where preventing access has more deleterious effect than granting it, if the underlying risk is low. Moreover, it has developed a model that facilitates automatic risk-based access control configuration in both DAC and RBAC cases. In both approaches, we first quantify the risk associated with the requested access, and permit access if it is less than a pre-specified threshold value. Also, in case of RBAC, it has developed method to determine situational role for a user. Our approaches to quantify the risk associated with granting an access are based on the technique of classification. The first approach, evaluates the risk of granting a permission based on the existing user-permission assignments (UPA), whereas second approach estimates risk associated with requested role or all such roles through which a user can acquire requested authorization. We have shown that the risk estimates can be used for facilitating automatic configuration of RBAC and DAC. Computational experiments using both synthetic and benchmark real datasets confirm the viability of our approaches to facilitate automated risk based access control.

Using attribute semantics for Configuration Management.

In this dissertation, we also investigated the configuration management problems that arise due to a requests from users of collaborating organizations that do not have explicit access to resources. This dissertation has proposed an efficient methodology to derive credential requirements for roles having permission to access requested object, based on local access control policies using existing access control data. Data mining techniques are employed to automate the process of determining credential requirements for external user to facilitate coalition based access control. Moreover, comparison of our approaches to existing method is also performed and proposed approach is supported by extensive experiments on both real and synthetic data.

Detection and Resolution of Misconfiguration in Access Control

Finally, we proposed approaches to automate the process of misconfiguration detection and resolution in deployed RBAC and DAC. Traditional access control system lacks an ability to anticipate potential misconfigurations. Therefore, these systems fail to gracefully react to configuration errors. Given, an access control data, our proposed methodologies facilitate the automated process of identifying inconsistent assignments in the system which are either mistakenly granted (over-assignment) or denied (under-assignment). Moreover, we have proposed methods to rectify them. Our proposed approaches rely on combining multiple classifiers to compute a misconfiguration score for a particular assignment. Computed score is then used to determine whether a given assignment is a misconfiguration or not. The goal of combining output of multiple classifiers is to leverage strength of each classifier, and to reduce the impact of classification errors made by an independent classifier. We have shown that the automated approaches for configuration management significantly alleviates the burdensome task of system operators to maintain the system configuration on regular basis. This may further prevent many severe service outages and downtime.

7.2 Future Research Plans

There are several possible future directions of this work. However, we plan to extend our proposed approaches to address limitations of our existing work, and to offer efficient solutions to more advanced access control problems. Specifically, we plan to work on the following:

Incorporating Attributes Data to Quantify Risk.

Our approaches to quantify risk are based on following assumptions:

(i.) The user submitting an access request is local,

(ii.) Either the RBAC or permission based authorization system is deployed within an organization.

(iii.) Existing permissions of a user are used to assess the risk associated with the authorization requested.

For future work, we plan to extend our approaches for handling access requests when a user belongs to an inter domain organization and existing permissions of user are either not known or they are not relevant. In such situation, credentials of a user may serve as useful information for determining whether a user qualifies for a requested authorization or not. It will be interesting to develop an efficient and multidimensional model that takes object-attributes data, user-attributes data, and permission authorizations of local user into an account while risk is computed. This can be done by employing our dynamic risk computation method within attribute based access control (ABAC) mechanism [56], [33]. And, while the work on configuration management was underway, research on cloud based security become more prevalent. Rapidly increasing interest of businesses in adoption of cloud deployments, compel us to extend our work to address security configuration challenges in cloud based environment where the argument around conflict among security and privacy issues often comes up. It will be interesting to apply our work in distributed

and collaborative environments where users may belong to an inter-domain organization.

Extension of method to automatically detect and resolve misconfiguration in access control.

In our existing work, we assume that core RBAC is implemented within an organization. As a part of future work we plan to extend our approaches to handle more complex extensions of RBAC, one of which is Hierarchical RBAC. Though it is quite challenging because of the inherent structural complexity in hierarchical RBAC, but we first plan to look in to an effective method to convert hierarchical RBAC into flat RBAC without loosing the information about the relationship among entities, and then applying our existing work to automate the error detecting and removal process. Moreover, we also plan to study how the optimal mix of classifiers can be selected. We also plan to look into the methods which help system administrators to select suitable threshold values for tasks of different nature.

