UNIVERSITY OF BONN

MASTER THESIS

Extraction and Fusion of Identity Data

Author:
YUJIE DIAO

Supervisors:
PROF. J. LEHMANN
H. JABEEN
Examiner:
PROF. S. AUER

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Institute of Computer Science
University of Bonn

September 7, 2016

Declaration of Authorship

I, YUJIE DIAO, declare that this thesis titled, "Extraction and Fusion of Identity Data" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. Except for such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

Abstract

It is crucially important to provide new staff members with company identities so that they have access to the resources they need to do their jobs. Equally important but often overlooked, is the need to retract a staff member's access when they leave the company. Identity Management (**IdM**), also known as Identity and Access Management (**IAM**), is considered a critical foundation for realizing the business benefits in terms of privacy, security cost saving, management control, operational efficiency and more. IAM is the security discipline that enables the right individuals to access the right resources at the right times for the right reasons. This thesis targets identity deprovisioning in identity life–cycle management, one component in the IAM framework. The task is to analyze the attributes presented in a user object stored in a Microsoft Active Directory (**MS AD**) such that it is classified as active or passive. User attributes will be retrieved via the LDAP protocol from the Active Directory servers of SMS group GmbH and stored in ARFF files in order to utilize the data mining software, Weka. Furthermore, the results returned from different machine learning algorithms with different parameter settings will be compared and analyzed so that the most proper algorithms and improvements on them for this task are suggested. Finally, the attributes which are closely concerned with this classification task are concluded.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to Dr.–Ing. habil. Ferdinand Klaus. I would like to thank him for his suggestions and encouragements throughout my thesis. His support has been very crucial to its materialization. I would like to extend my gratitude to my supervisor Prof. Dr. Jens Lehmann. His guidance has enlightened me through every step during the completion of my thesis. I would also like to thank Prof. Dr. Sören Auer for his support and help in understanding the concepts related to my thesis. Last but not least, I wish to thank Ms. Hajira Jabeen, Mr. Michael Biegler, Mr. Sascha Weins, and Mr. Stefan Halbfas for helping me with my questions and doubts from time to time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

Directory Service (Directory Service, Webopedia) is a network service that identifies all resources such as e–mail addresses, PCs, printers, etc., on a network and makes them accessible to users and applications. There are a number of directory service implementations, among which Microsoft's Active Directory is largely used by most of the enterprises and organizations all over the world. It was first released with Windows 2000 Server edition, and supported and extended by successive versions of Windows. The LDAP or Lightweight Directory Access Protocol (RFC4510) is an open, vendor–neutral, industry standard application protocol for accessing and maintaining distributed directory information services over an Internet Protocol network.

Active Directoy or Active Directory Domain Services (**AD DS**)[1] has been at the center of IT infrastructure for over a decade, and its features, adoption, and business–value have grown release after release. AD DS (Microsoft, 2014a) enables centralized, secure management of an entire network, which might span a building, a city, or multiple locations throughout the world. In Windows Server 2012 R2 (Microsoft, 2014e), AD DS has been enhanced to allow IT risk management while also enabling IT to empower their users to be productive from a variety of devices. This paper will focus on the release of Windows Server 2012 R2.

Despite the benefits and conveniences brought by Active Directory services, problems arise. One crucial problem is identity life–cycle management in AD DS.

Identity life–cycle management is one component in Identity and Access Management. IAM (Microsoft, 2004) refers to the processes,

[1] AD DS is one deployment of Active Directory. The other one is Active Directory Lightweight Directory Service (**AD LDS**). We use Active Directory and AD DS alternatively in this paper. They both refer to directory services implemented by Microsoft.

technologies, and policies for managing digital identities[2] and controlling how identities can be used to access resources. As shown in Fig.1.1 (Microsoft, 2006b), the main topics in IAM include directory services, identity life–cycle management, access management, and how applications should integrate with the infrastructure.

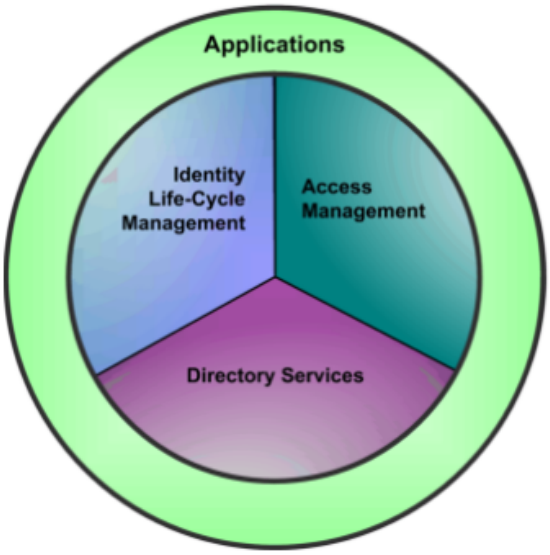


Figure 1.1: Main topics of the IAM framework

Identity life–cycle management (Microsoft, 2006a) consists of the processes and technologies that keep digital identities current and consistent with governing policies. It includes identity synchronization, provisioning, deprovisioning, and the ongoing management of user attributes, credentials, and entitlements. Deprovisioning is the process of deleting or deactivating an identity. Deprovisioning (Microsoft, 2006c) ensures that accounts are systematically disabled or deleted and that entitlements are revoked when employees leave the organization. Deprovisioning should be done quickly to prevent intentional attacks from former employees. A broadly–known incident of this would be the Sony hack, which happened at the end of 2014. A hacker group which identified itself as "Guardian of Peace" (**GOP**) leaked a release of confidential data from the film studio Sony Pictures Entertainment. At first, it was believed that the country of North Korea was involved, due to the subject of the movie, *The Interview*, a comedy about a plot to assassinate North Korean leader Kim Jong–un. But information and events (Biddle, 2014) have led some authorities and cybersecurity experts to believe that a disgruntled former employee, along with some current employees, was responsible for the hack.

[2]The unique identifier and descriptive attributes of a person, group, device or service (Microsoft, 2006a). They refer to user objects stored in Active Directory in this context.

Additionally, and perhaps most importantly, extra storage and maintenance for unnecessary identities can be a huge cost for the organization.

1.2 Problem Specification

SMS group GmbH[3] is a worldwide leading company in the metallurgical industry. It has more than 4,000 employees in Germany and around 14,000[4] employees globally. From January 1, 2015, the two Business Areas, SMS Siemag and SMS Meer, as well as SMS GmbH merged into SMS group GmbH. Figure 1.2 briefly shows the changes in directories during the reorganization.

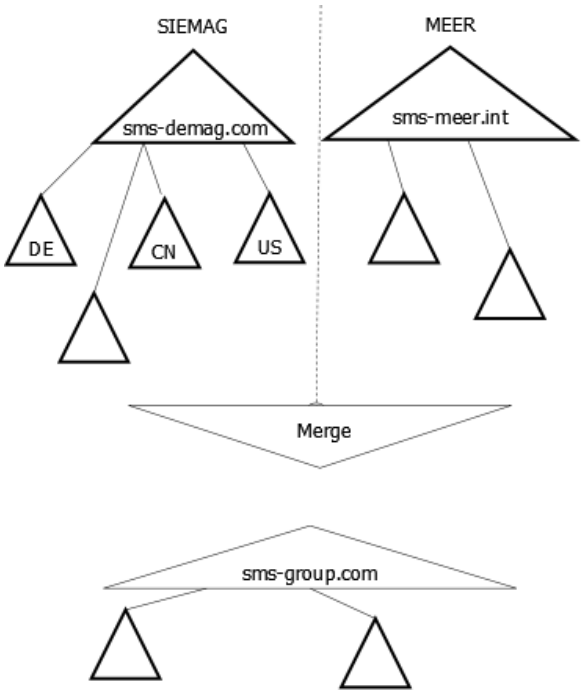


Figure 1.2: Reorganization in SMS group GmbH

There were two directories, `sms-demag.com` for SMS Siemag[5] and `sms-meer.int` for SMS Meer, respectively. In the restructuring, the data under the two directories were migrated and integrated to the common domain `sms-group.com`. During this processing, the merging of some data, such as status and configuration information, could go wrong. For example, a user who is marked as deactivated could be set active again after being integrated into the new directory.

The problem we are tackling here is to attest if the user objects are "healthy". Our task is to mine through the domain `sms-group.com`

[3] `www.sms-group.com`

[4] The average number of employees over the year 2014, including apprentices.

[5] SMS Demag AG was renamed as SMS Siemag AG on March 31, 2009.

and detect identities whose deprovisioning has not been done when it should have been done according to our analysis results.

1.3 Thesis Contribution

We developed an interface to access Active Directory servers and extract identity data from them. The data is stored relationally as ARFF files. We then tested a few machine learning algorithms on a set of training data with different sets of attributes and different parameter settings. Finally, 6 classifiers were decided to complete this classification task. Afterwards, we wrote a program to implement the 6 classifiers, which loads training data from a given file, builds the classifiers and predicts other users' statuses.

Currently, this task is achieved by querying users whose last log on times were 90 days ago, which is ambiguous. We are utilizing machine learning methodology on a set of attributes, which supply information other than only last log on time because other information, e.g. deparment, could also be relevant in this classification task. Furthermore, we analyzed the results obtained from different algorithms and which attributes are closely concerned with this task.

Our interface for extracting data can be utilized in any organization whose directory design follows AD specification[6]. Accordingly, the program to classify users can be utilized to the extracted identity data.

1.4 Thesis Structure

The rest of the thesis is organized as follows: In Chapter 2, we introduce directory and directory services, specially one implementation of it: Microsoft Active Directory. In Chapter 3, we present the approaches of how inactive users are detected currently and the concepts of machine learning which will be adopted in our approach. Afterwards, we present our approach in Chapter 4. In Chapter 5, we explain our analysis of the attributes present for a user object in AD, along with our implementation to extract these data. In Chapter 6, we elaborate on how we conducted the experiments of fusing the extracted identity data from the last step, as well as our analysis of the results. In addition, we describe how to use some good results to fulfil our task of classifying users. In Chapter 7, we summarize and conclude our work, and propose some future work for further improvements. Additionally, at the end of the paper, some of the data appearing in our work is supplied in Appendix A.

[6]It can be used to extract data from any LDAP directories. But not every LDAP directory has the attributes that are defined in AD, especially the attribute `sAMAccountName`, which is the identifier and is essential for our identity data.

Chapter 2

Fundamentals

This chapter provides an overview of directory and directory services, as well as some details of Active Directory. We then introduce the protocol LDAP. The details of search operations via LDAP are also given. Afterwards, we describe how to filter user objects in AD via LDAP.

2.1 Directory and Directory Services

A directory is a collection of data, e.g. a telephone book. In information technology, a directory (Directories, ApacheDS) is used for a special kind of data storage. It allows the structured storage and efficient retrieval of objects, which are often derived from the real world, such as persons or IT equipment. It is similar to a database (Directory Service, Microsoft), but typically contains more descriptive, attribute–based data. And the data in a directory is read more often than it is written. Hence, directories are tuned to respond quickly to high–volume search operations. Directories (Directory Service, Microsoft) may replicate data widely to increase availability and reliability, and thus reduce response time. As a consequence, temporary inconsistencies between replicas exist, which is acceptable as long as all the replicas are updated eventually.

A directory service is a solution, which offers users' processes on a directory, such as query, update, delete, and so on. The web directory provided by the Open Directory Project[1] is a good example of a directory service. It catalogs web pages and is specifically designed to support browsing and searching. More sophisticated directories are designed to store information on many kinds of real–world and conceptual objects. X.500 (ITU-T Rec. X.500) specifies a set of standards for directory services. Directory Access Protocol (**DAP**) (ITU-T Rec. X.511) is the standard for accessing an X.500 directory service. However, DAP is considered too complex, so LDAP was created to simplify the protocol. In the suite of LDAP specifications, (RFC4512) defines the directory information models that are to be accessed via LDAP.

[1] `http://dmoz.org`

2.1.1 Directory Information Models

The information held in a directory is collectively known as the Directory Information Base (**DIB**). The DIB contains two classes of information:

1. User information, e.g. information provided and administrated by users.
2. Administrative and operational information, e.g. information used to administer and/or operate the directory.

2.1.2 Directory Information Tree

An LDAP directory is a collection of entries, which consists of a set of attributes. Each attribute has a type and contains one or more values. The attribute type governs whether the attribute can have multiple values. The set of entries are organized hierarchically in a tree structure known as the Directory Information Tree (**DIT**), which traditionally reflects geographic and/or organizational boundaries. Figure 2.1 (OpenLDAP, 2016) shows an example of LDAP directory tree using traditional naming.

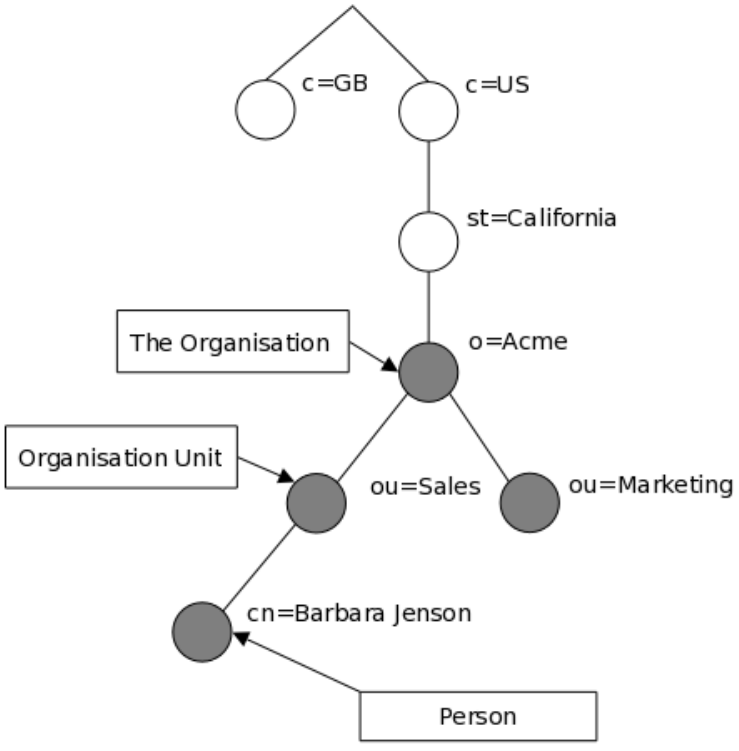


Figure 2.1: An example of LDAP directory tree

An entry is referenced by its distinguished name (**DN**), which is the concatenation of its relative distinguished name (**RDN**) and its

ancestor's DN. For example, the entry for Barbara Jenson in Fig. 2.1 has an RDN of `cn=Barbara Jenson` and a DN of `cn=Barbara Jenson,ou=Sales,o=Acme,st=California,c=US`. An entry's RDN must be unique among all the siblings in its subtree.

