Université de Montréal

SLA Violation Prediction: A Machine Learning Perspective

par Reyhane Askari Hemmat

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

Octobre, 2016

Résumé

Le cloud computing réduit les coûts de maintenance des services et permet aux utilisateurs d'accéder à la demande aux services sans devoir être impliqués dans des détails techniques d'implémentation. Le lien entre un fournisseur de services cloud et un client est régi par une *Validation du Niveau Service* (VNS) qui définit pour chaque service le niveau et le coût associé. La VNS contient habituellement des paramètres spécifiques et un niveau minimum de qualité pour chaque élément du service qui est négocié entre les deux parties.

Cependant, une ou plusieurs des conditions convenues dans une VNS pourraient être violées en raison de plusieurs problèmes tels que des problèmes techniques occasionnels. Du point de vue d'apprentissage automatique, le problème de la prédiction de violation de la VNS équivaut à un problème de classification binaire.

Nous avons exploré deux modèles de classification en apprentissage automatique lors de cette thèse. Il s'agit des modèles de classification de Bayes naïve et de Forêts Aléatoires afin de prédire des violations futures d'une certaine tâche utilisant ses traits caractéristiques. Comparativement aux travaux précédents sur la prédiction d'une violation de la VNS, nos modèles ont été entraînés sur des ensembles de données réels introduisant ainsi de nouveaux défis. Nous avons validé le tout en utilisant *Google Cloud Cluster trace* comme avec l'ensemble de données.

Les violations de la VNS étant des évènements rares ($\sim 2.2\%$), leur classification automatique reste une tâche difficile. Un modèle de classification aura en effet une forte tendance à prédire la classe dominante au détriment des classes rares. Pour répondre à ce problème, il existe plusieurs méthodes de ré-échantillonages telles que *Random Over-Sampling, Under-Sampling, SMOTH, NearMiss, One-sided Selection, Neighborhood Cleaning Rule.* Il est donc possible de les combiner afin de ré-équilibrer le jeu de données.

Mots clés: Cloud Computing, Validation du Niveau Service, Apprentissage Automatique, Classification Déséquilibrée, Forêts Aléatoires, Classification de Bayes Naïve

Summary

Cloud computing reduces the maintenance costs of services and allows users to access on demand services without being involved in technical implementation details. The relationship between a cloud provider and a customer is governed with a *Service Level Agreement* (SLA) that is established to define the level of the service and its associated costs. SLA usually contains specific parameters and a minimum level of quality for each element of the service that is negotiated between a cloud provider and a customer.

However, one or more than one of the agreed terms in an SLA might be violated due to several issues such as occasional technical problems. Violations do happen in real world. In terms of availability, Amazon Elastic Cloud faced an outage in 2011 when it crashed and many large customers such as Reddit and Quora were down for more than one day. As SLA violation prediction benefits both user and cloud provider, in recent years, cloud researchers have started investigating models that are capable of prediction future violations. From a Machine Learning point of view, the problem of SLA violation prediction amounts to a binary classification problem.

In this thesis, we explore two Machine Learning classification models: Naive Bayes and Random Forest to predict future violations using features of a submitted task. Unlike previous works on SLA violation prediction or avoidance, our models are trained on a real world dataset which introduces new challenges. We validate our models using *Google Cloud Cluster trace* as the dataset.

Since SLA violations are rare events in real world ($\sim 2.2\%$), the classification task becomes more challenging because the classifier will always have the tendency to predict the dominant class. In order to overcome this issue, we use several re-sampling methods such as *Random Over-Sampling, Under-Sampling, SMOTH, NearMiss, One-sided Selection, Neighborhood Cleaning Rule* and an ensemble of them to re-balance the dataset.

Keywords: Cloud Computing, Service Level Agreements, Machine Learning, Unbalanced Classification, Random Forest, Naive Bayes

Contents

List of Figures

List of Tables

List of Abbreviations

AWS	Amazon Web Services
BaaS	Backend as a Service
DaaS	Data as a Service
ERD	Entity Relationship Diagram
IaaS	Infrastructure as a Service
MSE	Mean Square Error
NCR	Neighborhood Cleaning Rule
NIST	National Institute of Standards and Technology
NaaS	Network as a Service
PGA	Parallel Genetic Algorithm
PaaS	Platform as a Service
QoS	Quality of Service
ROC	Receiver Operating Characteristics
SECaaS	Security as a Service
SLA	Service-Level Agreement
SLO	Service Level Objectives
SMOTE	Synthetic Minority Over-sampling Technique
STaaS	Storage as a Service
SaaS	Software as a Service
T-SNE	T-Distributed Stochastic Neighbor Embedding
VM	Virtual Machines
WS	Web Server
WSLA	Web Service Level Agreement

Acknowledgments

For the ancestors who paved the path before me, upon whose shoulders I stand. This is dedicated to my parents Ataollah and Zahra and to my brothers Mohammad Ali and Mohammad Hossein.

I would like to express my deepest gratitude to my supervisor Prof. Abdelhakim Hafid for his unwavering support, collegiality, and mentorship throughout this thesis.

I must also acknowledge great help and support from my friend Mohammad Pezeshki.

1 Introduction

1.1 Motivation and Statement of the Problem

Cloud computing provides a convenient way to access different IT resources such as servers, storage, databases and a wide range of application services over the Internet. The main appeal of cloud computing is that users do not get involved in details of service management, hardware maintenance, or software licenses.

In recent years, cloud computing is becoming the most cost-effective and reliable way of building and deploying different IT services. The superiority of cloud computing comes from the fact that it provides extensive computing and storage services on scalable and dynamic environment. According to Wang et al. (2010), cloud computing has five unique characteristics among other computing paradigms; (1) User-centric interfaces: cloud computing is accessed using simple and user-friendly environments, (2) On-demand service provisioning: based on user requirements of a service, different amounts of resources can be allocated, (3) QoS guaranteed offer: cloud computing guarantees a minimum level of *Quality of Service* (QoS), based of a *Service Level Agreement* (SLA), (4) Autonomous System: system management including both hardware and software are all done autonomously without involving users, and (5) Scalability and flexibility: cloud computing allows upscaling or downscaling IT resources easily.

Among the above-mentioned cloud characteristics, in this thesis we particularly, focus on the Quality of Service and Service Level agreements in cloud computing. "*Quality of service represents the set of those quantitative and qualitative characteristics of a distributed multimedia system necessary to achieve the required functionality of an application*" (Vogel et al., 1995). In order to guarantee a minimum level of QoS, a careful management of IT resources is essential. However, due to systems' complexities, the task of managing resources in an efficient way is a challenging problem. Management of resources and handling variable volumes of user requirements are a part of SLA between users and cloud providers.

QoS management involves helping users to find the required characteristics of the demanded service and adaptation of IT resources in such a way to respect SLA and to optimize the system performance and efficiency. Generally speaking, the problem of resource adaptation including resource reallocation in a complex system with an enormous number of tasks is an NP-hard problem (Darmann et al., 2010). Consequently, it is inevitable that QoS agreed in SLA not be always respected. In the case that the effective QoS does not comply with the minimum QoS agreed in SLA, QoS manager issues an instance of *SLA violation*.

QoS manager allocates different amounts of resources (CPU, memory, or storage) and also determines the agreements in SLA based on four sources of information: (1) The requested IT resources for each user task, (2) The available resources of the computing system, (3) Information about the minimum QoS agreed in SLA, and (4) The historical information about the system's load. QoS manager, usually using a heuristic method, decides how to prevent SLA violation. For example, in the application of video streaming such as YouTube, QoS manager may delay the video by a few seconds in order to buffer and prevent interruption in the middle of video. On the other hand, in some other applications such as video conference of Google Hangouts, in which significant delay is not acceptable, QoS manager may reduce the resolution of video or the sound quality to prevent any violation of the service. Therefore, it is desirable to be able to predict when an SLA violation may occur beforehand.

SLA violation prediction benefits both cloud providers and customers. From a cloud provider's point of view, SLA violation results in paying penalties in terms of both money and reputation. By predicting violations ahead of time, providers can reallocate the requests and resources to prevent future violations. All the process of resource allocation is done behind the scene; thus, from a customer point of view, better resource allocation results in a trustworthy provider. Moreover, customers would like to receive the service on demand and without any interruptions. Thus, a system in which a cloud provider or a third party could provide the prediction of SLA violations for the customer can be very insightful.

It is worth mentioning that violations do happen in the real world. As an example, Amazon Elastic Cloud faced an outage in 2011 when it crashed and many large customers such as Reddit and Quora were down for more than one day[1].

1.2 Contributions

In this thesis, we propose to use Machine Learning in order to predict SLA violations. Violation prediction can be seen as a classification problem in the terminology of Machine Learning. A classifier predicts whether a coming request will be violated or not. Each request is presented to the model using five different features: the priority of the task, the requested amount of disk space, the requested amount of CPU, the requested amount of memory, and also scheduling class which indicates latency-sensitivity of the task. We explore Random Forest and Naive Bayes classifiers. For the Naive Bayes we also explore two assumptions over the features vector: Bernoulli and Gaussian distributions.

Previous research mostly relies on heuristic methods for prediction of violations. Although Machine Learning has been used in different areas of QoS management, the experiments are done mostly in very restricted setting which is not necessarily scalable to real world data. However, this research takes a systematic machine learning approach applied on real-world data that provides an insightful set of experiments. We use 20k records of *Google Cloud Cluster trace* dataset containing $\sim$ 97.8% unviolated and $\sim$ 2.2% violated examples. Thus, the dataset is highly unbalanced and the classification task becomes more challenging because the classifier will always have the tendency to predict the dominant class. This problem usually biases the classifier to always predict no violation which is not desirable. We address this issue by using multiple classifiers aggregated and averaged in order to achieve a single reliable result. Specifically, in terms of algorithm, we use *random forest* classification model and in terms of data, we use different re-sampling methods.

We show that our proposed model achieves a remarkable performance of 99.88% accuracy[2] in prediction of violations. In addition, by analyzing the model and visualization of different re-sampling methods, we provide insightful and actionable

1. Amazon Elastic Compute Cloud (Amazon EC2). Available at https://aws.amazon.com/ec2/
2. Full table of results including other metrics is presented in Section 6.2

information on how to overcome the skewness of the dataset and train unbiased classification models. It is also worth mentioning that we extract human-interpretable results from the model which suggests that *requested memory* is the most correlated feature with violation occurrence. Finding the importance of each feature can then help the provider to implement a management system that can improve the performance of its cloud services.

1.3 Organization of this Thesis

In Chapter 2, we introduce fundamentals of Cloud Computing, its architecture, cloud's service model and deployment models. We will also give a brief definition of Service Level Agreement and Quality of Service. In Chapters 3, we give a description of the terminologies and basic concepts in machine learning. We define different machine learning models such as classification and regression. We also present how the performance of a model is measured in machine learning.

In Chapter 4, we present an overview of existing contributions on SLA violation prediction; in particular, we present the limitations of these contributions and how our proposed model aims to overcome them. Chapter 5 presents the proposed method that is used to predict SLA violation. Chapter 6 presents the details of the evaluation and the implementation of our proposal. Finally, Chapter 7 concludes the thesis and presents future work.

2 Cloud Computing

Cloud computing is a new paradigm for providing various hosting services over the internet. It has recently become so prevalent that it is hard to picture using many services and applications without cloud computing. There has been many driving forces that has led to the popularity and advancement of cloud computing. Computing power and storage have had rapid development and the hardware cost has been decreased. The emergence of exponentially growing data, the necessity of a new business model and technology led to the concept of cloud computing. See Figure 2.1 for a graphical illustration of a cloud system.

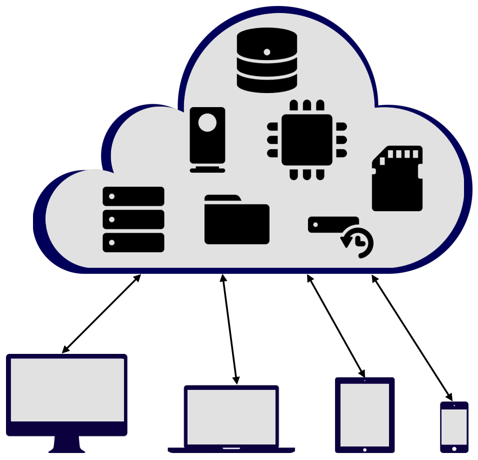


Figure 2.1 – A depiction of a cloud system.

Cloud computing reduces the maintenance costs of the service and also allows users to access on demand services without being involved in technical implementation details. Business owners do not need to plan ahead for provisioning and enterprises can start small and increase the resources as they grow.

According to the National Institute of Standards and Technology (NIST) (Mell and Grance, 2011), the definition of cloud computing is: *Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services)* *that can be rapidly provisioned and released with minimal management effort or service provider interaction.*

In this chapter, we present the architecture of cloud and its service and deployment models. Then cloud computing's characteristics are presented and related technologies such as grid computing, utility computing and autonomic computing are briefly discussed. Finally, we will introduce the concept of Quality of Service and Service Level Agreements in cloud.

2.1 Cloud Architecture and Layered Model

To better understand cloud computing, we need to first describe the architecture of cloud. Cloud architecture is usually defined in a layered model. Modular and layered structures such as OSI model and cloud layered architecture simplify the separation and definition of different parts of the system and reduce management overheard.

The architecture of cloud consists of four layers (Zhang et al., 2010): Hardware, Infrastructure, Platform and Application. We briefly discuss these layers in the following sections.

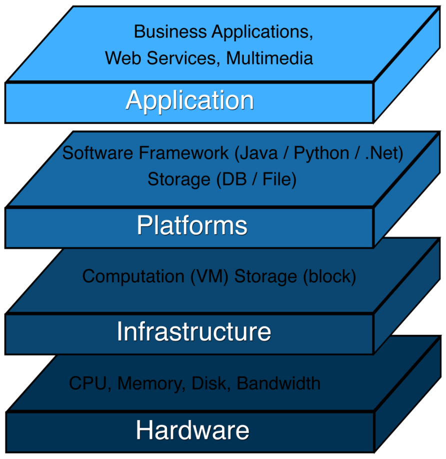


Figure 2.2 – Cloud architecture and layered model: Hardware, Infrastructure, Platform and Application layers.

2.1.1 Hardware Layer

This layer mostly includes the physical hardware that actually runs the cloud. It is usually a networked collection of data centers connected through switches and routers. Inside data centers are racks of servers, storage arrays, cooling infrastructure, power converters and backup generators (Zhang et al., 2010). Fault tolerance in this layer is managed via redundancy of several inexpensive physical hardware.

2.1.2 Infrastructure Layer

This layer is the foundation of Cloud Computing. It provides the virtualization technology that makes cloud flexible and scalable. Virtual machines (VMs) are deployed on hardware with different operating systems. A virtual machine creates logic structures that seem to operate just like the physical machine. Virtual machines are created and deleted at will which enables users to have dynamic resource allocation and maximum resource utilization.

2.1.3 Platform Layer

This layer includes operating systems and web platforms on top of the infrastructure layer. It provides a container for application development or APIs for cloud application development without the need to manage the hardware, virtual machines or operating systems. The cloud platform acts as a container where web applications with storage and database can be created.

2.1.4 Application Layer

The application layer is the most visible layer to end-users. Applications are accessed by users though a web portal. Cloud applications do not require the user to handle software upgrades and patches. Applications at this level are fast at processing real time data and are highly scalable.

2.2 Cloud Deployment Models

According to the NIST (Mell and Grance, 2011) definition of cloud, there are four cloud deployment models known as public, private, community and hybrid clouds. These four deployment models specify who is the customer of the services that are provided by these clouds.

2.2.1 Public Cloud

The services provided in a public cloud are open to use for general public, meaning customers that are external to the provider's organization. The services can be free or in a *pay as you go* manner. The service can be sold, managed and operated by end-users, businesses or organizations. Public cloud service providers usually own and operate the infrastructure at their data center and access is generally, provided via the Internet.

2.2.2 Private Cloud

Private clouds are owned or leased by one large or mid size organization. They can be hosted externally or internally. The services are not in a pay as you go manner because the hardware, storage, network and the whole infrastructure is dedicated to the organization.

Security is a key aspect in private clouds (Mell and Grance, 2011). The usage of dedicated hardware, storage and network can ensure higher levels of security.

