Πανεπιστήμιο Πειραιώς
Τμήμα Ψηφιακών Συστημάτων

Collaborative Filtering for Implicit Feedback Datasets

Δημιουργία Συστήματος Συνεργατικού Φιλτραρίσματος για Σύνολα Δεδομένων Έμμεσης Ανατροφοδότησης

MSc Dissertation for the postgraduate program

"Digital Systems and Services"

by

Poulopoulos Dimitrios

Supervisor: Kyriazis Dimosthenis

Piraeus, March 2017

Acknowledgements

This MSc dissertation completes the cycle of the postgraduate program “Digital Systems and Services”, University of Piraeus.

I would like to thank my family for supporting me in every step and in any way, and my partner Evi, for standing by me through this exciting and fulfilling journey.

Last but not least, I would like to thank the faculty of University of Piraeus, and especially my professors Dimostheni Kyriazi, Christo Doulkeridi and Marino Themistocleous, for their invaluable guidance and counsel.

University of Piraeus

Department of Digital Systems

Postgraduate program “Digital Systems and Services”

Poulopoulos Dimitrios

Preface

Recommender systems, or recommendation systems, is a way to predict the behavioral patterns of a user, his preferences or dislikes, in order to provide personalized recommendations. These systems work based on prior explicit or implicit feedback.

The much more extensively researched explicit feedback systems gather their knowledge directly from the users, in the form of a simple rating. For example, Netflix uses a rating scale, from one to five stars, to determine whether a user enjoyed a specific movie or not. On the other hand, implicit feedback systems work passively in the background, tracking different sorts of user behavior, such as browsing activity, watching habits or purchase history.

Thus, in the case of implicit feedback systems, we do not have any direct indication of the user’s preferences and, specifically, we do not have any significant evidence on which items a user dislikes.

The aim of this study is to analyze the unique properties of implicit feedback datasets, especially tailored to fit the retail sector, confronting a vast variety of consumer products, their price differences and user-item interaction sparsity.

Πανεπιστήμιο Πειραιώς

Τμήμα Ψηφιακών Συστημάτων

Π.Μ.Σ. “Ψηφιακά Συστήματα και Υπηρεσίες”

Πουλόπουλος Δημήτριος

Πρόλογος

Τα συστήματα συστάσεων ή προτάσεων, αποτελούν ένα τρόπο για την κατανόηση των προτύπων συμπεριφοράς ενός χρήστη, τις προτιμήσεις του, με σκοπό τη δημιουργία προσωποποιημένων συστάσεων. Τα συστήματα αυτά, λειτουργούν βασισμένα σε ιστορικά στοιχεία, τα οποία αποτελούνται από σύνολα δεδομένων άμεσης ή έμμεσης ανατροφοδότησης.

Τα σύνολα δεδομένων άμεσης ανατροφοδότησης, εξάγουν τα στοιχεία τους κατευθείαν από τους χρήστες, για παράδειγμα μέσω μιας μορφής βαθμολόγησης. Για παράδειγμα, το Netflix χρησιμοποιεί μια κλίμακα βαθμολόγησης, μεταξύ ενός και 5 άστρων, με σκοπό να εξάγει από το χρήστη άμεσα, αν διασκέδασε με μια ταινία ή όχι. Αντίθετα, τα συστήματα που βασίζονται σε έμμεση ανατροφοδότηση, λειτουργούν σιωπηλά στο περιθώριο, συλλέγοντας δεδομένα που μπορεί να αποκαλύψουν τις προτιμήσεις ενός χρήστη, όπως τις συνήθειες του καθώς πλοηγείται στο διαδίκτυο, ή το ιστορικό αγορών του κ.α..

Με αυτό τον τρόπο, στην περίπτωση ενός συστήματος που βασίζεται σε δεδομένα έμμεσης ανατροφοδότησης, δεν έχουμε μια ισχυρή και άμεση ένδειξη για τις προτιμήσεις του κάθε χρήστη. Συγκεκριμένα, δεν έχουμε καμία ένδειξη για την οποιαδήποτε αρνητική εμπειρία είχε ένας συγκεκριμένος χρήστης.

Ο σκοπός της συγκεκριμένης εργασίας είναι να αναλύσει τις ιδιότητες των συνόλων δεδομένων έμμεσης ανατροφοδότησης, συγκεκριμένα για τον τομέα των πωλήσεων.

Table of Contents

Table of Figures

Definitions

Chapter One

1 Introduction

With the expansion of internet technologies, as well as the mobile and cloud computing, in a huge variety of application areas data is being produced, processed and consumed at unprecedented scale. This tendency comes with enormous challenges and opportunities, to which researchers, and also the industry sector, respond with great excitement and eagerness, searching for insightful ways to extract useful information from these data. Such big data analysis is everywhere and fuels our modern society, showing its influence on a broad range of services, including, but not limited to, retail, financial services and life sciences.

Gartner hype cycle for emerging technologies shows a movement toward services, which either frame or utilize the so-called world of big data, to create immersive, personalized experiences in the form of personal assistants and cognitive advisors or machine learning applications such as recommendation systems (Figure 1-1).

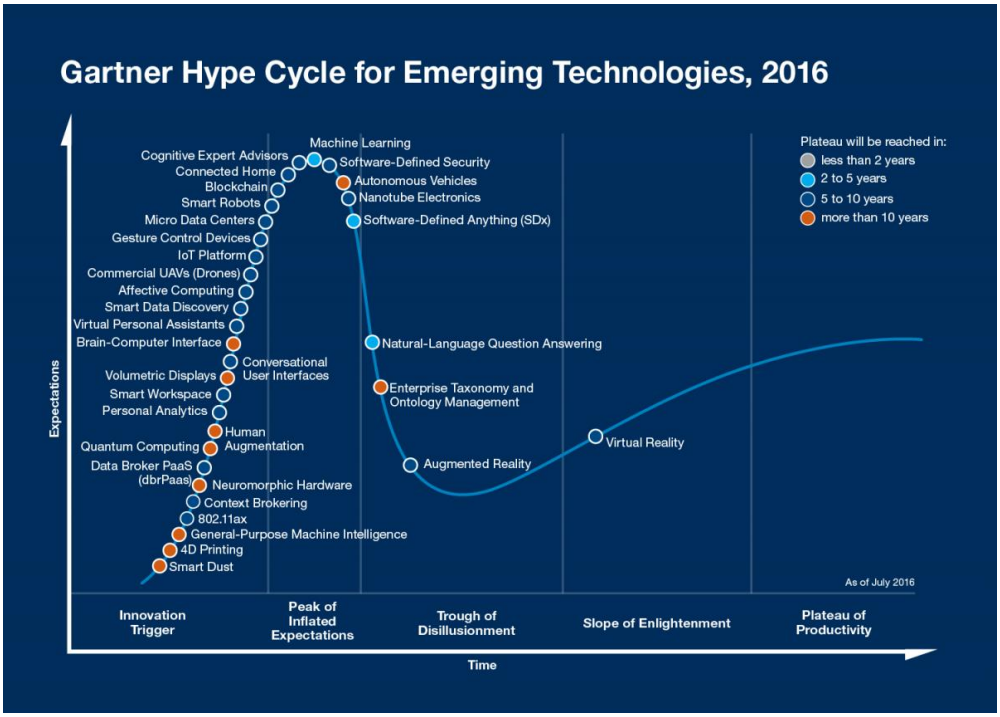


Figure 1-1 - Gartner Hype Cycle for Emerging Technologies, 2016

This need comes from the fact that people are awash in a flood of data, which converts any information into noise. To surpass this obstacle, big organizations, that either create or manage content, turn to solutions provided by machine learning and recommendation systems to create a much more personalized and tailored experience for their users.

1.1 Problem statement

The field of machine learning has been developing at a high pace the recent years, yielding excellent results both in the scientific and industry fields. It is considered an active area of research, but at the same time commercial products, applications and services are being created and utilized everywhere in our modern society.

Definition 1.1: Machine Learning

Machine learning is the subfield of computer science that gives computers the ability to learn without being explicitly programmed [1]***.***

Machine learning has many subfields, such as computer vision, natural language processing or speech recognition. It evolved from the study of pattern recognition, although the origins of the latter can be traced in engineering rather than computer science. It explores the design and implementation of algorithms that can learn from data (training examples) and make predictions about the state of the environment given a specific input.

1.1.1 Recommender systems

Recommender systems, or recommendation systems, aim to discover behavioral patterns, in order to provide a personalized experience to the users.

Definition 1.2: Recommender Systems

Recommender Systems (RSs) are software tools and techniques providing suggestions for items to be of use to a user [2].

Chris Anderson, who is widely-known for his 2004 article entitled “The Long Tail”, said that we are leaving the age of information and entering the age of recommendation [3]. He stated that in the context of the information overload that we absorb every day, arguing that unless we have a way to filter it out and extract what is important to us, information reduces to useless noise.

In his book “The Paradox of Choice”, the American psychologist Barry Schwartz presents many examples and experiments, which show why having many options is not always a good idea. He

concludes that more is less [4]. In his famous experiment, his team put six different flavors of jam in a store and prompt the clients to taste and buy. In the next phase, they used more than 60 flavors of jam and, interestingly enough, the sales of the store dropped.

Moreover, it seems that we are leaving the era of search and entering the era of discovery. The difference is that when searching, a user is actively looking for something. Discovery means that something the user did not know existed, or even did not know how to ask for it, finds him [5].

However, the meaning of the word recommendation remains an area of confusion in the minds of many. Recommendation should not be interpreted as personalized search, where the user forms an explicit query using filters to extract the desirable result. In recommendation, the query is implicit, and it formulates by the user and the context.

1.1.2 The value of recommendation

Maybe the first example that comes to mind when talking for recommender systems is that of Netflix. This organization owes a lot to the development of such algorithms. By 2014 over 65% of the movies watched on Netflix are recommended. Today, the percentage has raised to roughly around 80%.

Other organization has also discovered the benefits of such systems. Google news has seen a 38% boost to its click through rate, while Amazon reported that 35% of its sales come from recommendations.

1.1.3 The recommendation problem

The recommendation problem can be mathematically described by a utility function, that is trying to estimate or predict how a user would like an item.

This can be achieved by looking on user's previous behavior, his relationship to other users, item similarity, context and many more kind of information.

Thus, more formally, let $\boldsymbol{U}$ be a set of all users and let be $\boldsymbol{I}$ a set of every possible recommendable item. Let $\boldsymbol{f}$ be the utility function, measuring the preference that a user $\boldsymbol{u}$ would show to an item $\boldsymbol{i}$.

$$\boldsymbol{f : U \times I \rightarrow R}$$

Where $\boldsymbol{R}$ is a totally ordered set.

For each user $\boldsymbol{u} \in \boldsymbol{U}$ the system chooses items $\boldsymbol{i} \in \boldsymbol{I}$ that maximize the utility function $\boldsymbol{f}$**.** The value of the utility function $\boldsymbol{f}$ is usually a scalar number, represented by rating. Finally, the new set $\boldsymbol{I}'$ satisfies the following statement:

$$\forall\, \boldsymbol{u} \in \boldsymbol{U}, \boldsymbol{I'}_{\boldsymbol{u}} = \boldsymbol{argmax}\big(\boldsymbol{f}(\boldsymbol{u}, \boldsymbol{i})\big), \boldsymbol{i} \in \boldsymbol{I}$$

1.1.4 The recommendation pipeline

The recommendation process is a two-step process, where the first half is an offline process and second half comes online, and in many cases real-time.

During the offline process, also called the learning process, the system tries to train a machine learning algorithm, and build a model from the data it has. Similarly, the learning algorithms may produce useful clusters, which can be used for market segmentation and better understanding the users' needs.

The online process is where the recommendation happens. A user connects to the system and the recommendations are being formulated based on a specific context (Figure 1-2).