2.1.3 Directory Schema

Directory Schema, i.e. the administrative and operational information in a DIB, is the set of definitions and constraints concerning the structure of the DIT. Attribute Syntaxes define the possible ways entries are named and the information that can be held in an entry; Matching Rules define the ways in which attributes' values are matched; Attribute Types define an object identifier (**OID**) and a set of names that may be used to refer to a given attribute, and associate that attribute with a syntax and set of matching rules; Object Classes define named collections of attributes and classify them into sets of required and optional attributes; and other rules.

2.2 Active Directory

AD DS (Microsoft, 2014a) is the distributed directory service that is included with Microsoft Windows Server operating systems. AD DS is the information hub of the operating system. It stores and organizes objects, such as users, computers, devices, etc., on a network into a hierarchical container structure that is known as the logical structure.

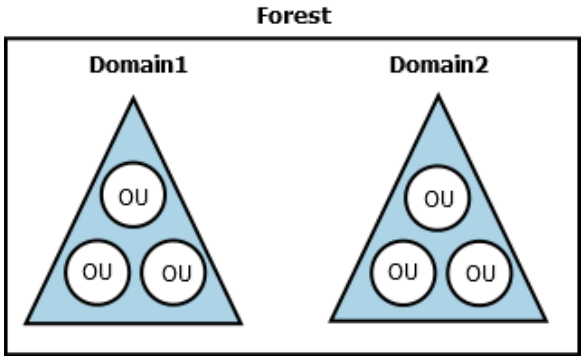


Figure 2.2: AD DS logical structure architecture

2.2.1 AD DS Logical Structure

The AD DS logical structure includes the Active Directory forest, domains in the forest, and organizational units (**OUs**) in each domain. Figure 2.2 (Microsoft, 2014b) illustrates the relationships of OUs, domains, and forests in the logical structure architecture. A forest is the

security boundary for an organization and defines the scope of authority for administrators. Domains in the forest provide partitioning of data, which enables organizations to replicate data only where it is needed. An Active Directory domain also supports a number of other core functions that are related to administration, including network–wide user identity, authentication, and trust relationships. OUs simplify the delegation of authority to facilitate the management of large numbers of objects.

Domain Controller.

A domain controller (Microsoft, 2014c) is a server that has the AD DS server role installed. When you want to create a new forest, a new domain, or an additional domain controller in an existing domain, you configure the server as a domain controller by installing AD DS.

By default, a domain controller stores information about the domain where it is located, as well as the schema and configuration information in the entire forest. Every domain controller stores all the objects contained in this domain. A domain controller which is designated as a Global Catalog (**GC**) server stores the objects from all the domains in the forest. However, for the objects which are not in the domain where the GC server is, only a limited set of attributes are stored in a partial replica. The attributes that are replicated to a GC server are those which are most likely to be used for search, e.g. `cn` (common name). Whether this attribute is replicated to GC is defined by the attribute `isMemberOfPartialAttributeSet` in the **attributeSchema** (see Sec. 2.2.2) object of this attribute. Global Catalog makes it possible for clients to search AD DS without having to be referred from server to server until a domain controller which is in the domain that contains the queried object is found.

2.2.2 Active Directory Schema

As defined in (Active Directory Schema, Microsoft), the Microsoft Active Directory schema contains formal definitions of every object class that can be created in an Active Directory forest, the schema also contains formal definitions of every attribute that can exist in an Active Directory object.

Similarly to how objects are defined in an LDAP directory, each object in Active Directory represents a single entity, which contains a set of attributes. An object is an instance of an object class which is defined by a **classSchema** (Characteristics of Object Classes, Microsoft) object. The attributes of a **classSchema** object specify the characteristics of the class, such as class identifiers, possible attributes, possible parents, etc. An attribute contained in an object entity is an object defined by an **attributeSchema** (Characteristics of Attributes,

Microsoft) object. The properties of an **attributeSchema** object specify the characteristics of the attribute, such as attribute identifiers, the type of data contained by instances of the attribute, if it has only one value or multiple values, etc.

The schema container contains all of the **classSchema** and **attributeSchema** objects that define the classes and attributes that can exist in a directory forest. There is one schema per forest, which is replicated and kept updated in every domain. This way, all objects in one forest are created using the same definition and hence, uniformly.

Object Class.

As introduced above, an object class is defined by a **classSchema** object. There are four categories of classes (Microsoft, 2015n) in Active Directory: structural classes, abstract classes, auxiliary classes, and 88 classes. Abstract classes are templates that are used to derive new classes. They cannot be instantiated in the directory. Structural classes are classes that can have instances. Auxiliary classes contain a set of attributes. Adding an auxiliary class to the definition of a structural or an abstract class adds the auxiliary class's attributes to the definition. 88 classes do not fall into any of the preceding categories. It can be used as any of the different classes above.

The attribute `objectClass` is a multivalued attribute that appears in every object in the directory. It contains a sequence of class names. For example, the `objectClass` attribute in a user object from SMS group looks like in Fig. 2.3. The class `top` is the top level class from which all classes are derived. The class `user` is the class that a user object is initiated from. The classes in the list form a superclass chain, i.e., class `person` is the superclass of class `organizationalPerson`, which is the superclass of class `user`.

```
attribute: objectClass
value: top
value: person
value: organizationalPerson
value: user
```

Figure 2.3: The objectClass attribute in a user object

Object Category.

Each instance of an object class has the attribute `objectCategory` (Microsoft, 2015k). It is a single–valued attribute which specifies an object class name and is used to group objects of this class or other derived classes. When an object is created, the system sets its

`objectCategory` to the value specified by the `defaultObjectCategory` attribute of its object class. For most classes (Object Class & Object Category, Microsoft), the `defaultObjectCategory` is the distinguished name of the class's **classSchema** object. However, some classes refer to another class as their `defaultObjectCategory`. For example, the `user, person, organizationalPerson`, and `contact` classes all identify the `person` class in their `defaultObjectCategory` attribute. Figure 2.4 shows the value of attribute `objectCategory` in a user object from SMS group.

```
attribute: objectCategory
value: CN=Person,CN=Schema,CN=Configuration,DC=sms-group,DC=com
```

Figure 2.4: The objectCategory attribute in a user object

2.3 LDAP

LDAP provides access to distributed directory services that act in accordance with X.500 data and service models. The latest version is LDAPv3 (RFC4511).

LDAP utilizes a client–server model. One or more LDAP servers contain the data which makes up the directory. A client starts an LDAP session by connecting to an LDAP server. Table 2.1 lists the default port numbers that LDAP servers listen to. It then sends an operation request to the server. The server performs the necessary operation(s) in the directory and sends responses in return. The server responds with a direct answer and/or with a reference which points to another LDAP server where the client can get additional information. The operations are generally independent of one another. Each operation is processed as an atomic action, leaving the directory in a consistent state. There is no requirement for synchronous behavior on either part of clients or servers. The client does not need to wait for a response before sending the next request, and the server may send the responses in any order. No matter which LDAP server a client connects to, it sees the same view of the directory.

Table 2.1: Default port numbers used by LDAP

Port	TCP	UDP	Description
389	TCP	UDP	LDAP
636	TCP	UDP	LDAP over TLS/SSL (LDAPS)
3268	TCP	UDP	msft-gc, Microsoft Global Catalog (LDAP GC)
3269	TCP	UDP	msft-gc-ssl, Microsoft Global Catalog over SSL (LDAP GC SSL)

2.3.1 Search Operations

All protocol operations are encapsulated in a common envelope, the LDAPMessage. Operations are provided for: binding to the server; adding and deleting an entry from the directory; changing an existing entry; and most frequently, searching for information in the directory. We mainly consider the search operations in our implementation. The search operation is used to request a server to return a set of entries matching a complex search criterion subject to access controls and other restrictions.

Search Filters.

(RFC4515) defines a human–readable string representation of LDAP search filters. The basic syntax of search filter is:

attribute operator value

The attribute can be any attributes defined in an LDAP directory. The operator can be "=" (equal), "$\sim$=" (approx), ">=" (greaterorequal), "<=" (lessorequal), and other extensible types. The boolean operator "&" (and), "|" (or), and "!" (not) can be used to form combinations of filters. The value can be a concrete value of the attribute, e.g., the search filter `(sn=diao)` will return all objects which have the attribute `sn` (surname) with a value `diao`. It may also contain the wildcard character, e.g., the search filter `(&(sn=diao)(mail=*uni-bonn.de))` will return all objects that have both the last name `diao` and at least one email address which ends with `uni-bonn.de`. The matching rules for the different operations are defined in the directory schema, e.g. case insensitive when matching `cn`.

Search Scope.

Another aspect we need to consider is the search scope. The scope can be constrained to the entry, which is named in search base: itself (base), the immediate subordinates of the entry (one level), to all its subordinates including the entry itself (subtree level).

Search Size Limit.

Size limit restricts the maximum number of entries to be returned as a search result. A value of zero indicates that no size limit restriction from the client is given. However, the returned number of results cannot be larger than the default size limit that is set server–side, e.g., the server administrator specifies that a normal client cannot retrieve more than one thousand entries in one search.

Search Result with Referrals.

If the queried object is not located on the server that the session is connected to, the server may return one or more referrals which direct the search to another set of servers for continuing the operation. The client could choose to follow or not to follow the returned referral(s) in the search controls. Following a referral would slow down the search operation.

2.3.2 Query Active Directory

LDAP syntax filters can be used in many situations to query Active Directory, e.g. using PowerShell scripts (Searching Active Directory with Windows PowerShell, Microsoft). As described in Sec. 2.3.1, there are three elements in an LDAP syntax filter clause. The attribute must be the `lDAPDisplayName`[2] of an Active Directory attribute. (LDAP Syntax Filters, Microsoft) describes search filter syntax that is used for Active Directory in detail.

Filters for User Objects.

We want to search Active Directory for a particular type. More specifically, we want to search for Active Directory objects that represent users. This can be done by searching on the `objectClass` attribute for `(objectClass=user)`. Another method is to search on a particular category, i.e. using the `objectCategory` attribute. As explained in Sec. 2.2.2, every object is a member of more than one class because of inheritance. However, it only belongs to one object category. Therefore, searches that use `objectClass` can be less accurate than searches that use `objectCategory`. For example, the search filter `(objectClass=user)` returns both user and computer objects, while the search filter `(objectCategory=user)` returns only user objects.

The attribute `sAMAccountType` (Microsoft, 2015c) specifies the account type of the security principal objects[3] in Active Directory. The value `0x30000000` stands for user object. Therefore, the search filter `(sAMAcounntType=805306368)` also returns user objects and it is said to be more efficient (LDAP Syntax Filters, Microsoft).

Paged Search.

Searches of Active Directory performed without paging return a maximum of the first 1000 records (Paged Search, Microsoft). With a

[2]This attribute (Microsoft, 2015f) specifies the name used by LDAP clients to read and write the attribute by using the LDAP protocol.

[3]In Active Directory, the user, computer, and group object classes are examples of security principal object classes (though not every group object is a security principal object).

paged search, the results are returned as individual pages, each of which contains a predetermined number of records and new pages are returned until the end of the result set is reached. Paged search offers benefits to both sides. For example, the client can be more responsive in presenting the results to end users as the results are returned page by page; the server only needs to give a response with a specified size to each client, which makes the operation scalable.

Chapter 3

Related Work

In Active Directory, there are several date and timestamp attributes associated with a user object. The most relevant attributes in our task are considered to be `lastLogon` and `lastLogonTimestamp`. We will elaborate on these two attributes and explain how they differ from each other in the next section, followed by an introduction of machine learning methods.

3.1 lastLogon vs lastLogonTimestamp

According to (Microsoft, 2015e), the attribute `lastLogon` specifies the last time the user logged on, and the attribute `lastLogonTimestamp` is the time that the user last logged on to the domain. The value of `lastLogon` is not replicated, i.e., it is updated only on the domain controller that validates the logon request. The attribute `lastLogonTimestamp` is introduced in Windows Server 2003 and higher. It is replicated so all domains have the same value for the attribute. However, it does not provide real–time logon information. With default settings, the value of `lastLogonTimestamp` will be 9–14 days behind the current date (Pyle, 2009).

3.2 Identify Inactive Users in AD

One big problem in Active Directory is the existence of stale accounts. It is highly possible that a large number of user accounts who have not logged into the domain for years, or at all, exist in a company's AD. These accounts need to be identified and deleted (or disabled).

3.2.1 PowerShell Scripts

(McFerron, 2011) suggested to accomplish this task with Windows PowerShell and the AD module. As explained in the previous section, the value of `lastLogonTimestamp` is replicated among domain controllers every 9 to 14 days. The AD module also displays it in an easy–to–read format called `lastLogonDate`. Because there are some instances when this value is not updated, he also suggested to look at `pwdLastSet`.

In order to identify inactive users in Active Directory, one way is to use the **Get–ADUser** cmdlet and filter the accounts that have `lastLogonTimestamp` and `pwdLastSet` values that are over 90[1] days old, as shown in the following script (McFerron, 2011). In addition, you can output the list of inactive accounts to a `csv` file by using the parameter `export-csv <FileName>` in the command.

```
$90Days = (get-date).adddays(-90)

Get-ADUser -SearchBase "OU=OUName,DC=DomainName" -filter {(lastlogondata
-notlike "*" -OR lastlogondate -le $90days) -AND (passwordlastset -le
$90days) -AND (enabled -eq $True)} -Properties lastlogondate, passwordlast-
set | Select-Object name, lastlogondate, passwordlastset
```

In Windows Server 2003 Domain Functional Level[2], one can also use the **Search–ADAccount** (Search-ADAccount, Microsoft) command to search for accounts that have not logged in within a given time period or since a specified time. The `TimeSpan` parameter is used to specify a time period, and the `DateTime` parameter is used to specify a specific time.

3.2.2 Dsquery Command

Another way to find inactive users is to use **Dsquery**. **Dsquery** (Microsoft, 2012) is a command–line tool that is built into Windows Server 2008. It is available if you have the AD DS server tool installed. The command `Dsquery user -inactive <NumberOfWeeks> -limit`[3] `0` searches for users who have been inactive for at least the number of weeks that you specify. Furthermore, the parameter `-stalepwd <NumberOfDays>` searches for objects who have not changed their passwords for at least the number of days that you specify. Furthermore, you can export the search result by adding `> <FileName>`.