2.2.3 Community Cloud

Community clouds have shared infrastructures for a specific community that share a common goal (security, compliance, jurisdiction, etc.). They can be managed internally or by a third party and are hosted either internally or externally. Compared to a private cloud the costs can be shared in a community cloud and the services are provided in a pay as you go manner. On the other hand, compared to public cloud, a higher level of security that is more compatible with organizations is provided.

2.2.4 Hybrid Cloud

In hybrid clouds, the infrastructure is composed of two or more other cloud models (private, public or community) that will be separated from each other but bounded with a standard technology. They provide multiple benefits of different deployment models. Organizations can use a hybrid of public and private clouds to store sensitive information in a private cloud connected to an application that is deployed in a public cloud.

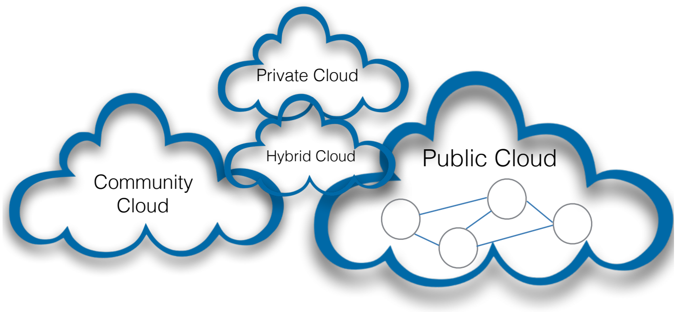


Figure 2.3 – Hybrid Cloud uses the infrastructure of two or more of the Public, Private or Community clouds.

2.3 Cloud Computing Characteristics

According to NIST definition of cloud computing, a cloud system has five essential characteristics; *on demand self-service, broad network access, resource pooling, rapid elasticity* and *measured service* (Mell and Grance, 2011). There are also several common characteristics in cloud systems; *service oriented*, *multi-tenancy* and *geographic distribution*, to name a few. We briefly discuss each of these characteristics.

2.3.1 On Demand Self-service

On demand self service or automation is a property that enables customers to perform all actions needed to acquire or release a service without any human interaction and in a pay as you go manner. The transition usually takes place immediately, although depending on the architecture and the resource availability of the provider it may be delayed (Zhang et al., 2010).

A customer does not need to have huge investment in a service from the start and can scale the required service up to a significant level without any disruption on host operations. Moreover, the traditional provisioning model for resources was based on the peak demand of the service whereas in a cloud system, resources are acquired on-demand which can considerably lower the costs. The customer is charged only for the resources used under a subscription-based billing method.

2.3.2 Broad Network Access

A cloud system should be accessible over a network. This characteristic is called *Broad Network Access*. Computing capabilities in a cloud system are available from a wide range of locations over the network and accessed through standard mechanisms.

Comparing to the mainframe era when resources were scarce and costly, nowadays, broad network access of cloud systems has become possible (Williams, 2012). The reason is that the network bandwidth and access has increased and also the cost associated with networks has decreased.

2.3.3 Resource Pooling

In a cloud system, providers offer a pool of computing, network, storage and services to various costumers. Providers have the flexibility of dynamically assigning the pooled resources and services to multiple customers. The customers will share resources adjacent to other customers (Zhang et al., 2010). This characteristic allows providers to manage their resources by maximizing resource utilization and minimizing the operating costs, power consumption and cooling. This leads to offering resources with considerable lower prices.

2.3.4 Rapid Elasticity

An elastic system can adapt to workload changes by provisioning and deprovisioning the resources in an autonomic way. Rapid elasticity is the ability of fast provision and deprovision of resources such that available resources match the current requested resources as much as possible.

In a cloud system, provision of resources can be so quick that the resources can appear unlimited to the customer and provision can be performed in any quantity at any time.

2.3.5 Measured Service

In a cloud system, resource usage is automatically monitored, controlled and reported by using metering capabilities with some level of abstraction. The measurement tools provide both the provider and the customer an account of what has been used. This allows transparency between the provider and the customer.

2.3.6 Service Oriented

Cloud systems operate with a service-driven model. In such systems, service management and preferably autonomous service management are key aspects. Each provider offers the resources under an agreement called *Service Level Agreement* (SLA) (Casalicchio and Silvestri, 2013). SLA can be negotiated between a provider and a customer and is an essential part of cloud systems to ensure maximum availability of services for customers. With a violation of SLA, the provider has to pay penalties, thus SLA assurance is a critical objective for every provider.

2.3.7 Multi-tenancy

Multi-tenancy is a property that allows a system to serve a single instance of a resource or application for multiple customers which are called tenants in this case (Krebs et al., 2012). This system requires secure physical and logical separation of resources which are controlled by one tenant in a shared environment. It is worth mentioning, that per-tenant reporting and quota management are key aspects in a multi-tenant system (Krebs et al., 2012).

The layered structure of cloud allows adding new features to the system by changing the entire infrastructure once for all customers whereas in a dedicated hardware per customer environment, changes need to be done per device (AlJahdali et al., 2014).

2.3.8 Geographic Distribution

As mentioned previously, one of the main characteristics of cloud systems is broad network access. Hence, to achieve higher performance, localization can be used. Many of cloud providers have data centers distributed around the world in order to gain the maximum service utility (Attiya and Welch, 2004).

2.4 Related Technologies

Several technologies share close characteristics with cloud computing. We will briefly describe *Grid Computing, Utility Computing* and *Autonomic Computing* and discuss their common characteristics with *Cloud Computing*.

2.4.1 Grid Computing

According to Foster et al. (2008), grid computing is a resource provisioning model where computing resources are distributed. Usually large number of servers (nodes) are connected to each other through a network to form a grid. Large and computing-intensive workloads are sent to grid which are then shared between nodes in a paralleled manner. Thus, it requires software that can divide the computation task into pieces. This rises the concern of software management or handling failure of a node.

Scalability and multi-tenancy are the shared features between cloud and grid computing. However, in cloud computing the resources are allocated and deallocated on demand and at a more granular level by using virtualization (Foster et al., 2008). Virtualization amounts to running multiple operating systems and application on a single physical server. Cloud and grid computing both offer SLAs to provide resources for a guaranteed uptime. Storage computing in a grid is designed for data-intensive tasks. It is not financially preferred for storing small objects, but cloud computing can be implemented in non-grid environments (Foster et al., 2008).

2.4.2 Utility Computing

According to Rappa (2004), *Utility Computing* is "the on-demand delivery of infrastructure, applications, and business processes in a security-rich, shared, scalable, and standards based computer environment over the Internet for a fee".

Utility computing is mostly used in cases where the peak of usage is rare. By using virtualization, it tries to minimize the operating costs while maximizing the actual usage of resources. Thus, the main leverage of utility computing is the economics (Degabriele and Pym, 2007).

Utility computing can be considered as the backbone of cloud computing but cloud computing provides more features and flexibility (Kaur and Singh, 2015). Cloud computing can be used internally in an organization. It also has unlimited scalability, utility based pricing, network access and on-demand self service.

2.4.3 Autonomic Computing

Autonomic computing is the self-managing feature of distributed computing. In autonomic computing, the control of the system, application and resources are done without human interaction thus removing the burden of management. The system adapts to unpredictable changes and hides the system's complexity from the users (Kephart and Chess, 2003).

Autonomic computing reduces maintenance costs by hiding the system complexity. The autonomic feature is also included in cloud computing but with a different purpose. Autonomic resource allocation and SLA monitoring are two features of cloud computing but they aim to reduce the cost and maximize the resource utilization rather than reducing the complexity (Zhang et al., 2010).

2.5 Service Model

The service model architecture of cloud computing suggests that the resources provisioned in cloud be provided as services in an on-demand fashion. These services are usually grouped into three categories; *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS), *Software as a Service* (SaaS). Each layer can be used as a service provider for the upper layer (Mell and Grance, 2011).

There are also other less popular service models for cloud such as; *Network as a Service* (NaaS), *Storage as a Service* (STaas), *Security as a Service* (SECaaS), *Data as a Service* (DaaS), *Backend as a Service* (BaaS) and etc. or the generally as called in Banerjee et al. (2011), *Everything as a Service* that are provided by cloud providers at different layers.

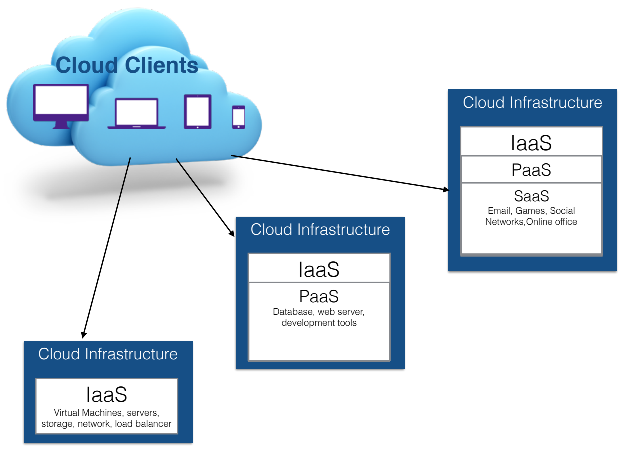

Cloud Clients
Cloud Infrastructure
IaaS
PaaS
SaaS
Email, Games, Social Networks,Online office
Cloud Infrastructure
IaaS
PaaS
Database, web server, development tools
Cloud Infrastructure
IaaS
Virtual Machines, servers, storage, network, load balancer

Figure 2.4 – Cloud Service Model: Three well-know cloud service model are IaaS, PaaS and SaaS providers.

2.5.1 Infrastructure as a Service (IaaS)

In the IaaS model, the infrastructure provider leases disk, CPU, memory, hardware, network and other infrastructure components to an end-user or other cloud providers. An IaaS provider handles tasks including backup, maintenance and resiliency planning. Also, the administrative tasks are automated. The user has the ability to dynamically scale using virtualization technology but has no control over operating systems and applications that are running on the server (Mell and Grance, 2011). Famous IaaS providers are Amazon Web Services (AWS), Windows Azure, Google Compute Engine and Rackspace Open Cloud.

2.5.2 Platform as a Service (PaaS)

In the PaaS model, the platform provider can rent resources from an infrastructure provider or use its own infrastructure to deliver hardware and software tools that are usually used for application development (Mell and Grance, 2011). Thus, the platform is used by end-users or software providers for deployment of software applications without the cost and complexity of acquiring and managing the underlying hardware and software layers. A user in PaaS can upgrade operating system features but has no control over the underlying cloud infrastructure such as operating systems, disk, CPU, memory, hardware and network. Famous PaaS providers are Appear IQ, Mendix, Amazon Web Services (AWS) Elastic Beanstalk, Google App Engine and Heroku.

2.5.3 Software as a Service (SaaS)

In the SaaS model, the software provider uses a platform or an infrastructure provider or uses its own infrastructure to deliver a software to end-users. SaaS removes the burden of maintenance, installation, acquisition, provisioning and licensing of software (Mell and Grance, 2011). Organizations or individuals do not need to install the software on their data centers or on their computers. Instead, the software is available through a web browser or an API. The user has no control over the underlying infrastructure that is running the software such as operating systems, disk, CPU, memory, hardware, network. Most of the time, end-users do not even need to configure the software. Famous SaaS providers are Salesforce, Oracle, SAP, Intuit and Microsoft.

2.6 Quality of Service in Cloud Computing

"*Quality of service represents the set of those quantitative and qualitative characteristics of a distributed multimedia system necessary to achieve the required functionality of an application*" (Vogel et al., 1995). By measuring the QoS of a system, the performance can be improved and guaranteed in advance. Therefore, QoS measurement increases the reliability and availability of the system. In cloud systems, QoS is an essential aspect as cloud customers would like to have a measure of the cloud's performance and a cloud provider would like to find the best trade off between the provided service and the cost. In the infrastructure level of cloud computing, there are several QoS parameters that can be measured (Meegan et al., 2012):

— **Compute**: availability, outage length, server reboot time.
— **Network**: availability, packet loss, bandwidth, latency, mean/max jitter.
— **Storage**: availability, input/output per second, max restore time, processing time, latency with internal compute resource.

Cloud providers guarantee the QoS with *Service Level Agreements* (SLAs). Also, *Service Level Objectives* (SLOs) are given as quantitative or qualitative parameters of an SLA such as throughput, availability and response time (Sturm et al., 2000).

We discuss the definition of SLA and SLA management and life cycle in the following sections.

2.6.1 Service Level Agreements

The relationship between a cloud provider and a customer is governed with a *Service Level Agreement* (SLA). SLA is negotiated between parties and a level of the service, QoS and its associated costs are agreed upon. SLA usually contains specific parameters and a minimum level of quality for each element of the service that is negotiated between the provider and the customer (Casalicchio and Silvestri, 2013). A common framework for SLA definition is *web service-level agreement* (WSLA) (Ludwig et al., 2003). Components of a WSLA is shown in Table 2.1.

From an application hosting point of view, SLA has two different types: Infrastructure SLA and Application SLA. Infrastructure SLA guarantees a level of reliability on infrastructures such as power, data center, latency and etc. by dedicating resources solely to the customer. An example is shown in Table 2.2. Application SLA is appropriate for hosting models on which multiple applications are co-located. In such a setting, cloud resources are available to applications according the application demands. Hence, in application SLA, cloud providers ensure meeting application demands. An example of application SLA is shown in Table 2.3.

For example, SLA can indicate a 99.99 % availability for requests of CPU, disk and memory. An SLA might also contain constraints on the response time for each request.

SLA is an important part of each contract because a provider would like to

Table 2.1 – Components of a Web Service Level Agreement (Buyya et al., 2010).

Service-Level Parameter	Describes an observable property of a service whose value is measurable.
Metrics	These are definitions of values of service properties that are measured from a service-providing system or computed from other metrics and constants. Metrics are the key instrument to describe exactly what SLA parameters mean by specifying how to measure or compute the parameter values.
Function	A function specifies how to compute a metric's value from the values of other metrics and constants. Functions are central to describing exactly how SLA parameters are computed from resource metrics.
Measurement directives	These specify how to measure a metric.

Table 2.2 – An example of infrastructure SLA (Buyya et al., 2010).

Hardware availability	99 % uptime in a calendar month.
Power availability	99.99 % of the time in a calendar month.
Data center network availability	99.99 % of the time in a calendar month.
Backbone network availability	99.999 % of the time in a calendar month.
Service credit for unavailability	Refund of service credit prorated on downtime period.
Outage notification guarantee	Notification of customer within 1 hr of complete downtime.
Internet latency guarantee	When latency is measured at 5-min intervals to an upstream provider, the average doesn't exceed 60 msec.
Packet loss guarantee	Shall not exceed 1 % in a calendar month.

allocate the least amount of resources for each customer to reduce the cost of its server infrastructure. At the same time, the provider needs to avoid having

Table 2.3 – An example of application SLA (Buyya et al., 2010).

Service-level parameter metric	• Website response time (e.g., max of 3.5 sec per user request).
Function	• Latency of web server (WS) (e.g., max of 0.2 sec per request). • Latency of DB (e.g., max of 0.5 sec per query) • Average latency of WS = (latency of web server 1 + latency of web server 2) /2 • Website response time = Average latency of web server + latency of database
Measurement directive	• DB latency available via http://mgmtserver/em/latency. WS latency available via http://mgmtserver/ws/instanceno/latency
Service-level objective	• Service assurance.
Penalty	• Website latency < 1 sec when concurrent connection < 1000. • 1000 USD for every minute while the SLO was breached.

penalties due to failure of providing the agreed service. The failure of providing a service is called an *SLA violation*. The customer would like to receive the service on demand and without any interruptions. Despite these high availability rates, violations do happen in real world and have caused both the provider and the customer heavy costs (Leavitt, 2009).

2.6.2 SLA Management Life Cycle

According to Gallizo et al. (2009) SLA management has a life cycle of six phases:

— SLA Contract Definition

— Basic Schema with the Quality of Service (QoS) Parameters

— SLA Negotiation

— SLA Monitoring

— SLA Violation Detection

— SLA Enforcement

We will briefly describe these phases in the following subsections. Figure 2.5 illustrates this life cycle.