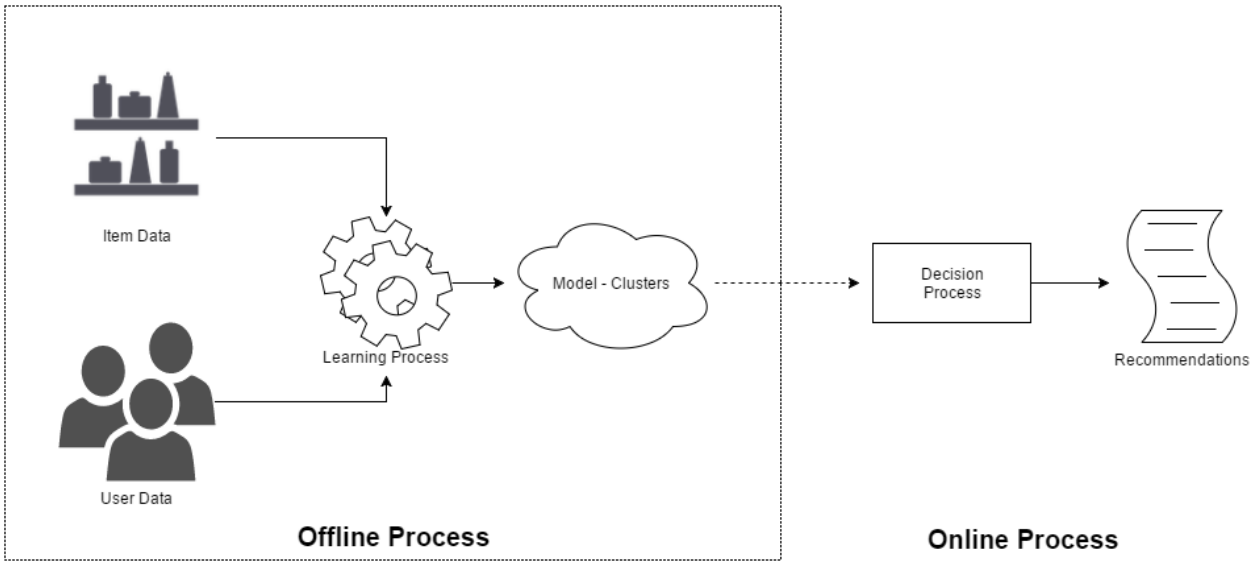


Figure 1-2 - The recommendation pipeline

1.1.5 Approaches to recommendation

There are many approaches on how to build a recommendation system and each has its advantages and weaknesses. The most common approach is that of collaborative filtering. With this approach recommendations are generated based on the users' past behavior. This approach can be either user-based or item-based.

Using user-based collaborative filtering, the system tries to discover similarities between users and recommend common favorable items. On the other hand, in item-based collaborative filtering, the engine tries to find similar items to those a user already has expressed a preference, and form the recommendations. The item-oriented approach is favorable in many cases [6] [7] [8], because of better scalability and improved accuracy. Furthermore, it's easier to explain the reasoning behind the predictions of item-based approaches. This work mostly explores the field of collaborative filtering.

Other approaches are those of content-based algorithms, where recommendations are based on item features, demographic, which recommends items based on user features, social recommendations, which take advantage of trust-based structures that take shape in social networks and hybrid approaches, which combine any of the above.

1.1.6 Explicit Vs Implicit feedback

The kind of data that recommender systems utilize to learn are of two kinds. Implicit feedback and explicit feedback. Explicit feedback means that the system takes advantage of direct feedback from the user. The user willingly express his preference or dislike about an item usually in the form of a scalar number, a rating. In the case of implicit feedback, the system tries to disclose user preferences, monitoring different sorts of behavior, such as browsing activity or purchases. Sources of implicit feedback that have been explored in the past are time spent reading [9] or URL references in Usenet postings [10].

Although it may seem that explicit feedback has an advantage, in most cases, the opposite is true. Implicit feedback is easier to extract, immediately applicable, readily available and in most cases more accessible. Also, implicit feedback usually consists of much more data and, even though it seems counter intuitively, explicit data are noisy. For example, a Netflix user might give a great movie a rating of five stars, but this is something that is called an aspirational rating. That does not mean, though, that this is the kind of entertainment he calls for after a long and tiring day. In this case a user is more likely to ask for a comedy or something that will allow him to relax, but will only rate with two or three stars. Thus, there is some sort of contradiction between the explicit information that the system gets from the users and what the users want to watch.

1.2 Research objectives

This work explores the mechanisms of extracting meaningful implicit feedback, from user-item interactions, and ways to transform and use it to create a system of recommendations tailored to fit the world of retail. In such cases, gathering explicit information is either impossible or too expensive, because it is time and resource consuming.

It is important to analyze the unique characteristics of implicit feedback datasets, which prevent the application of already successful algorithms, that were designed with explicit feedback datasets in mind [11].

i. By observing the users' behavioral patterns, we can extract with some confidence that someone likes the item he is interacting with. But, in such cases, there is no negative feedback. For example, if a user chooses not to purchase a product, this does not mean that he dislikes it. It may very well be the case that he does not even know the product's existence, or that it is too expensive for him now. Nevertheless, it might still be in his wish list. On the other hand, a user might purchase a product as a gift, not showing his true intentions. Furthermore, in the case of explicit feedback, there is an indication whether a user likes a product or not. This allows the algorithms to treat every other field, the empty ratings, as missing data. That is not possible in the case of implicit feedback, because concentrating only on the available information leaves most negative feedback untouched, as it is expected to be found in missing data.
ii. Implicit feedback can be inherently noisy. As shown in the aforementioned example, purchasing an item does not necessarily show preference, because it may be a gift.
iii. The numerical output of an algorithm using explicit feedback datasets indicates preference, whereas the numerical output using implicit information indicates confidence. A recurring event is more likely to capture the user's opinion, but the output given by an algorithm using implicit feedback datasets is merely the probability that a user will like an item.
iv. Evaluation of such algorithms also differs. In the presence of ratings, a clear metric like the mean squared error is sufficient. However, in our case, parameters like stock or product competition should be considered.

1.3 Document outline

This work consists of six chapters that cover in full length the development of this research thesis on recommender systems.

Chapter One, Introduction: This chapter specified the problem statement and brought forward the research objectives for the successful completion of the dissertation.

Chapter Two, Theoretical Background: This chapter presents the theoretical background about recommender systems and machine learning in general. It describes the learning problem, argues that learning is in fact feasible, discusses error and noise and explains the theory of generalization.

Chapter Three, Related Work: In this section State of the Art techniques, already in use, are presented. This chapter forms an extended report of recommender system techniques, compare them and emerge open research areas that exist in the field.

Chapter Four, Proposed Model: In this chapter, the proposed model for building a recommendation system, using implicit feedback datasets and tailored to fit the retail sector is presented. The model is described and analyzed in in full detail.

Chapter Five, Experimental Study: In this chapter the evaluation process of the proposed model is examined and presented. The chosen criteria are explained and the results of this process are discussed.

Chapter Six, Summary and Future Research: The final section of this document presents the summary and conclusions of this research. Also, it argues on possible open areas of future research and modifications.

Chapter Two

2 Theoretical background

The problem of pattern recognition is an ancient one and dates back to the age of the first astronomical observations. The field is concerned with the automatic discovery of arrangements in data through computer algorithms. Using these regularities in data machine learning systems can learn to classify objects, predict values or detect useful information otherwise invisible.

2.1 The components of machine learning

Machine learning, in its foundations, has three components. When those three ingredients exist, then machine learning techniques are ready to be applied and utilized.

The first of these components is that the existence of a pattern. If no pattern exists, someone could still try and apply machine learning techniques, but most likely he would be trapped in a dead-end.

The second component is that there is a target function, which the system tries to compute and mimic its behavior. But, certainly, will never exactly discover it. In other words, there is a pattern but we cannot describe it mathematically. If we could, machine learning would not be of much use, because we would have a simple mathematical formula to apply.

The last, and most important, component is the data itself. We need to have data on the problem, because machine learning, in its essence, is trying to learn from data.

2.2 The “movie” example

In this example, we try to predict the preference of a user for a movie, in other words, how a user would rate a particular movie. A simple solution is described by the figure below (Figure 2-1):

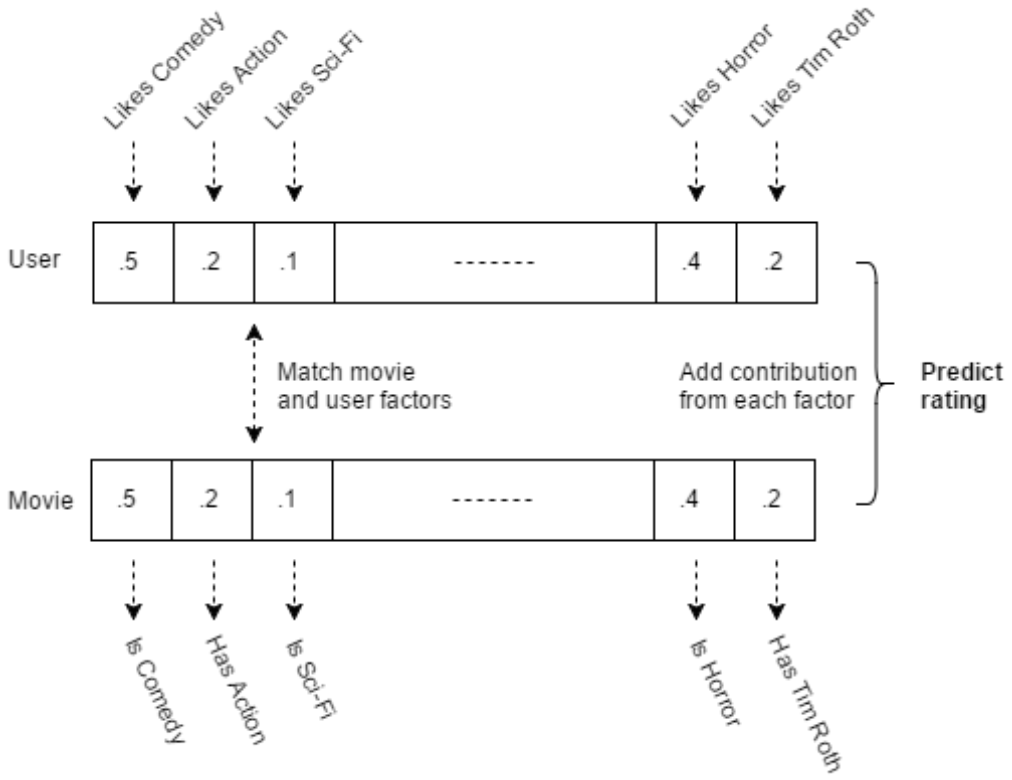


Figure 2-1 - The "movie" example

In this example, a user is described as a vector of factors, comprising a user profile. These factors express the user's preferences for certain movie characteristics, e.g. does the user likes comedy? Does he prefer blockbusters?

On the other side, there are movies which are described by corresponding characteristics, features, e.g. if they are a comedy or has a lot of action.

The system compares the active user to the movie, match the movie and user factors, add the contribution from each factor and predicts how this user would rate the movie. If this rating exceeds a specific threshold, the system decides that the user would like this movie and proceeds to recommend it.

The problem with the previous approach is that this is not machine learning. To create such a system, an engineer would have to watch every movie, extract their characteristics and then, interview each user separately, asking them for their preferences.

The learning approach reverse engineers the process that was mentioned above. It starts from the rating and then tries to find which factors would be consistent with that rating. The system randomly initializes the movie and user factors. This would produce an error in the calculation of the rating that actually took place. Then, it tries to gently manipulate those factors in order to

reduce the error. This process is going on for thousands of ratings and hundreds of iterations over them, until the algorithm converges, meaning that it has minimized the error below a certain acceptable value. In this case, the factors are meaningful and consistent with the ratings, thus, the system can predict how a new viewer would rate an unseen before movie.

2.3 Types of Machine Learning

Machine Learning is a broad subject and is divided in three main categories. There are also subsets of these categories, but we can argue that these are subfields or hybrids of the main three ones [12].

The first approach is called ***supervised learning***. In such applications, the data that are used to train the machine learning algorithms take the form of examples. In other words, the input vectors come along with their corresponding target vectors, in an input-output (result) relationship. The field of supervised learning is further divided by classification and regression. In classification, the aim is to assign each input to a finite number of labels. On the other hand, when the system tries to come up with a real-valued output, such us predicting the value of stock prices, the task is called regression.

Other machine learning problems fall under the field of ***unsupervised learning***. In this approach, the data consist of the input but there is no corresponding, target output. Unsupervised learning tries to discover groups of similar examples inside the data, a technique that is called clustering. Other types of unsupervised learning are density estimation, which determines the distribution of data within the input space, and dimensionality-reduction, which projects data from a high-dimensional space to a lower dimension space, for compression or visualization.

The third field of machine learning is called ***reinforcement learning*** [13]. This problem is concerned with taking certain steps in order to maximize a reward, a feedback that is given to the system. The process is therefore explained by the trial and error paradigm. Appropriate examples of such approaches can be found in the gaming industry, where AI agents, which learn to play the game using reinforcement learning techniques, are playing the role of user opponents. Other examples, of reinforcement learning systems working in the real world, can be found in energy efficiency optimization programs, where Google is using similar techniques to reduce the cost for cooling data centers by 40%.

2.4 A probabilistic approach to learning

In machine learning and pattern recognition problems there is a target function $\boldsymbol{f}$, from which the training examples are generated from. When the system uses a machine learning algorithm, it tries to come up with a hypothesis function $\boldsymbol{h}$, that approximates the target function well.