However, one can not be certain if this account is needed or not in the future even it has not been logged on in the last three months. More factors need to be considered in order to make the decision. That is what we are exploring, retrieving more attributes of accounts and applying machine learning algorithms to predict the probability of a user being inactive.

[1] Replace this with the number of days back you want to look.

[2] Functional levels (Microsoft, 2014d) determine the available AD DS domain or forest capabilities. When you deploy AD DS, set the domain and forest functional levels to the highest value that your environment can support.

[3] Specify the number of objects to be returned that matches the search criteria. With a value of 0, it returns all matching objects. If you do not specify this parameter, it displays the first 100 results by default.

3.3 Machine Learning

Following Alan Turing's proposal (Turing, 1950) in his paper "Computational Machinery and Intelligence" that "Can machine think?", (Harnad, 2008) replaced the question by "Can machines do what we (as thinking entities) can do?". (Mitchell, 1997a) provided a formal definition of machine learning:

Definition *A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*

Machine learning appears in many fields, including statistics, artificial intelligence, computational complexity, etc. In the field of data analysis, machine learning is a method that builds a model from a training data set and makes predictions or decisions on another data set with the learned model.

3.3.1 Machine Learning Styles

There are typically three broad categories of machine learning styles based on the nature of the learning "signal" or "feedback" available to a learning system. They are (Russell and Norvig, 1995):

- **Supervised learning** Input data has a known label or result. A model is prepared through a training process, which continues until a desired level of accuracy on the training data is achieved.
- **Unsupervised Learning** Input data is not labeled and does not have a known result. A model is prepared by deducing structures present in the input data.
- **Reinforcement Learning** A computer program interacts with a dynamic environment in which it receives some evaluation of its action but is not told the correct action.

Between supervised and unsupervised learning is semi–supervised learning, where the input data is a mixture of labeled and unlabeled examples.

3.3.2 Machine Learning Tasks

Machine learning can be used to solve problems like classification, regression, clustering, and so on. In classification, data is divided into two or more classes. The learner must produce a model that assigns an unseen input entry to one or more of these classes. Classification is typically tackled in a supervised way. Regression is also a supervised problem. The output variable takes continuous values

rather than class labels as in classification. Clustering classifies a set of inputs into groups. Unlike classification, the groups are unknown beforehand, which makes it typically an unsupervised task.

3.3.3 Machine Learning Algorithms

There are a number of machine learning algorithms in each learning style and for every learning task. The most popular ones, and those we are going to consider in our work, are the decision tree algorithm C4.5, k–Nearest Neighbor (k–**NN**), Naive Bayes, and the Support Vector Machine (**SVM**) algorithm.

As our goal is to classify users into active or passive categories, it is a classification problem, which should be solved in a supervised way. We have a set of attributes for each user entry, which can serve as the foundations for decision tree algorithms. On the other hand, the values of the attributes can be measured by similarity in order to assign a new user with the label of the class which contains users with most similar values to this user. Bayes' Theorem can also be used for this classification task. It will give an accurately computed probability for each prediction. The SVM algorithm is designed to find a hyperplane which separates the data with a maximum margin, which is also suitable for our classification task.

Chapter 4

Approach

As shown in Fig. 4.1, we first extract identity data from AD domain controller(s), store it in ARFF files. In this stage, we need to analyze the attributes that a user object contains and decide which attributes will be possibly useful for our task. Parallel to this task is to train some learning models and evaluate their performances. Finally, we decide on a few trained models and use them to predict other users' statuses.

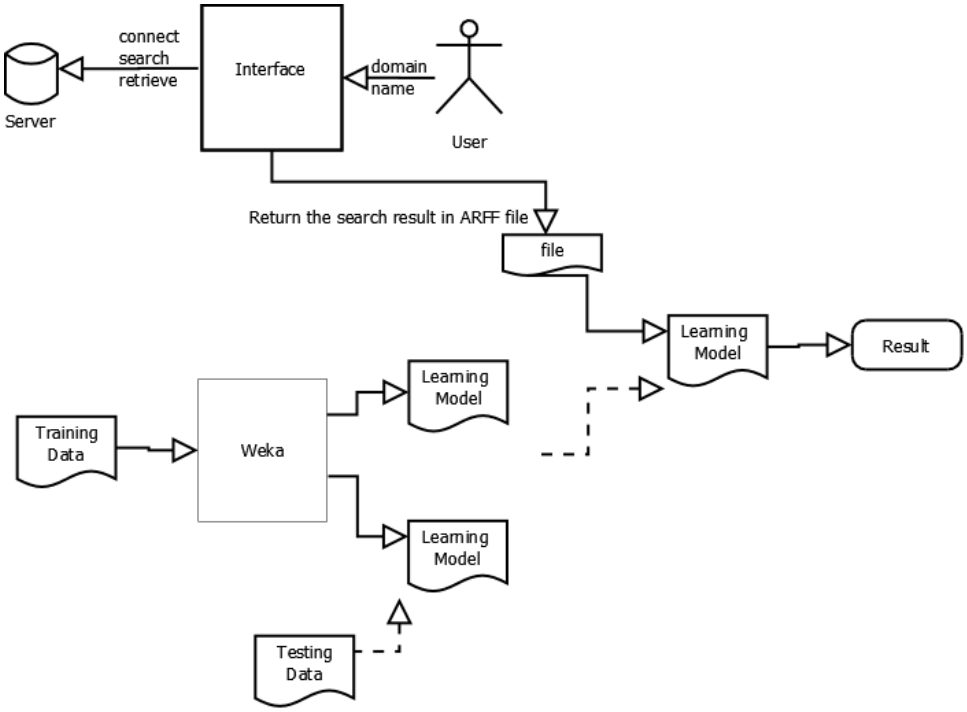


Figure 4.1: Illustration of our approach

The training data and testing data comes from the Human Resource (**HR**) system of the company. It contains a different set of attributes than the attributes for a user object in AD. The entries in HR systems are synchronized to AD via a Meta Directory, which is Forefront Identity Manager (**FIM**) in our scenario. It maps an object in the HR system to an object in AD and at the same time, adds some control information (e.g., which groups this object belongs to). For more information, see (Syschronization Service in FIM, Microsoft).

We retrieve a set of training data from the HR system, for which the class labels (supplied in attribute `status`) are clear. Accordingly, we retrieve the records of these training data from AD and store them into ARFF files, one set as the training data and another set as the testing data. In the training phase, we need to analyze the attributes further and decide which attribute sets to use for different algorithms. We also consider the parameter settings of each algorithm in order to obtain the optimal learning models.

Chapter 5

Extraction of the Identity Data

In this chapter, we elaborate on our analysis of the attributes in AD and our implementation details of retrieving data from AD. We further describe two more functions in the implementation which are used to extract training data and testing data.

5.1 Analysis of Attributes in AD

As we understand from Sec. 2.2.1, a DC contains all the information of the objects which reside in the same domain while a GC additionally contains a subset of information of the objects which reside in other domains. We connect to a DC server via port 386 and to a GC server via port 3286 (Tab. 2.1). We retrieve all the attributes and their values of one user via ports 386 and 3268[1], respectively. We observe that some attributes which exist in the record that we retrieved via port 389 are not present in that which we retrieved via port 3268. Furthermore, the record that we retrieved via port 3268 contains more attributes than those which are specified to be replicated globally. For example, for the attribute `pwdLastSet`, `isMemberOfPartialAttributeSet` is not present in the definition of the **attributeSchema** object (Microsoft, 2015b). However, it is present in the record retrieved from a GC and has a value.

A detailed record of user `DIAO` is supplied in Appendix A.1 as an example. After a thorough analysis and consideration, we decided to retrieve the following attributes for a user object.

5.1.1 sAMAccountName

The attribute `sAMAccountName` (Microsoft, 2015c) specifies the logon name used to support clients and servers running LAN manager and older versions of the operating system, such as Windows NT 4.0 operating system, Windows 95 operating system, and Microsoft Windows 98 operating system. We retrieve this attribute to identify the objects. As shown in Fig. 5.1, its values are unique in the whole

[1]Note that the DCs we have access to are also GCs. Thus, we can access them as normal DC servers via port 389 or as GC servers via port 3268.

domain[2]. A user's DN (Fig. 2.1) is absolutely distinct in the entire domain. The attribute `userPrincipalName` (Microsoft, 2015d) is also distinct as it specifies the user principal name (**UPN**) in an Internet–style logon name. By convention, the UPN should map to the user email name. Although UPN is shorter and easier to remember than DN, it is still a long string with redundant information (e.g. the suffix of the email address). Therefore, we chose `sAMAccountName` as the identity of a user. Another advantage is that it maps to the attribute `User-ID` which is one of the identifiers in the HR system.

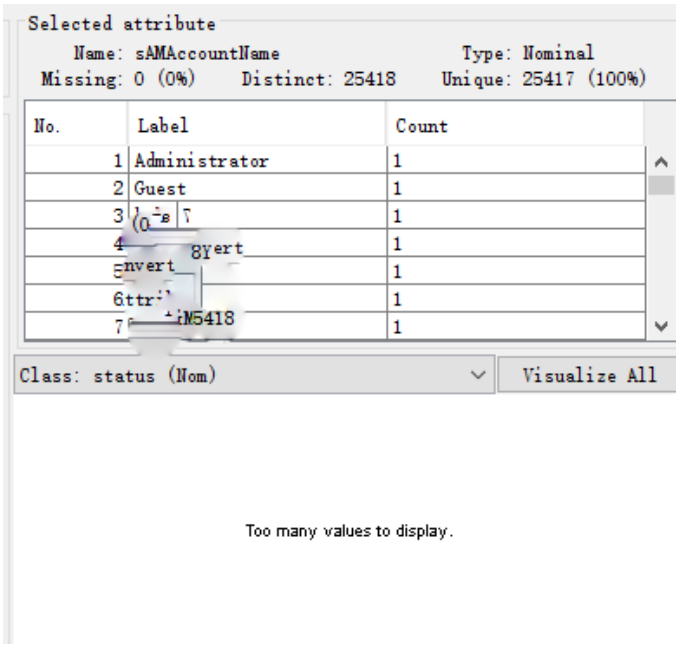


Figure 5.1: The attribute `sAMAccountName`

5.1.2 whenCreated & whenChanged

According to (Microsoft, 2015h), the attribute `whenCreated` specifies the date and time when this object was created. This value is replicated and is in GCs. The attribute `whenChanged` specifies the date when this object was last changed. This value is not replicated but exists in GCs. They use Generalized–Time syntax (Generalized-Time syntax, Microsoft) to store time values. These dates are strings in the format "YYYYMMDDHHMMSS.0Z". The "Z" indicates no time differential. Active Directory stores date/time as Greenwich Mean Time (**GMT**). For example, "20160201101940.0Z" corresponds to "Mon, 01 Feb 2016 10:19:40 GMT". These two dates attributes, especially the

[2]These 25419 records were retrieved from the domain `dc=sms-group,dc=com` on June 16, 2016. There was one duplicate value for this attribute. All the statistics shown below in this section are based on this data except for special statement. All the statistics are analyzed by Weka's Explorer.

attribute `whenChanged`, are considered relevant in our classification task. If the user has not changed any information or it has not been changed by administrators in a long time, it might indicate that this account is stale.

5.1.3 logonCount & badPwdCount

The attribute `logonCount` (Microsoft, 2015j) specifies the number of times that the account has successfully logged on, and the attribute `badPwdCount` (Microsoft, 2015m) specifies the number of times the user tried to log on to the account by using an incorrect password. Neither of them exists in GCs. Both of them are recorded in a numeric format. A value of 0 indicates that the value is unknown. These values reflects the account's logon statistics in a way, such as how often it is logged on and how frequently the user inputs the password incorrectly.

5.1.4 lockoutTime & badPasswordTime

The attribute `lockoutTime` (Microsoft, 2015i) specifies the date and time (UTC) that this account was locked out, and the attribute `basPasswordTime` (Microsoft, 2015l) specifies the last time and date that an attempt to log on to this account was made with an invalid password. These timestamps are stored as large integers which are called file times. A file time (File Times, Microsoft) is a 64–bit value that represents the number of 100–nanosecond intervals that have elapsed since 12:00 A.M. January 1, 1601 Coordinated Universal Time (**UTC**). A value of 0 in these attributes means that the account is not currently locked out and that the last invalid password time is unknown, respectively. They reflect the state of the account.

5.1.5 pwdLastSet

The attribute `pwdLastSet` (Microsoft, 2015b) specifies the date and time that the password for this account was last changed. It is stored in a file time format as `lockoutTime` and `badPasswordTime` are. This attribute plays a role in computing the expiration time for the user's current password, see (Microsoft, 2015o). Figure 5.2 shows the statistics of this attribute's values in the domain, from which we can see that more than half of the values of this attribute are 0s[3]. As it is essential to change passwords within a period of time in the system, `pwdLastSet` reflects the user's activity to some extent.

[3]That is 12:00 A.M. January 1, 1601 UTC. Since Weka processes date format to milliseconds after Unix Epoch (00:00:00 UTC on 1 January 1970), the values shown in Weka are negative.