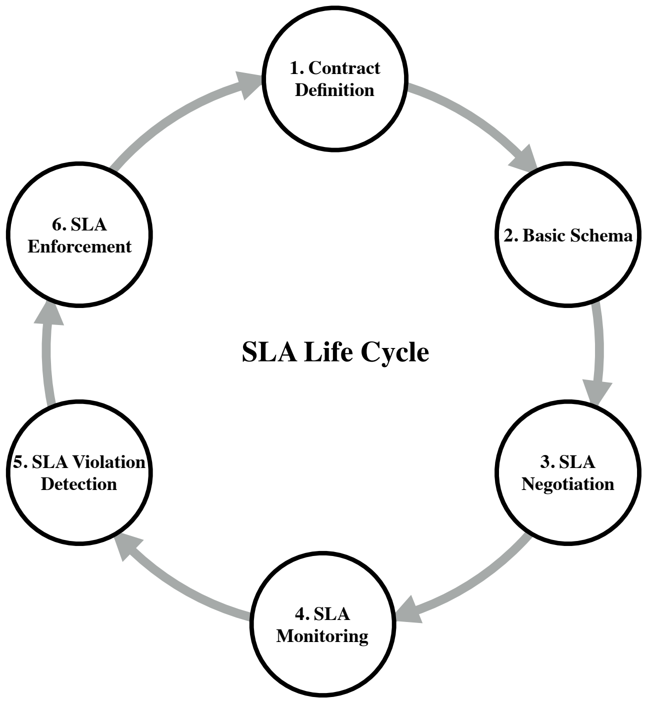


Figure 2.5 – SLA Life Cycle.

SLA Contract Definition

In this phase, the service and its corresponding price, QoS parameters with a basic schema and also the penalty policy is defined. SLAs are usually defined using standard or base templates or by customization of these base templates.

Basic Schema with the Quality of Service (QoS) Parameters

QoS parameters must be included in an SLA, covering different types of (virtualized) physical resources (e.g. for the network resources QoS parameters may be bandwidth, jitter, delay; for the computing resources the parameters may be CPU, memory, etc).

SLA Negotiation

In this phase a customer discovers a service provider that meets the customer's needs. The terms and conditions of the SLA are negotiated and agreed upon in this phase. A cloud provider needs to analyze the SLA in terms of scalability, availability and performance of its services in order to avoid penalties before agreeing on the specification of SLA. By the end of this phase, parties start to commit to the agreement.

SLA Monitoring

In this phase, the provider's performance in delivery of the service is measured against the contract. An essential part of SLA monitoring is to be able to predict violations enabling providers to reallocate the resources accordingly before occurrence of violations.

SLA Violation Detection

In this phase the parameters inside SLA are calculated and any deviation is determined. In case of SLA violation, SLA enforcement is conducted.

SLA Enforcement

This phase is to enforce penalties for SLA violation. In this phase appropriate actions are taken place when the violation has been detected in the previous phase. The concerning parties are notified and penalty charges are taken place. After SLA enforcement, SLA might also terminate due to timeout or violation.

3 Prediction Models

Machine learning is the study and development of programs and algorithms that can learn from historical data and make prediction when exposed to new data. There are three general types of algorithms that are used to solve different problems in machine learning: supervised learning algorithms, unsupervised learning algorithms and reinforcement learning (Murphy, 2012).

— **Supervised Learning** aims to find a function that maps the input to the output given a labeled dataset[1].

— **Unsupervised Learning** aims to find the structures and patterns inside the input given an unlabeled dataset.

— **Reinforcement Learning** aims to find a function that outputs a sequence of actions that optimizes costs or rewards.

The focus of this thesis is on supervised learning. Consequently, after a review of some terminologies in machine learning, *Supervised Learning* is introduced in more details. Next, key concepts in machine learning such as *Generalization, Bias-Variance Trade Off, Overfitting, Regularization,* and *Cross Validation* are presented. Finally, we discuss how a model is evaluated in machine learning and specifically discuss *Confusion Matrix, Accuracy, Precision and Recall, F_β and ROC curves.*

3.1 Terminology

In this section, we introduce the basic terminology of machine learning which is used in the rest of this chapter. In a typical machine learning supervised task, a *dataset* is given in a set of rows and columns. Each row of the dataset corresponds to a single *datapoint* which is called a *training example* or a *training instance.*

1. In machine learning terminology the output variables or the targets are sometimes referred as labels. Thus, a dataset with inputs and their desired outputs is called a labeled dataset.

The columns are called *input variables*, *features* or *attributes*. Each datapoint is associated with one or more than one *label(s)*, *targets*, or *output variables*.

The dataset is typically splitted into two sets; *training set* and *test set*. The training set is used to learn the underlying factors of variation in data, while the test set is used for the final evaluation. First, the model is trained given the training set and during testing, an example represented by its features is provided to the model and the output is the predicted label.

3.2 Supervised Machine Learning: Concepts and Definitions

In supervised machine learning, two pieces of information are provided to the algorithm: a set of input instances $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m\}$ and a corresponding set of targets $Y = \{y_1, y_2, ..., y_m\}$. Typically, each of these m input instances contains a set of n features $\mathbf{x} = \{x_1, x_2, ..., x_n\}$. Generally speaking, each feature x_i can take any value, either numerical (values are real numbers) or categorical (values are members of an unordered set). However, depending on the task at hand, features may be required to be converted to certain types.

There is always a true function $f^*(.)$ that maps any possible $\mathbf{x}$ into the best possible y. However, we never have access to this unknown function. Accordingly, supervised learning amounts to approximating the function $f^*(.)$ based on the information provided in the X and Y sets. The process of approximating $f^*(.)$ using a function $f_\theta(.)$ in which θ is a set of parameters is called *learning*.

Learning algorithms learn the parameters θ of the function $f_\theta(.)$ by minimizing the errors that the model makes. Formally, a function that maps the discrepancy between the output prediction of the model and the true target into a real number is called the *loss function* (Murphy, 2012).

If the true target y is a discrete variable, the prediction task is called *Classification*. On the other hand, if y is continuous, the task is called *Regression*. In the following subsections, after formally introducing learning, we discuss these two types of supervised learning algorithms in more details.

3.2.1 Learning

Approximating function $f^*(.)$ using function $f_\theta(.)$ corresponds to extracting the underlying factors of variation from data instances and mapping them to the output. These underlying factors could be a table of probabilities, a structure of a graph, or weights depending on which learning algorithm is used for knowledge discovery. Generally, *learning* amounts to finding the best parameters θ in order to minimize a loss function over all the examples in the dataset (Murphy, 2012). Therefore, the learning process can be formulated as follows,

$$\hat{\theta} = argmin_\theta\{\sum_{i=1}^{m} l(y_i, o_i; \theta)\}, \tag{3.1}$$

in which $\hat{\theta}$ is the *learned* set of parameters, y_i and o_i are the target and output of the model for the i^{th} sample.

3.2.2 Classification

In a supervised classification task, the prediction output y is from one of the total C distinct classes $\{1, 2, ..., C\}$. In order to get prediction for new examples, the model can simply output a class label or the output can be a set of probabilities. Each probability corresponds to one of C classes that indicates how probable it is that the unseen input $\mathbf{x}$ belongs to a specific class. In models that output probabilities, to get a discrete prediction out of the model, either the class with the highest probability is chosen or the class label is drawn by sampling from the output distribution.

3.2.3 Regression

Similar to a classification task, in regression problems, the goal is to learn a mapping function from an n-dimensional vector $\mathbf{x}$ into a real-valued number o as the prediction. *Mean Square Error* (MSE) is a common loss function used to measure the performance of regression models. Consequently, learning amounts to reducing the MSE between the model prediction and the true target which is defined as follows,

$$\mathrm{MSE}(O,Y) = \sum_{i=1}^{n} ||o_i - y_i||_F^2 \tag{3.2}$$

The parameters of the model are then selected such that MSE is minimized.

3.3 Generalization

The goal of machine learning is to train models that are able to predict the labels for new unseen examples. As a result, generalization to new examples is an important side of each learning algorithm. We usually look for models that perform well on testing data as well as on training data. As a result, we need to prevent learning algorithms from simply memorizing training data; instead, these algorithms need to learn the underlying factors of variation.

3.3.1 Bias-Variance Trade off

In order to determine how accurate a model is, we need to understand what are the reasons behind errors. Bias and variance of a prediction model help us formally measure these errors. To define bias and variance over a model, we need to assume that we are able to train the same model multiple times with different randomly selected data points. In this thesis, each trained model is called a *model instance.* Errors in predictions that are caused by bias and variance are called *error due to bias* and *error due to variance* respectively (Sammut and Webb, 2011) (Geurts, 2002).

Bias corresponds to the distance between the expected prediction of the model and the true target (Wasserman, 2013). Considering $f(x)$ as the model, the bias is defined as follows:

$$\text{bias} = |E[f(x)] - y|^2, \tag{3.3}$$

where $E[.]$ is the expectation and y is the true target. On the other hand, variance corresponds to the variability in different predictions of multiple instances of a model (Wasserman, 2013):

$$\text{variance} = |f(x) - E[f(x)]|^2. \tag{3.4}$$

The total error of a model in terms of bias and variance is defined as follows:

$$\text{error} = E[(f(x) - y)^2] = \text{bias}^2 + \text{variance}. \tag{3.5}$$

Given the limited amount of data, there is always a trade-off between bias and variance. The trade-off happens in a way that reducing one may lead to increasing the other. As a result, minimizing the total error requires a careful balance between bias and variance. A graphical illustration of this trade-off is shown in Figure 3.1.

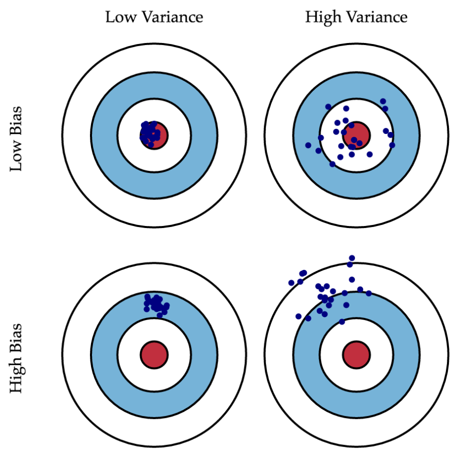


Figure 3.1 – Dart chart: A graphical illustration of bias-variance trade-off. Consider a classification problem as throwing darts at a dart-board. If darts land in very different parts of the board, the model has "high variance". If their mean is close to the center of the board, the model has "low bias". Similarly, "low variance" and "high bias" can be defined. The above four dart boards corresponds to these situations (Moore and McCabe, 1989).

3.3.2 Overfitting Problem

Overfitting is the case when a prediction model performs very good on the training data but achieves a significantly lower performance on the test data. This usually means that the model has memorized the whole training data including the noises instead of the underlying factors of variation that are essential for generalization. Overfitting could have different reasons including having small amount of data or too many model parameters (Bishop, 2006). Another reason for the problem of overfitting is unbalanced data which is one of the important aspects of this thesis. A typical example of overfitting is shown in Figure 3.2.

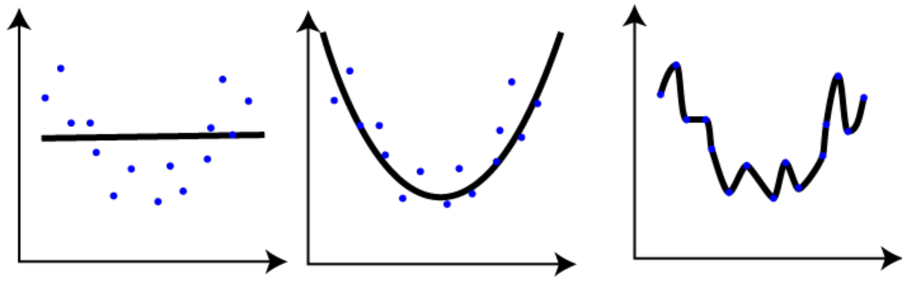


Figure 3.2 – Left: the model is underfitted or equivalently has high bias. The reason is that we are trying to approximate a second order polynomial function using a linear function. Right: the model is overfitted because a high order polynomial function is used. Although the error on the training set is close to zero, the model has a high variance. Middle: the model is just fitted. The Figure is adopted from Bishop (2006).

One potential reason for overfitting could be high capacity of a model. Model capacity is the ability of a model to fit a range of functions. Higher the model's capacity is, the wider range of functions can possibly be approximated (Bishop, 2006). It is worth mentioning that dealing with overfitting or underfitting in such situations corresponds to the trade-off between bias and variance. As the model complexity increases, bias decreases while variance *might* increase in an overfitting setting.

Overfitting due to Unbalanced Data

If data contains a significantly large number of examples for one class and a few examples for the other(s), the data is called unbalanced or skewed. In such scenarios, classifiers may end up performing well on the majority class while performing poorly on other minorities. This type of overfitting simply happens because the classification objective assumes that errors from different classes have the same costs (Ganganwar, 2012). Consequently, fewer number of examples for one class leads to less error for that specific class.

3.3.3 Regularization

A group of different techniques used to avoid the problem of overfitting is called regularization. In most regularization techniques, some kind of prior knowledge is imposed on the model. For example, if number of examples in one class is not enough, resampling might be used to correct the class distributions. Or in some other cases, the model complexity might be controlled (Bishop, 2006).

Usually, as parameters of a model grow in size, the model complexity increases which may lead to overfitting (Bishop, 2006). One of the basic solutions is to add a penalty term to the loss function in order to penalize models with high capacity. By adding this constraint, overfitting can be prevented as a result of preferring simpler models by the learning algorithm. Specifically, for supervised problems, the unregularized loss function in Equation 3.3 is altered with the following regularized one,

$$\hat{\theta} = argmin_{\theta}\{\sum_{i=1}^{n} l(y_i, o_i; \theta) + \lambda J(\theta)\}, \tag{3.6}$$

where $J(\theta)$ is a constraint on the parameters and the coefficient λ controls the balance between two learning objectives.

3.3.4 Cross Validation

In order to find the parameters of the model that generalize the best, we need to know if the model has been overfit. Cross validation helps us to find an overfit model. Overfitting happens when the error rate in the training set decreases but the error on the test set increases. As shown in Figure 3.3, as we increase the complexity of the model, the error rate in the training set decreases but at some point the error in the test set passes the minimum and increases. When the error in the test set increases with higher model complexity the model is overfit.

In cross validation, the dataset is divided into training and validation sets. To increase the validity of the model, *k-fold cross validation* is used where the dataset is partitioned into k equal subsets. We define d as the complexity order of the model. For each order-d hypothesis class:

— Repeat k times:

    — Set aside one of the subsets.

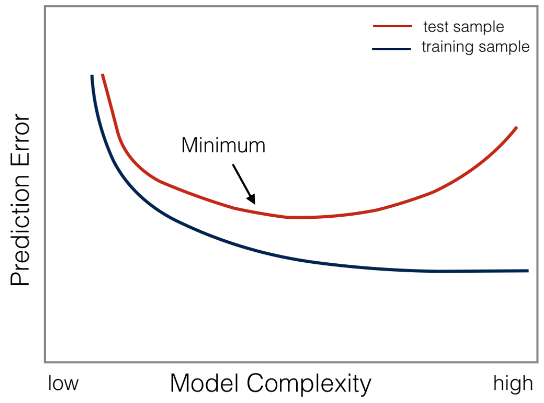


Figure 3.3 – Test and training error as the function of model complexity. Figure is adopted from Murphy (2012).

    — Use the rest of the data points to find θ (model parameters).
    — Compute prediction error on the held-out subset.
- — Average the prediction error over the k rounds/folds. Use this as the estimated true prediction error for order-d hypothesis class

The goal is to find d with the lowest estimated true prediction error. It is worth mentioning that k-fold cross validation increases the computation k-times. Thus, with larger datasets or complex models, smaller values of k is preferred.

3.4 Performance Evaluation

In this section, we introduce the common error metrics used for a classification: (1) Confusion Matrix, (2) Accuracy, (3) Precision and Recall, (4) F_β Score, and (5) ROC Curves (Stehman, 1997). Error metrics help us indicate how good the model will perform when exposed to unseen data. Thus, after the model is trained on the training set and the best performing[2] model is chosen, it will be tested on an intact test set. This approach helps us select a model which will have good performance on unseen data.