The target function is unknown and it remains unknown throughout the process of learning. Thus, even if the system forms a hypothesis function that approximates the training data points generated by $\boldsymbol{f}$, the target function can still take any value outside the data given to the system.

Consequently, the question is if we can infer something outside the sample data that we have. Is learning feasible?

2.4.1 A simple probabilistic experiment

Consider the following situation; there is a box filled with black and white tennis balls. We are going to generate a sample from this box (Figure 2-2).

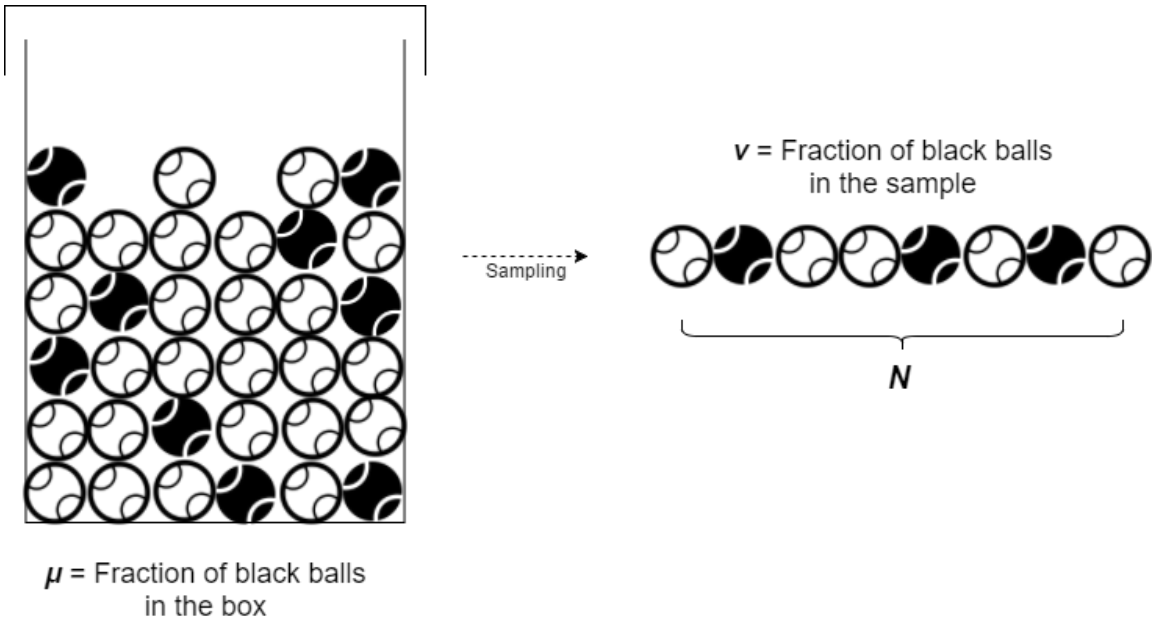


Figure 2-2 - Tennis balls probabilistic experiment

Formally, let $\boldsymbol{\mu}$ be the probability of picking a black tennis ball. So,

$$\boldsymbol{P}[\,\boldsymbol{picking\ a\ black\ ball}] = \boldsymbol{\mu}$$

The probability $\boldsymbol{\mu}$ is independent of the repetitions of the experiment, so every time we reach into the box, the probability of picking a black ball is $\boldsymbol{\mu}$. According to this, the probability of picking a white tennis ball is $\mathbf{1} - \boldsymbol{\mu}$**.** So,

$$\boldsymbol{P}[\ \boldsymbol{picking\ a\ white\ ball}] = \mathbf{1} - \boldsymbol{\mu}$$

The value of $\boldsymbol{\mu}$ is unknown to us. Sampling from the box, we pick $\boldsymbol{N}$ balls independently. So now, we call the frequency of black tennis balls in the sample, the fraction of black tennis balls, $\boldsymbol{\nu}$. The question is: Does $\boldsymbol{\nu}$ says anything about $\boldsymbol{\mu}$? In other words, can we depend on the fraction of black balls in sample to make inferences about the fraction of black balls in the box (population)?

2.4.2 Possible Vs Probable

In science and engineering, we can go a huge distance by settling not for certain but almost certain. It opens a world of possibilities. In accordance to probability theory, if the number of samples is big enough, the sample frequency $\boldsymbol{\nu}$ is likely to be close to the population frequency $\boldsymbol{\mu}$.

More formally, having a large number of samples, $\boldsymbol{N}$, makes it possible to consider the sample frequency $\boldsymbol{\nu}$ to be within a tolerance level $\boldsymbol{\varepsilon}$ from the population frequency $\boldsymbol{\mu}$.

Thus, per Hoeffding's inequality, the probability that $\boldsymbol{\nu}$ does not track $\boldsymbol{\mu}$ out of sample is:

$$\boldsymbol{P}[\ |\boldsymbol{\nu} - \boldsymbol{\mu}| > \boldsymbol{\varepsilon}|\ ] \leq\ \mathbf{2}\boldsymbol{e}^{-\mathbf{2}\boldsymbol{\varepsilon}^{\mathbf{2}}\boldsymbol{N}}$$

This inequality perfectly demonstrates the importance of a big sample. According with the aforementioned formula, this probability can be small number because of the negative exponential on the right side. However, the smaller out tolerance level $\boldsymbol{\varepsilon}$ is, the greater this number becomes, because the $\boldsymbol{e}^{\mathbf{2}}$ in the exponent will severely dampen the number $\boldsymbol{N}$. Nevertheless, using the Hoeffding's inequality we can say that $\boldsymbol{\nu}$ is probably, approximately, equal to $\boldsymbol{\mu}$. Probably because of the inequality and approximately because of $\boldsymbol{\varepsilon}$.

The Hoeffding's inequality is valid for every positive integer $\boldsymbol{N}$ and every $\boldsymbol{\varepsilon}$ greater than zero. Moreover, the Hoeffding's inequality bound does not depend on the unknown to us quantity $\boldsymbol{\mu}$. The tradeoff happens between $\boldsymbol{N}$ and $\boldsymbol{\varepsilon}$. Usually, the number of sample points $\boldsymbol{N}$ is predetermined from the data we have in our disposal. So, the number of $\boldsymbol{\varepsilon}$ we choose as our tolerance, should be such that the number of sample data we have can compensate.

Finally, the accurate statement would be that $\boldsymbol{\nu}$ tends to be close to $\boldsymbol{\mu}$. This means that $\boldsymbol{\mu}$ affects the way $\boldsymbol{\nu}$ is behaving and not the other way around. However, the form of the probability is symmetric, thus, using a subtle logical leap, instead of saying that $\boldsymbol{\nu}$ approximates $\boldsymbol{\mu}$ well, we infer that $\boldsymbol{\mu}$ tends to be close to $\boldsymbol{\nu}$.

2.4.3 Connection to learning

Let us now try and connect the previous simple probabilistic example to the learning problem. In the case of the box, the unknown quantity that we are trying to approximate is the number $\boldsymbol{\mu}$. The corresponding quantity in the learning problem, discussed on the preface of this subchapter, is the full-fledged function $\boldsymbol{f : X \rightarrow Y}$.

Next, we assume that each tennis ball from the previous example is a point $\boldsymbol{x \in X}$**.** Thus, the box containing the balls is now the input space $\boldsymbol{X}$. Let us now correspond the white balls to the assumption that the hypothesis, $\boldsymbol{h}$, learnt from the system agrees with the target function $\boldsymbol{f}$. Consequently, the black balls represent the points that the two functions disagree.

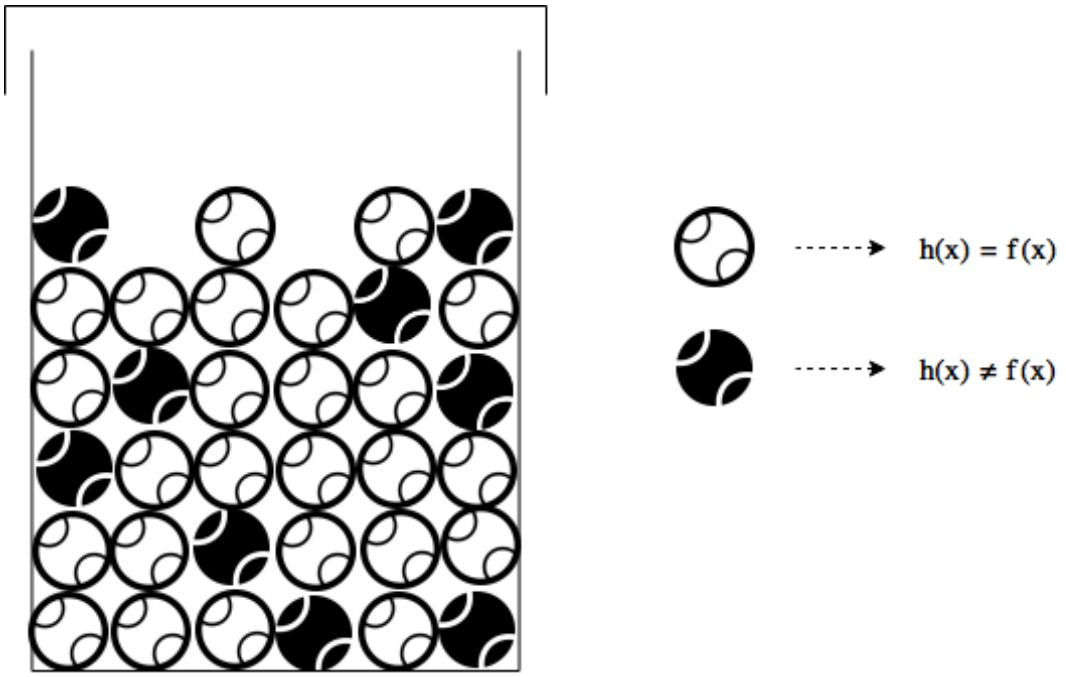


Figure 2-3 - Connecting the box experiment to learning

The key point we should make, in order to relate those examples, is to introduce a probability distribution over the input space $\boldsymbol{X}$. So now, the points in this space are not just generic data points. There is a probability of picking one versus another. Let $\boldsymbol{P}$ capture this probability. Despite of demanding a probability distribution over the input space, we do not need to have any prior knowledge about its properties. Moreover, we do not even need to know the value of $\boldsymbol{P}$. The choice of a different probability distribution though, will always affect the outcome of the experiment, the probability of getting a black or a white tennis ball, thus, it could affect the value of $\boldsymbol{\mu}$. But being able to bound the accuracy of the inferences we make independently of $\boldsymbol{\mu}$, through the Hoeffding's inequality, we can continue our machinery given any $\boldsymbol{P}$.

The problem with making this connection to learning though, is that we are verifying that $\boldsymbol{\nu}$ generalizes to $\boldsymbol{\mu}$ for a specific hypothesis $\boldsymbol{h}$. This is called verification and not learning. Someone

comes up with this specific hypothesis $\boldsymbol{h}$, and through our connection we are verifying that this particular hypothesis generalizes well to the population, using the Hoeffding's inequality. But what are the chances of picking a right hypothesis in the first place? The idea of learning is searching the whole space to find the best hypothesis, that agrees with the training data and generalizes well to the population. Thus, in order to relate the box example to learning, we need to choose from multiple hypotheses the one that has the minimum generalization error (Figure 2-4).