In our existing work, we assume that inconsistent assignments present in either UA matrix or *UPA* matrix. We are not considering a possibility of inconsistencies present in *UA* or *PA* matrices. In order to overcome this limitation, we plan to incorporate these two matrices as well to our model. We also plan to evaluate the performance of our proposal presented in chapter 6 using real data sets.

Updateable security models.

Our main goal for future work is to extend our existing models to facilitate automatic update of access control system. Survey of literature shows that approaches to automatically incorporate updates in access control are still missing. For example, in most organizations, role engineering process is considered as one time process. Although, over a period of time, there could be need of new set of roles, or even existing roles might need new set of configuration. Traditional systems lacks an ability to incorporate such changes automatically. We plan to address these issues in our

future work.

REFERENCES

[1] Google drive, dropbox, box and icloud reach the top 5 cloud storage security breaches list. https://psg.hitachi-solutions.com/credeon/blog/google-drive-dropbox-box-and-icloud-reach-the-top-5-cloud-storage-secur note = Nov 20, 2014 8:00:00 AM.

[2] Owl web ontology language guide. available at http://www.w3.org/tr/owl-guide/.

[3] Why gmail went down: Google misconfigured load balancing servers (updated). http://arstechnica.com/information-technology/2012/12/why-gmail-went-down-google-misconfigured-chromes-sync-server/. 12/11/2012, 4:25 PM.

[4] V. Atluri and J. Warner. Automatic enforcement of access control policies among dynamic coalitions. In *International Conference on Distributed Computing and Internet Technology*, December 2004.

[5] J. Bacon, K. Moody, and W. Yao. A model of oasis role-based access control and its support for active security. *ACM Transactions on Information and System Security*, 5(4):492–540, November 2002.

[6] N. Badar, J. Vaidya, V. Atluri, and B. Shafiq. Risk based access control using classification. *In 5th Symposium on Configuration Analytics and Automation (SafeConfig 2012)*.

[7] L. A. Barroso, J. Clidaras, and U. Hlzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second Edition*. 2013.

[8] L. Bauer, S. Garriss, and M. K. Reiter. Detecting and resolving policy misconfigurations in access-control systems. In *SACMAT*, pages 185–194, 2008.

[9] L. Bauer, Y. Liang, M. K. Reiter, and C. Spensky. Discovering access-control misconfigurations: new approaches and evaluation methodologies. In *Proceedings of the second ACM conference on Data and Application Security and Privacy*, CODASPY '12, pages 95–104, New York, NY, USA, 2012. ACM.

[10] V. Bharadwaj and J. Baras. A framework for automated negotiation of access control policies. *Proceedings of DISCEX III*, 2003.

[11] L. Breiman. Random forests. *Mach. Learn.*, 45:5–32, October 2001.

[12] A. Brucker and D. Hutter. Information flow in disaster management systems. In *Availability, Reliability, and Security, 2010. ARES '10 International Conference on*, pages 156 –163, feb. 2010.

[13] A. D. Brucker and H. Petritsch. Extending access control models with break-glass. In *Proceedings of the 14th ACM symposium on Access control models and technologies*, SACMAT '09, pages 197–206, New York, NY, USA, 2009. ACM.

[14] C. J. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167. 10.1023/A:1009715923555.

[15] J. Cendrowska. Prism: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27(4):349 – 370, 1987.

[16] C.-C. Chang and C.-J. Lin. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, May 2011.

[17] B. Chen and L. He. An Extensible Framework for RBAC in Dynamic Ad-Hoc Coalitions. In *International Conference on Network Computing and Information Security*, 2011.

[18] P.-C. Cheng, P. Rohatgi, C. Keser, P. Karger, G. Wagner, and A. Reninger. Fuzzy multi-level security: An experiment on quantified risk-adaptive access control. In *Security and Privacy, 2007. SP '07. IEEE Symposium on*, pages 222 –230, may 2007.

[19] S. A. Chun and V. Atluri. Risk-based access control for personal data services. *Algorithms, Architectures and Information Systems Security*, 3:263, 2009.

[20] S. A. Chun, J. H. Kwon, and H. Lee. Situation role-based privacy control using dynamic credentials for emergency health services. *SIGHIT Rec.*, 2(1):5–5, Mar. 2012.