2. On the validation set.

3.4.1 Confusion Matrix

A confusion matrix is a table used to describe the performance of a classification model. To be able to construct the confusion table, the true targets must be available. As shown in Figure 3.4, a Confusion Matrix contains four values to describe the performance of a classification model: *false positive*, *false negative*, *true positive*, and *true negative*. False positive (resp. negative) is the number of mistakenly classified examples that are classified as 1 (resp. 0) where the actual targets are 0 (resp. 1). Similarly, true positive (resp. negative) is the number of correctly classified examples that are classified as 1 (resp. 0). Generally speaking, the first term (false or true) indicates if the classification result matches with the actual target. The second term (negative or positive) indicates the prediction of the classifier. Based on the confusion matrix, accuracy, precision and recall, F_β and ROC curves are defined.

	Positive (Predicted)	**Negative (Predicted)**
Positive (Actual)	True Positive	False Positive
Negative (Actual)	False Positive	True Negative

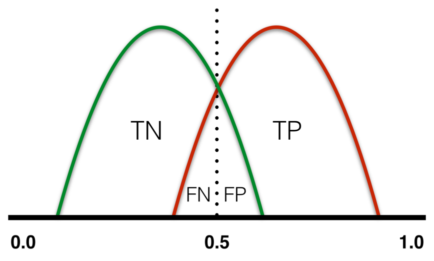


Figure 3.4 – Left: A confusion matrix; The table contains information about actual and predicted targets of a binary classifier. Right: A graphical illustration of the confusion matrix; Red and Green are indicating the real classes while the dotted line corresponds to the threshold of a classifier. The right side of the dotted line is labeled as positive and the left side is labeled as negative.

3.4.2 Accuracy

Accuracy indicates the number of correct predictions from all predictions made by a classification model. Formally, it is defined as,

$$accuracy = \frac{\text{True positives} + \text{True negatives}}{\#\text{ of all examples}}. \tag{3.7}$$

3.4.3 Precision and Recall

Classification accuracy alone is not necessarily the best measurement to evaluate the performance of a classifier and specifically in classification tasks with an imbalanced targets distribution. Suppose a case where we have a majority class (for instance 90 % of the targets belong to this class). A naive model that *always* predicts the majority class achieves high accuracy (Ganganwar, 2012). However, such a classification model that always outputs the same prediction is useless. Thus, accuracy could mislead us to prefer a model with high accuracy while there are cases where we can not merely rely on accuracy.

Precision and recall are two other useful measures that can help us evaluate a classification model more correctly. These two metrics are defined as follows,

$$precision = \frac{\text{True positives}}{\text{\# of all positive predictions}}, \tag{3.8}$$

$$recall = \frac{\text{True positives}}{\text{\# of all positive examples}}. \tag{3.9}$$

3.4.4 F$_\beta$ Score

Precision and recall can also be seen as a measure of model exactness and model completeness respectively. However, in order to compare different models with different precisions and recalls, we need a balance between these two. The metric F_β combines precision and recall into a single value such that different models are compared more easily:

$$F_\beta = (1 + \beta^2) \ . \ \frac{\text{precision . recall}}{(\beta^2 \ . \ \text{precision}) + \text{recall}}, \tag{3.10}$$

where β is single number controlling the balance between precision and recall (Salton and Buckley, 1988).

3.4.5 Receiver Operating Characteristics (ROC) curves

Receiver operating characteristic (ROC) curve is a visualization of the performance of a binary classifier as its *discrimination threshold* changes. The *discrimi-*

nation threshold is the cut-off applied on the predicted probability of a test example that assigns it to a particular class. In an ROC curve, true positive and false positive rates are plotted on vertical and horizontal axis respectively.

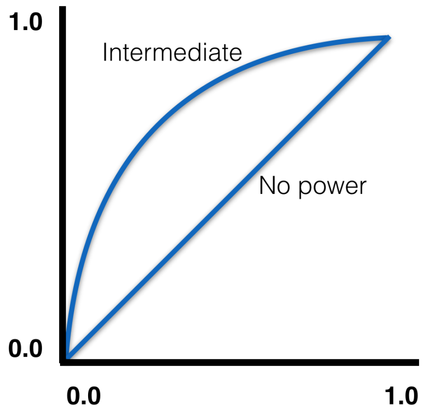


Figure 3.5 – In an ROC curve, the best ideal model would go straight up to left-upper corner and then straight to the right-upper corner. An untrained model with no discrimination is the diagonal one. Usually all classifiers are somewhere between the ideal one and the one with no power.

4 Related Works

The assurance of quality in cloud and proposed approaches approaches have been proposed since cloud computing. Researchers quickly realized that to achieve higher revenue, cloud providers can offer much more resources to customers than their available resources. This is due to the fact that not all customers use their maximum requested resources at the same time. Indeed, there is an underlying distribution that describes customers' behaviors and can be used to manage the requests and resources. As described in section 2.6.2, SLA management provides a framework to effectively manage resource allocation to meet the requested QoS. Cloud is mainly considered to have three service models; Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS); in this thesis we mainly focus on the IaaS.

Different aspects of SLA management in IaaS can be categorized as follows: Load Prediction, Resource Scheduling, and SLA Violation Prediction. In this chapter we briefly describe the related works in each of these categories.

4.1 Load Prediction

A variety of contributions have been proposed for load prediction. Using regression based models, Barnes et al. (2008) proposed an extension of a regression based model for work load prediction in parallel applications. Zhang et al. (2011) proposed a model that predicts the characterization of tasks such as wait time and machine resource utilization on Google's production clusters. Ganapathi et al. (2010) proposed an extension of statistical models to predict resource requirements based on workloads in cloud computing. Carrington et al. (2003) introduced a new model to predict load values using a set of basis operations used in a specific application. Akioka and Muraoka (2004) and Dabrowski and Hunt (2008) used Hidden

Markov Models for host load predictions on large-scale computational grids.

In a closely related work to this thesis, Di et al. (2014) proposed a recent work on host load predictions in a cloud environment using Google Compute Cluster dataset. Authors proposed a Bayesian model to predict the mean load over long-term and consecutive future time intervals. The key idea in this paper is to generate posterior distribution from the prior distribution and the recent fluctuations of load. Ali-Eldin et al. (2012) proposed a provisioning model that monitors and also predicts future loads. Based on the predicted loads, an autonomous elasticity module controls the number of allocated virtual machines and heuristically decreases it by a factor of three.

4.2 Resource Scheduling

In resource scheduling, the fundamental goal is to optimize the usage of infrastructure while maintaining the highest QoS for provided services. In Zheng et al. (2011), the authors use a parallel genetic algorithm (PGA) for the virtual machines scheduling task and load balancing in an IaaS cloud. Their method improves the availability and reliability of the cloud system.

In Zhang and Cao (2013), a resource management scheduling is modeled with a dynamic sequential decision model where future demands are foretasted. Resource management is done through basic operations such as switching on/off hosts, renting remote cloud machines, making machines standby and activating machines. Grey Forecast Model is the core of their intelligent system where based on the historical data, the required number of Virtual Machines is predicted. It is worth mentioning that this work used synthetic data which is generated using a *Poisson* distribution in a simulation environment.

Wang and Vassileva (2007) manage different services based on SLA and feedback from users in a peer-to-peer web application. In Wu et al. (2014), authors proposed a detection scheme that prevents unfair ratings to compete with other services. Alhamad et al. (2010) proposed an SLA-based trust model for cloud that selects the provider based on a selection scheme; in this scheme, although no violation is predicted, the customers are grouped according to business needs and the most trusted cloud provider is selected based on the customer's non-functional requirements.

4.3 SLA Violation Prediction

SLA violation prediction is an essential task in cloud systems as an SLA violation might cause interruptions for the customers' availability of service and force penalties on the provider. Emeakaroha et al. (2010) proposed *LoM2HiS* model which maps monitored low-level metrics to high-level SLA parameters. While no learning is involved, the paper uses heuristics, based on predefined threat thresholds to predict potential future violations. Wu et al. (2011) proposed *Profmin-VMminAvaiSpace*, an algorithm that maps users' requirements into infrastructure resources to provide a reliable service and at the same time maximize resource allocation in order to prevent violations.

Authors in Jules et al. (2014) use a Naive Bayes model to predict SLA violations. Despite its good performance, the dataset is generated using simulation which does not necessarily represent a real environment. It contains 40% violations and neglects the fact that in real world, violations are very rare (∼2.0%). In a similar work for predicting SLA violations in composite services, in Leitner et al. (2010), the authors propose a regression machine learning model; the regression model is implemented using the WEKA framework which cannot be scaled to real world environments.

The authors in Uriarte et al. (2015), use unsupervised learning to cluster the resource usage and duration of services to avoid violations of Google Cluster trace dataset. If a violation happens inside a cluster of services, the other services inside the cluster will be assigned to other resources to avoid the violation. This helps in violation avoidance in the cluster but there is no explicit prediction of SLA violation for each service.

5 Methodology

In this chapter, we propose machine learning models and techniques to tackle the problem of SLA violation prediction. Previous contributions on SLA violation prediction or avoidance have mainly neglected the challenges of using real world data. We, on the other hand, use a real world dataset *(Google Cloud Compute Trace)* to propose machine learning models that can best tackle real world data.

SLA violation prediction can be simply deemed as a classification problem. We define two classes: violated and unviolated; the objective is to predict if a coming request will be violated or not (will be served). After general analysis of the dataset, we observe that the dataset is highly skewed meaning that the number of violated tasks are much smaller than the number of unviolated ones [1]. Thus, we are facing an unbalanced classification task. An aspect that has been overlooked in most of existing contributions as they do not use real world data to solve the problem.

The rest of this chapter is organized as follows. In section 5.1 we first analyze the dataset and understand its characteristics, features and class distribution; this helps to discover models that can best make predictions with such features. Then, we discuss how SLA violations can be found and defined in the *Google Cluster Trace* (Reiss et al., 2011) dataset which provides information for the availability of service in the infrastructure layer of Google's cloud clusters. In Section 5.2, we present two approaches that we adopt in order to tackle the unbalanced classification problem: algorithm-based approach (Ensemble methods in Section 5.3) and data-based approach (Re-sampling techniques in section 5.4). Finally, in Section 5.5, we present the machine learning models that can best tackle the classification task of SLA violation prediction on a skewed dataset; we also present the implementation details.

1. For example, Google Cloud Compute Trace has 97.80% availability versus 2.20% violations.

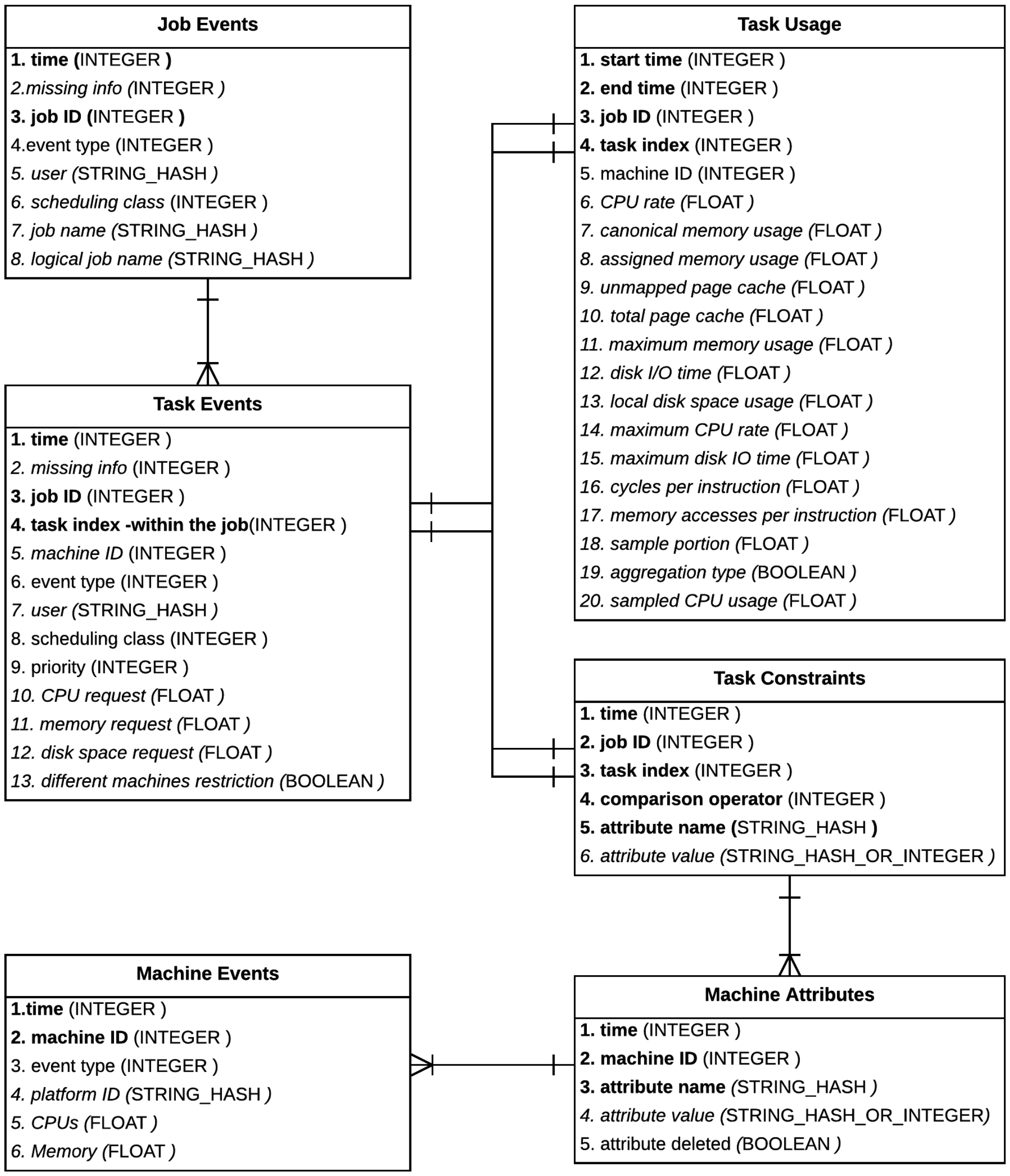
5.1 Dataset

The dataset, we did consider, contains 29-day trace of Google's Cloud Compute published in 2011. For security reasons, part of the trace has been omitted or obfuscated. For example, the values for CPU, disk and memory have been re-scaled by dividing each value by their corresponding largest value in the trace. Also the names of the users' applications have been hashed. The trace has six separate tables: *Job Events*, *Task Events*, *Task Usage*, *Machine Events*, *Machine Attributes*, and *Task Constraints*. The entity relationship diagram of the database is shown in Figure 5.1.

User's application submits its required resources as jobs to the cluster. Each job has several tasks. The state transition diagram of jobs and tasks is depicted in Figure 5.2. The *Job Events* table traces the event cycle of the jobs that were submitted to the cluster. The tasks inside each job are tracked in *Tasks Events* table. Each task is then assigned to a specific machine. *Machine Events* table shows removal or addition of a machine to the cluster or update of its resources. *Machine attributes* table shows the attributes of each machine such as kernel version, clock speed and presence of an external IP address (Reiss et al., 2011). Tasks can have constraints (e.g. A task may have zero or more task placement constraints, which restrict the machines on which the task can run.) on machine attributes which are recorded in the *Tasks Constraints* table.

Metrics such as *requested CPU*, *requested memory*, *requested disk space*, *scheduling class* and *priority* of the task are all recorded in tasks events table. The *Task Usage* table contains the actual usage of resources for each task. It contains information such as *assigned memory* and *memory usage* (Reiss et al., 2011).