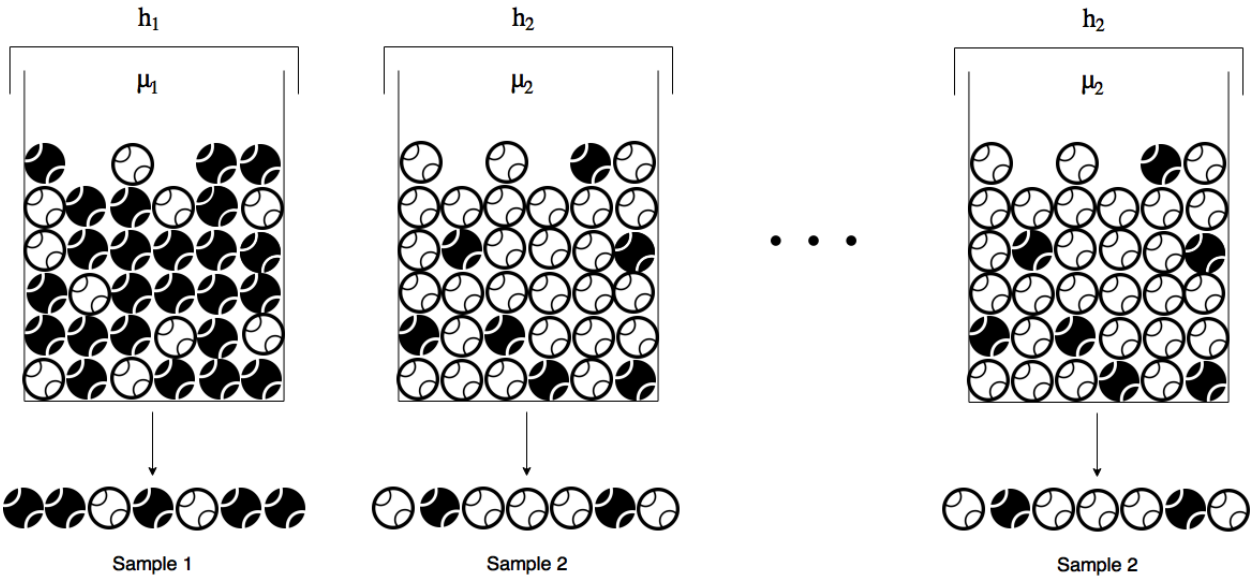


Figure 2-4 - Box example with multiple hypotheses

Generalizing the box example, we can assume that we have a finite number of hypotheses to choose from. Some of them generalizes well out of sample and some don't. Our objective is to choose the one with the least error. In the figure below we denote $\boldsymbol{E_{in}(h)}$ the error of the hypothesis $\boldsymbol{h}$ in the sample we have (training data), and $\boldsymbol{E_{out}(h)}$ the error of the hypothesis $\boldsymbol{h}$ out of sample (population). Thus, if $\boldsymbol{E_{out}(h)}$ is small, then the hypothesis generalizes well, and by picking this hypothesis the system has learnt. So, the Hoeffding's inequality now becomes:

$$\boldsymbol{P}[\,|\boldsymbol{E_{in}(h)} - \boldsymbol{E_{out}(h)}| > \boldsymbol{\varepsilon}|\,] \leq \boldsymbol{2e^{-2\varepsilon^2 N}}$$

This means that the probability that the error the system has in-sample, deviates from the error the system has when it generalizes is hopefully a small number. In other words, the in-sample error tracks the out of sample error well. Now the above figure transforms to Figure 2-5, shown below:

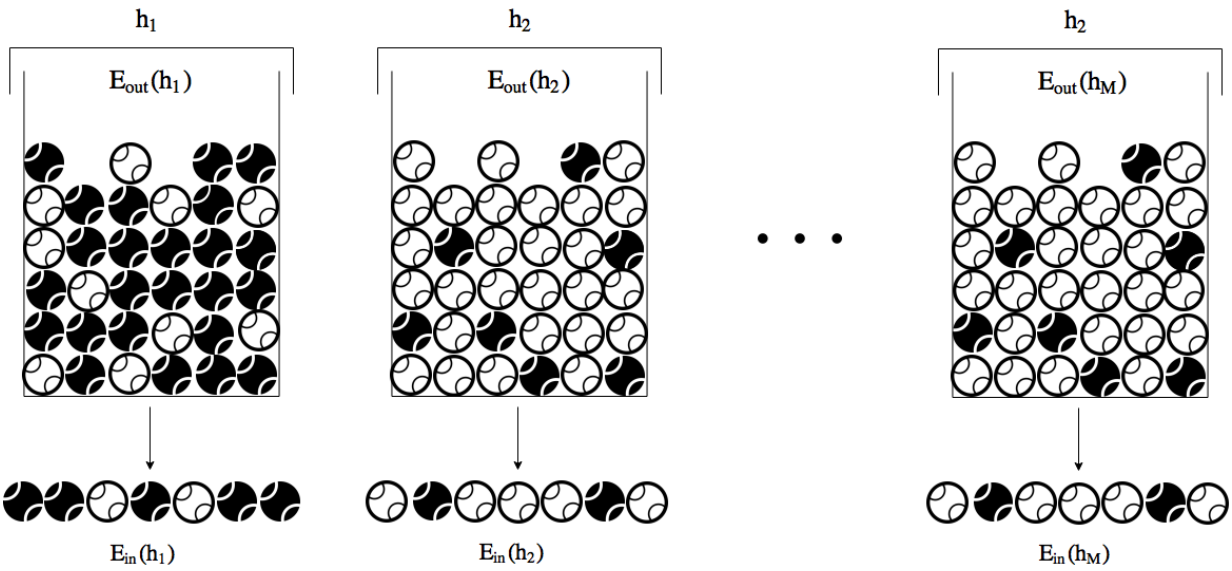


Figure 2-5 - Multiple hypotheses error

The problem now is that the Hoeffding's inequality doesn't apply to multiple boxes, so, multiple hypotheses. In a coin analogy, if a coin is tossed ten times, the probability of getting ten heads is 0.1%. But if 1000 coins are tossed ten times each, the probability of some coin gives ten heads rises to 63%. Although the Hoeffding's inequality applies to all of them separately, there is still a small chance that the experiment will give ten heads. If those small probabilities are disjoined, we will end up with a significant probability value, that this, almost impossible outcome, will happen.

In order to deal with multiple hypotheses properly we should transform the Hoeffding's inequality as shown below. Now, instead $\boldsymbol{h}$ denotes the final hypothesis, the one the system chooses:

$$\boldsymbol{P}[\,|\boldsymbol{E_{in}}(\boldsymbol{h}) - \boldsymbol{E_{out}}(\boldsymbol{h})| > \boldsymbol{\varepsilon}|\,] \leq \sum_{\boldsymbol{m=1}}^{\boldsymbol{M}} \boldsymbol{P}[\,|\boldsymbol{E_{in}}(\boldsymbol{h_m}) - \boldsymbol{E_{out}}(\boldsymbol{h_m})| > \boldsymbol{\varepsilon}\,] \leq \sum_{\mu=1}^{M} \boldsymbol{2e^{-2\varepsilon^2 N}}$$

This derives using the union bound, where we make the pessimistic assumption that the individual probabilities for each hypothesis from $\boldsymbol{h_1}$ … $\boldsymbol{h_M}$ are non-overlapping, disjoined, so the overall probability for the hypothesis $\boldsymbol{h}$ we choose, is less than the sum of the probabilities of every hypothesis. So, in other words we are considering the worst-case scenario.

Finally, to capture the learning problem, the Hoeffding's inequality is written as:

$$\boldsymbol{P}[\,|\boldsymbol{E_{in}}(\boldsymbol{h}) - \boldsymbol{E_{out}}(\boldsymbol{h})| > \boldsymbol{\varepsilon}|\,] \leq \boldsymbol{2Me^{-2\varepsilon^2 N}}$$

Thus, intuitively, the more complex the model the more we lose the link between in-sample and out of sample performance. This is a typical problem of performing well in-sample but generalizing poorly in the population, that this work addresses in the subchapters to come.

2.5 Error measures

Previously, in the probabilistic approach to machine learning, it is mentioned that the goal of the system is to learn the hypothesis $\boldsymbol{h}$ that approximates well the target function $\boldsymbol{f}$. Thus, there should be a measure of error, which is a quantitative measure.

So, the error measure is formally defined as $\boldsymbol{E}(\boldsymbol{h}, \boldsymbol{f})$, a functional that takes two parameters, the hypothesis $\boldsymbol{h}$ and the target function $\boldsymbol{f}$, and measures how well $\boldsymbol{h}$ approximates $\boldsymbol{f}$. Therefore, the goal of the system is to minimize this function, with a value of zero meaning that the system has found the perfect hypothesis.

The result of $\boldsymbol{E}$ is the summed difference of each point, between $\boldsymbol{h}$ and $\boldsymbol{f}$. Thus, there is a pointwise definition $\boldsymbol{e}(\boldsymbol{h}(\boldsymbol{x}), \boldsymbol{f}(\boldsymbol{x}))$, that takes as parameters the value of the hypothesis $\boldsymbol{h}$ at a particular point $\boldsymbol{x}$ and the value of the target function $\boldsymbol{f}$ at the same point.

Examples include the squared error, $\boldsymbol{e}\big(\boldsymbol{h}(\boldsymbol{x}), \boldsymbol{f}(\boldsymbol{x})\big) = (\boldsymbol{h}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x}))^{\boldsymbol{2}}$, or the binary error used in classification problems, defined as $\boldsymbol{e}\big(\boldsymbol{h}(\boldsymbol{x}), \boldsymbol{f}(\boldsymbol{x})\big) = [\![\boldsymbol{h}(\boldsymbol{x}) \neq \boldsymbol{f}(\boldsymbol{x})]\!]$, where it returns one if the statement inside the squared brackets is true and zero otherwise. So, if $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x})$ at $\boldsymbol{x}$, then this function returns zero, so the error is equal to zero.

Finally, the overall error is defined as the average of the pointwise error, and formally, the in-sample and out of sample overall error is defined as follows:

- In-sample error

$$\boldsymbol{E_{in}}(\boldsymbol{h}) = \frac{\boldsymbol{1}}{\boldsymbol{N}} \sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} \boldsymbol{e}(\boldsymbol{h}(\boldsymbol{x_n}), \boldsymbol{f}(\boldsymbol{x_n}))$$

- Out of sample error

$$\boldsymbol{E_{out}}(\boldsymbol{h}) = \mathbb{E}_{\boldsymbol{x}} [\boldsymbol{e}(\boldsymbol{h}(\boldsymbol{x}), \boldsymbol{f}(\boldsymbol{x}))]$$

For the out of sample error, $\mathbb{E}_{\boldsymbol{x}}$ is the expected value with respect to $\boldsymbol{x}$.

2.5.1 False positive and false negative

There are two types of error, when trying to choose the error measure the system should use. We will borrow a medical example to describe them. The first is called false positive, where a patient in taking a test for some disease and the test wrongfully comes back positive. The second type of error is called false negative, where the test comes back negative, although the patient suffers from this disease.

2.5.2 How to choose an error measure

According to the definition of false positive and false negative types of error, the system should penalize each of those in a way that the final hypothesis performs better, in agreement with the chose type of error.

For example, in the case of the patient mentioned above, the false positive error does not look that serious. It may be the cause of elevated anxiety for the patient, or inconvenience because of the follow-up tests he should take, but overall the case of being ill and not being informed, the case of false negative, is worse.

Thus, penalizing the false negative type of error more makes sense for that system. Putting it in a matrix form, the system architect could choose to penalize the false positive case with a lower penalty weight than the case of false negative.

		f	
		1	0
h	1	No error	False positive (1)
	0	False Negative (100)	No error

Figure 2-6 - False positive and False negative matrix

The outcome of this is that there is not some method in choosing and error measure versus another. It is not an analytic question. It is an application domain question. Thus, the error measure is usually specified by the user of the system and its context.

However, this is not always happened. Alternatively, as a second choice the system architect could resolve to so-called plausible measures, measures that he could argue that they have merit. In such a case, the error measure could simply be the squared error.

Another form of error that could be chosen is the so-called practical measure. In this case the system architect could choose an error measure that helps him solve the problem and justify the solution easier.

2.6 Noise

The case of a clean target function $\boldsymbol{f}$ is very rare. In reality, the target function is not always a function. In other words, the target function $\boldsymbol{f}$ does not always return a unique value for every point in its domain, thus, mathematically it is not a function. Two independent training examples can return the same outcome and make the same prediction. There is always information that is not captured and contributes noise to the result.

To address this issue, we use a target probability distribution $\boldsymbol{P}(\boldsymbol{y}|\boldsymbol{x})$ instead of the target function $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$. This means that some outcomes $\boldsymbol{y}$, are more possible than others, given the training data points.

Therefore, $\boldsymbol{x}$ is always generated from the unknown input probability distribution $\boldsymbol{P}(\boldsymbol{x})$, that was introduced before to gain the advantages of the Hoeffding's inequality. But now, what changes is that $\boldsymbol{y}$ instead of being deterministic once we have the generated value of $\boldsymbol{x}$, is now probabilistic, given that specific $\boldsymbol{x}$, and is given by the target probability distribution, that remains unknown, just like the target function $\boldsymbol{f}$ that was used until now. Thus, every pair $(\boldsymbol{x}, \boldsymbol{y})$ is now generated by the joint distribution $\boldsymbol{P}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{P}(\boldsymbol{x})\boldsymbol{P}(\boldsymbol{y}|\boldsymbol{x})$.

The system now has a noisy target, given by the aforementioned target probability distribution, that can be thought of as the deterministic target that we had up until now, plus noise. Intuitively, if we define the target function $\boldsymbol{f}$ to be the expected value of $\boldsymbol{y}$ given $\boldsymbol{x}$, $\boldsymbol{f}(\boldsymbol{x}) = \mathbb{E}(\boldsymbol{y}|\boldsymbol{x})$, then whatever remains can be thought of noise.

2.7 The learning diagram

Below is presented and analyzed the learning diagram, the diagram that sums up every component that forms and frames what is supervised machine learning. It incorporates the knowledge that is built up so far in a concise and intuitive way [14].