[21] E. Cohen, W. Winsborough, R. Thomas, and D. Shands. Models for coalition-based access control (cbac). *SACMAT*, 2002.

[22] T. Das, R. Bhagwan, and P. Naldurg. Baaz: A system for detecting access control misconfigurations. In *USENIX Security Symposium*, pages 161–176. USENIX Association, 2010.

[23] D. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R. Chandramouli. Proposed nist standard for role-based access control. *TISSEC*, 2001.

[24] A. Ferreira, D. W. Chadwick, P. Farinha, R. J. C. Correia, G. Zhao, R. Chilro, and L. F. C. Antunes. How to securely break into rbac: The btg-rbac model. In *ACSAC*, pages 23–31, 2009.

[25] A. Ferreira, R. Cruz-Correia, L. Antunes, P. Farinha, E. Oliveira-Palhares, D. W. Chadwick, and A. Costa-Pereira. How to break access control in a controlled manner. In *Proceedings of the 19th IEEE Symposium on Computer-Based Medical Systems*, CBMS '06, pages 847–854, Washington, DC, USA, 2006. IEEE Computer Society.

[26] D. L. Fisher. Data, documentation, and decision tables. *Commun. ACM*, 9(1):26–31, Jan. 1966.

[27] K. Frikken, M. Atallah, and J. Li. Attribute-based access control with hidden policies and hidden credentials. *Computers, IEEE Transactions on*, 55(10):1259–1270, 2006.

[28] J. Gray. Why do computers stop and what can be done about it?, 1985.

[29] M. A. Hall. *Correlation-based Feature Subset Selection for Machine Learning*. PhD thesis, University of Waikato, Hamilton, New Zealand, 1998.

[30] R. Housley, W. Polk, W. Ford, and D. Solo. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. *RFC 3280*, April 2002.

[31] J. Hu, Y. Zhang, R. Li, and Z. Lu. Role updating for assignments. In *Proceedings of the 15th ACM symposium on Access control models and technologies*, SACMAT '10, pages 89–98, New York, NY, USA, 2010. ACM.

[32] V. C. Hu, D. Ferraiolo, D. R. Kuhn, I. T. L. N. I. of Standards, and Technology). *Assessment of access control systems [electronic resource] / Vincent C. Hu, David F. Ferraiolo, D. Rick Kuhn*. U.S. Dept. of Commerce, National Institute of Standards and Technology [Gaithersburg, Md.], 2006.

[33] V. C. Hu, D. Ferraiolo, R. Kuhn, A. R. Friedman, A. J. Lang, M. M. Cogdell, A. Schnitzer, K. Sandlin, R. Miller, K. Scarfone, V. C. Hu, D. Ferraiolo, R. Kuhn, A. R. Friedman, A. J.

Lang, M. M. Cogdell, A. Schnitzer, K. Sandlin, R. Miller, K. Scarfone, and S. Cybersecurity. Guide to attribute based access control (abac) definition and considerations (draft), 2013.

[34] S. Kandala, R. Sandhu, and V. Bhamidipati. An attribute based framework for risk-adaptive access control models. *Availability, Reliability and Security, International Conference on*, 0:236–241, 2011.

[35] H. Khurana, S. Gavrila, R. Bobba, R. Koleva, A. Sonalker, E. Dinu, V. Gligor, and J. Baras. Integrated security services for dynamic coalitions. *Proc. of the DISCEX III*, 2003.

[36] R. Krishnan, J. Niu, R. Sandhu, and W. H. Winsborough. Group-centric secure information-sharing models for isolated groups. *ACM Trans. Inf. Syst. Secur.*, 14(3):23:1–23:29, Nov. 2011.

[37] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust-management framework. In *IEEE Symposium on Security and Privacy*, pages 114–130, 2002.

[38] N. Li, W. H. Winsborough, and J. C. Mitchell. Distributed credential chain discovery in trust management. *Journal of Computer Security*, 11(1):35–86, 2003.

[39] H. Liu and H. Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[40] S. Marinovic, R. Craven, J. Ma, and N. Dulay. Rumpole: A flexible break glass access control model. *SACMAT*, 2011.

[41] L. C. Molina, L. Belanche, and A. Nebot. Feature selection algorithms: A survey and experimental evaluation. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, ICDM '02, pages 306–, Washington, DC, USA, 2002. IEEE Computer Society.