The features, *requested CPU*, *requested memory*, *requested disk space*, *scheduling class* and *priority* for each task are fed to the classifier model. Table 5.1 summarizes features, their types and descriptions. Table 5.2 also provides an example for each of two classes. A total of $20,000$ datapoints are used for training and validation the model. In a 3-fold cross validation, one third of data is used for validation and the rest is used for training. These features are considered as high level features that can semantically fully represent each task. Other criterion such as state and load of each machine when a request is taken place can also be considered as features. However, we avoided using too many features in order to prevent the model from



Job Events
1. time (INTEGER)
2.missing info (INTEGER)
3. job ID (INTEGER)
4.event type (INTEGER)
5. user (STRING_HASH)
6. scheduling class (INTEGER)
7. job name (STRING_HASH)
8. logical job name (STRING_HASH)
Task Usage
1. start time (INTEGER)
2. end time (INTEGER)
3. job ID (INTEGER)
4. task index (INTEGER)
5. machine ID (INTEGER)
6. CPU rate (FLOAT)
7. canonical memory usage (FLOAT)
8. assigned memory usage (FLOAT)
9. unmapped page cache (FLOAT)
10. total page cache (FLOAT)
11. maximum memory usage (FLOAT)
12. disk I/O time (FLOAT)
13. local disk space usage (FLOAT)
14. maximum CPU rate (FLOAT)
15. maximum disk IO time (FLOAT)
16. cycles per instruction (FLOAT)
17. memory accesses per instruction (FLOAT)
18. sample portion (FLOAT)
19. aggregation type (BOOLEAN)
20. sampled CPU usage (FLOAT)
Task Events
1. time (INTEGER)
2. missing info (INTEGER)
3. job ID (INTEGER)
4. task index -within the job(INTEGER)
5. machine ID (INTEGER)
6. event type (INTEGER)
7. user (STRING_HASH)
8. scheduling class (INTEGER)
9. priority (INTEGER)
10. CPU request (FLOAT)
11. memory request (FLOAT)
12. disk space request (FLOAT)
13. different machines restriction (BOOLEAN)
Task Constraints
1. time (INTEGER)
2. job ID (INTEGER)
3. task index (INTEGER)
4. comparison operator (INTEGER)
5. attribute name (STRING_HASH)
6. attribute value (STRING_HASH_OR_INTEGER)
Machine Events
1.time (INTEGER)
2. machine ID (INTEGER)
3. event type (INTEGER)
4. platform ID (STRING_HASH)
5. CPUs (FLOAT)
6. Memory (FLOAT)
Machine Attributes
1. time (INTEGER)
2. machine ID (INTEGER)
3. attribute name (STRING_HASH)
4. attribute value (STRING_HASH_OR_INTEGER)
5. attribute deleted (BOOLEAN)

Figure 5.1 – Google's cluster trace dataset ERD (Entity Relationship Diagram). The dataset contains the above five different tables. This ERD is used to define and find violated tasks based on the definition in Section 5.1.1

overfitting.

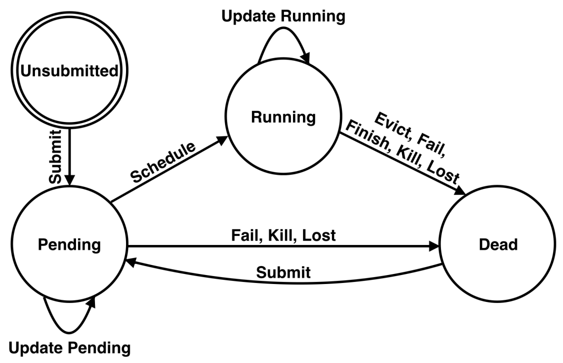


Figure 5.2 – The state transition diagram of a task on Google Cluster machines (Reiss et al., 2011).

5.1.1 Data Analysis

Figure 5.3 illustrates the mechanism of resource allocation in Google's Cluster. It shows the state of the cluster at 500 random *snapshots*. We define a *snapshot* as a moment in time when the total/sum of requested resources is calculated. Similarly, available or allocated resources are calculated at each snapshot. In Figure 5.3, the total requested memory, assigned memory, memory usage and available memory of the cluster at each snapshot are calculated using Task Events, Task Usage and Machine Events tables. Since all the requested resources are not used at the same time, it is the nature of cloud to allocate less resources than requested resources and even accept more requests than its available resources. Figure 5.3 shows that at all of the 500 snapshots, the requested memory to the cluster is much higher than the actual usage of memory. Google scheduler has reserved a safe margin between the assigned memory and usage of memory at these snapshots. Thus, the availability rate is very high and violations are rare.

Violation Detection

In order to identify SLA violations, we need to have specific details of QoS (Quality of Service) parameters and Service Level Objectives (SLOs). Although we do not have access to the details of SLA for this dataset, we can find violations in the availability of the service using the trace.

Figure 5.2 shows the state transition diagram for jobs and tasks in the trace. We define a violation in the availability of the service when a task is evicted and

Feature	Description	Type
`cpu_requested`	A floating point number in $[0, 1]$ indicating the normalized amount of requested cpu.	Continuous
`mem_requested`	A floating point number in $[0, 1]$ indicating the normalized amount of requested memory.	Continuous
`priority`	An integer that is mapped into a sorted set of values, with 0 as the lowest priority and 10 as the highest.	Categorical
`disk`	A floating point number in $[0, 1]$ indicating the normalized amount of requested disk space.	Continuous
`sched_cls`	An integer in [0, 3] representing how latency-sensitive the task is.	Categorical

Table 5.1 – The description and type of features used as the input for the classification model.

Feature	Violated Example	Un-violated Example
`cpu_requested`	0.0436	0.0327
`mem_requested`	0.0632	0.0392
`priority`	3	8
`disk`	0.0082	0.0023
`sched_cls`	2	0

Table 5.2 – Two sample examples (datapoints) from the dataset for each class: violated and un-violated.

never re-scheduled successfully after the eviction. According to the documentation of the trace (Reiss et al., 2011), eviction of a task is due to *"overcommiting of the scheduler or because the machine on which it was running became unusable (e.g. taken offline for repairs), or because a disk holding the task's data was lost"*. Thus, all the tasks that were evicted and never re-scheduled successfully after the eviction were detected as cases of violations. The percentage of non evicted tasks to the total tasks submitted to the cluster is 97.8%. Thus, the cluster has only 2.2% violations for the availability of the service. Our goal is to use this data and predict future violations.

As previously mentioned, violation prediction can be simply considered as a classification problem where we predict whether at a specific time in the future,

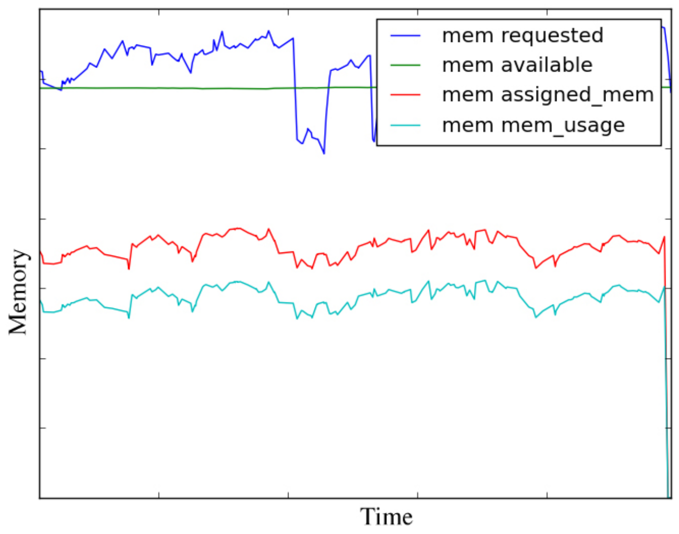


Figure 5.3 – The Figure shows 500 snapshots of the requested, available, assigned and used memory of the cluster.

the provider will have violation or not. Since the availability rate is very high (97.8%) and violations do not happen most of the time, machine learning models have the tendency of always predicting the absence of violations, which is not desirable. Thus, in Section 5.2 we introduce some techniques in machine learning to handle the skewness of data, such as re-sampling techniques and applying models that have better performance with skewed datasets.

Features such as the amount of requested CPU, disk and memory of the violated tasks and also the available resources at the time of request can be studied to predict future violations.

5.2 Tackling Unbalanced Data

In typical classification algorithms, the distribution of classes are usually considered to be balanced, either implicitly or explicitly (Provost, 2000). Consequently, naively applying common classification models on unbalanced data may lead to poor performance of classifiers.

In most of probabilistic classification models, such as Naive Bayes, the probability of occurrence of the rare class can be very close to zero. Furthermore, in some other classification algorithms, such as decision trees, criterion for feature selection simply ignores the imbalance between different classes (Drummond and Holte, 2000). Given a small amount of data for one class, feature selection methods could easily fail as there is no significant change in model performance by adding or eliminating a feature (Provost, 2000).

Based on the assumption that poor classification performance is mainly due to mis-classification of rare classes, we mainly focus on adaptation of learning algorithms for rare classes. We investigate two general approaches to solve this problem. The first method has an algorithm-based approach where it tries to improve the results using an aggregation of several classifiers. We discuss this method in more details in section 5.2.1. The second method has a data-based approach where it changes the training data distribution in a smart way in order to bias the classifier towards the rare class. This method re-samples the data set in a way that the two classes will have close distributions. The data-based approach is described in more details in section 5.2.2.

5.2.1 Algorithm-based Approach

The algorithm-based approach for tackling unbalanced data is the ensemble methods. Ensembling is a form of model regularization aimed to compensate errors of one classifier by building more complex classifiers from simpler ones. Ensembling amounts to training several classifiers in order to aggregate their predictions to get a final prediction with higher performance. As introduced in Chapter 3, in the concept of bias-variance trade-off, due to data skewness, variance of a classifier could be relatively high. Ensembling multiple classifiers usually helps reducing the variance significantly. Random Forests is one of the successful ensemble methods which is introduced in Section 5.5.4.

5.2.2 Data-based Approach

As a data-based approach, re-sampling the training data could bias the classifier towards the rare class in order to make the error on the rare class as significant as the other classes. The re-sampling can be done on any of the classes. There are two forms of re-sampling: *Over Sampling* and *Under Sampling*. An over sampler creates new data points in the minority class and an under sampler deletes data points from the majority class. Different forms of re-sampling techniques are introduced in Section 5.4.

5.3 Ensemble Methods

The core idea behind ensemble methods is to build a complex and powerful classifier by combining the prediction results of simpler ones that are called *base algorithms*. The base algorithm can be a Decision Tree, Naive Bayes or Support Vector Machines. Ensemble of classifiers achieve higher performance as they reduce the bias or variance or both (Opitz and Maclin, 1999).

There are several ways to create an Ensemble of classifiers, but the two general approaches are: *Bagging* and *Boosting*. These approaches are illustrated in Figure 5.4. We briefly describe each of them in the following subsections.

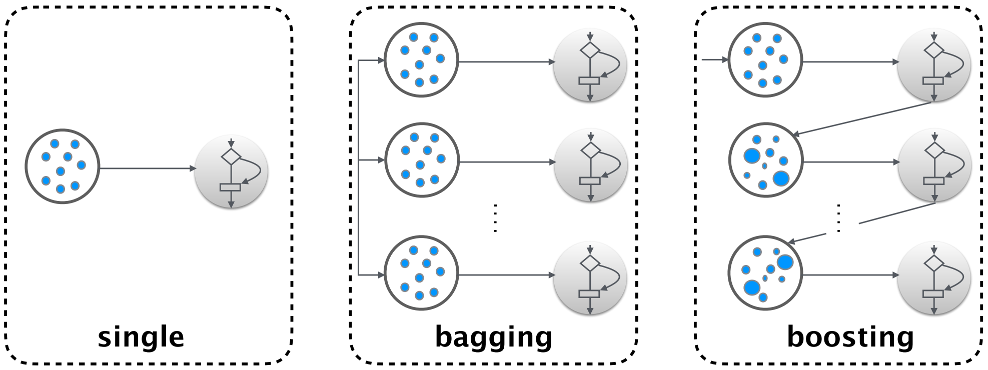


Figure 5.4 – In a single model, the complete dataset is given to the model in one iteration. In bagging, the dataset is divided into several sets randomly sampled with replacement from the original dataset. The sets are then fed to the model in parallel. In boosting, random sampling with replacement over weighted data is used. The data is sequentially given to a set of weak learners.

5.3.1 Bagging

Bagging aims to reduce the variance and avoid over-fitting (Opitz and Maclin, 1999). It generates several sets by sampling *with replacement* from the original dataset. These sets have the same size and some datapoints might be repeated in each of them. This kind of sampling is called *bootstrapping* (Felsenstein, 1985).

Bootstrapping increases the size of the training set which reduces the variance. The sets are fed to the model and their results are combined by either averaging (regression) or choosing the majority class (classification) (Breiman, 1996).

5.3.2 Boosting

Boosting aims to reduce the bias and also the variance (Opitz and Maclin, 1999). It combines a set of weak learners into a stronger one. At each round, the weighted training set is fed to a weak learner which gets added to the final classifier with a weight that is usually related to its accuracy. After a weak learner is added, the data is re-weighted such that the datapoints that were classified correctly will lose weight. This allows the succeeding learners focusing on the datapoints that were misclassified. The process is repeated n times and the result will be a combination of n weak learners as a strong learner (Freund and Schapire, 1995).

5.4 Data Resampling

Several models to re-sample the database and create a more balanced dataset are presented in this section. They can be divided into three general categories: Over Sampling, Under Sampling and combination of both. *Random Over-sampling* and *Random Under-sampling* are baseline methods to combat the skewness of the dataset and balance the class distribution. We will also briefly describe several other techniques ; *SMOTE, Borderline-SMOTE, Tomek links, One-sided Selection, Neighborhood Cleaning Rule, Near Miss (1, 2, 3), SMOTE-Tomek Links* and *SMOTE-ENN.*

5.4.1 Over Sampling Techniques

Random Over-sampling

In random over-sampling, the samples of the minority class are randomly picked and duplicated. The method continues to create duplicate datapoints until the two classes have roughly the same number of datapoints.

SMOTE

SMOTE ((Synthetic Minority Over-sampling Technique)) (Chawla et al., 2002) is a re-sampling method that generates new "synthetic" datapoints of the minority class using interpolation between the current datapoints. One major drawback of this technique is that it may add new datapoints in the space of the majority class.

Borderline-SMOTE

Borderline-SMOTE (Han et al., 2005) tries to generate synthetic samples only on the border line of the two classes. The datapoints which are in the borderline of the classes are more prone to miss-classification and thus need more attention. It is based on SMOTE for generating new datapoints. The synthetic datapoints are created on the border line and between the minority datapoints and its selected nearest neighbors.

5.4.2 Under Sampling Techniques

Random Under-sampling

Random Under-sampling is the case of randomly deleting data points from the dominant class until both classes have roughly the same size. This method might delete the datapoints in the decision boundary that are important in the process of decision making.

Tomek links

Tomek links (Tomek, 1976) is an under sampling method. Tomek links technique removes the borderline and noisy datapoints. It takes two samples, E_i and E_j and computes d(E_i, E_j) as their distance. A (E_i, E_j) is a Tomek link if there is no sample E_k that $d(E_i, E_k) < d(E_i, E_j)$ or $d(E_j, E_k) < d(E_i, E_j)$. After finding the links, the datapoints from the dominant class are removed.

One-sided Selection

One-sided Selection (Kubat et al., 1997) uses combination of Tomek links and an extension of Nearest Neighbor which is defined as an optimization problem for finding closest points (Gowda and Krishna, 1979). One-sided Selection finds the safe samples and removes the unsafe samples from the majority class. Tomek links removes the samples near the border line and Nearest Neighbor removes the samples that are far from the border line.

Neighborhood Cleaning Rule

Neighborhood Cleaning Rule (NCR) Laurikkala (2001) improves Edited Nearest Neighbor Rule (ENN)(Wilson, 1972) to remove some datapoints from the majority class. ENN cleans the data such that any datapoint whose class is different from the class of at least its two nearest neighbors is removed. NCR finds the three nearest neighbors of each datapoint (E_i). If E_i's three neighbors are from the minority class, and E_i was classified as the majority class, then E_i is removed. But, if E_i belongs to the minority class and its neighbors are from the majority class, then its three neighbors will be removed.

NearMiss

NearMiss 1, 2 and 3 algorithms (Mani and Zhang, 2003) are under-sampling methods. *NearMiss 1* removes the datapoints from the majority class whose average distance to three closest datapoints in the minority class is the smallest. *NearMiss 2* chooses the majority class datapoints whose average to all datapoints in the minority class is the smallest. *NearMiss 3* removes a given number of majority class datapoints for each datapoint in the minority class. The selection is done by applying a clustering and selecting a sample from each clustered neighborhood.

5.4.3 Combination of Over Sampling and Under Sampling Techniques

SMOTE-Tomek links

Since SMOTE over-sampling might lead to over-fitting and Tomek links under sampling might remove important datapoints, the ensemble of these two methods provides better results. In SMOTE-Tomek links (Batista et al., 2003), we first over-sample the minority class with SMOTE and then under-sample using Tomek links both the majority and minority classes producing a more balanced dataset.

SMOTE-ENN

SMOTE-ENN (Batista et al., 2004) is also a combination of SMOTE and ENN. SMOTE is used as the over-sampler for minority class and then ENN provides data cleaning for both classes.