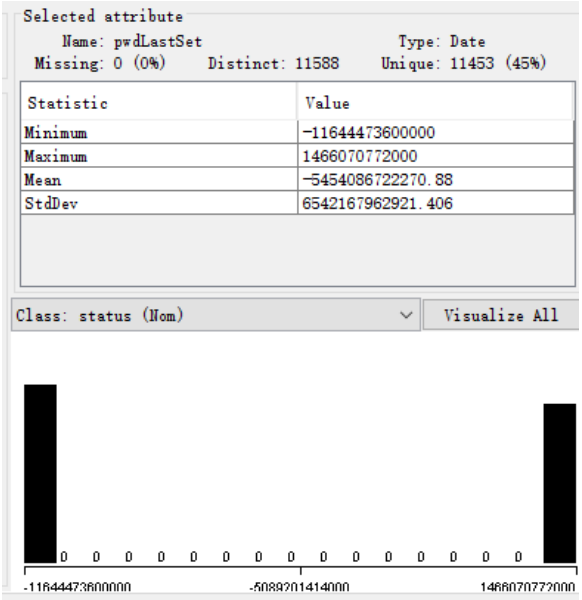


Figure 5.2: The attribute `pwdLastSet`

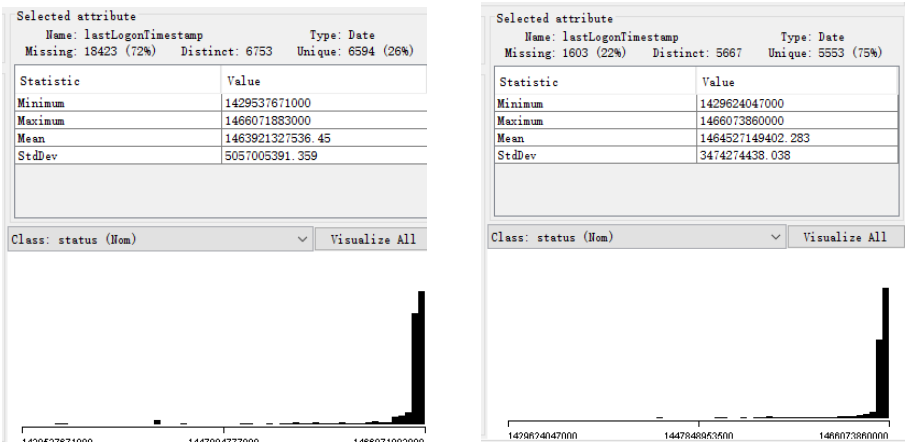
5.1.6 lastLogon & lastLogonTimestamp

As introduced in Sec. 3.1, `lastLogon` records users' accurate last logon time but is not replicated. In order to obtain a user's real last logon time by reading this attribute, one has to retrieve the values of this attribute from all the existing domain controllers and take the most recent one. The attribute `lastLogonTimestamp` has been introduced since Windows Server 2003. It is defined as a member of the partial attribute set, which is replicated in all GCs. One only needs to retrieve the value of this attribute from one of the GCs to obtain a user's last logon time. Both attributes are stored in file times.

The attribute `msDS-LogonTimeSyncInterval` (Microsoft, 2015g) specifies the frequency (in days) with which the last logon time for a user/computer is updated, and is recorded in the `lastLogonTimestamp` attribute. The default value is 14. The following procedure (Pyle, 2009) shows how the value of `lastLogonTimestamp` is updated:

1. Value of the `msDS-LogonTimeSyncInterval` $= I$
2. User logs on to the domain
3. The `lastLogonTimestamp` attribute value of the user is retrieved as T
4. $I -$ (Random percentage of 5) $= X$
5. Current data $- T = Y$
6. $X \leq Y \rightarrow$ update `lastLogonTimestamp`

7. $X > Y \rightarrow$ do not update `lastLogonTimestamp`



(a) `lastLogonTimestamp` in the domain **(b)** `lastLogonTimestamp` in an OU

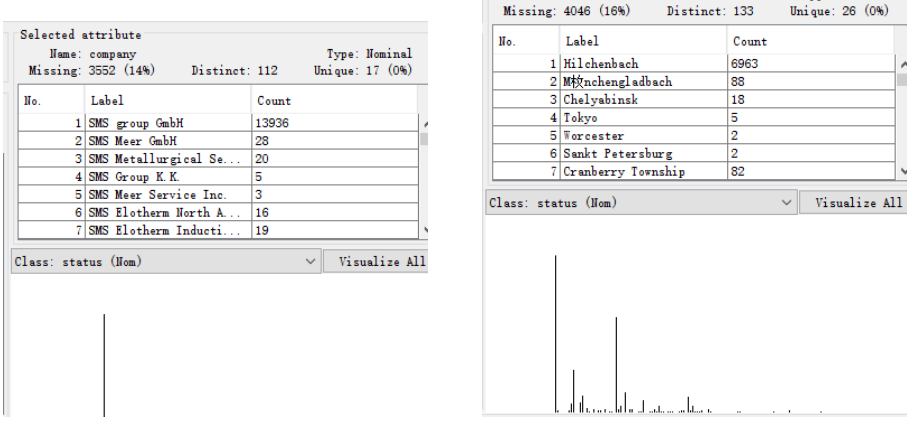
Figure 5.3: The attribute `lastLogonTimestamp`

Figure 5.3a shows the statistics of `lastLogonTimestamp` in the domain `dc=sms-group,dc=com`. Unfortunately, 72% of this value is missing. Figure 5.3b shows the statistics of that in the OU `ou=de, dc=sms-group,dc=com`, in which 22% records do not have values for this attribute.

5.1.7 company, department & l

The attributes `company`, `department` and `l` are documented in (Microsoft, 2015a). Employees working for different companies or departments in different places might have different logon behaviors, e.g., the holiday regulations are different in different countries, meaning that users are allowed to be absent for different durations of time. As a matter of fact, only the attribute `l`, which represents the name of a locality such as a town or city, is a member of the partial attribute set. Neither `company` nor `department` is defined as a member of the partial attribute set. However, we can see from Fig. 5.4 that the proportion (14% in Fig. 5.4a) of missing values for `company` is not so large as an attribute which is not replicated is expected to be[4]. On the contrary, it is smaller than that (16% in Fig. 5.4b) of `l`. The same appears for attribute `department`, whose missing value proportion is 15% in the entire domain, which is not shown here. Hence, we also take these two attributes into consideration.

[4]As the data was retrieved from a DC located in Mönchengladbach.



(a) The attribute `company` **(b)** The attribute `l`

Figure 5.4: The attributes `company` & `l`

5.2 Extract Identity Data from AD

Having decided which attributes to retrieve, we can connect to an LDAP server and read data from it.

5.2.1 Connect to LDAP Servers

There are numerous ways to connect to an LDAP server. Java LDAP[5] (**JLDAP**), developed by Novell, enables you to write applications to access, manage, update, and search for information stored in directories accessible using LDAPv3. Mozilla implemented a Software Development Kit (**SDK**) following The C LDAP Application Program Interface (The C LDAP API), Mozilla LDAP C SDK[6]. Java Naming and Directory Interface (**JNDI**)[7] is an API included in Java SE Platform to access the directory and naming service. Microsoft also provides Active Directory Service Interfaces[8] (**ADSI**) for accessing the features of directory services. We use JNDI to connect to and operate on Active Directory servers as it is designed especially for the Java platform using Java's object model.

As shown in Fig. 5.5, our interface requires users to input the hostname of the domain controller which is to be connected to, the search base for this execution, the user's credential in the domain, as well as the file path for the output. We connect to the DC via port 389 and follow the returned referral(s) from the server. This set–up

[5] `https://www.novell.com/developer/ndk/ldap_classes_for_java.html`

[6] `https://wiki.mozilla.org/LDAP_C_SDK`

[7] `http:/www.oracle.com/technetwork/java/overview-142035.html`

[8] `https://msdn.microsoft.com/en-us/library/windows/desktop/aa772170(v=vs.85).aspx`

of operation is slower than connecting to the server via port 3268 and not following referral(s), but it is necessary because we are retrieving some attributes which are not present in the partial attribute set of GCs and the difference of consumed time is not significant. The search base is given as `dc=sms-group,dc=com` as we require global–level data. Paged search and subtree search scope are utilized so that all records which match the search criteria are returned. We can see from Fig. 5.6 that the page size was set to 100.

Figure 5.5: The interface of retrieving identity data

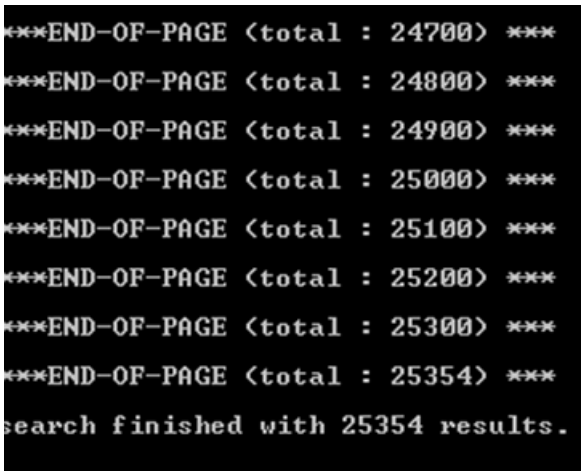


Figure 5.6: Search result

5.2.2 Filter User Objects

As introduced in Sec. 2.3.2, there are at least two ways to filter user objects in AD. In Fig. 5.7, we show how we use both methods to retrieve all the user objects. The two methods did not differ much in terms of time consumption but gave the same number of search results.

```
String searchFilter =  //"(&(objectCategory=person)(objectClass=user))";
                       "(sAMAccountType=805306368)";
```

Figure 5.7: Search filter

5.2.3 Parse the Attributes

As discussed in Sec. 5.1, there are 4 formats of attributes which we want to retrieve, i.e. numeric, string, Generalized–Time syntax, and File Times. All the attributes are retrieved first as an **Attribute** object via the `getAttribute()` method of the **SearchResult** class. The **Attribute** object, which contains the name of the attribute and the value(s) of it, is then processed into strings. The attributes we are extracting are all single–valued attributes. If the returned **Attribute** object is `null`, we fill the value string with `?` as we will store the values into ARFF files. `?` stands for missing value in ARFF syntax. For numeric attributes, we simply write the returned string values to the ARFF file. For string attributes, we wrap the non–null values with double quotes and write them to the ARFF file. It is necessary to wrap string values with single or double quotes in ARFF files because the parser in Weka does not read the value correctly if there are spaces or other special characters in the string.

Parse Generalized–Time Attributes.

The attributes `whenCreated` and `whenChanged` are stored in Generalized–Time syntax in AD. After extracting the value as a string, we parse it to the local time and in the date format as defined in the ARFF header, shown in Fig. 5.8. We also add one more attribute, the Unix Epoch, in the output ARFF file to display the time as a numeric value.

```
private static String parseADDate(String ldapDate) {
    String[] parts = ldapDate.split("[.]");
    String dateTimePart = parts[0];  //take the date string before .0Z
    SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMddhhmmss");
    sdf.setTimeZone(TimeZone.getTimeZone("GMT"));  //Z means UTC time, to the local timezone
    SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

    try {
        Date tempDate = sdf.parse(dateTimePart); //parse the string to a date
        return formatter.format(tempDate);  //format it as what we want
    } catch (ParseException ex) {
        System.out.println("Parsing LDAP Date exception \n");
        ex.printStackTrace();
    }
    return null;
}
```

Figure 5.8: Parse Generalized–Time attributes

Parse Attributes Stored in File Times.

The attributes `lockoutTime`, `badPasswordTime`, `pwdLastSet`, `lastLogon`, and `lastLogonTimestamp` are stored in file times. We use the following function (Fig. 5.9[9]) to convert it to Unix Epoch and format it. Similarly, we add one more attribute in the output ARFF file for each attribute to display the time in Unix Epoch.

[9]The computation of time is from `http://goliferay.blogspot.de/2015/11/convert-18-digit-ldap-timestamps-to.html`

```
private static String parseADTimestamp(String ldapTimestamp) {
    long nanoseconds = Long.parseLong(ldapTimestamp);  // 100 nanoseconds
    long mills = (nanoseconds/10000000);  // To seconds
    long unix = (((1970-1601)*365)-3+Math.round((1970-1601)/4))*86400L;
    long timeStamp = mills - unix;

    Date date = new Date(timeStamp*1000L); // To milliseconds
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    //sdf.setTimeZone(TimeZone.getTimeZone("GMT"));

    return sdf.format(date);
}
```

Figure 5.9: Parse attributes stored in file times

5.2.4 The Search Result

Figure 5.10 shows the header of the output ARFF file, which represents the relations of the attributes. We can see the 13 attributes which we have analyzed and chosen to retrieve for one user object. For each date/time attribute, we have one more numeric attribute to display the value in Unix Epoch. Finally, the attribute `status` is the class label which is a nominal class with a value of either "P" for "Passive" or "A" for "Active". It is not an attribute in the AD user object, and the value is unknown when retrieving user objects from AD.

```
%1. Title: User Objects Attributes
%
%2. Sources: ldap://svhilads02.sms-group.com:389
%
%3. Access from: cn=diao,ou=fim,ou=user,ou=dehil,ou=de,dc=sms-group,dc=com
%
@RELATION users

@ATTRIBUTE sAMAccountName string
@ATTRIBUTE whenCreated DATE "yyyy-MM-dd HH:mm:ss"
@ATTRIBUTE whenCreatedNum numeric
@ATTRIBUTE whenChanged DATE "yyyy-MM-dd HH:mm:ss"
@ATTRIBUTE whenChangedNum numeric
@ATTRIBUTE logonCount numeric
@ATTRIBUTE lockoutTime DATE "yyyy-MM-dd HH:mm:ss"
@ATTRIBUTE lockoutTimeNum numeric
@ATTRIBUTE badPwdCount numeric
@ATTRIBUTE badPasswordTime DATE "yyyy-MM-dd HH:mm:ss"
@ATTRIBUTE badPasswordTimeNum numeric
@ATTRIBUTE pwdLastSet DATE "yyyy-MM-dd HH:mm:ss"
@ATTRIBUTE pwdLastSetNum numeric
@ATTRIBUTE lastLogon DATE "yyyy-MM-dd HH:mm:ss"
@ATTRIBUTE lastLogonNum numeric
@ATTRIBUTE lastLogonTimestamp DATE "yyyy-MM-dd HH:mm:ss"
@ATTRIBUTE lastLogonTimestampNum numeric
@ATTRIBUTE company string
@ATTRIBUTE department string
@ATTRIBUTE l string
@ATTRIBUTE status {P,A}

@DATA
```

Figure 5.10: The header of the output ARFF file

One record in the data section looks like this: `"DIAO","2016-02-01 11:19:40",1454321980000,"2016-06-16 12:11:07",1466071-867000,61,?,?,0,"2016-06-08 12:18:28",1465381108000,`

"2016-04-25 11:42:36",1461577356000,"2016-06-20 13:14:58",1466421298000,"2016-06-16 12:11:07",1466071867000,"SMS group GmbH","ITCN","Hilchenbach",?.

5.3 Retrieve Training Data

As we introduced in Chapter 4, the data in AD is synchronized from an HR system via Forefront Identity Manager (**FIM**). We obtained a set of data which was extracted from the HR system. One entry in the dataset represents one user. However, it contains different attribute sets. Some attributes are not synchronized to AD, e.g., the attribute `Personal Number` in HR system does not appear in AD. Some attributes from the HR system are mapped to an attribute with a different name in AD, e.g., the attribute `User-ID` in HR system appears as `sAMAccountName` in AD. Most importantly, there is the attribute `status` in the HR system. Therefore, we can accordingly retrieve the user objects from AD via the common identification attribute `User-ID/sAMAccountName`. Furthermore, we add the label value for each record which comes from the `status` value. Thus, we will get the same attribute set for the training data as for the data to be classified.