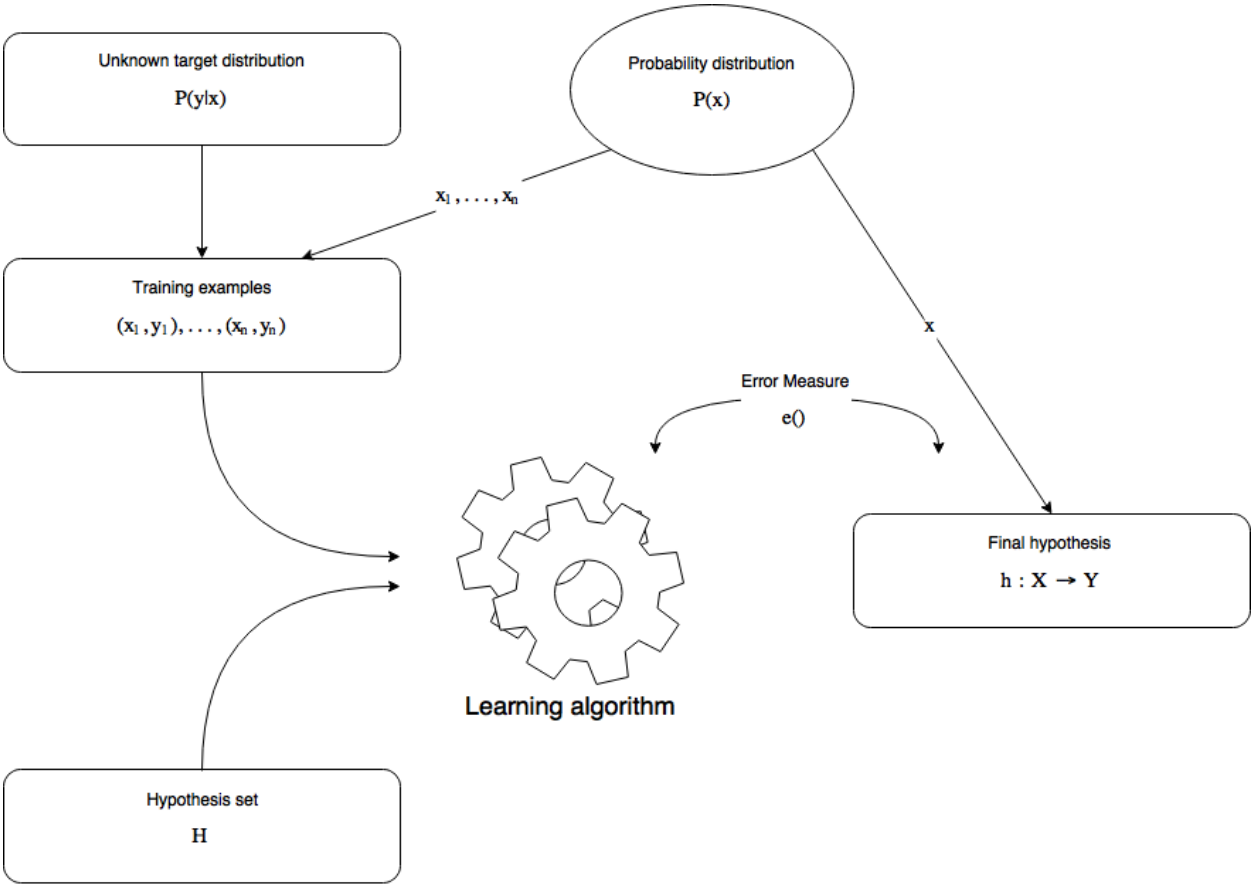


Figure 2-7 - The learning diagram

In the learning diagram, we can see that the input probability distribution $\boldsymbol{P}(\boldsymbol{x})$, generate the training points $\boldsymbol{x}$, and in turn the target distribution, instead of deterministically generating pairs of $(\boldsymbol{x}, \boldsymbol{y})$ training examples, it uses the data points $\boldsymbol{x}$ to generate the labels $\boldsymbol{y}$ in a probabilistic way.

In turn the training examples are given to the learning algorithm, which chooses by a hypothesis set $\boldsymbol{H}$, the right hypothesis, that approximates the unknown target distribution well. Last but not least, the performance of any chosen hypothesis, as well as the final hypothesis $\boldsymbol{h}$, are measured by the error measure of choice.

2.8 Training and testing

Formally, we can see the difference between testing and training using the Hoeffding's inequality. In the case of testing, the system has already a fixed hypothesis and in turn, it applies it on data that it has not yet presented with, trying to come up with the right predictions. This is shown below, using the already seen before Hoeffding's formula:

$$\boldsymbol{P}[\,|\boldsymbol{E_{in}}(\boldsymbol{h}) - \boldsymbol{E_{out}}(\boldsymbol{h})| > \boldsymbol{\varepsilon}|\,] \;\leq\; \boldsymbol{2e^{-2\varepsilon^2 N}}$$

In other words, this means that the probability that the in-sample error does not track well the out of sample error, is low.

On the other hand, during the process of training, the system is presented with numerous examples and consequently, it constantly refines its results, trying to figure out the best hypothesis to fit the training data. Formally, this is shown below using the same formula with one small addition; the parameter $\boldsymbol{M}$. This parameter accounts for the number of the hypotheses included in the hypothesis set, $\boldsymbol{H}$, from which the system is trying to pick the best one.

$$\boldsymbol{P}[\,|\boldsymbol{E_{in}}(\boldsymbol{h}) - \boldsymbol{E_{out}}(\boldsymbol{h})| > \boldsymbol{\varepsilon}|\,] \;\leq\; \boldsymbol{2Me^{-2\varepsilon^2 N}}$$

According with how we defined the parameter $\boldsymbol{M}$ in the corresponding chapter, there is a major problem. Even the simplest machine learning algorithm has $\boldsymbol{M}$ to be equal to infinity. Thus, the above statement does not really make sense. To establish the feasibility of learning in an actual model, we need to replace this quantity with something more friendly.

2.8.1 Refining the union bound assumption

To replace the quantity $\boldsymbol{M}$ in the latter form of the Hoeffding's inequality transformation, we need to understand what it means. As we have seen before, in order to measure the performance of the final hypothesis that he system picked, we need to take into account every possible hypothesis. Thus, we said that the probability that the chosen hypothesis does not track the out of sample error well, is no greater than the sum of the corresponding probabilities of every hypothesis in the set.

$$\boldsymbol{P}[\,|\boldsymbol{E_{in}}(\boldsymbol{h}) - \boldsymbol{E_{out}}(\boldsymbol{h})| > \boldsymbol{\varepsilon}|\,] \leq \sum_{\boldsymbol{m=1}}^{\boldsymbol{M}} \boldsymbol{P}[\,|\boldsymbol{E_{in}}(\boldsymbol{h_m}) - \boldsymbol{E_{out}}(\boldsymbol{h_m})| > \boldsymbol{\varepsilon}\,] \;\leq\; \sum_{\mu=1}^{M} \boldsymbol{2e^{-2\varepsilon^2 N}}$$

This sum can be depicted using a Venn diagram, as shown below:

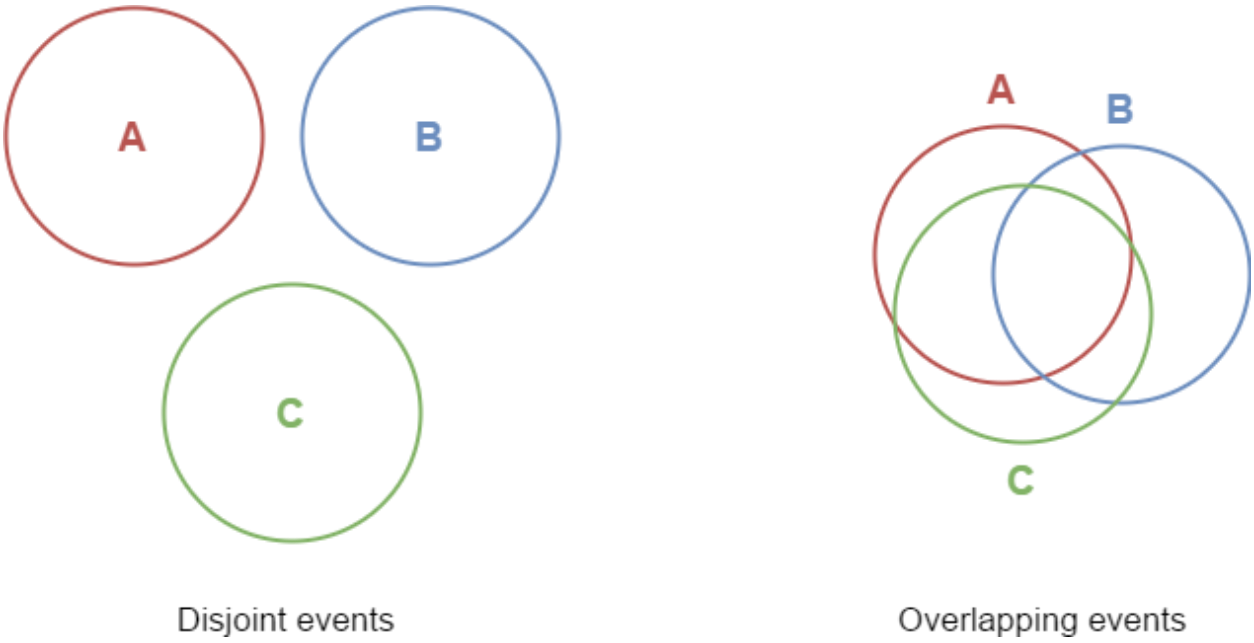


Figure 2-8 - Summing unwanted events

The events $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ depict three different hypotheses, taken from the hypothesis set $\boldsymbol{H}$ (Figure 2-8). These events can be disjoint, or coincident, or independent, in which case the would be proportionally overlapping etc. The union bound considers the worst-case scenario, that those events are disjoint. Thus, to replace the $\boldsymbol{M}$ quantity, we need to consider the overlaps.

In order to achieve that, we need to abstract from the hypothesis set, a quantity that is sufficient to characterize the overlaps, thus, lessen the bound, without trying to analyze the intricate details of every situation and its correlations. Formally, we are trying to suggest that how a hypothesis $\boldsymbol{m}$ performs is comparable to a hypothesis $\boldsymbol{n}$:

$$|\boldsymbol{E_{in}}(\boldsymbol{h_m}) - \boldsymbol{E_{out}}(\boldsymbol{h_m})| \approx |\boldsymbol{E_{in}}(\boldsymbol{h_n}) - \boldsymbol{E_{out}}(\boldsymbol{h_n})|$$

That, in other words, means that the events are overlapping. If the probability that $\boldsymbol{h_m}$ exceeds $\boldsymbol{\varepsilon}$ is a number $\boldsymbol{a}$, then the probability of $\boldsymbol{h_n}$ to exceeds $\boldsymbol{\varepsilon}$ is barely different.

2.8.2 Replacing the $\boldsymbol{M}$ quantity

To justify the way we are replacing the quantity $\boldsymbol{M}$, we need to consider a finite sample of data, instead of the whole input space. Therefore, the number $\boldsymbol{M}$ reduces to how many different patterns can I recognize in the sample.

For example, if we consider a classification algorithm, which tries to classify a finite number of data points, we count the ways that those data points can be classified. This way, we are counting

the different hypotheses in a restricted set, instead of the input space. We are going to call the hypotheses restricted to this set, dichotomies.

Considering the above notion, a hypothesis $\boldsymbol{h}$ is a function that is defined as:

$$\boldsymbol{h}: \boldsymbol{X} \rightarrow \{-\mathbf{1}, \mathbf{1}\}$$

A dichotomy is also a function $\boldsymbol{h}$, but it is defined as:

$$\boldsymbol{h}: \{\boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_N}\} \rightarrow \{-\mathbf{1}, \mathbf{1}\}$$

Thus, a dichotomy is just like a hypothesis, but its domain is not the whole input space, but a restricted sample. So, the number of hypotheses, $|\boldsymbol{H}|$, is infinite. However, the number of possible dichotomies, $|\boldsymbol{H}(\boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_N})|$ is at most $\mathbf{2}^{\boldsymbol{N}}$. We assume that a classification algorithm can classify every point of the sample set, as -1 or 1. The number of hypothesis that we can impose on this training sample is infinite. Yet, not every hypothesis alters the classes that was given by the previous one. Thus, we conclude to the number $\mathbf{2}^{\boldsymbol{N}}$. This number becomes a candidate for replacing $\boldsymbol{M}$.