5.5 Classification Models

In this section, we briefly describe the machine learning models that we use for the task of SLA violation prediction; *Naive Bayes Classifier*, *Decision Tree* and *Random Forest Classifier*. We also briefly describe the implementations details.

5.5.1 Naive Bayes Classifier

From a probabilistic point of view, the conditional probability of class k among K different classes given a vector representation of n distinct features $\mathbf{x} = \{x_1, ..., x_n\}$ can be written as $P(C_k|\mathbf{x})$. According to the Bayes theorem (Vapnik, 1999), the above probability can be reformulated as follows:

$$P(C_k|\mathbf{x}) = \frac{P(\mathbf{x}|C_k)P(C_k)}{P(\mathbf{x})}, \tag{5.1}$$

in which $P(C_k|\mathbf{x})$ is called the *posterior* meaning our updated knowledge conditioned on the observed data. Two probabilities $P(\mathbf{x}|C_k)$ and $P(C_k)$ are called the *likelihood* and the *prior* respectively.

In a classification setup, the denominator $P(\mathbf{x})$ is the same for all classes. In practice, training such a Bayesian classifier amounts to maximizing the nominator for the target class and minimizing it for the other classes (Friedman et al., 2001). The nominator is the joint probability of features and classes $P(C_k, x_1, ..., x_n)$ which according to the chain rule (Schum, 1994), can be reformulated as follows:

$$\begin{aligned} P(C_k, x_1, ..., x_n) = & P(C_k) * \\ & P(x_1 | C_k) * \\ & P(x_2 | x_1, C_k) * \\ & P(x_3 | x_2, x_1, C_k) * \\ & ... \\ & P(x_n | x_{n-1}, ..., x_1, C_k). \end{aligned}$$

Consequently, since $P(C_k | x_1, ..., x_n) \propto P(C_k, x_1, ..., x_n)$, a classifier can be defined as follows:

$$\hat{c} = \underset{k \in \{1,...,K\}}{\operatorname{argmax}} P(C_k, x_1, ..., x_n).$$

In practice, for large number of features, n, it is challenging to train such a classifier. One of the simple, yet effective probabilistic classifiers is known as *Naive Bayes* (Duda et al., 1973). *Naive Bayes* algorithm has an assumption that given the class label C_k, all the features $\{x_1, ..., x_n\}$ are independent of each other. The adjective *naive* comes from the fact that the assumption of class conditional independence is simplistic. A graphical illustration of this classifier is shown in Figure 5.5.

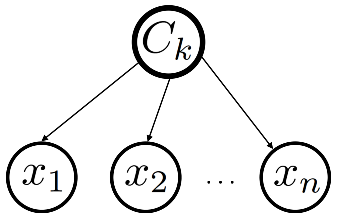


Figure 5.5 – Bayesian network representation of the naive Bayes classifier. According the the graph representation, conditioned on the class C_k, x_i's are independent of each other.

5.5.2 Naive Bayes Implementation

As described in Section 5.5.1, Naive Bayes classifier uses the Bayes theorem (Equation 5.1). In most of Naive Bayes implementations, the prior is computed by using an estimate for the class probability from the training set [2]. To model the likelihood term, one can assume a distribution over the data represented by features vector. In this work, we have chosen two distributions: (1) Bernoulli distribution and (2) Gaussian distribution. As the task at hand has two states of being violated or not violated, Bernoulli distribution is a desirable prior which assigns a success probability of p and a failure probability of $1 - p$. Gaussian distribution is also a common prior distribution which provides easy inference and is mathematically convenient. A Gaussian distribution can approximate a wide range of distributions with two single parameters; mean and variance. In the following sub-sections, we explain these extensions in more details.

Bernoulli Naive Bayes

In a Bernoulli Naive Bayes (McCallum et al., 1998), the likelihood is assumed to have a Bernoulli distribution. Features are assumed to be independent booleans and are required to be binary-valued. Thus, if other types of features were represented to the model, the input is binarized. The decision rule is as follows:

$$p(\mathbf{x}|C_k) = \prod_{i=1}^{n} p_{ki}^{x_i}(1 - p_{ki})^{(1-x_i)}, \tag{5.2}$$

where p_k is the probability of feature x_i appearing in a sample belonging to class C_k.

Gaussian Naive Bayes

In Gaussian Naive Bayes (McCallum et al., 1998), the likelihood is assumed to have a Gaussian distribution:

$$P(\mathbf{x}|C_k) = \frac{1}{\sqrt{2\pi\sigma_{C_k}^2}} exp\big(- \frac{(x_i - \mu_{C_k})^2}{2\sigma_{C_k}^2}\big), \tag{5.3}$$

where σ_{C_k} and μ_{C_k} are estimated using maximum likelihood.

2. Prior for a given class is computed as $P(C_k) = \frac{\text{number of data point belong to class } C_k}{\text{Total number of data points}}$.

5.5.3 Decision Tree Classifier

Decision Tree (Breiman et al., 1984) is a family of scalable classifiers that enjoy the advantage of human-interpretable results. Formally, a classification decision tree is a tree in which each leaf represents a target class, each internal node represents a condition, and each branch corresponds to the outcome of the condition in the parent node.

As a simple example (Mitchell et al., 1997), consider a set of features {*Outlook, Humidity, Wind*} and the target is *PlayTennis* which takes values of *"Yes"* or *"No"*. A trained decision tree is shown in Figure 5.6.

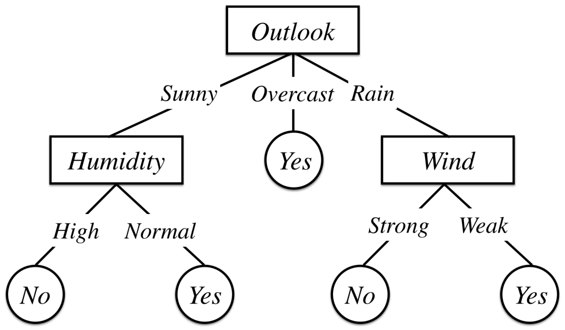


Figure 5.6 – A graphical illustration of a Decision Tree: Classification starts from the top node towards leaves by testing the *Outlook*. After moving to one of the left or the right subtrees, a test on *Humidity* or *Wind* determines the class label (Mitchell et al., 1997).

Construction of a decision tree amounts to finding the appropriate conditions on nodes and ordering them from root to the leaves. A condition is a test on one of the features of the given datapoint. Among different types of tests, we use *Gini Impurity* (Breiman et al., 1984) on each feature as the criterion that splits nodes to their children. *Gini impurity* measures the probability of being wrongly classified for a random datapoint, if the classification is based on the distribution of the targets. We chose Gini Impurity as it is slightly more robust to mis-classification compared to other criterion such as entropy.

5.5.4 An Ensemble of Decision Tree Classifier: Random Forest

Random Forest is an ensemble learning method. Decision Tree is the base algorithm for building the Random Forest model. From a geometrical point of view, a decision tree leads to a hierarchical partitioning over the feature space. Starting from the top node in the tree, each node divides the feature space into two or more partitions. Consequently, as the tree gets deeper, more complicated partitioning is done. However, in the case of over-fitting, the partitioned space is over complicated and yields a small error on the training data while a relatively larger error on the test data.

One of the successful ways to overcome the issue of over-fitting is to use ensemble of decision trees or *Random Forrest* (Breiman, 2001). Random Forrest is an *ensemble* of decision trees which the final prediction is the result of the aggregation of each decision tree.

Given a training set $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m\}$ and a corresponding set of targets $Y = \{y_1, y_2, ..., y_m\}$, where each $\mathbf{x}_i$ has a set of n features $\mathbf{x} = \{x_1, x_2, ..., x_n\}$, do the following k times to train a random forest:

— **Select a set of random samples with replacement from X called X_k.**

— **Build a decision tree on X_k. Where the tree is trained on a random subset of the features ($n' < n$).**

As illustrated in Figure 5.7, k different trees are trained. For a classification task, the class prediction for an unseen sample can be made by choosing the majority class. The performance of random forest depends on two aspects of the model (Breiman, 2001): the correlation between two trees and the accuracy of each tree. By increasing the size of n', the strength of each tree increases but also the correlation between two trees increases. Thus, the value of n' needs to be optimized.

5.5.5 Random Forest Implementation

Random forest is an ensemble of decision trees. As the number of trees in a forest increases, better overall performance is achieved (Oshiro et al., 2012). However, it is computationally expensive to create large forests. There is certain point in which adding more trees does not significantly improve the performance while it adds more computations. Considering the size of our dataset and computational resources, in all our experiments, we use ten trees. The bootstrap samples are also created with the same size as the original dataset with random sampling with replacement.

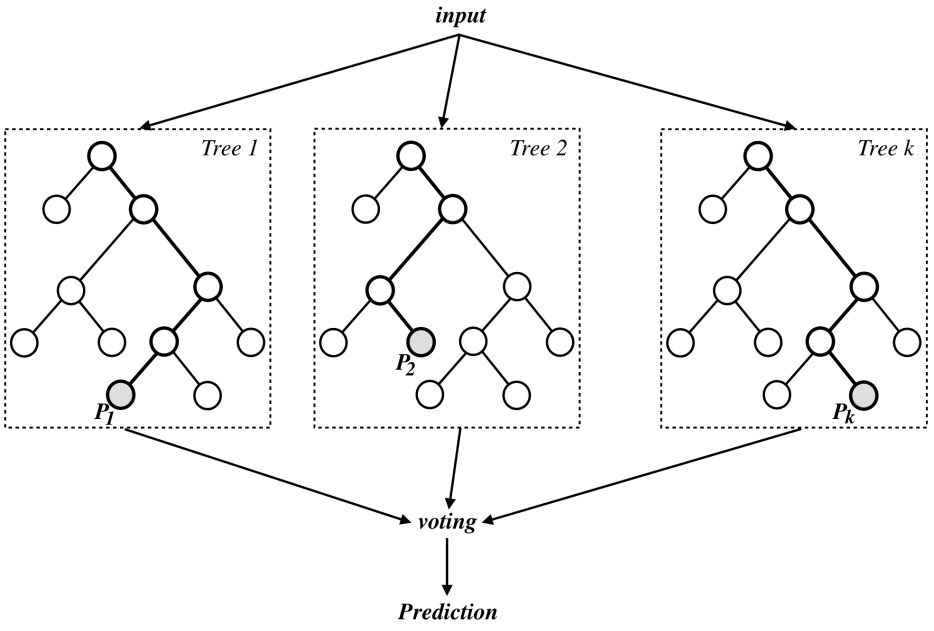


Figure 5.7 – In a random forest, k different decision trees are trained using k different subsets of the dataset. During test time, a sample *input* point is fed to all trees and predictions $P_{1..k}$ are generated. A voting is then applied on all predictions to make a single final prediction.

6 Results and Discussion

In the previous chapters, several methods and algorithms for the classification task of SLA violation prediction and handling skewed data were introduced. In this chapter, we first report and analyze the results of several machine learning and re-sampling techniques in Section 6.2. In Section 6.1 we explain the toolkit and environments in which the experiments are done. Finally, in Section 6.3, we discuss the results.

In all our experiments we use *3-fold cross validation*. The dataset is randomly split into three partitions and the prediction model is trained three times. During each training, two-third of the dataset is used as the training set and fed to the model and one-third as the test set. The aggregated results of the three runs on the model are reported as the final result.

6.1 Environments and Toolkits

In this section, the details of the environment and toolkit that were used for the implementation are described. The source code for this work is available in the author's github[1].

6.1.1 Python

Python is a general-purpose, interpreted, dynamic programming language that is widely used for data analysis. The robust collection of scientific, statistical and mathematical tools in python allows easier implementation of Machine Learning models.

Libraries such as *NumPy* (Van Der Walt et al., 2011) (Python's Numerical Library), *SciPy* (Jones et al., 2001) (Python's scientific library), Scikit-Learn and

1. https://github.com/ReyhaneAskari/SLA_violation_classification

Imbalanced-learn are built on top of python to provide easy computation and analysis on data. In this work we used python 2.7 along with many other libraries.

6.1.2 Scikit-Learn

Scikit-Learn (Pedregosa et al., 2011) is a machine learning library built on top of python, Scipy and NumPy. Scikit-Learn provides various tools for data mining and analysis and is also open source and commercially usable. It features different classification, regression and clustering algorithms such as Random Forests, Gradient Boosting, k-means and Naive Bayes.

6.1.3 Imbalanced-learn

Imbalanced-learn (Lemaître et al., 2016) is a python library built on top of scikit-learn, Scipy and Numpy. It offers many re-sampling techniques for unbalanced data such as over-sampling, under-sampling and combination of both.

6.1.4 T-SNE (t-Distributed Stochastic Neighbor Embedding)

T-SNE (Maaten and Hinton, 2008) is an effective dimensionality reduction technique which aims to preserve local structures in the high dimensional data while bringing data into a lower dimensional space.

6.2 Results

For the sake of readability, results are presented in the following three subsections; *classification with under-sampling methods*, *classification with over-sampling methods*, and *classification with a combination of Over-sampling and Under-Sampling*. For a general overview of all classification models and re-sampling methods, a summary including F_1 score is presented in Figure 6.1. Tables of results contain experiments for Naive Bayes and Random Forest classifiers, each with both Bernoulli or Gaussian likelihood assumptions. Also, the ROC curves of the Random Forest which have the best results are also shown in Figure 6.2. The following models resulted in almost zero F_1 score thus, have been removed from the tables: Naive Bayes (Gaussian or Bernoulli), Naive Bayes (Gaussian or Bernoulli) with SMOTE, Naive Bayes (Gaussian or Bernoulli) with SMOTE borderline, Naive Bayes (Gaussian or Bernoulli) with Random Over-Sampling, Naive Bayes (Gaussian or Bernoulli) with SMOTE Tomek Links, and Naive Bayes (Bernoulli) with SMOTE ENN.

In our evaluation, we used error metrics (precision, recall and F_1 score) rather than accuracy. Indeed, in skewed datasets, accuracy can not be a good error metric to find the best performing classifier. Two classes are available: 2.2 % of the samples are represented as violated class and 97.8 % of the samples are represented as unviolated. Consider a classifier that predicts there will be no violations. It has an accuracy of 97.8 % but Precision and Recall of zero. Thus, precision, recall and F_β score will help us find the better performing algorithm.

In order to have a better understanding of how each resampling method works, different methods are visualized in 2D. Technically, since the data is in an 5-dimensional space, visualization is impossible. To overcome this limitation, we use t-SNE to bring the data into a 2-dimensional space. Specifically, we select a small subset of the dataset (1000 data-points) and apply t-SNE to bring them into a 2D space, we then visualize the application of any of the resampling methods.

6.2.1 Classification with Under-Sampling

Table 6.1 collects the results of models with different Under-Sampling methods. A visualized comparison between data with no resampling and with different under-sampling techniques is also shown in Figure 6.3.

6.2.2 Classification with Over-Sampling

Table 6.2 collects the results of models with different Over-Sampling methods. A visualized comparison between data with no re-sampling and with different over-sampling techniques is shown in Figure 6.4.

Model / Method	Acc	ROC	Precision	Recall	$F_{0.5}$	F_1	F_2
RF	0.9708	0.99	0.9353	0.6885	0.8728	0.7932	0.7269
NB (Gaus)-under sampled	0.5514	0.68	0.5293	0.8611	0.5735	0.6556	0.7652
NB (Bern)-under sampled	0.6420	0.72	0.6788	0.5280	0.6421	0.5940	0.5526
RF-under sampled	0.9486	0.99	0.9361	0.9800	0.9446	0.9576	0.9709
NB (Gaus)-near miss 1	0.7594	0.75	0.7892	0.7023	0.7702	0.7432	0.7181
NB (Bern)-near miss 1	0.7159	0.71	0.8056	0.5629	0.7416	0.6627	0.5990
RF-near miss 1	0.8350	0.91	0.9546	0.7146	0.8945	0.8174	0.7525
NB (Gaus)-near miss 2	0.7506	0.74	0.7882	0.6908	0.7666	0.7363	0.7083
NB (Bern)-near miss 2	0.7078	0.71	0.8076	0.5519	0.7391	0.6557	0.5892
RF-near miss 2	0.8314	0.91	0.9629	0.7064	0.8977	0.8150	0.7462
NB (Gaus)-near miss 3	0.86712	0.88	0.9436	0.9086	0.9364	0.9258	0.9154
NB (Bern)-near miss 3	0.9129	0.72	0.9129	1.0	0.9291	0.9544	0.9812
RF-near miss 3	0.9747	0.98	0.9927	0.9826	0.9907	0.9876	0.9846

Table 6.1 – Full results of Naive Bayes (NB) and Random Forest (RF) classification algorithms with Under-Sampling techniques. Results are achieved using 3-fold cross validation.