We do not retrieve the training data by giving `sAMAccountName` values in the search filter in an LDAP query, which is tedious and time consuming. Instead, we write a program to retrieve records from the ARFF file obtained from the last step by giving `sAMAccountName` values. The program searches the user record with a given `sAMAccountName` in the ARFF file which contains all the user objects in the domain `dc=sms-group,dc=com`. It replaces the value of `status` (which is `?`) with the input `status` value and writes the record to another file.

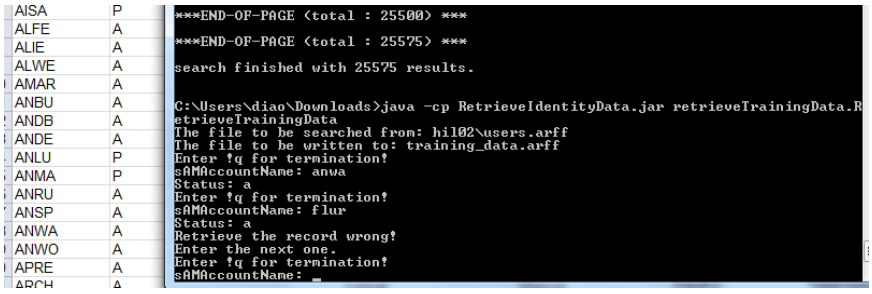
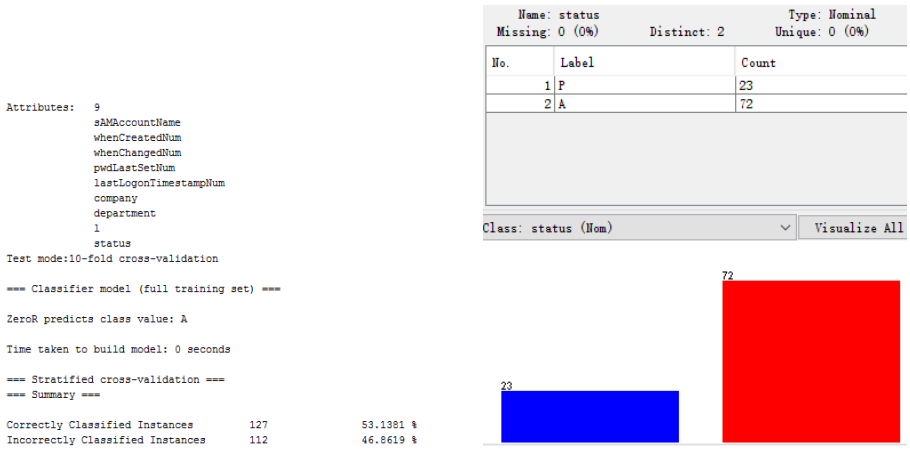


Figure 5.11: Retrieve training data

Figure 5.11 shows how we retrieve training data. On the left side is the dataset from the HR system. The two columns are `User-ID` and `status`, respectively. We input the file path to search from and the file path to store the results in our program. One single record is retrieved by giving the `sAMAccountName` value and the `status`

value. If the given user does not exist, it displays `Retrieve the record wrong!` and continues. The program terminates when the user inputs `!q` in the `sAMAccountName` field. Note that the header of the file which is searched from is copied to the file which stores the search results before the searching starts.

In addition, we write another program to retrieve testing data when needed. As we have an abundance of data, we can use an extra dataset to evaluate the performances. The second program additionally requires the file path of training data as one of the inputs. The difference of this program from the previous program is that it further confirms if the record you want to retrieve already exists in the training data.



(a) Training data statistics **(b)** Testing data statistics

Figure 5.12: Training data & testing data

The statistics of the training data and testing data we have retrieved are shown in Fig. 5.12. Figure 5.12a shows the statistics of the training data in the result of classifier **ZeroR**. **ZeroR** simply takes a new data point and assigns it with the majority class label. Therefore, the accuracy (53.14% in our case) of **ZeroR** is typically taken as the baseline of experiments. There are 239 entries in the training data, 53.14% of which are active users. Figure 5.12b displays the distribution of the testing data in a table and a chart, which is the interface of Weka Explorer Preprocess. 23 users are passive and 72 users are active.

Chapter 6

Fusion of the Identity Data

After extracting the data, we are working on the fusion of it, i.e., to infer the user's status from the information conveyed by the attributes. We start this chapter with the machine learning algorithms that we have chosen to do this task. We then give a detailed description how we conducted experiments with these algorithms on the identity data. Afterwards, we analyze the obtained results. And finally, we choose a few classifiers according to our analyses and implement them to classify users.

6.1 Machine Learning Algorithms in Weka

Weka[1] is a collection of machine learning algorithms for data mining tasks. It is an open source software issued under the **GNU General Public License**[2]. As explained in Chapter 4, we use Weka to train learning models. And as introduced in Sec. 3.3.3, we chose to consider 4 machine learning algorithms. Table 6.1 lists the algorithms and their implementations in Weka.

Table 6.1: Machine learning algorithms in Weka

Algorithm	Implementation in Weka
C4.5	J48
k–NN	IBk
Naive Bayes	NaiveBayes
SVM	SMO
SVM	LibSVM

C4.5 is described in (Quinlan, 1993). It is implemented as J48[3] in Weka. The details of k–NN can be found in (Mitchell, 1997b). The value of k is the number of neighbors. In Weka, it is implemented as IBk. In the description of this algorithm, k is said to be the number of neighbors as the k in k–NN. However, the cited paper (Aha, Kibler,

[1] `http://www.cs.waikato.ac.nz/ml/weka/`

[2] `https://www.gnu.org/licenses/gpl-3.0.html`

[3] This Pentaho page `http://wiki.pentaho.com/display/DATAMINING/Data+Mining+Algorithms+and+Tools+in+Weka` contains an overview of all algorithms and tools included in Weka.

and Albert, 1991) describes IB1, IB2 and IB3 as different algorithms. We believe the former description is correct. The value of k can be set to any number apart from $\{1, 2, 3\}$. There are also discussions on this topic in the mailing list[4], where they suggest the former description. The Bayes theorem and the naive Bayes classifier are described in (Mitchell, 1997c). (John and Langley, 1995) describes another way to estimate continuous values for naive Bayes classifier, using kernels instead of Gaussian distributions. (Chang and Lin, 2011) gives a practical guide to SVM classification for beginners, which also describes the use of LibSVM. LibSVM implements an SMO–type algorithm. We include the Java release of it in the CLASSPATH so that we can use it in Weka. SMO or Sequential Minimal Optimization is described in (Platt, 1998). It is an algorithm for solving large quadratic programming (**QP**) optimization problems that are required in the training of SVM.

6.2 Experiments

The performances of these algorithms are closely related to the chosen attributes. We have selected 13 attributes (Sec. 5.1) that we consider relevant for this classification task. Now we will look into the statistics of the values of these attributes and make a further reduced selection.

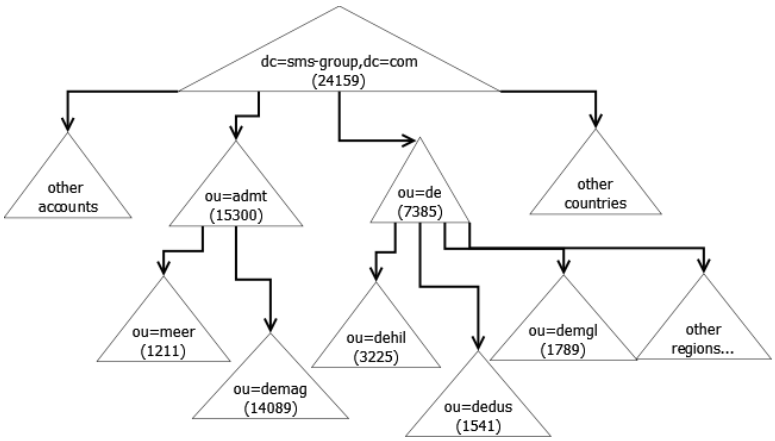


Figure 6.1: The structure of the directory

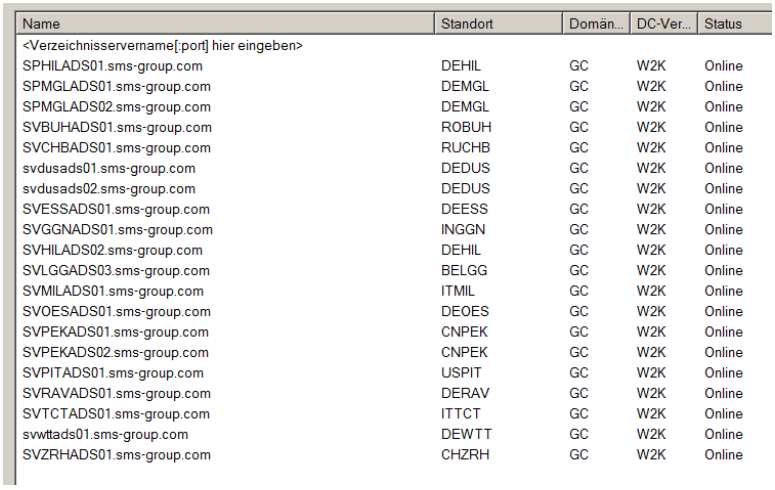
6.2.1 Further Analysis of the Selected Attributes

As shown in Fig. 6.1, the domain `dc=sms-group,dc=com` contains many different OUs. The data was retrieved on June 16, 2016. We extracted data from each OU and obtained this data distribution. The numbers in the parentheses are the numbers of user objects from the

[4] `http://marc.info/?l=wekalist&m=124251718130305`

OUs. The data in LDAP directories is very dynamic. One hour later, you may obtain a different numbers of objects.

There are 20 DCs which we have access to. They are designated as GCs as well, shown in Fig. 6.2. The data in our analysis here was retrieved from the DC `spmglads01.sms-group.com`.



Name	Standort	Domän...	DC-Ver...	Status
<Verzeichnisservername[:port] hier eingeben>				
SPHILADS01.sms-group.com	DEHIL	GC	W2K	Online
SPMGLADS01.sms-group.com	DEMGL	GC	W2K	Online
SPMGLADS02.sms-group.com	DEMGL	GC	W2K	Online
SVBUHADS01.sms-group.com	ROBUH	GC	W2K	Online
SVCHBADS01.sms-group.com	RUCHB	GC	W2K	Online
svdusads01.sms-group.com	DEDUS	GC	W2K	Online
svdusads02.sms-group.com	DEDUS	GC	W2K	Online
SVESSADS01.sms-group.com	DEESS	GC	W2K	Online
SVGGNADS01.sms-group.com	INGGN	GC	W2K	Online
SVHILADS02.sms-group.com	DEHIL	GC	W2K	Online
SVLGGADS03.sms-group.com	BELGG	GC	W2K	Online
SVMILADS01.sms-group.com	ITMIL	GC	W2K	Online
SVOESADS01.sms-group.com	DEOES	GC	W2K	Online
SVPEKADS01.sms-group.com	CNPEK	GC	W2K	Online
SVPEKADS02.sms-group.com	CNPEK	GC	W2K	Online
SVPITADS01.sms-group.com	USPIT	GC	W2K	Online
SVRAVADS01.sms-group.com	DERAV	GC	W2K	Online
SVTCTADS01.sms-group.com	ITTCT	GC	W2K	Online
svwttads01.sms-group.com	DEWTT	GC	W2K	Online
SVZRHADS01.sms-group.com	CHZRH	GC	W2K	Online

Figure 6.2: Domain controllers in `sms-group`

Data from Different OUs.

We observed that the data under domain `dc=sms-group,dc=com` has large proportions of missing values on some attributes. Hence, we further checked the data in all the OUs. Table 6.2 shows the missing value proportions of these attributes from some of them.

Table 6.2: The missing value proportions of some attributes from different OUs

Domains	Missing Values (%)					
	lC	lT	bPC	bPT	lL	lLT *
`dc=sms-group,dc=com`	87	57	56	89	87	72
`ou=dehil,ou=de`	84	25	24	90	84	29
`ou=demgl,ou=de`	17	17	14	20	17	16
`ou=dedus,ou=de`	74	8	8	90	84	29
`ou=dehdn,ou=de`	96	19	19	97	96	26
`ou=meer,ou=admt`	100	40	40	100	100	100
`ou=demag,ou=admt`	100	74	74	100	100	100

*lC=`logonCount`, lT=`lockoutTime`, bPC=`badPwdCount`
bPT=`badPasswordTime`, lL=`lastLogon`, lLT=`lastLogonTimestamp`

The names of the OUs in the table are the first parts of their full names. The full names are suffixed with the domain name `dc=sms-group,dc=com`. We can see that the data in OU `ou=demgl,ou=de,dc=sms-group,dc=com` contains more values compared to the large

proportions of missing values in other OUs. That's because some attributes, such as `logonCount, badPasswordTime, lastLogon`, are not replicated to GCs. The data was retrieved from a DC which is located in Mönchengladbach, and this OU is the delegation of offices in Mönchengladbach.

We did the same analysis for data retrieved from other DCs, which displays the same phenomenon. One unexpected behavior of the data is with the attribute `lastLogonTimestamp`. Its value is replicated globally. The large missing value proportion in the domain `dc=sms-group,dc=com` seems unreasonable since a missing value of this attribute implies that this object has never logged on to the domain. As the data that we retrieved from different DCs shows the same statistics in this attribute, there is no problem in replication. From this table, we can see that the data in OUs `ou=meer,ou=admt, dc=sms-group,dc=com` and `ou=demag,ou=admt,dc=sms-group,dc=com` is the main reason that there are many missing values globally.

Data from Different DCs.

Next, we retrieved user information from different DCs, shown in Tab. 6.3. The data is from user `DIAO`. There are 7 DCs. Two are in Mönchengladbach as `spmglads01(2).sms-group.com`. Two are in Hilchenbach as `sphilads01` and `svhilads02.sms-group.com`. One is in Düsseldorf as `svdusads01.sms-group.com`. One is in Milan as `svmilads01.sms-group.com`. And one is in Peking as `svpekads01.sms-group.com`.