[42] I. Molloy, L. Dickens, C. Morisset, and P.-C. Cheng. Risk based access control decisions under uncertainty. *Report on Risk Based Access Control March*, 2011.

[43] I. Molloy, N. Li, T. Li, Z. Mao, Q. Wang, and J. Lobo. Evaluating role mining algorithms. In B. Carminati and J. Joshi, editors, *SACMAT*, pages 95–104. ACM, 2009.

[44] R. Mukkamala, V. Atluri, and J. Warner. A distributed service registry for resource sharing among ad-hoc dynamic coalitions. In *Lecture Notes in Computer Science*. IFIP, December 2005.

[45] R. Mukkamala, V. Kamisetty, and P. Yedugani. Detecting and resolving misconfigurations in role-based access control (short paper). In *Proceedings of the 5th International Conference on Information Systems Security*, ICISS '09, pages 318–325, Berlin, Heidelberg, 2009. Springer-Verlag.

[46] Q. Ni, E. Bertino, and J. Lobo. Risk based access control systems built on fuzzy inferences. *ASIAACCS*, 2010.

[47] N. Nissanke and E. J. Khayat. Risk based security analysis of permissions in rbac. In *Proceedings of the 2 nd International Workshop on Security In Information Systems, Security In Information Systems*, pages 332–341. INSTICC Press, 2004.

[48] C. Philips, E. Charles, T. Ting, and S. Demurjian. Towards information assurance in dynamic coalitions. *IEEE IAW, USMA*, February 2002.

[49] C. Philips, T. Ting, , and S. Demurjian. Information sharing and security in dynamic coalitions. *SACMAT*, 2002.

[50] D. M. W. Powers. Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation. Technical Report SIE-07-001, School of Informatics and Engineering, Flinders University, Adelaide, Australia, 2007.

[51] R. B. Prasad Naldurg and T. Das. Understanding policy intent and misconfigurations from implementations: Consistency and convergence. 2011.

[52] J. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[53] J. Quinlan. Improved use of continuous attributes in c4. 5. *arXiv preprint cs/9603103*, 1996.

[54] J. R. Quinlan. Induction of decision trees. *Mach. Learn*, pages 81–106, 1986.

[55] M. Sanderson and W. B. Croft. Deriving concept hierarchies from text. In *SIGIR*, pages 206–213, 1999.

[56] R. Sandhu. Attribute-based access control models and beyond. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '15, Singapore, April 14-17, 2015*, page 677, 2015.

[57] R. Sandhu, D. Ferraiolo, and R. Kuhn. The nist model for role-based access control: towards a unified standard. In *Proceedings of the fifth ACM workshop on Role-based access control*, RBAC '00, pages 47–63, New York, NY, USA, 2000. ACM.

[58] B. Shafiq, J. Vaidya, A. Ghafoor, and E. Bertino. A framework for verification and optimal reconfiguration of event-driven role based access control policies. In *SACMAT*, pages 197–208, 2012.

[59] J. Vaidya, V. Atluri, and Q. Guo. The role mining problem: Finding a minimal descriptive set of roles. In *In Symposium on Access Control Models and Technologies (SACMAT*, pages

175–184, 2007.

[60] J. Vaidya, V. Atluri, and J. Warner. Roleminer: mining roles using subset enumeration. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, pages 144–153, New York, NY, USA, 2006. ACM.

[61] J. Vaidya, V. Atluri, J. Warner, and Q. Guo. Role engineering via prioritized subset enumeration. *Dependable and Secure Computing, IEEE Transactions on*, 7(3):300 –314, july-sept. 2010.

[62] L. Wang, D. Wijesekera, and S. Jajodia. A logic-based framework for attribute based access control. In *FMSE'04*, October 2004.

[63] J. Warner, V. Atluri, and R. Mukkamala. An attribute graph based approach to map local access control policies to credential based access control policies. In *ICISS*, pages 134–147, 2005.

[64] J. Warner, V. Atluri, and R. Mukkamala. A credential-based approach for facilitating automatic resource sharing among ad-hoc dynamic coalitions. In *IFIP*, August 2005.

[65] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, San Francisco, Oct. 1999.

[66] T. Yu, M. Winslett, and K. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Transactions on Information and System Security*, 6(1):1–42, February 2003.