2.8.3 The growth function

The growth function counts the maximum number of dichotomies, that can be used to separate any $\boldsymbol{N}$ points, in a classification process. More formally:

$$\boldsymbol{m_H}(\boldsymbol{N}) = \max_{\boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_N} \in \boldsymbol{X}} |\boldsymbol{H}(\boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_N})|$$

Thus, we can bound the growth function as follows:

$$\boldsymbol{m_H}(\boldsymbol{N}) \leq \mathbf{2}^{\boldsymbol{N}}$$

For example, we will apply the growth function definition on a simple, binary classification algorithm, the perceptron. If we are given three data points, $\boldsymbol{N} = \mathbf{3}$, the value of the growth function is $\boldsymbol{m_H}(\boldsymbol{N}) = \mathbf{8}$. But if we are given four data points, the value of the growth function reduces to 14, and not 16. This is depicted in the figure below (Figure 2-9).

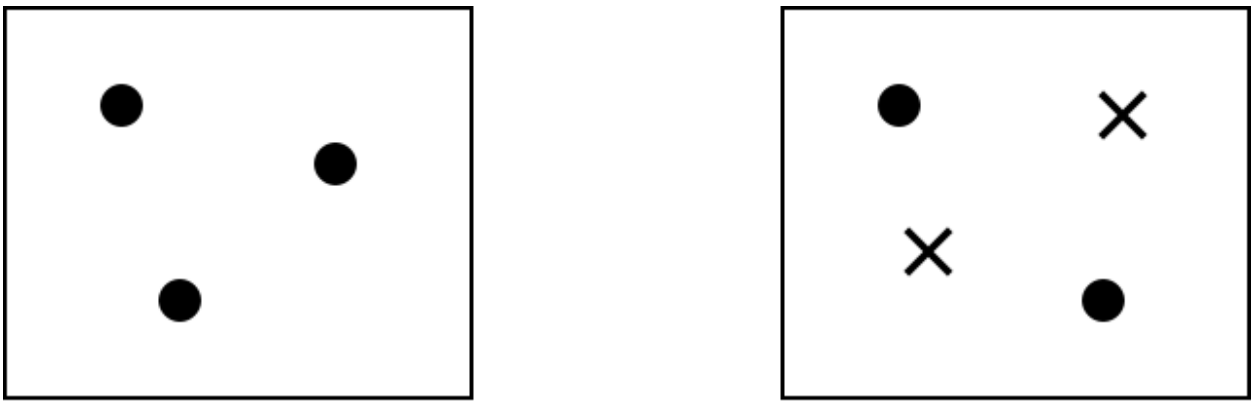
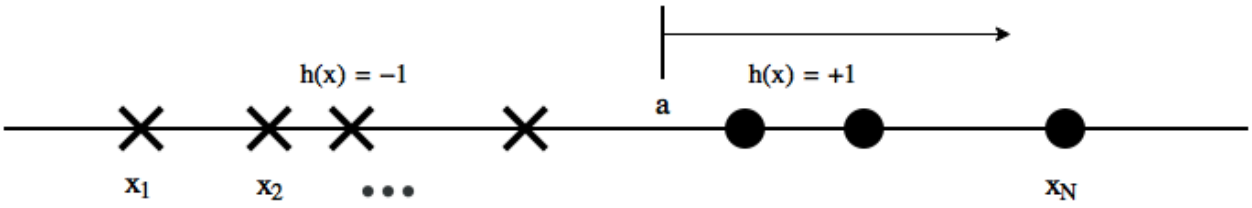
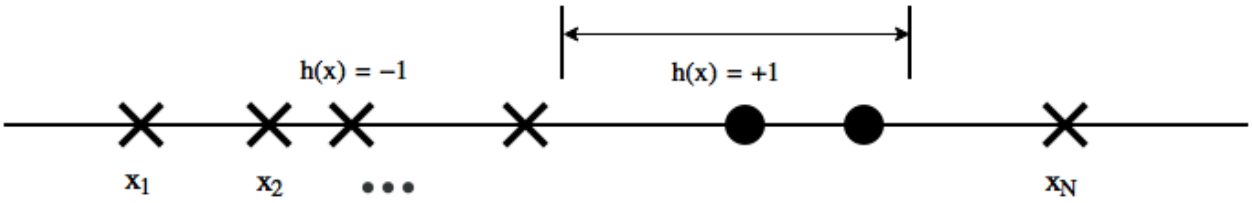


Figure 2-9 - Perceptron algorithm breaking point

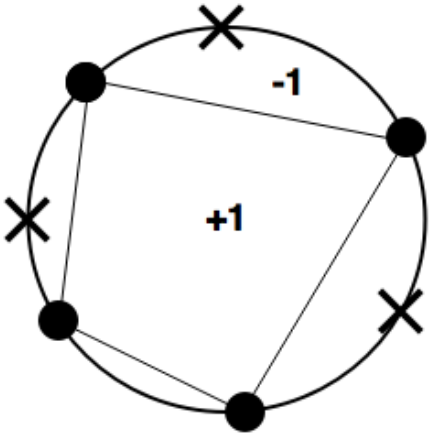
As we can see, given three data points, we can come up with eight different classifiers. But in the case of four data points, we cannot classify the data the way they are depicted above. This is true even if we reverse the classes of the data points. Thus, in total, we are two classifiers short. Hence the 14 classifiers and not 16. We now can expand and measure the value of the growth function in the examples depicted in the figure below.



$$h(x) = \text{sign}(x - a)$$



$$h(x) = \text{sign}(x - a)$$



$h(x) = +1$

if area is convex

Figure 2-10 - Growth function examples

In the first example, the most classifiers that we can get are $\boldsymbol{N+1}$. It is a simple linear model, but in the second example, we ha $\binom{N+1}{2} + \mathbf{1} = \frac{1}{2}\boldsymbol{N^2} + \frac{1}{2}\boldsymbol{N} + \mathbf{1}$**.** In the third example, we can arrange the given data points in such a way, that we can get the maximum number of classifiers, which is $\boldsymbol{2^N}$.

What we need, finally, is to replace the quantity $\boldsymbol{M}$ with a friendlier measure. If that number was a polynomial, then we would be in great shape because the negative exponential in Hoeffding's inequality would kill any polynomial, given a fair amount of data points, $\boldsymbol{N}$.

2.8.4 Break point

The definition of the break point, for a dichotomy $\boldsymbol{h}$, is the point at which the system fails to get all possible classifiers. For example, as we have seen before, for the perceptron algorithm the break point is the number four. Generally, if the breaking point of an algorithm is $\boldsymbol{N}$, that means that we can classify $\boldsymbol{N-1}$ points in whatever way we want and then, start failing at the $\boldsymbol{N^{th}}$ point.

Definition 2.5: Break Point

If no data set of size k can be shattered by H, then k is the breaking point for H.

Intuitively, this definition translates to $\boldsymbol{m_H(H) < 2^k}$. Thus, for the examples of positive rays, positive intervals and convex areas, depicted in the figure above (Figure 2-10), the breaking points are respectively:

- Positive rays

  $$\boldsymbol{m_H(H) = N+1, k = 2}$$

  because for $\boldsymbol{N = 2, m_H(2) < 2^2}$

- Positive intervals

  $$\boldsymbol{m_H(H) = \frac{1}{2}N^2 + \frac{1}{2}N + 1, k = 3}$$

  because for $\boldsymbol{N = 3, m_H(3) < 2^3}$

- Convex sets

  $$\boldsymbol{m_H(H) = 2^N, k = \infty}$$

Thus, the result is that if there is any break point, the growth function is polynomial in $\boldsymbol{N}$. Therefore, all we need to declare that learning is feasible is that there is a break point.

2.9 The theory of generalization

The important next stop on this roadmap is to prove that the growth function, $\boldsymbol{m_H(H)}$, is a polynomial and, finally, prove that it can replace the $\boldsymbol{M}$ quantity.

2.9.1 Bounding the growth function

To show that the growth function is a polynomial, all we need to prove is that it is bounded by a polynomial. We are not trying to prove that the growth function is a specific polynomial. So, we show that:

$$\boldsymbol{m_H(H) \leq a\ polynomial}$$

As a reminder, having the growth function be less or equal to a polynomial, gives us a great advantage concerning the feasibility of learning. In this case, if we replace the $\boldsymbol{M}$ quantity by the value of the growth function, the negative exponential in the Hoeffding's inequality will kill any exponential, eventually, given the right amount of data.

Now, we should define a key quantity, that we will call $\boldsymbol{B(N,k)}$, and is defined as the number of classifiers we can derive, given $\boldsymbol{N}$ data points and a break point $\boldsymbol{k}$.

$$\boldsymbol{K(N,k)}: \boldsymbol{Maximum\ number\ of\ dichotomies\ on\ N\ points, given\ a\ break\ point\ k}$$

Thus, we can define the following example; we have $\boldsymbol{N}$ data points and we are trying to shatter the data, i.e. get as many classifiers as I can, under the constraint that there is a break point $\boldsymbol{k}$. This is shown on the figure below (Figure 2-10).

Every row is a unique combination of classes (+1, -1), but, if we keep the last data point, $\boldsymbol{x_N}$, separated, the remaining $\boldsymbol{x_1, \dots, x_{N-1}}$ data points can fall into one of the following categories. They define either unique rows, independently of the last element $\boldsymbol{x_N}$, or they can be duplicates that have either +1 or -1 in the last position.

We are keeping the last data point separated, so we can divide the matrix of classified data points into three big bins; in the first one, that we name $\boldsymbol{S_1}$, we are keeping the data points that can only have +1 or -1 as the class of the last element. The $\boldsymbol{S_2}$ bin is divided into two categories. The rows from $\boldsymbol{x_1, \dots, x_{N-1}}$ are the same, but the last element is always +1 for the bin $\boldsymbol{S_2^+}$ and -1 for $\boldsymbol{S_2^-}$. In other words, the first row in either $\boldsymbol{S_2^+}$ or $\boldsymbol{S_2^-}$ is the same, except the last element $\boldsymbol{x_n}$.

		Number of rows	x_1	x_2	•••	x_{N-1}	x_N
S_1		α	+1	+1	•••	+1	+1
S_2	S_2^+	β	+1	-1	•••	+1	+1
	S_2^-	β	+1	-1	•••	+1	-1

Figure 2-11 - Bounding the growth function [14]

From the figure above, we can derive that $\boldsymbol{B(N,k) = \alpha + 2\beta}$. Thus, we should estimate the value of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$.

First, we should concentrate on $\boldsymbol{x_1, \dots, x_{N-1}}$ columns (Figure 2-12). In this, new, submatrix the region $\boldsymbol{S_2^+}$ and $\boldsymbol{S_2^-}$ have duplicate rows, so we can reduce the total number of rows to only the rows that are within the regions $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. In those regions, we can be sure by design, that there are no duplicates either within the same region or between regions $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$.

For the original matrix, we could not find all possible patterns on any $\boldsymbol{k}$ columns. So, by extension, we cannot find all possible patterns on any $\boldsymbol{k}$ columns on the newly created submatrix. This is because if we could, then these patterns would serve as all possible patterns on the full, original matrix. Thus, we can argue that the total number of rows $\boldsymbol{\alpha + \beta}$, are less or equal to $\boldsymbol{B(N-1,k)}$.

$$\boldsymbol{\alpha + \beta \ \leq B(N-1,k)}$$

We say that it is less or equal than and not equal, because the original matrix was constructed in such way, that it maximizes the number of patterns that we can get on $\boldsymbol{N}$ data points. The new submatrix was constructed in a special way, so we cannot be certain that it is maximizing the number of rows.