Model / Method	Acc	ROC	Precision	Recall	$F_{0.5}$	F_1	F_2
RF	0.9708	0.99	0.9353	0.6885	0.8728	0.7932	0.7269
RF - Random over sampling	0.9688	0.99	0.9258	0.7058	0.8715	0.8009	0.7410
RF - SMOTE	0.9690	0.99	0.9404	0.6908	0.8770	0.7965	0.7295
RF - SMOTE borderline 1	0.9704	0.99	0.9189	0.7313	0.8740	0.8144	0.7624
RF - SMOTE borderline 2	0.9696	0.99	0.9158	0.7318	0.8720	0.8135	0.7625

Table 6.2 – The results of Random Forest (RF) classification algorithm with Over-Sampling methods. Results are achieved using 3-fold cross validation.

6.2.3 Classification with Combination of Under-Sampling and Over-Sampling

As some of the re-sampling methods are combinations of both under-sampling and over-sampling, the results are shown in Table 6.3 and their visualizations are shown in Figure 6.5.

Model / Method	Acc	ROC	Precision	Recall	$F_{0.5}$	F_1	F_2
RF	0.9708	0.99	0.9353	0.6885	0.8728	0.7932	0.7269
RF - SMOTE Tomek links	0.9683	0.99	0.9452	0.6920	0.8808	0.7990	0.7312
NB (Gaus) - SMOTE ENN	0.7457	0.82	0.1701	0.7325	0.2010	0.2761	0.4409
RF - SMOTE ENN	0.9988	1.0	0.9987	0.9972	0.9984	0.9980	0.9975

Table 6.3 – The results of Naive Bayes (NB) and Random Forest RF) classification algorithms with combination of Over-Sampling and under-Sampling techniques. Results are achieved using 3-fold cross validation.

6.3 Discussion

In order to evaluate the models on skewed datasets, the F_1 score is the principal metric. Thus, the best performing model in terms of F_1 score is the random forest classification algorithm on the dataset re-sampled using the SMOTE + ENN technique. This method has also a very high accuracy rate of 99.88 %.

Random Forest has better performance because tree based classifiers are less sensitive to class distributions and a classifier family with a much higher capacity than naive Bayes. Thus, even with no re-sampling technique it has an acceptable performance (accuracy = 97 % and $F_1 = 0.79$). On the other hand, Naive Bayes classifiers are highly biased with class distribution and do not have any acceptable results without re-sampling techniques.

As discussed in the previous chapter, Random Forest is an ensemble method and reduces the error by reducing the variance. There are two features contributing in the reduction of variance: averaging and random sampling. Each tree in the Random Forest has a very high variance but averaging reduces the variance as long as the trees are not co-related. Since training many trees on the same dataset generates strongly co-related trees, different subsets of the dataset are fed to decision trees. By adding randomness to the sampling process, trees are trained even more independently which in case of large number of trees, the gain for averaging can be more dramatic.

Random Forest also provides human interpretable results and one can find the importance of each feature in the classification task. According to the trained model, the five features shown in Figure 6.6 are sorted based on their average contributions in classification task. It can be seen that the requested memory has the highest contribution (almost twice as much as the others). Such information can further help the provider to find which part of the infrastructure needs more attention to build a cloud infrastructure that can better serve the requests. In this case, building a task scheduler that better allocates the memory requests can have twice the effect in reducing the violations compared to paying attention to the allocation of disk or CPU.

Among re-sampling methods, combined methods such as SMOTE-ENN and SMOTE-Tomek links have good results. SMOTE-ENN has better performance than SMOTE-Tomek links because the Tomek links algorithm removes the noise and also the datapoints near the borderline of the two classes. Borderline datapoints are important as they are more prone to miss-classification and are thus important datapoints to be kept. Also ENN provides a more in depth data cleaning rule and removes any sample whose three nearest neighbors is miss-classified, which helps with better re-sampling the dataset.

```
Methods
├── Random Under Sampling
│   ├── Naive Bayes
│   │   ├── Gaussian ........................................ 0.656.
│   │   └── Bernoulli ....................................... 0.594.
│   └── Random Forest ....................................... 0.958.
├── Near miss 1
│   ├── Naive Bayes
│   │   ├── Gaussian ........................................ 0.743.
│   │   └── Bernoulli ....................................... 0.663.
│   └── Random Forest ....................................... 0.817.
├── Near miss 2
│   ├── Naive Bayes
│   │   ├── Gaussian ........................................ 0.736.
│   │   └── Bernoulli ....................................... 0.656.
│   └── Random Forest ....................................... 0.815.
├── Near miss 3
│   ├── Naive Bayes
│   │   ├── Gaussian ........................................ 0.926.
│   │   └── Bernoulli ....................................... 0.954.
│   └── Random Forest ....................................... 0.988.
├── Other Random Forest Variants
│   ├── One-Sided Selection ................................. 0.805.
│   ├── Neighborhood Cleaning Rule .......................... 0.998.
│   ├── Random over sampling ................................ 0.801.
│   ├── SMOTE borderline 1 .................................. 0.814.
│   ├── SMOTE borderline 2 .................................. 0.813.
│   └── SMOTE ............................................... 0.796.
│       ├── Tomek links ..................................... 0.799.
│       └── ENN ............................................. 0.998.
└── Other methods* .......................................... ~
                                                              0.0.
```

Figure 6.1 – A hierarchical depiction of different sampling methods and the models used for each. The number associated with each methods indicates the F_1 score. Other methods*: Since the data is highly unbalanced, the other models mostly overfit and learn to always predict the most dominant class. These models include Naive Bayes, Naive Bayes One-Sided Selection, Naive Bayes Neighborhood Cleaning Rule, Naive Bayes Random over sampling, Naive Bayes SMOTE and its variants.

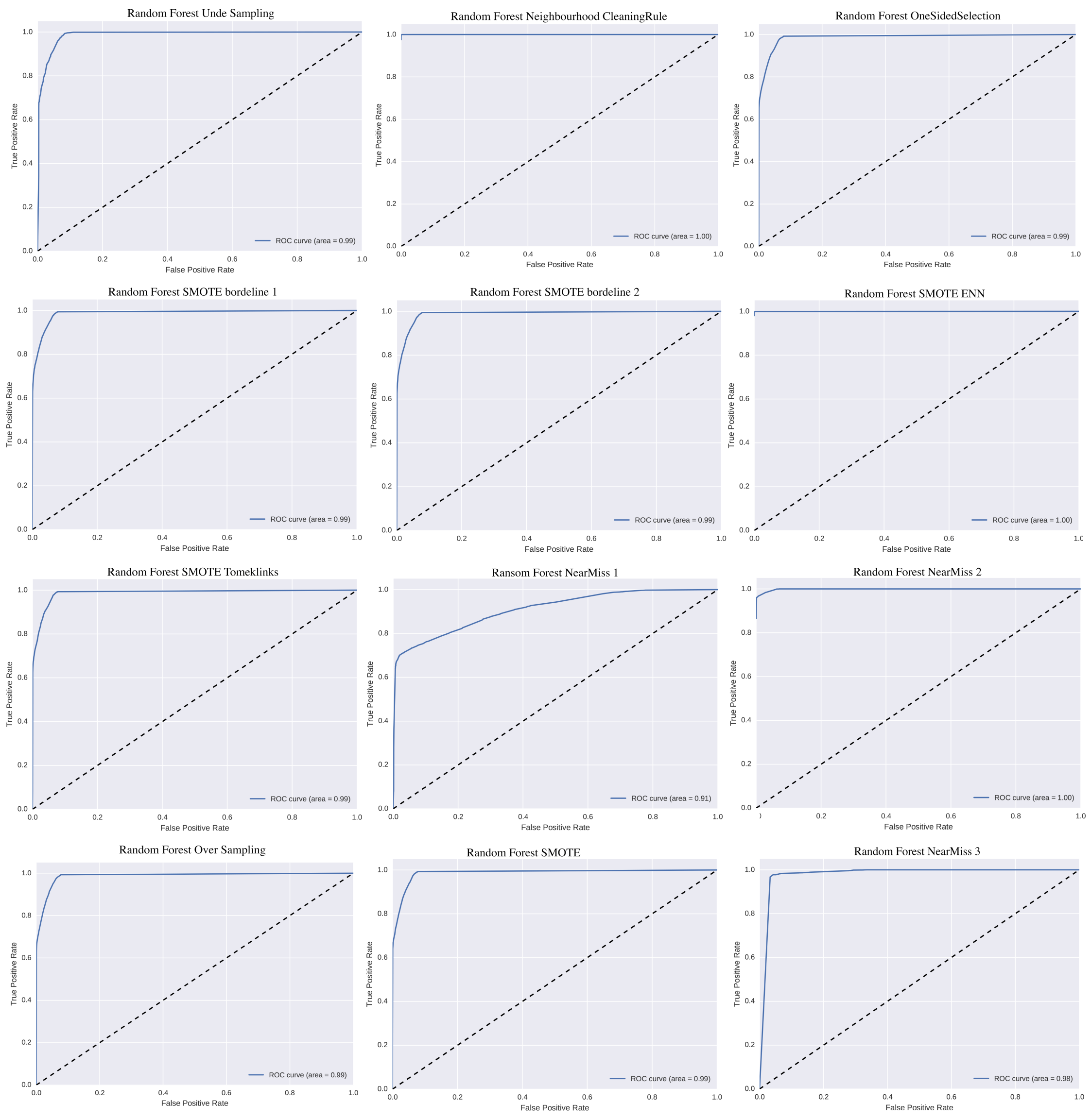


Figure 6.2 – ROC curves of different sampling methods imposed on the random forest algorithm. ROC curves represent the performance of binary classifiers over different cut-off points of the algorithm. The area under the curve is considered as a single number presenting the trade-off between sensitivity (true positive rate) and specificity (true negative rate).

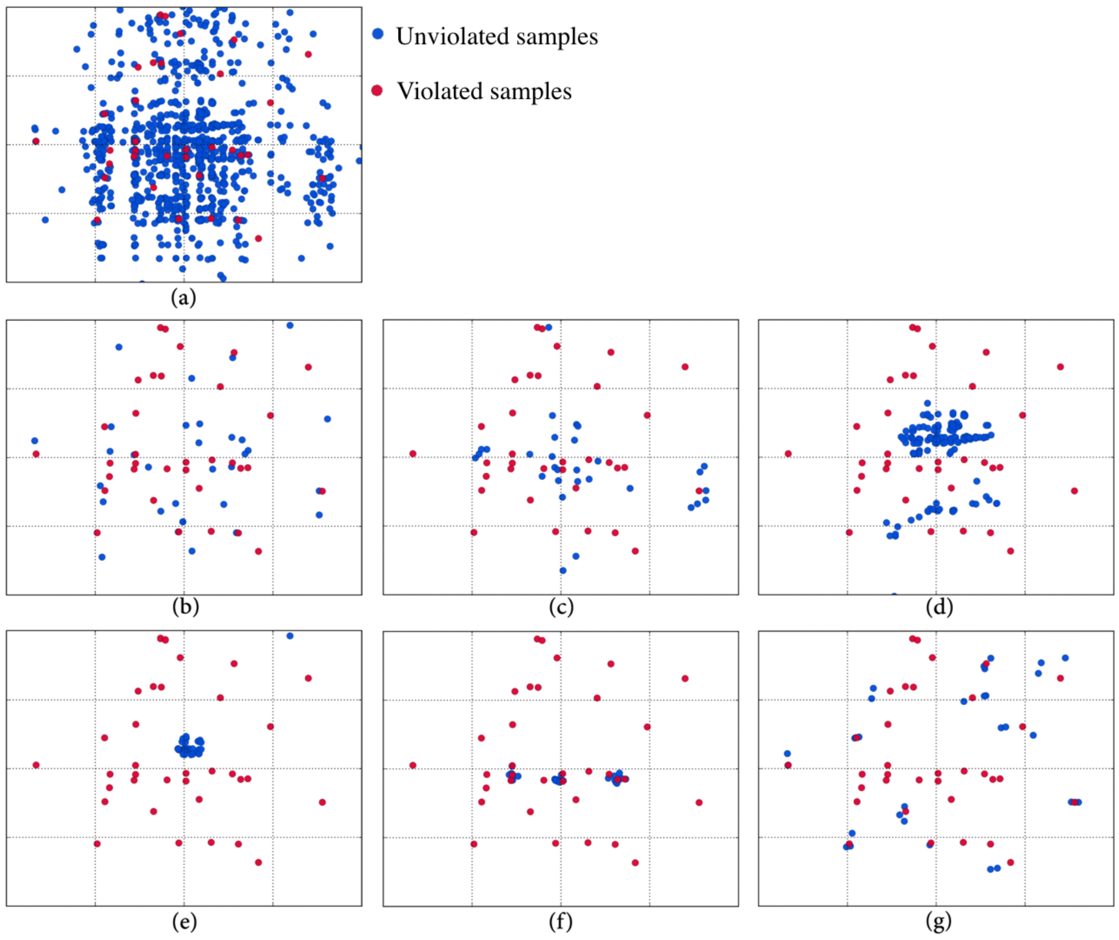


Figure 6.3 – A 2D t-SNE visualization of different under sampling methods. **(a)**: No Resampling, **(b)**: Random Under-Sampling, **(c)**: One-Sided Selection, **(d)**: Neighborhood Cleaning Rule, **(e)**: Near-Miss 1, **(f)**: Near-Miss 2, **(g)**: Near-Miss 3.

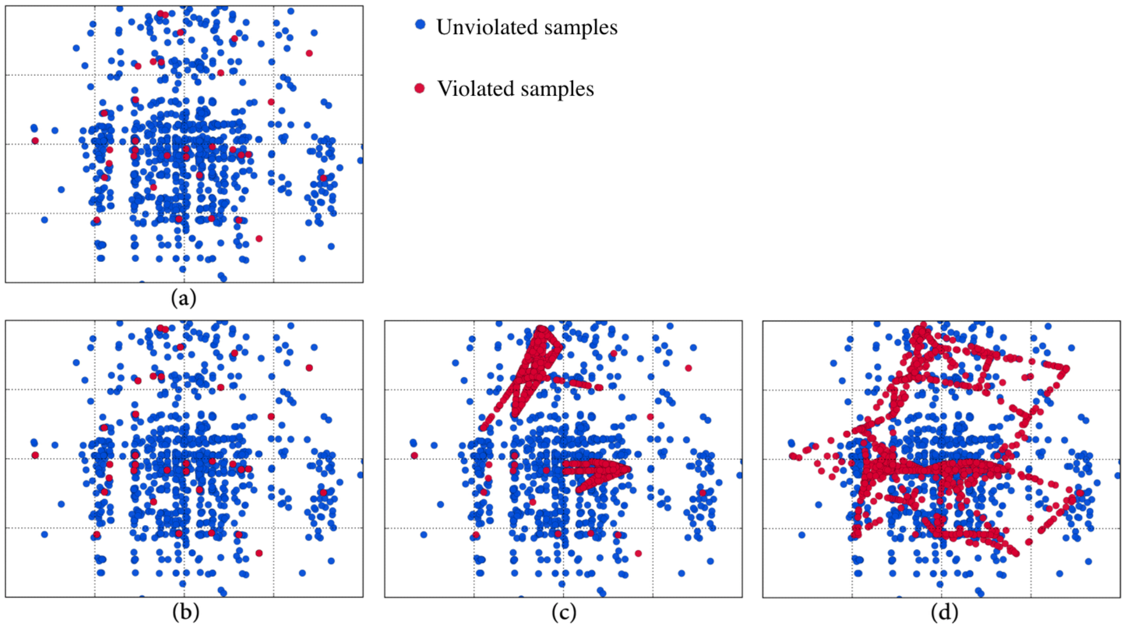


Figure 6.4 – A 2D t-SNE visualization of different over sampling methods. **(a)**: No Resampling, **(b)**: Random Over-Sampling, **(c)**: SMOTE Borderline 1 and 2, **(d)**: SMOTE. Note that the similarity between **(b)** and **(a)** is because Random Over-Sampling, duplicates the existing points randomly.