Table 6.3: The values of some attributes in my record from different DCs

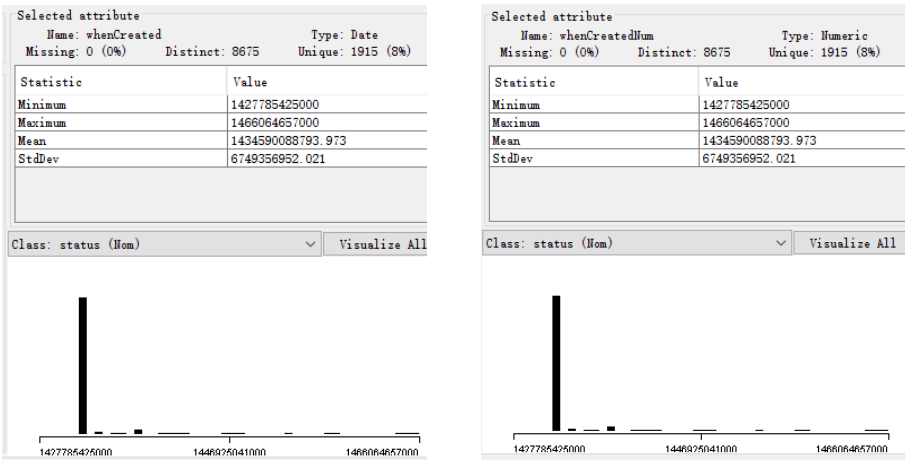
DCs	whenChanged	lT	bPC *	lLT
spmglads01	2016-06-16 12:11:07	?	0	2016-06-16 12:11:07
spmglads02	2016-06-16 12:11:12	?	0	2016-06-16 12:11:07
sphilads01	2016-06-16 12:23:10	?	0	2016-06-16 12:11:07
svhilads02	2016-06-16 12:23:10	?	0	2016-06-16 12:11:07
svdusads01	2016-06-16 12:24:57	?	?	2016-06-16 12:11:07
svmilads01	2016-06-16 12:32:32	?	?	2016-06-16 12:11:07
svpekads01	2016-06-17 16:26:04	?	?	2016-06-16 12:11:07

*lT=`lockoutTime`, bPC=`badPwdCount`, lLT=`lastLogonTimestamp`

We can see that the values of `whenChanged` vary from DC to DC with delays. The attribute of `lockoutTime` is not present. In some DCs, the values of `badPwdCount` are 0s while not present in some other DCs. In fact, in some DCs, the value of `badPasswordTime` is given, which means this account did log on with an incorrect password at least once. The value of 0 indicates unknown. The value of `lastLogonTimestamp` is synchronized in all DCs.

Final Attribute Set.

From the analysis above, we finally decided to train our data on 9 attributes, including the class label `status`. We stored date/time attributes both in date format and numeric format as Unix Epoch. From the data distribution shown in Fig. 6.3a and Fig. 6.3b we see that Weka reads dates as Unix Epoch values. To skip this process in Weka, we take the numeric presentation of each date attribute, i.e. *`attributeName`*`Num`.



(a) `whenCreated` stored in date format **(b)** `whenCreated` stored in numeric format

Figure 6.3: The attribute `whenCreated` in date and numeric formats

Weka cannot handle string values. We have to convert string attributes to nominal attributes, i.e., to list all the possible values in the ARFF header. There is the filter **StringToNominal** for this purpose, shown in Fig. 6.4. The number(s) in the **attributeRange** field is(are) the index(ices) of the attribute(s) starting from 1.

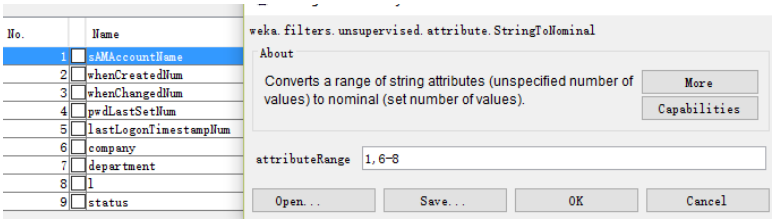


Figure 6.4: The filter **StringToNominal**

Additionally, Weka requires the ARFF headers of training data and testing data to be exactly the same. Apart from the same attributes and the same syntaxes for the attributes, the value sets of nominal attributes are required to be equal. Hence, we convert these string attributes to nominal with the ARFF file obtained from the

search base `dc=sms-group,dc=com` and copy them to other smaller data sets. The resulting ARFF header is shown in Fig. 6.5.

```
@relation 'users-weka.filters.unsupervise

@attribute sAMAccountName {x-weinssas,Adm
@attribute whenCreatedNum numeric
@attribute whenChangedNum numeric
@attribute pwdLastSetNum numeric
@attribute lastLogonTimestampNum numeric
@attribute company {'SMS group GmbH','SMS
@attribute department {6972.0,000000,SC,6
@attribute l {Hilchenbach,Mönchengladbach
@attribute status {P,A}

@data
```

Figure 6.5: The ARFF header of the final selected attributes

6.2.2 Select Attribute Set for Each Algorithm

To obtain better learning models, we select attributes for each algorithm to train on. There is the **Select attributes** function (WEKA Manual, 2015) in Weka, as shown in Fig. 6.6. There are two parameters in this method, **Attribute Evaluator** and **Search Method**. **Attribute Evaluator** determines what method is used to assign a worth to each subset of attributes. **Search Method** determines what style of search is performed. This example selects the best subset of attributes for classifier **J48**. The result is shown in Fig. 6.7.

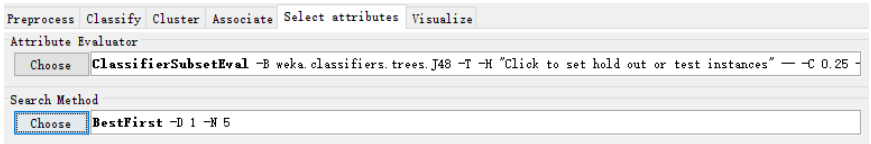


Figure 6.6: The function `Select Attributes`

We obtained the following attribute sets for different classifiers. For the classifier **J48**, most search methods return the attributes `whenChanged` and `pwdLastSet`; a few additionally include `whenCreated`; **ExhaustiveSearch** returns the attribute set `whenCreated, whenChanged, lastLogonTimestamp` and `l`. For the classifiers **IBk** and **NaiveBayes**, most search methods return the attribute `sAMAccountName`. For the classifier **SMO**, **RaceSearch** returns the attributes

pwdLastSet and l; **RankSearch** returns the set sAMAccountName, whenChanged, pwdLastSet, company and department; other methods return the attribute sAMAccountName. For the classifier **LibSVM**, most search methods return the attribute pwdLastSet.

```
Evaluation mode:evaluate on all training data



=== Attribute Selection on all input data ===

Search Method:
        Best first.
        Start set: no attributes
        Search direction: forward
        Stale search after 5 node expansions
        Total number of subsets evaluated: 36
        Merit of best subset found:    0.05

Attribute Subset Evaluator (supervised, Class (nominal): 9 status):
        Classifier Subset Evaluator
        Learning scheme: weka.classifiers.trees.J48
        Scheme options: -C 0.25 -M 2
        Hold out/test set: Training data
        Accuracy estimation: classification error

Selected attributes: 2,3,4 : 3
                     whenCreatedNum
                     whenChangedNum
                     pwdLastSetNum
```

Figure 6.7: The best subset of attributes for the classifier **J48**

6.3 Results and Analyses

We ran these machine learning algorithms with different attribute sets and parameter settings. The following sections show you some results and analyses.

6.3.1 J48 Models

We first trained some J48 models on the entire attribute set with different **confidenceFactor** values. This parameter is used for pruning. Smaller values incur more pruning. The default value is 0.25. We then trained more J48 models with different attribute sets as we observed that the presence of some attributes complicates the tree structure. Table 6.4 shows the results of some J48 models. The accuracies on training data and testing data, the number of leaves, and the tree size are given. All accuracies on training data are evaluated with 10 folds cross–validation.

Table 6.4: Some J48 models

Models	Training (%)	Testing (%)	No. of Leaves	Tree Size
Pruned	88.70	75.79	2	3
Unpruned	90.80	85.26	2290	2294
Partly pruned	89.54	81.05	113	115
AttrSet[1]	88.70	80	8	15
AttrSet[2]	83.68	82.11	141	150

AttrSet[1] = {`whenCreated, whenChanged, pwdLastSet, lastLogonTimestamp`}

AttrSet[2] = {`whenCreated, whenChanged, lastLogonTimestamp, l`}

The first model was trained on the entire attribute set with the default value for **confidenceFactor**. This model performs poorly on testing data. In fact, it only considers the attribute `pwdLastSet`, which gives a tree with two leaves. We then enabled the **unpruned** parameter. The trained model (the second row) performs best on both training data and testing data. However, the generated tree is too large. It includes the attributes `company, department` and `l`, whose value sets are very large. The **confidenceFactor** was set to 0.5 in the third model. The accuracies are slightly better than the 0.25 pruned tree and only slightly worse than the unpruned tree, and the size of the generated tree is acceptable.

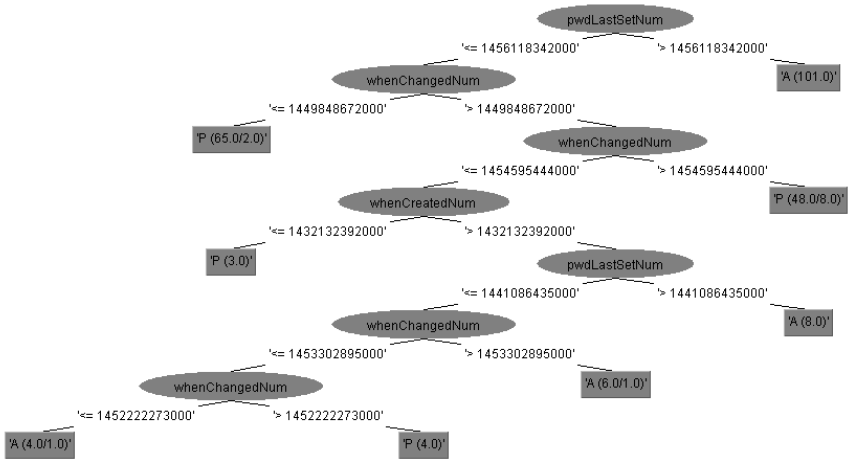


Figure 6.8: The tree view of trained model on AttrSet[1]

The last two models were trained on different attribute sets. They are both pruned. It turned out there is not much difference between pruned and unpruned tree in these two models. The accuracies are the same. The tree sizes vary a bit. AttrSet[1] only contains date/-time attributes, whose tree view is given in Fig. 6.8. We can see that the attribute `pwdLastSet` is the dominant attribute, followed

by `whenChanged` and `whenCreated`. AttrSet[2] is the result of **ExhaustiveSearch** on classifier **J48**. It has the best accuracy on testing data after the unpruned model.

6.3.2 IBk & NaiveBayes

For **IBk**, we did experiments with different values of k (the number of neighbors). They were all trained on the entire attribute set. The default value $k = 1$ gave very good accuracies, 83.69% on training data and 84.21% on testing data. Other values of k did not show a significant improvement. On the contrary, some of them performed worse. We also tried to adjust other parameters, such as **distanceWeighting** and **nearestNeighbourSearchAlgorithm**. It turned out that the default setting with **distanceWeighting** disabled and applying **LinearNNSearch** with **EuclideanDistance** is the optimal option.

For **NaiveBayes**, we did experiments with different attribute sets. The entire attribute set and the attribute set excluding `sAMAccountName` behaved the same in terms of accuracies: 84.10% on training data and 83.16% on testing data. They are also the best accuracies we obtained with the **NaiveBayes** classifier. Hence, we consider this model as one of our final classifiers for predicting.

6.3.3 SVM Models

Table 6.5 compares some **SVM** models in their accuracies on the training data and testing data, as well as the time needed to build the models. Similar to the **J48** models, the accuracies on the training data are evaluated with 10 folds cross–validation.

Table 6.5: Some SVM models

Models	Training (%)	Testing (%)	Speed (s)
SMO	84.10	83.16	4.72
LibSVM	53.14	76.84	0.54
AttrSet[3]	83.26	84.21	0.08
AttrSet[4]	85.36	81.05	4.09
AttrSet[5]	83.68	83.16	0.18

AttrSet[3] = {`pwdLastSet, l`}

AttrSet[4] = {`sAMAccountName, whenChanged, pwdLastSet, company, department`}

AttrSet[5] = {`whenChanged, pwdLastSet, department`}

The first model was built with the **SMO** classifier in Weka on the entire attribute set. The second one was built with the **LibSVM** classifier on the entire attribute set. We did not change any parameters for either of them. We can see that to build an SVM with **LibSVM** is indeed faster than with **SMO**. However, the accuracy suffers.

LibSVM uses RBF (Radial Basis Function) kernels by default while **SMO** uses linear kernels by default.

The final three models, AttrSet^3, AttrSet^4, and AttrSet^5, were trained with **SMO** on different attribute sets. They were evaluated with the default settings. When we changed the kernel to RBF with the default parameter value, $\gamma = 0.01$ (in **LibSVM**, $\gamma = 0$), the obtained accuracies are slightly worse than those trained with linear kernels. The model trained on AttrSet^3 gives good accuracy on the training data, the best accuracy on the testing data, and needs the least time to be built. It should be included as one of our final classifiers. However, we included the fourth model (on AttrSet^4) instead in our program to classify users. Further experiments with other functions could be conducted to choose better parameter values which could give better accuracies.

6.4 Classify Users

In this section, we illustrate how to use the classifiers we have chosen to classify users. As analyzed in the previous section, we include the last three **J48** models, **IBk** and **NaiveBayes** with default settings on the entire attribute set, and the fourth SVM model. In our program, the training data is loaded and processed according to each classifier's requirements, e.g. remove some attributes. The parameters of the classifiers are set, e.g. set the value of **confidenceFactor**. Figure 6.9 shows the code to build the first two **J48** models with the Weka API in Java.

```
//J48 classifier with confidence factor 0.5 on full attributes set
J48 cls1 = new J48();
cls1.setConfidenceFactor((float) 0.5);
cls1.buildClassifier(trainingData);
//System.out.println(Arrays.toString(cls1.getOptions()));

//J48 classifier with default settings on attributes
//sAMAccountName, whenCreated, whenChanged, pwdLastSet, lastLogonTimestamp
//remove some attributes using filter
Remove rm2 = new Remove();
rm2.setAttributeIndices("6-8"); //Remove attributes 6,7,8
J48 cls2 = new J48();
FilteredClassifier fc2 = new FilteredClassifier();
fc2.setFilter(rm2);
fc2.setClassifier(cls2);
fc2.buildClassifier(trainingData);
```

Figure 6.9: Build J48 classifiers with the Weka API in Java

In our interface, the file paths of the training data, the user data to be classified, and result file need to be given. We also added one more function in our **RetrieveIdentityData** program (see Sec. 5.2) to retrieve some users by giving their `User-ID`. Although it does not

take much time to classify user objects from the domain `dc=sms-group,dc=com`.