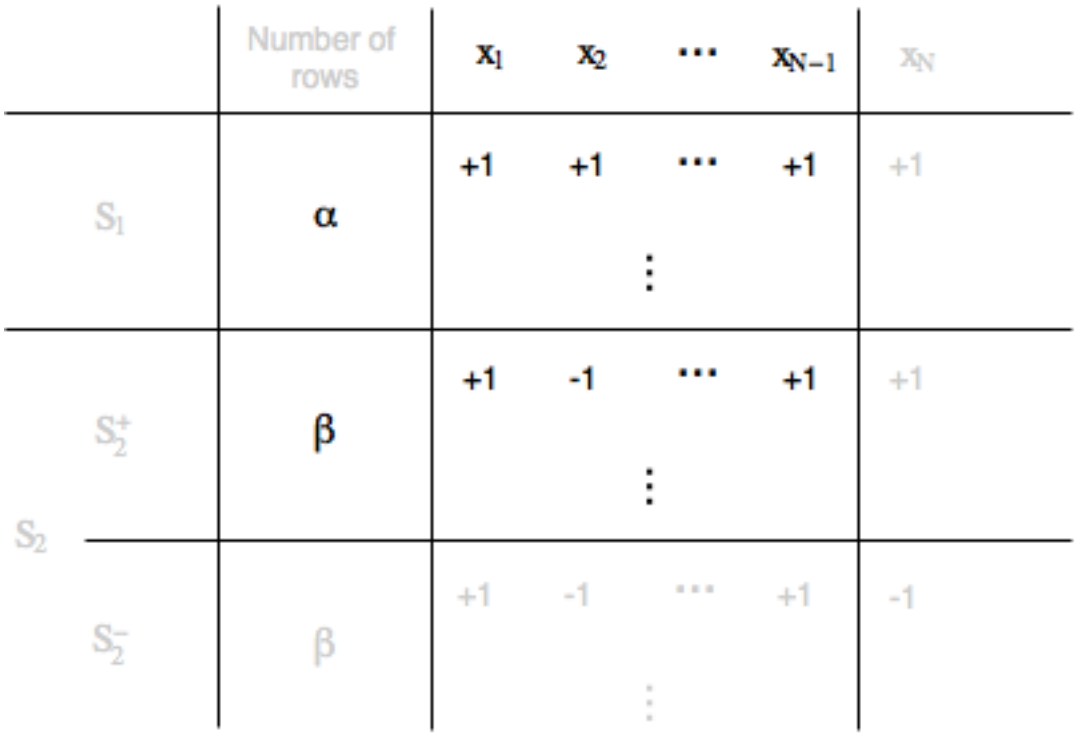


Figure 2-12 - Recursive bound on the growth function – Part I

Next, we should concentrate on estimating the quantity $\boldsymbol{\beta}$. Thus, we should concentrate on either the $\boldsymbol{S_2^+}$ or $\boldsymbol{S_2^-}$ regions.

In those regions, we can argue that the break point is $\boldsymbol{k-1}$. This is because, if we could get all possible patterns on $\boldsymbol{k-1}$ columns, then by adding the $\boldsymbol{x_N}$ extension we could get all possible patterns on $\boldsymbol{k}$ columns.

Having establish that the break point is $\boldsymbol{k-1}$, we can estimate the value of b as:

$$\boldsymbol{\beta \leq B(N-1, k-1)}$$

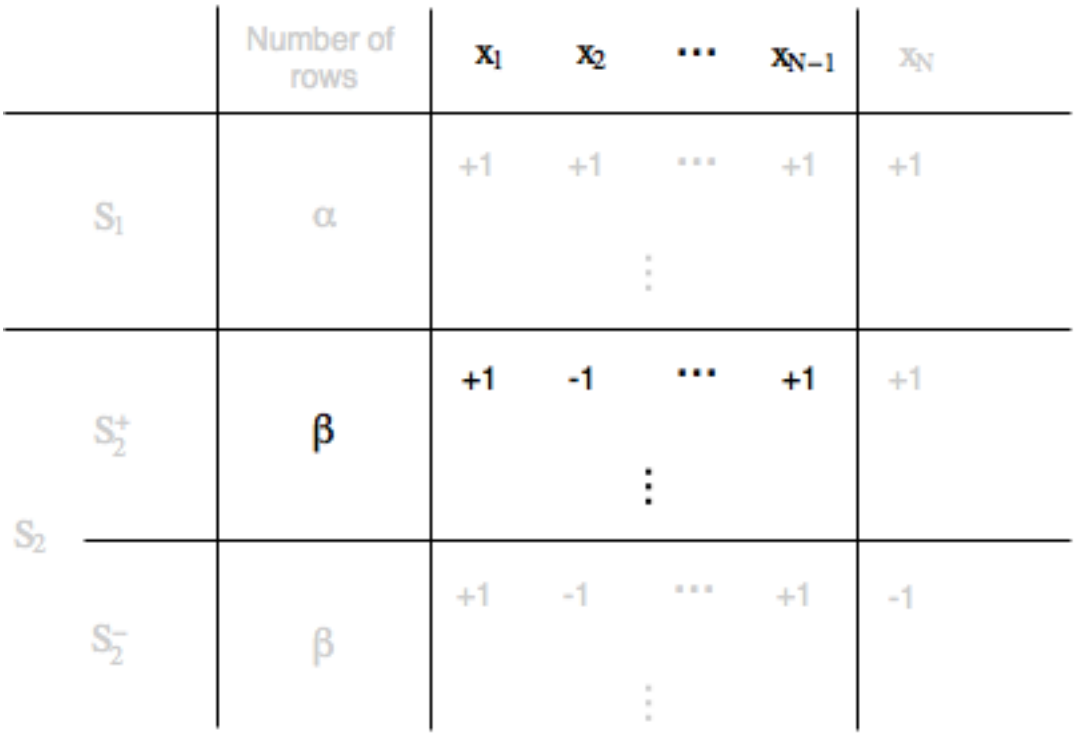


Figure 2-13 - Recursive bound on the growth function - Part II

Finally, we should put everything together. By putting together the three formulas computing the quantities $\boldsymbol{\alpha}, \boldsymbol{\beta}$, we get:

$$\boldsymbol{B(N,k) \leq B(N-1,k) + B(N-1,k-1)}$$

2.10 Numerical computation of $B(N,k)$

Having compute an estimate for the upper bounds of the key quantity $\boldsymbol{B(N,k)}$, we can form a table that computes numerically the value of this quantity, given $\boldsymbol{N}, \boldsymbol{k}$ (Figure 2-14).

		$\boldsymbol{k}$				
		1	**2**	**3**	**4**	**…**
$\boldsymbol{N}$	1	1	2	2	2	…
	2	1	3	4	4	…
	3	1	4	7	8	…
	4	1	5	…	…	…
	5	1	…	…	…	…
	…	…	…	…	…	…

Figure 2-14 - Computation of B(N, k)

We have two base key values. The one is when we have the break point equal with one and when the number of data points is equal to one. For the first base value, if we have a break point of 1, then the value of the quantity $\boldsymbol{B(N,k)}$ is equal to one, no matter how many data points we have. On the other hand, if the value of data points is equal to one, then we can only have two different patterns, +1 and -1. Thus, no matter the valued of the break point, except the case in which it is one, the value of $\boldsymbol{B(N,k)}$ is two.

We can easily fill the rest of the table now by adding the values that the $\boldsymbol{B(N,k)}$ takes, recursively, using the formula we derived in the previous subchapter:

$$\boldsymbol{B(N,k) \leq B(N-1,k) + B(N-1,k-1)}$$

Formally we can say that the value of the upper bound of $\boldsymbol{B(N,k)}$ is given by the formula:

$$\boldsymbol{B(N,k) = \sum_{i=0}^{k-1} \binom{N}{i}}$$

We will prove the aforementioned formula, analytically. So, starting from the boundary conditions we verify that the formula returns the expected values from Figure 2-14, given the $\boldsymbol{N}$ and $\boldsymbol{k}$.

Next, we should prove the induction step:

$$\boldsymbol{\sum_{i=0}^{k-1} \binom{N}{i} = \sum_{i=0}^{k-1} \binom{N-1}{i} + \sum_{i=0}^{k-2} \binom{N-1}{i}}$$

$$\boldsymbol{= 1 + \sum_{i=1}^{k-1} \binom{N-1}{i} + \sum_{i=1}^{k-1} \binom{N-1}{i-1}}$$

$$\boldsymbol{= 1 + \sum_{i=1}^{k-1} \left[\binom{N-1}{i} + \binom{N-1}{i-1} \right]}$$

$$\boldsymbol{= 1 + \sum_{i=1}^{k-1} \binom{N}{i} = \sum_{i=0}^{k-1} \binom{N}{i}}$$

So, the formula stands. With the above argument, we conclude that the maximum number of patterns on $\boldsymbol{N}$, given the fact that we have a break point $\boldsymbol{k}$, is a $\boldsymbol{N^{k-1}}$ degree polynomial. Thus,

$$\boldsymbol{m_H(H) \leq \sum_{i=0}^{k-1} \binom{N}{i}}$$

2.11 The Vapnik–Chervonenkis inequality

We now have seen that the maximum number of patterns on $\boldsymbol{N}$, given the fact that we have a break point $\boldsymbol{k}$, in other words the growth function, $\boldsymbol{m_H(H)}$, is bounded by a $\boldsymbol{N^{k-1}}$ degree polynomial. This means that if we can substitute the $\boldsymbol{M}$ quantity of the Hoeffding's inequality, with the growth function, then we can consider learning feasible, in any case.

The only problem is that the growth function considers only what happens in-sample. Thus, we need to get a method to relate the in-sample error with the out of sample error, in order to generalize well. The way to do this, is by completely ignoring the out of sample error. So, returning to the box and balls example, instead of taking sampling the box once, we do it twice.

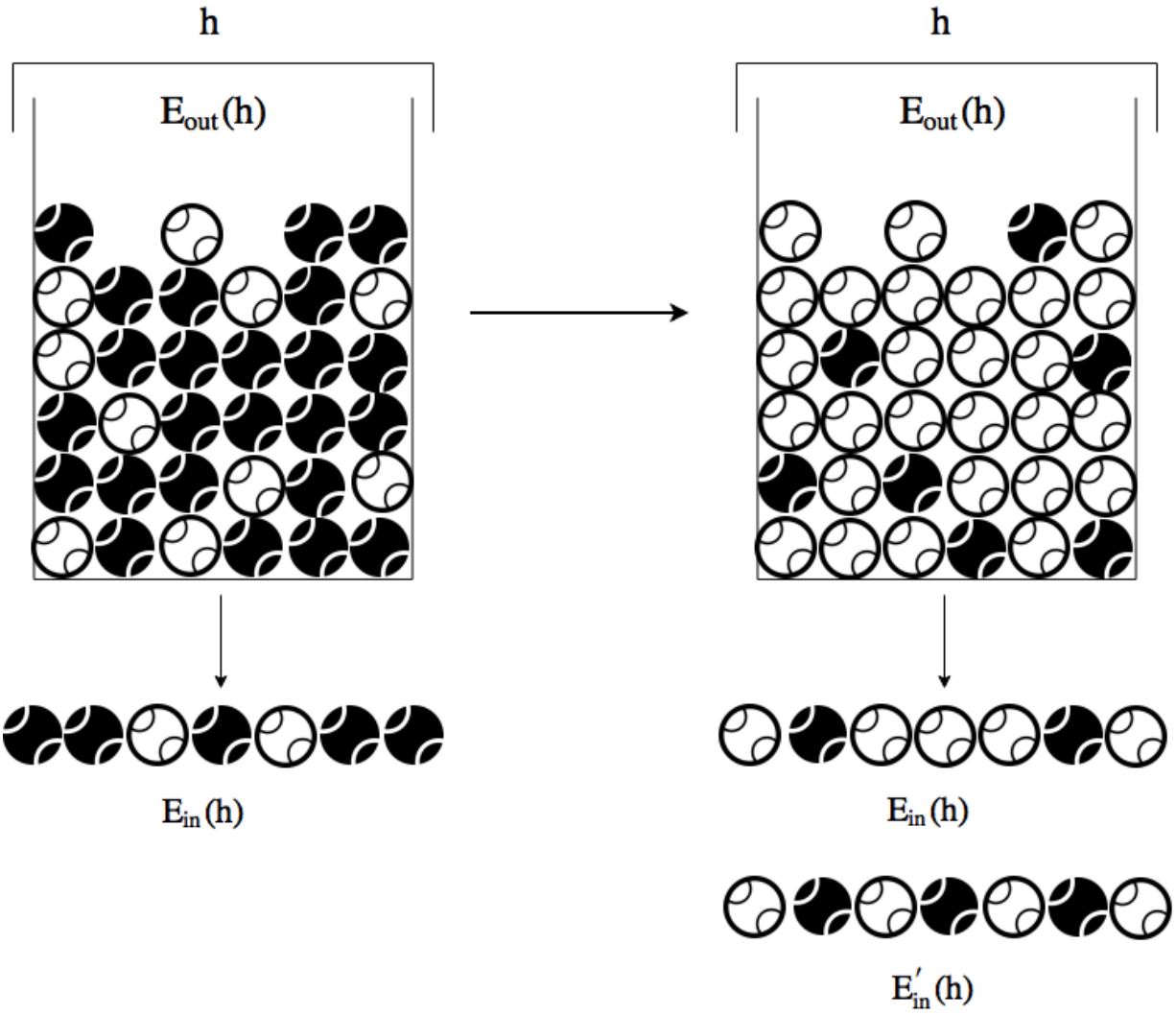


Figure 2-15 - The Vapnik–Chervonenkis example

Using this simple logic, we can now compare the two generated samples, by studying if $\boldsymbol{E_{in}(h)}$ tracks $\boldsymbol{E'_{in}(h)}$ well.

According to the Hoeffding's inequality, in the case of one box, the two in-sample errors track the out of sample error well. Thus, we expect them to track each other as well. But if we consider the case of multiple boxes, then they start to grow apart. For example, there is a probability that in some case, the first sample would consist of only black balls and the second with only white ones.