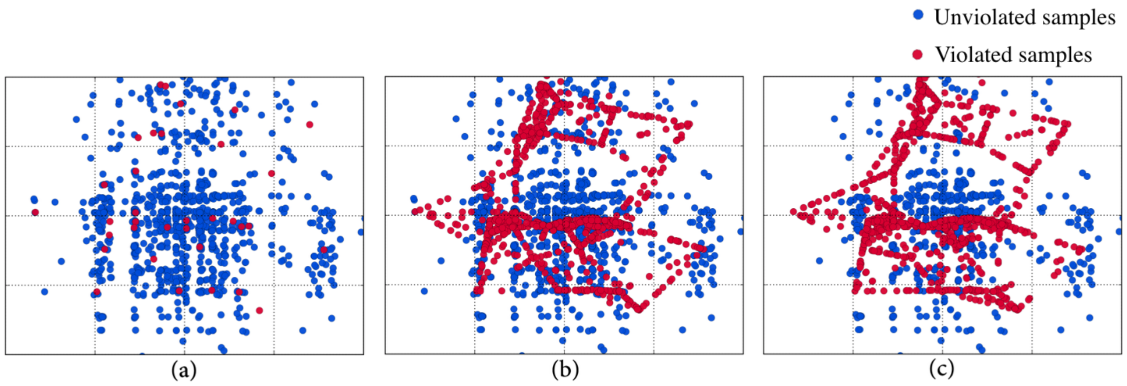


Figure 6.5 – A 2D t-SNE visualization of different combined sampling methods. **(a)**: No Resampling, **(b)**: SMOTE Tomek Links, **(b)**: SMOTE ENN

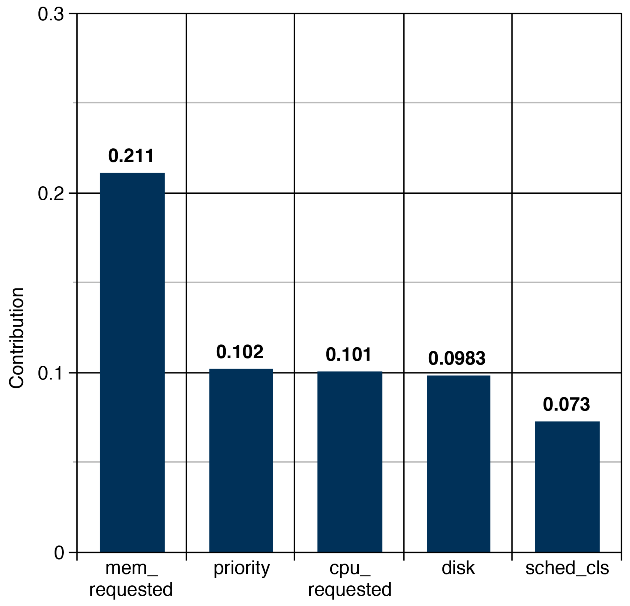


Figure 6.6 – The average contribution of each feature based on the best trained random forest algorithm. The average is taken over all test examples.

7 Conclusion

The thesis systematically compares the performance of two Machine Learning classification models on the task of SLA violation prediction. As discussed, in such a classification task, the data is skewed meaning that the number of violated tasks are much smaller than the number of unviolated ones. Consequently, the thesis also explores several methods of handling unbalanced data.

To summarize our contribution, we find that the Random Forest algorithm performs the best when SMOTE + ENN is used as a data re-sampling method. Among other related works on SLA violation prediction or avoidance or QoS management, our models are trained on a real world dataset which introduces new challenges that have been neglected in previous works (such as highly unbalanced classification problem), to the best of our knowledge. It is worth mentioning that the Random Forest model is human-interpretable and the results suggests that `mem_requested` is the most important feature in predicting violations. Moreover, thanks to the relatively high speed of random forest, it can be used in real-time prediction applications.

Despite the impressive results achieved by random forest, one drawback of random forest is that it is not trivial to update the knowledge representation of the model based on the new coming examples. One of the future works might be to explore other models that can be easily updated when receiving more training data. Another remaining question in the area of SLA violation avoidance is how to take advantage of the prediction of classifier in order to avoid violation. This work can also be extended using other machine learning models such as Support Vector Machines, and Neural Networks.

Bibliography

Akioka, S. and Y. Muraoka (2004). Extended forecast of cpu and network load on computational grid. In *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on*, pp. 765–772. IEEE.

Alhamad, M., T. Dillon, and E. Chang (2010). Sla-based trust model for cloud computing. In *Network-Based Information Systems (NBiS), 2010 13th International Conference on*, pp. 321–324. IEEE.

Ali-Eldin, A., M. Kihl, J. Tordsson, and E. Elmroth (2012). Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control. In *Proceedings of the 3rd workshop on Scientific Cloud Computing Date*, pp. 31–40. ACM.

AlJahdali, H., A. Albatli, P. Garraghan, P. Townend, L. Lau, and J. Xu (2014). Multi-tenancy in cloud computing. In *Service Oriented System Engineering (SOSE), 2014 IEEE 8th International Symposium on*, pp. 344–351. IEEE.

Attiya, H. and J. Welch (2004). *Distributed computing: fundamentals, simulations, and advanced topics*, Volume 19. John Wiley & Sons.

Banerjee, P., C. Bash, R. Friedrich, P. Goldsack, B. A. Huberman, J. Manley, C. Patel, P. Ranganathan, and A. Veitch (2011). Everything as a service: Powering the new information economy. *Computer 44*(3), 36–43.

Barnes, B. J., B. Rountree, D. K. Lowenthal, J. Reeves, B. De Supinski, and M. Schulz (2008). A regression-based approach to scalability prediction. In *Proceedings of the 22nd annual international conference on Supercomputing*, pp. 368–377. ACM.

Batista, G. E., A. L. Bazzan, and M. C. Monard (2003). Balancing training data for automated annotation of keywords: a case study. In *WOB*, pp. 10–18.

Batista, G. E., R. C. Prati, and M. C. Monard (2004). A study of the behavior of several methods for balancing machine learning training data. *ACM Sigkdd Explorations Newsletter 6*(1), 20–29.

Bishop, C. M. (2006). Pattern recognition. *Machine Learning 128.*

Breiman, L. (1996). Bagging predictors. *Machine learning 24*(2), 123–140.

Breiman, L. (2001). Random forests. *Machine learning 45*(1), 5–32.

Breiman, L., J. Friedman, C. J. Stone, and R. A. Olshen (1984). *Classification and regression trees.* CRC press.

Buyya, R., J. Broberg, and A. M. Goscinski (2010). *Cloud computing: Principles and paradigms*, Volume 87. John Wiley & Sons.

Carrington, L., A. Snavely, X. Gao, and N. Wolter (2003). A performance prediction framework for scientific applications. In *International Conference on Computational Science*, pp. 926–935. Springer.

Casalicchio, E. and L. Silvestri (2013). Mechanisms for sla provisioning in cloud-based service providers. *Computer Networks 57*(3), 795–810.

Chawla, N. V., K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer (2002). Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research 16*, 321–357.

Dabrowski, C. and F. Hunt (2008). Markov chain analysis for large-scale grid systems. Technical report, Draft NIST Internal Report.

Darmann, A., U. Pferschy, and J. Schauer (2010). Resource allocation with time intervals. *Theoretical Computer Science 411*(49), 4217–4234.

Degabriele, J. P. and D. Pym (2007). Economic aspects of a utility computing service. In *Proceedings of the first international conference on Networks for grid applications*, pp. 27. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Di, S., D. Kondo, and W. Cirne (2014). Google hostload prediction based on bayesian model with optimized feature combination. *Journal of Parallel and Distributed Computing 74*(1), 1820–1832.

Drummond, C. and R. C. Holte (2000). Exploiting the cost (in) sensitivity of decision tree splitting criteria. In *ICML*, pp. 239–246.

Duda, R. O., P. E. Hart, et al. (1973). *Pattern classification and scene analysis*, Volume 3. Wiley New York.

Emeakaroha, V. C., I. Brandic, M. Maurer, and S. Dustdar (2010). Low level metrics to high level slas-lom2his framework: Bridging the gap between monitored metrics and sla parameters in cloud environments. In *High Performance Computing and Simulation (HPCS), 2010 International Conference on*, pp. 48–54. IEEE.

Felsenstein, J. (1985). Confidence limits on phylogenies: an approach using the bootstrap. *Evolution*, 783–791.

Foster, I., Y. Zhao, I. Raicu, and S. Lu (2008). Cloud computing and grid computing 360-degree compared. In *2008 Grid Computing Environments Workshop*, pp. 1–10. Ieee.

Freund, Y. and R. E. Schapire (1995). A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pp. 23–37. Springer.

Friedman, J., T. Hastie, and R. Tibshirani (2001). *The elements of statistical learning*, Volume 1. Springer series in statistics Springer, Berlin.

Gallizo, G., R. Kuebert, K. Oberle, A. Menychtas, and K. Konstanteli (2009). Service level agreements in virtualised service platforms. *eChallenges 2009, Istanbul, Turkey*.

Ganapathi, A., Y. Chen, A. Fox, R. Katz, and D. Patterson (2010). Statistics-driven workload modeling for the cloud. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pp. 87–92. IEEE.

Ganganwar, V. (2012). An overview of classification algorithms for imbalanced datasets. *International Journal of Emerging Technology and Advanced Engineering 2*(4), 42–47.

Geurts, P. (2002). *Contributions to decision tree induction: bias/variance tradeoff and time series classification.* Ph. D. thesis, University of Liège Belgium.

Gowda, K. C. and G. Krishna (1979). The condensed nearest neighbor rule using the concept of mutual nearest neighborhood. *IEEE Transactions on Information Theory 25*(4), 488–490.

Han, H., W.-Y. Wang, and B.-H. Mao (2005). Borderline-smote: a new over-sampling method in imbalanced data sets learning. In *International Conference on Intelligent Computing*, pp. 878–887. Springer.

Jones, E., T. Oliphant, P. Peterson, et al. (2001). Open source scientific tools for python.

Jules, O., A. Hafid, and M. A. Serhani (2014). Bayesian network, and probabilistic ontology driven trust model for sla management of cloud services. In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, pp. 77–83. IEEE.

Kaur, H. and B. Singh (2015). International journal in applied studies and production management.

Kephart, J. O. and D. M. Chess (2003). The vision of autonomic computing. *Computer 36*(1), 41–50.

Krebs, R., C. Momm, and S. Kounev (2012). Architectural concerns in multi-tenant saas applications. *CLOSER 12*, 426–431.

Kubat, M., S. Matwin, et al. (1997). Addressing the curse of imbalanced training sets: one-sided selection. In *ICML*, Volume 97, pp. 179–186. Nashville, USA.

Laurikkala, J. (2001). Improving identification of difficult small classes by balancing class distribution. In *Conference on Artificial Intelligence in Medicine in Europe*, pp. 63–66. Springer.

Leavitt, N. (2009). Is cloud computing really ready for prime time. *Growth 27*(5), 15–20.

Leitner, P., B. Wetzstein, F. Rosenberg, A. Michlmayr, S. Dustdar, and F. Leymann (2010). Runtime prediction of service level agreement violations for composite services. In *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, pp. 176–186. Springer.

Lemaître, G., F. Nogueira, and C. K. Aridas (2016). Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *CoRR abs/1609.06570*.

Ludwig, H., A. Keller, A. Dan, R. P. King, and R. Franck (2003). Web service level agreement (wsla) language specification. *IBM Corporation*, 815–824.

Maaten, L. v. d. and G. Hinton (2008). Visualizing data using t-sne. *Journal of Machine Learning Research 9*(Nov), 2579–2605.

Mani, I. and I. Zhang (2003). knn approach to unbalanced data distributions: a case study involving information extraction. In *Proceedings of workshop on learning from imbalanced datasets*.

McCallum, A., K. Nigam, et al. (1998). A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, Volume 752, pp. 41–48. Citeseer.

Meegan, J., G. Singh, S. Woodward, S. Venticinque, M. Rak, D. Harris, and G. Malekkos (2012). Practical guide to cloud service level agreements version 1.0. *Cloud Standards Customer Council 36*.

Mell, P. and T. Grance (2011). The nist definition of cloud computing.

Mitchell, T. M. et al. (1997). Machine learning. wcb.

Moore, D. S. and G. P. McCabe (1989). *Introduction to the Practice of Statistics.* WH Freeman/Times Books/Henry Holt & Co.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective.* MIT press.

Opitz, D. and R. Maclin (1999). Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research 11*, 169–198.

Oshiro, T. M., P. S. Perez, and J. A. Baranauskas (2012). How many trees in a random forest? In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pp. 154–168. Springer.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12*, 2825–2830.

Provost, F. (2000). Machine learning from imbalanced data sets 101. In *Proceedings of the AAAI'2000 workshop on imbalanced data sets*, pp. 1–3.

Rappa, M. A. (2004). The utility business model and the future of computing services. *IBM Systems Journal 43*(1), 32.

Reiss, C., J. Wilkes, and J. L. Hellerstein (2011). Google cluster-usage traces: format+ schema. *Google Inc., White Paper*, 1–14.

Salton, G. and C. Buckley (1988). Term-weighting approaches in automatic text retrieval. *Information processing & management 24*(5), 513–523.

Sammut, C. and G. I. Webb (2011). *Encyclopedia of machine learning.* Springer Science & Business Media.

Schum, D. A. (1994). *The evidential foundations of probabilistic reasoning.* Northwestern University Press.

Stehman, S. V. (1997). Selecting and interpreting measures of thematic classification accuracy. *Remote sensing of Environment 62*(1), 77–89.

Sturm, R., W. Morris, and M. Jander (2000). {Foundations of Service Level Management}.

Tomek, I. (1976). Two modifications of cnn. *IEEE Trans. Systems, Man and Cybernetics 6*, 769–772.

Uriarte, R. B., S. Tsaftaris, and F. Tiezzi (2015). Service clustering for autonomic clouds using random forest. In *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, pp. 515–524. IEEE.

Van Der Walt, S., S. C. Colbert, and G. Varoquaux (2011). The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering 13*(2), 22–30.

Vapnik, V. N. (1999). An overview of statistical learning theory. *IEEE transactions on neural networks 10*(5), 988–999.

Vogel, A., B. Kerherve, G. von Bochmann, and J. Gecsei (1995). Distributed multimedia and qos: A survey. *IEEE multimedia 2*(2), 10–19.

Wang, L., G. Von Laszewski, A. Younge, X. He, M. Kunze, J. Tao, and C. Fu (2010). Cloud computing: a perspective study. *New Generation Computing 28*(2), 137–146.

Wang, Y. and J. Vassileva (2007). A review on trust and reputation for web service selection. In *Distributed Computing Systems Workshops, 2007. ICDCSW'07. 27th International Conference on*, pp. 25–25. IEEE.

Wasserman, L. (2013). *All of statistics: a concise course in statistical inference.* Springer Science & Business Media.

Williams, B. (2012). *The economics of cloud computing.* Cisco Press.

Wilson, D. L. (1972). Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics* (3), 408–421.

Wu, L., S. K. Garg, and R. Buyya (2011). Sla-based resource allocation for software as a service provider (saas) in cloud computing environments. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pp. 195–204. IEEE.

Wu, Q., X. Zhang, M. Zhang, Y. Lou, R. Zheng, and W. Wei (2014). Reputation revision method for selecting cloud services based on prior knowledge and a market mechanism. *The Scientific World Journal 2014*.

Zhang, C. and J. Cao (2013). A dynamic resource management strategy for cloud based on sequential decision model and demand forecast. In *Cloud and Service Computing (CSC), 2013 International Conference on*, pp. 77–83. IEEE.

Zhang, Q., L. Cheng, and R. Boutaba (2010). Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications 1*(1), 7–18.

Zhang, Q., J. Hellerstein, and R. Boutaba (2011). Characterizing task usage shapes in google compute clusters.

Zheng, Z., R. Wang, H. Zhong, and X. Zhang (2011). An approach for cloud resource scheduling based on parallel genetic algorithm. In *Computer Research and Development (ICCRD), 2011 3rd International Conference on*, Volume 2, pp. 444–447. IEEE.