In our result file, the meta information is given first, shown in Fig. 6.10. It records where the data is from, the attributes in the data, and the classifiers with parameter information.

```
#The training data is read from data\trainingData.arff
#The data to be classified is from data\Users.arff

The attributes are:
1. sAMAccountName
2. whenCreatedNum
3. whenChangedNum
4. pwdLastSetNum
5. lastLogonTimestampNum
6. company
7. department
8. l
9. status


*****1. J48 partly pruned on full attributes set      *****
[-C, 0.5, -M, 2, , , , , , , , , , ]
*****2. J48 pruned on selected attributes set         *****
[-F, weka.filters.unsupervised.attribute.Remove -R 6-8, -W, w
*****3. J48 unpruned on selected attributes set       *****
[-F, weka.filters.unsupervised.attribute.Remove -R 4,6,7, -W,
*****4. IBk (k=1) on full attributes set              *****
[-K, 1, -W, 0, -A, weka.core.neighboursearch.LinearNNSearch -
*****5. NaiveBayes on full attributes set             *****
[, , ]
*****6. SMO (linear kernel) on selected attributes set*****
[-F, weka.filters.unsupervised.attribute.Remove -R 2,5,8, -W,
```

Figure 6.10: The meta information in an output file

The prediction results of some users are given in Fig. 6.11. In Fig. 6.11a: user `DIAO` was classified as "Active" by all classifiers; user `WAGENTOM` was classified as "Passive" by 4 classifiers and as "Active" by the other 2 classifiers. The user in Fig. 6.11b, `BRB1`, was classified as "Passive" by all classifiers. Users with different prediction results (e.g. user `WAGENTOM`) are marked by "*"; users which are classified as "Passive" by all classifiers (e.g. user `BRB1`) are marked by "**".

```
DIAO - A - 0,00 | 1,00
 -     A - 0,00 | 1,00
 -     A - 0,04 | 0,96
 -     A - 0,00 | 1,00
 -     A - 0,00 | 1,00
 -     A - 0,00 | 1,00
WAGENTOM - P - 0,86 | 0,14
 -         P - 0,83 | 0,17
 -         P - 1,00 | 0,00
 -         P - 1,00 | 0,00
 -         A - 0,03 | 0,97
 -         A - 0,00 | 1,00 * different predicting results
```

(a) Prediction results 1

```
BRB1 - P - 0,81 | 0,19
 -     P - 0,83 | 0,17
 -     P - 0,94 | 0,06
 -     P - 1,00 | 0,00
 -     P - 0,85 | 0,15
 -     P - 1,00 | 0,00 ** predicted as Passive by all classifiers
```

(b) Prediction results 2

Figure 6.11: Prediction results of some users

Chapter 7

Conclusion and Future Work

7.1 Summary

In this master thesis, we analyzed the attributes stored in AD for user objects; we then extracted some of them and stored them into ARFF files; furthermore, we utilized machine learning algorithms on the data with Weka and analyzed the correlations between these attributes and users' statuses; finally, we implemented a program with a few chosen classifiers to predict users' statuses.

7.2 Conclusion

From our work, we can conclude that using machine learning to detect stale users is possible. There is much more information stored in AD than last log on information for identifying inactive users. In fact, the attribute `lastLogonTimestamp` did not take an important role in our process, contrary to the current methods for this task. The reason is the limited record of this attribute. Note that 72% of values are missing in the domain `dc=sms-group,dc=com`. And the main cause is the 100% missing values in two OUs. We do not know the internal reorganization during the migration and integration of the data from old directories to this new one.

Nevertheless, we found that attributes `pwdLastSet` and `company` are the dominant factors in the decision tree learning algorithm. If we suppress the factors `company, department,` and `l` (AttrSet[1]), as their value sets are large, `whenChanged` and `whenCreated`, following `pwdLastSet`, are the most influential factors. Attributes `pwdLastSet` and `l` gave the best performance in the SVM algorithm. In addition, k–NN and naive Bayes on all attributes also gave as good of an accuracy as C4.5 and SVM.

7.3 Future Work

In the future, we would consider to extend our work so that a more accurate decision of a user being "Active" can be given. For example, we could assign each of our classifiers with a weight in order to

compute the final probability. The weights could be calculated from their accuracies in testing and normalized. We would only output users whose probabilities of being "Active" are below a threshold for further investigation. This "threshold" would need to be carefully designed.

Because we initially considered that the last log on information would be the main factor for our classification task, which turned out to be the contrary, we also consider doing a further experiment with training data only from the OUs which do not have large missing value proportions in `lastLogonTimestamp`. We would like to see how these algorithms behave with a data set which does not have many missing values in `lastLogonTimestamp`.

Appendix A

Data Stream

A.1 Example User Data

Table A.1: Example User Data (1)

Attribute	Value(s)
objectCategory	CN=Person,CN=Schema,CN=Configuration,DC=sms-group,DC=com
adminDescription	true
whenCreated	20160201101940.0Z
badPwdCount	0
mDBUseDefaults	TRUE
codePage	0
publicDelegates	CN=diao,OU=FIM,OU=User,OU=DEHIL,OU=DE,DC=sms-group,DC=com
msExchELCMailboxFlags	16
mail	yujie.diao@sms-group.com
objectGUID	not readable*
msExchUserAccountControl	0
msExchMailboxSecurityDescriptor	not readable
memberOf	CN=S-T-ITOperations-SDE-C,OU=Teamfolder,OU=Fileservice, OU=Application,OU=DE,DC=sms-group,DC=com CN=v_ev_user,OU=Groups,OU=DEMAG,OU=ADMT, DC=sms-group,DC=com CN=glo_Users_BK,OU=Groups,OU=DEMAG,OU=ADMT, DC=sms-group,DC=com CN=v_standort_hil-sde,OU=Groups,OU=DEMAG,OU=ADMT, DC=sms-group,DC=com CN=glo_pb,OU=Groups,OU=DEMAG,OU=ADMT,DC=sms-group,DC=com CN=glo_pbhdn,OU=Groups,OU=DEMAG,OU=ADMT, DC=sms-group,DC=com CN=mig_done,CN=Users,DC=sms-group,DC=com CN=glo_intranet_user,OU=Groups,OU=DEMAG,OU=ADMT, DC=sms-group,DC=com CN=un_internet_emea_01,OU=Groups,OU=DEMAG,OU=ADMT, DC=sms-group,DC=com
msExchMailboxGuid	not readable
instanceType	4
objectSid	not readable
badPasswordTime	131098547085090154
proxyAddresses	x500:/o=SMS Group/ou=Exchange Administrative Group (FYDIBOHF23SPDLT)/cn=Recipients/ cn=8dc4bb7e629d48e3a11ceaf9e37680ec-diao x500:/o=SMS DEMAG AG/ou=Exchange Administrative Group (FYDIBOHF23SPDLT)/cn=Recipients/cn=DIAOf83 smtp:yujie.diao@smssiemag.com smtp:yujie.diao@sms.ex.sms-siemag.com smtp:yujie.diao@sms-siemag.de smtp:yujie.diao@sms-siemag.com smtp:yujie.diao@sms-metallurgy.com smtp:diao@target.sms-group.com smtp:diao@smssiemag.com smtp:diao@sms-siemag.de smtp:diao@sms-siemag.com smtp:diao@sms-metallurgy.com smtp:diao@sms-group.com smtp:diao@siemag.sms-group.com SMTP:yujie.diao@sms-group.com
dSCorePropagationData	20160223030241.0Z 16010101000001.0Z
objectClass	top person organizationalPerson user

Table A.2: Example User Data (2)

Attribute	Value(s)
company	SMS group GmbH
msExchWhenMailboxCreated	20160222230548.0Z
name	diao
sn	Diao
userAccountControl	512
primaryGroupID	513
lastLogon	131103031225978224
accountExpires	9223372036854775807
lastLogoff	0
adminDisplayName	true
uSNChanged	19632476
publicDelegatesBL	CN=diao,OU=FIM,OU=User,OU=DEHIL,OU=DE,DC=sms-group,DC=com
msExchRBACPolicyLink	CN=Default Role Assignment Policy,CN=Policies,CN=RBAC, CN=SMS Group,CN=Microsoft Exchange,CN=Services, CN=Configuration,DC=sms-group,DC=com
cn	diao
msExchVersion	88218628259840
msExchTextMessagingState	302120705 16842751
protocolSettings	HTTP$1$1$$$$$$ OWA$1
logonCount	44
msExchHomeServerName	/o=SMS Group/ou=Exchange Administrative Group (FYDIBO HF23SPDLT)/cn=Configuration/cn=Servers/cn=SPHILEXC04
msExchMDBRulesQuota	64
sAMAccountType	805306368
msExchRecipientTypeDetails	1
legacyExchangeDN	/o=SMS Group/ou=Exchange Administrative Group (FYDIBOHF23SPDLT)/cn=Recipients/ cn=8dc4bb7e629d48e3a11ceaf9e37680ec-diao
sIDHistory	not readable
givenName	Yujie
uSNCreated	9795713
displayName	Diao, Yujie (SMS group GmbH)
userPrincipalName	diao@sms-group.com
pwdLastSet	131060509569932250
whenChanged	20160606085016.0Z
lastLogonTimestamp	131096766169035322
department	ITCN
countryCode	0
mailNickname	diao
l	Hilchenbach
distinguishedName	CN=diao,OU=FIM,OU=User,OU=DEHIL,OU=DE,DC=sms-group,DC=com
homeMDB	CN=DE-DB31,CN=Databases,CN=Exchange Administrative Group (FYDIBOHF23SPDLT),CN=Administrative Groups, CN=SMS Group,CN=Microsoft Exchange,CN=Services, CN=Configuration,DC=sms-group,DC=com
msExchRecipientDisplayType	1073741824
msExchUMDtmfMap	emailAddress:985433426 lastNameFirstName:342698543 firstNameLastName:985433426
mAPIRecipient	TRUE
msExchPoliciesExcluded	{26491cfc-9e50-4857-861b-0cb8df22b5d7}
msExchUMEnabledFlags2	-1
showInAddressBook	CN=All Mailboxes(VLV),CN=All System Address Lists,CN=Address Lists Container,CN=SMS Group,CN=Microsoft Exchange,CN=Services,CN=Configuration,DC=sms-group,DC=com CN=Mailboxes(VLV),CN=All System Address Lists,CN=Address Lists Container,CN=SMS Group,CN=Microsoft Exchange, CN=Services,CN=Configuration,DC=sms-group,DC=com CN=All Recipients(VLV),CN=All System Address Lists,CN=Address Lists Container,CN=SMS Group,CN=Microsoft Exchange,CN=Services,CN=Configuration,DC=sms-group,DC=com CN=All Users,CN=All Address Lists,CN=Address Lists Container,CN=SMS Group,CN=Microsoft Exchange,CN=Services, CN=Configuration,DC=sms-group,DC=com CN=Default Global Address List,CN=All Global Address Lists,CN=Address Lists Container,CN=SMS Group, CN=Microsoft Exchange,CN=Services,CN=Configuration, DC=sms-group,DC=com
sAMAccountName	DIAO

The attributes which are not readable do not have a human–readable value. Those attributes marked in **bold** are also present in GCs.

A.2 Select Attributes

The following data stream is the result of **Select Attributes** for **SMO**:

```
=== Run information ===

Evaluator:     weka.attributeSelection.ClassifierSubsetEval -B weka.classif-
    iers.functions.SMO -T -H "Click to set hold out or test instances" -- -C
    1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.
    supportVector.PolyKernel -C 250007 -E 1.0"
Search:weka.attributeSelection.RaceSearch -R 0 -L 0.001 -T 0.001 -F 0 -N -1
    -J -1.7976931348623157E308 -A weka.attributeSelection.GainRatioAttribut-
    eEval --
Relation:      users-weka.filters.unsupervised.attribute.Remove-R2,4,6-12,14-
    16-weka.filters.unsupervised.attribute.Remove-R1
Instances:     239
Attributes:    8
               whenCreatedNum
               whenChangedNum
               pwdLastSetNum
               lastLogonTimestampNum
               company
               department
               l
               status
Evaluation mode:evaluate on all training data

=== Attribute Selection on all input data ===

Search Method:
   RaceSearch.
   Race type :  forward selection race
   Base set :  no attributes
   Cross validation mode :  10 fold
   Merit of best subset found :  0.167

Attribute Subset Evaluator (supervised, Class (nominal):  8 status):
   Classifier Subset Evaluator
   Learning scheme:  weka.classifiers.functions.SMO
   Scheme options:  -C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K weka.classifi-
       ers.functions.supportVector.PolyKernel -C 250007 -E 1.0
   Hold out/test set:  Training data
    Accuracy estimation:  classification error

Selected attributes:  3,7 :  2
                      pwdLastSetNum
                      l
```

A.3 J48 Classifier

The following data stream is the result of running the **J48** classifier on AttrSet[1]:

```
=== Run information ===

Scheme:weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     users-weka.filters.unsupervised.attribute.Remove-R2,4,6-12,14-
    16-weka.filters.unsupervised.attribute.Remove-R6-8
Instances:    239
Attributes:   6
              sAMAccountName
              whenCreatedNum
              whenChangedNum
              pwdLastSetNum
              lastLogonTimestampNum
              status
Test mode:10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree
------------

pwdLastSetNum <= 1456118342000
   whenChangedNum <= 1449848672000:  P (65.0/2.0)
   whenChangedNum > 1449848672000
      whenChangedNum <= 1454595444000
         whenCreatedNum <= 1432132392000:  P (3.0)
         whenCreatedNum > 1432132392000
            pwdLastSetNum <= 1441086435000
               whenChangedNum <= 1453302895000
                  whenChangedNum <= 1452222273000:  A (4.0/1.0)
                  whenChangedNum > 1452222273000:  P (4.0)
               whenChangedNum > 1453302895000:  A (6.0/1.0)
            pwdLastSetNum > 1441086435000:  A (8.0)
      whenChangedNum > 1454595444000:  P (48.0/8.0)
pwdLastSetNum > 1456118342000:  A (101.0)

Number of Leaves :    8

Size of the tree :    15

Time taken to build model:  0.03 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances    212    88.7029 %
Incorrectly Classified Instances   27    11.2971 %
Kappa statistic                0.7738
Mean absolute error            0.1369
```

```
Root mean squared error         0.2942
Relative absolute error         27.4881 %
Root relative squared error     58.958 %
Total Number of Instances    239

=== Detailed Accuracy By Class ===
                TP Rate  FP Rate  Precision  Recall  F-Measure  ROC Area  Class
                 0.902    0.126     0.863     0.902     0.882      0.932      P
                 0.874    0.098      0.91     0.874     0.892      0.932      A
 Weighted Avg.   0.887    0.111     0.888     0.887     0.887      0.932

=== Confusion Matrix ===
  a     b    <- classified as
 101   11 |         a=P
 16   111|         b=A

=== Re-evaluation on test set ===

User supplied test set
Relation:     users-weka.filters.unsupervised.attribute.Remove-R2,4,6-12,14-16-
    weka.filters.unsupervised.attribute.Remove-R6-8
Instances:    unknown (yet).  Reading incrementally
Attributes:    6

=== Summary ===

Correctly Classified Instances    76    80 %
Incorrectly Classified Instances   19    20 %
Kappa statistic                0.5191
Mean absolute error             0.2042
Root mean squared error         0.4081
Total Number of Instances    95

=== Detailed Accuracy By Class ===
                TP Rate  FP Rate  Precision  Recall  F-Measure  ROC Area  Class
                 0.783    0.194     0.563     0.783     0.655      0.837      P
                 0.806    0.217     0.921     0.806     0.859      0.837      A
 Weighted Avg.    0.8     0.212     0.834      0.8      0.81       0.837

=== Confusion Matrix ===
 a    b   <- classified as
 18   5 |         a=P
 14   58|         b=A
```

Bibliography

Aha, David W., Dennis Kibler, and Marc K. Albert (1991). "Instance-Based Learning Algorithms". In: *Machine Learning, 6, 37-66*.

ApacheDS. "1.2 - Some Backgroud. Directories, directory services and LDAP". In: *ApacheDS v2.0 Basic User's Guide*. https://directory.apache.org/apacheds/basic-ug/1.2-some-background.html [Accessed 19-May-2016].

Biddle, Sam (2014). "Reseacher: Sony Hack Was Likely an Inside Job by a Woman Named 'Lena'". In: *Gawker*. URL: http://gawker.com/researcher-sony-hack-was-likely-an-inside-job-by-a-wom-1676556756.

Bouckaert, R. R. et al. (2015). "5.6 Selecting Attributes". In: *WEKA Manual for Version 3-7-13*. University of Waikato, Hamilton, New Zealand.

Chang, Chih-Chung and Chih-Jen Lin (2011). "LIBSVM: A library for support vector machines". In: *ACM Transactions on Intelligent Systems and Technology* 2 (3). Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm, 27:1–27:27.

Harnad, Stevan (2008). "The Annotation Game: On Turing (1950) on Computing, Machinery, and Intelligence (PUBLISHED VERSION BOWDLERIZED)". In: ed. by Robert Epstein, Gary Roberts, and Grace Beber. Chapter: 3 Commentary On: Turing, A.M. (1950) Computing Machinery and Intelligence. Mind 49 433-460 Address: Amsterdam, pp. 23–66. URL: http://eprints.soton.ac.uk/262954/.

ITU-T (2012a). *Information technology - Open Systems Interconnection - The Directory: Abstract service definition*. International Standard ISO/IEC 9594-3. Telecommunication Standardization Sector of ITU. URL: http://www.itu.int/ITU-T/recommendations/rec.aspx?id=11738&lang=en.

— (2012b). *Information technology - Open Systems Interconnection - The Directory: Overview of concepts, models and services*. International Standard ISO/IEC 9594-1. Telecommunication Standardization Sector of ITU. URL: http://www.itu.int/ITU-T/recommendations/rec.aspx?id=11732&lang=en.

John, George H. and Pat Langley (1995). "Estimating Continuous Distributions in Bayesian Classifiers". In: *Eleventh Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers, San Mateo, pp. 338–345.

McFerron, Ken (2011). "Use PowerShell to Find and Remove Inactive Active Directory Users". In: *Microsoft TechNet Blogs: Hey, Scripting Guy! Blog*. URL: https://blogs.technet.microsoft.

com/heyscriptingguy/2011/11/30/use-powershell-to-find-and-remove-inactive-active-directory-users/.

Microsoft. "AccountInactive". In: *Microsoft TechNet Library: Search-ADAccount*. https://technet.microsoft.com/de-de/library/ee617247.aspx?f=255&MSPPError=-2147217396 [Accessed 08-June-2016].

— "Active Directory: LDAP Syntax Filters". In: *Microsoft TechNet Wiki*. http://social.technet.microsoft.com/wiki/contents/articles/5392.active-directory-ldap-syntax-filters.aspx [Accessed 27-May-2016].

— "Active Directory Schema". In: *Microsoft MSDN Documentation*. https://msdn.microsoft.com/en-us/library/ms675085(v=vs.85).aspx [Accessed 08-March-2016].

— "Characteristics of Attributes". In: *Microsoft MSDN Documentation*. https://msdn.microsoft.com/en-us/library/ms675578(v=vs.85).aspx [Accessed 08-March-2016].

— "Characteristics of Object Classes". In: *Microsoft MSDN Documentation*. https://msdn.microsoft.com/en-us/library/ms675579(v=vs.85).aspx [Accessed 08-March-2016].

— "File Times". In: *Microsoft MSDN Documentation*. https://msdn.microsoft.com/de-de/library/windows/desktop/ms724290(v=vs.85).aspx [Accessed 19-July-2016].

— "Object Class & Object Category". In: *Microsoft MSDN Documentation*. https://msdn.microsoft.com/en-us/library/ms677612(v=vs.85).aspx [Accessed 24-May-2016].

— "Retrieving Large Results Sets". In: *Microsoft MSDN Documentation*. https://msdn.microsoft.com/en-us/library/aa746459(v=vs.85).aspx [Accessed 27-May-2016].

— "Searching Active Directory with Windows PowerShell". In: *Microsoft TechNet Library*. https://technet.microsoft.com/en-us/library/ff730967.aspx [Accessed 27-May-2016].

— "String (Generalized-Time) syntax". In: *Microsoft MSDN Documentation*. https://msdn.microsoft.com/de-de/library/ms684436(v=vs.85).aspx [Accessed 19-July-2016].

— "Understanding Data Synchronization with External Systems". In: *Microsoft TechNet Library*. https://technet.microsoft.com/en-us/library/jj133850(v=ws.10)[Accessed 18-May-2016].

— "What is a Directory Service?" In: *Microsoft MSDN Documentation*. https://msdn.microsoft.com/en-us/library/aa367035(v=vs.85).aspx [Accessed 19-May-2016].

— (2004). "Identity and Access Management". In: *Microsoft MSDN Documentation*. URL: https://msdn.microsoft.com/en-us/library/aa480030.aspx.

— (2006a). "Chapter 2: Terminology and Initiatives". In: *Microsoft Identity and Access Management Series: Foundamental Concepts*. Microsoft, pp. 9–10. URL: https://www.microsoft.com/en-us/download/details.aspx?id=17974.
— (2006b). "Chapter 3: Microsoft Identity and Access Management Technologies". In: *Microsoft Identity and Access Management Series: Foundamental Concepts*. Microsoft, p. 28. URL: https://www.microsoft.com/en-us/download/details.aspx?id=17974.
— (2006c). "Chapter 3: Microsoft Identity and Access Management Technologies". In: *Microsoft Identity and Access Management Series: Foundamental Concepts*. Microsoft, p. 32. URL: https://www.microsoft.com/en-us/download/details.aspx?id=17974.
— (2012). "Dsquery". In: *Microsoft TechNet Library: Management and Tools*. URL: https://technet.microsoft.com/en-us/library/cc732952(v=ws.11).aspx.
— (2014a). "Active Directory Domain Services Collection". In: *Microsoft TechNet Library*. URL: https://technet.microsoft.com/en-us/library/cc780036(v=ws.10).aspx.
— (2014b). "Active Directory Structure and Storage Technologies". In: *Microsoft TechNet Library*. URL: https://technet.microsoft.com/en-us/library/cc759186(v=ws.10).aspx.
— (2014c). "Domain Controller Roles". In: *Microsoft TechNet Library: Active Directory Collection*. URL: https://technet.microsoft.com/en-us/library/cc780036(v=ws.10).aspx.
— (2014d). "Understanding Active Directory Domain Services (AD DS) Functional Levels". In: *Microsoft TechNet Library: Active Directory Services*. URL: https://technet.microsoft.com/en-us/library/dbf0cdec-d72f-4ba3-bc7a-46410e02abb0?f=255&MSPPError=-2147217396.
— (2014e). "What's New in Active Directory in Windows Server". In: *Microsoft TechNet Library*. URL: https://technet.microsoft.com/en-us/library/dn268294.aspx.
— (2015a). "2.118 Attribute company, 2.151 Attribute department, & 2.345 Attribute l". In: *[MS-ADA1]: Active Directory Schema Attributes A-L*. Microsoft, pp. 56,68,139.
— (2015b). "2.175 Attribute pwdLastSet". In: *[MS-ADA3]: Active Directory Schema Attributes N-Z*. Microsoft, p. 77.
— (2015c). "2.222 Attribute sAMAccountName & 2.223 Attribute sAMAccountType". In: *[MS-ADA3]: Active Directory Schema Attributes N-Z*. Microsoft, pp. 95–96.
— (2015d). "2.349 Attribute userPrincipleName". In: *[MS-ADA3]: Active Directory Schema Attributes N-Z*. Microsoft, p. 147.
— (2015e). "2.351 Attribute lastLogon & 2.352 Attribute lastLogonTimestamp". In: *[MS-ADA1]: Active Directory Schema Attributes A-L*. Microsoft, p. 142.

Microsoft (2015f). "2.356 Attribute lDAPDisplayName". In: *[MS-ADA1]: Active Directory Schema Attributes A-L*. Microsoft, p. 144.
— (2015g). "2.363 Attribute msDS-LogonTimeSyncInterval". In: *[MS-ADA2]: Active Directory Schema Attributes M*. Microsoft, pp. 155–156.
— (2015h). "2.370 Attribute whenChanged & 2.371 Attribute whenCreated". In: *[MS-ADA3]: Active Directory Schema Attributes N-Z*. Microsoft, p. 147.
— (2015i). "2.373 Attribute lockoutTime". In: *[MS-ADA1]: Active Directory Schema Attributes A-L*. Microsoft, p. 150.
— (2015j). "2.375 Attribute logonCount". In: *[MS-ADA1]: Active Directory Schema Attributes A-L*. Microsoft, p. 151.
— (2015k). "2.39 Attribute objectCategory". In: *[MS-ADA3]: Active Directory Schema Attributes N-Z*. Microsoft, p. 27.
— (2015l). "2.82 Attribute badPasswordTime". In: *[MS-ADA1]: Active Directory Schema Attributes A-L*. Microsoft, p. 43.
— (2015m). "2.83 Attribute badPwdCount". In: *[MS-ADA1]: Active Directory Schema Attributes A-L*. Microsoft, p. 43.
— (2015n). "3.1.1.2.4 Classes". In: *[MS-ADTS]: Active Directory Technical Specification*. Microsoft, pp. 126–128.
— (2015o). "3.1.1.4.5.26 msDS-UserPasswordExpired & 3.1.1.4.5.33 msDS-UserPasswordExpiryTimeComputed". In: *[MS-ADTS]: Active Directory Technical Specification*. Microsoft, pp. 248,251.
Mitchell, Tom M. (1997a). *Machine Learning*. McGraw Hill, p. 2. ISBN: 0070428077.
— (1997b). *Machine Learning*. McGraw Hill, pp. 230–236. ISBN: 0070428077.
— (1997c). *Machine Learning*. McGraw Hill, pp. 154–184. ISBN: 0070428077.
OpenLDAP (2016). "1.2. What is LDAP?" In: *OpenLDAP Software 2.4 Administration's Guide*. URL: `http://www.openldap.org/doc/admin24/index.html`.
Platt, John C. (1998). *Fast Training of Support Vector Machines using Sequential Minimal Optimization*. TechReport MSR-TR-98-14. Microsoft Research Lab - Redmond. URL: `https://www.microsoft.com/en-us/research/publication/sequential-minimal-optimization-a-fast-algorithm-for-training-support-vector-machines/`.
Pyle, Ned (2009). "'The LastLogonTimeStamp Attribute' - 'What it was designed for and how it works'". In: *Microsoft TechNet Blogs: Ask the Directory Services Team*. URL: `https://blogs.technet.microsoft.com/askds/2009/04/15/the-lastlogontimestamp-attribute-what-it-was-designed-for-and-how-it-works/`.
Quinlan, Ross (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA.
Russell, Stuart J. and Peter Norvig (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall, p. 528. ISBN: 0-13-103805-2.

Sermersheim, J. (2006). *Lightweight Directory Access Protocol (LDAP): The Protocol*. RFC 4511. RFC Editor. URL: http://www.rfc-editor.org/rfc/rfc4511.txt.

Smith, M. and T. Howes (2006). *Lightweight Directory Access Protocol (LDAP): String Representation of Search Filters*. RFC 4515. RFC Editor. URL: http://www.rfc-editor.org/rfc/rfc4515.txt.

Smith, M. et al. (2000). *The C LDAP Application Program Interface <draft-ietf-ldapext-ldap-c-api-05.txt>*. IETF Docs. Network Working Group INTERNET-DRAFT. URL: http://www-archive.mozilla.org/directory/ietf-docs/draft-ietf-ldapext-ldap-c-api-05.txt.

Turing, Alan (1950). "Computing Machinery and Intelligence". In: *Mind* LIX (236), p. 433. URL: http://mind.oxfordjournals.org/content/LIX/236/433.

Webopedia. *Directory Service*. http://www.webopedia.com/TERM/D/directory_service.html. [Online; accessed 3-Mar-2016].

Zeilenga, K. (2006a). *Lightweight Directory Access Protocol (LDAP): Directory Information Models*. RFC 4512. RFC Editor. URL: http://www.rfc-editor.org/rfc/rfc4512.txt.

— (2006b). *Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map*. RFC 4510. RFC Editor. URL: http://www.rfc-editor.org/rfc/rfc4510.txt.