However, there is a great advantage. The simple argument that characterizes the Vapnik–Chervonenkis idea, is that now, we can describe the problem we have with multiple boxes, using the two samples. Having done that, we are completely in the realm of dichotomies. The only difference is that now we have a bigger sample to work with. Thus, the Hoeffding's inequality transforms to the Vapnik–Chervonenkis inequality, as shown below:

$$\boldsymbol{P[\,|E_{in}(h) - E_{out}(h)| > \varepsilon|\,] \;\leq\; 4m_H(2N)e^{-\frac{1}{8}\varepsilon^2 N}}$$

We now have twice the sample, so the growth function transforms to $\boldsymbol{m_H(2N)}$, but nonetheless it is still bounded by a polynomial in $\boldsymbol{N}$.

Finally, the Vapnik–Chervonenkis inequality is a statement that holds true, for any hypothesis that has a break point, and moreover, $\boldsymbol{m_H(2N)}$ is bounded by a polynomial in $\boldsymbol{N}$ with the order of the polynomial decided by the break point. So, we conclude that learning is feasible, given enough training examples $\boldsymbol{N}$.

2.12 The problem of Overfitting

Overfitting is a problem that causes the performance of machine learning algorithms to drop, proportionally to their complexity. When trying to approximate the unknown target distribution, which is the goal of every machine learning algorithm, we use a finite set of training examples. In this case, the learning algorithm tries to minimize the in-sample error, by choosing the right hypothesis from a hypothesis set $\boldsymbol{H}$.

The problem appears when the complexity of the model increases. The final hypothesis that will be chosen by the algorithm may yield great results in the training set, fitting every data point. From calculus, we know that an infinite Taylor series can approximate any function. But this does not guarantee a good generalization performance.

For example, we will use the sine function, generate random data points as training examples from it and finally try to learn from the data points to approximate the sin function.

Figure 2-16 shows the "unknown" target function we are trying to approximate:

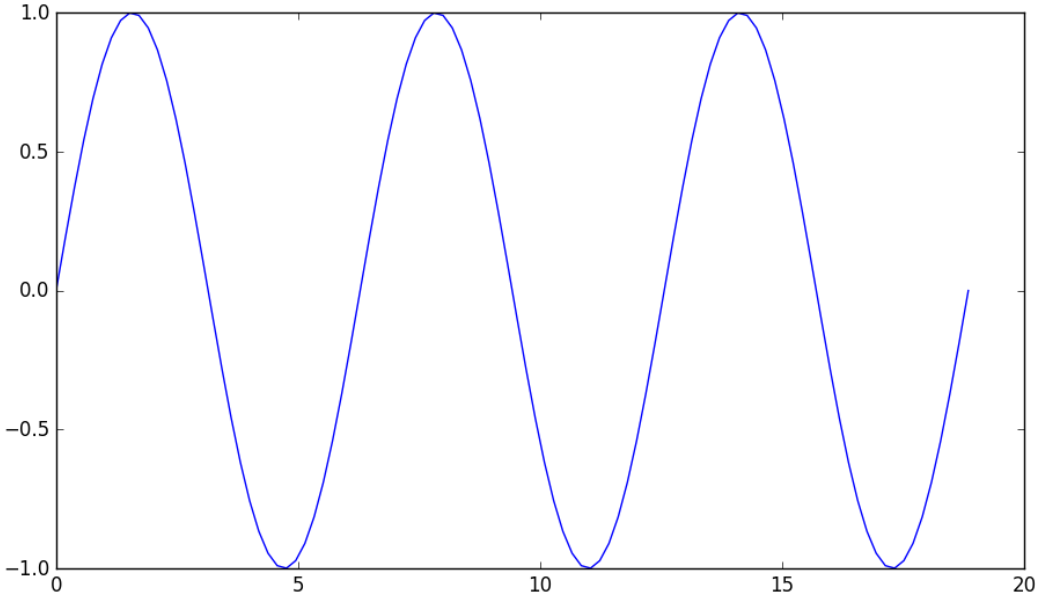


Figure 2-16 - The sin function

The next step is to generate random data points from this function. Figure 2-17 depicts the process:

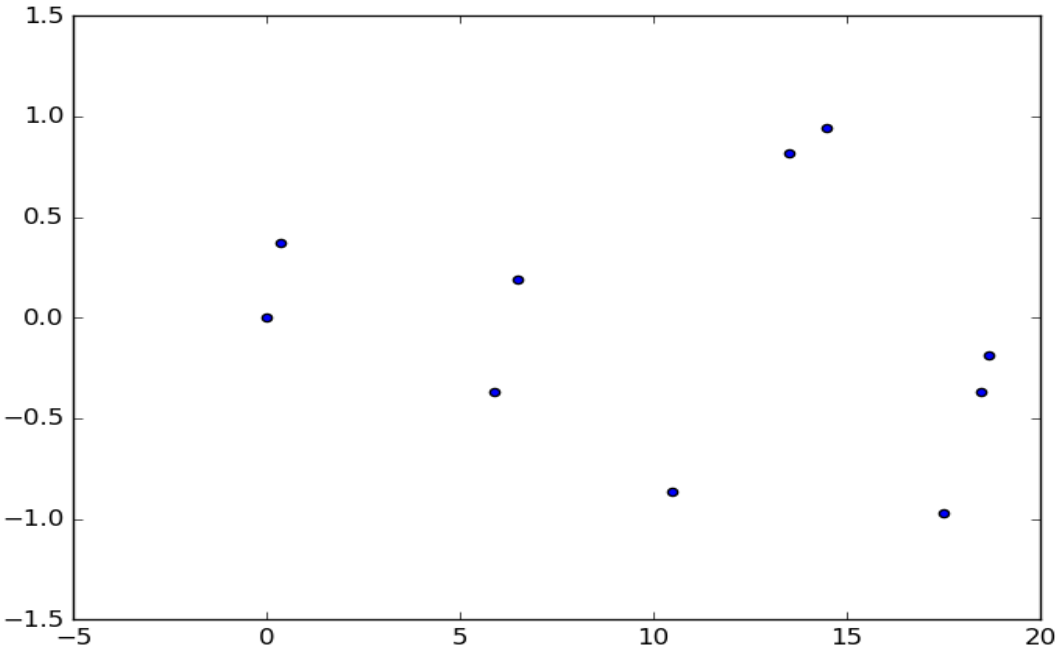


Figure 2-17 - Generating random data points

In the following figures is depicted how different degree polynomials approximate the target function. Note that because the data points are generated randomly every time, they differ from those of Figure 2-17.

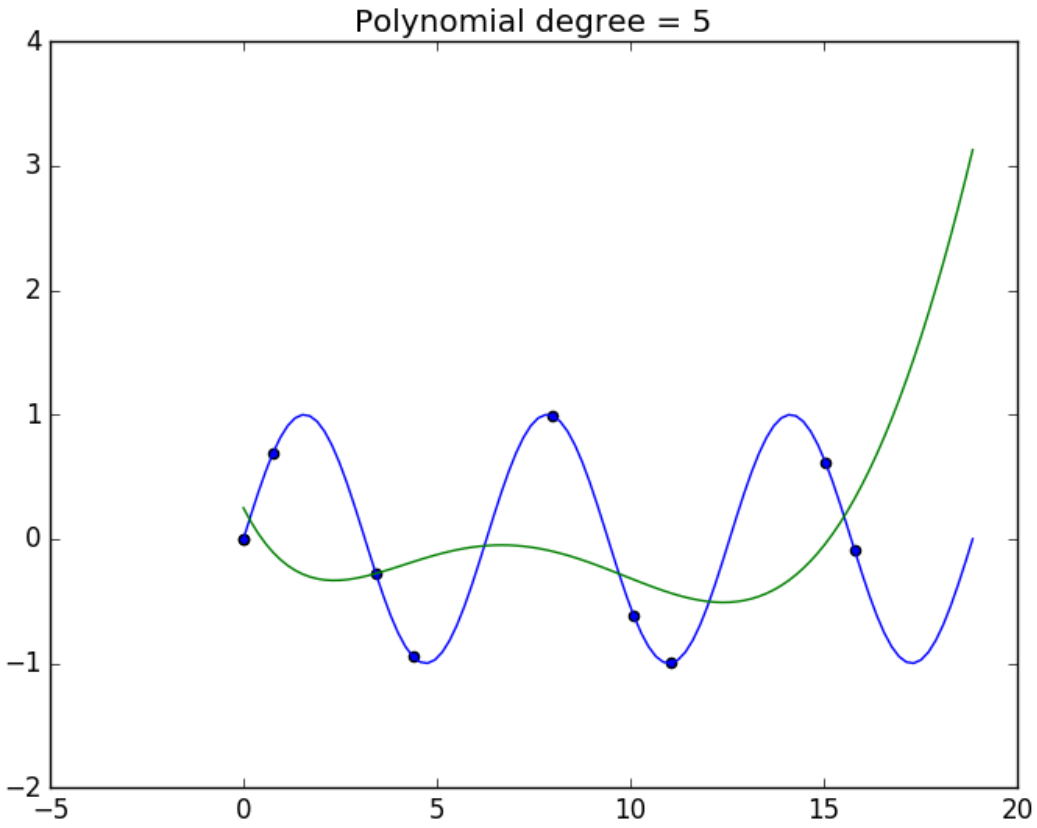


Figure 2-18 - Sine approximation with a degree five polynomial

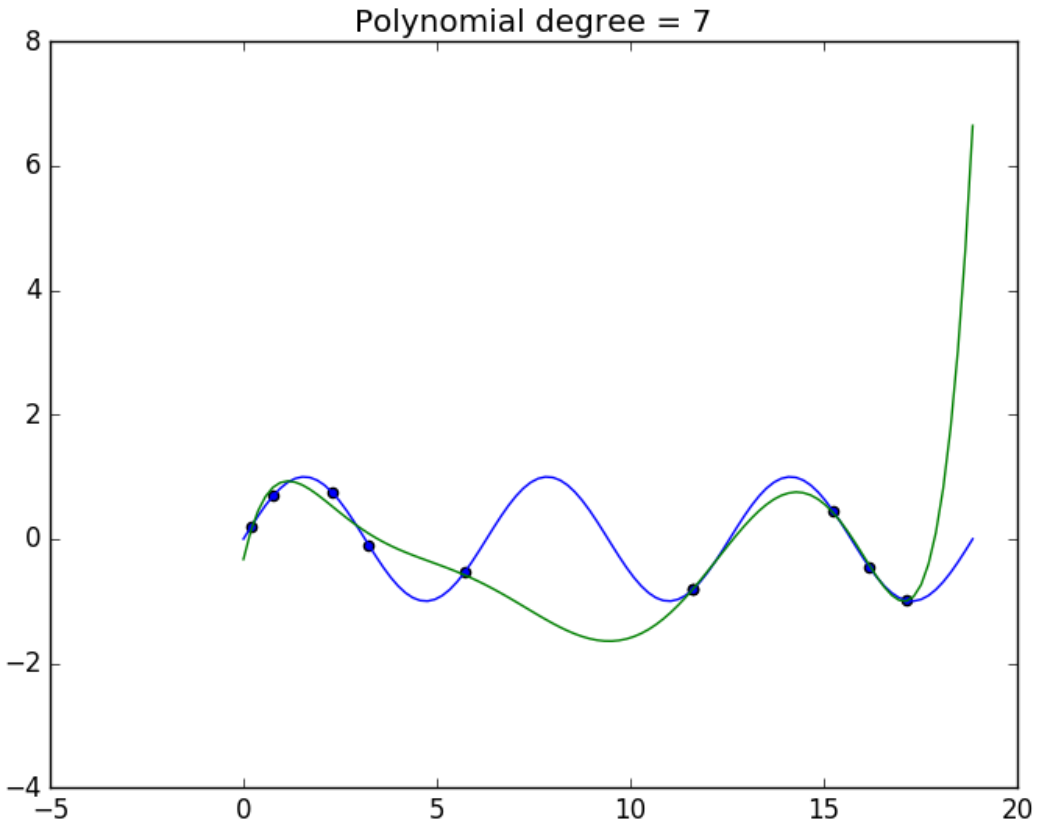


Figure 2-19 - Sine approximation with a degree seven polynomial

As we can see, a degree five polynomial does not fit well the data points. It may pass through some of them, but it actually underfits the data.

For a degree seven polynomial we have a different case. In this example, a degree seven polynomial fits almost every data point perfectly. Thus, if we were to choose this as out final hypothesis, we would have an in-sample error very close to zero. Although, we can see that in the areas that we do not have a sufficient number of examples, our model generalizes poorly.

This is the problem of overfitting. We can reduce the amount of in-sample error as much as we want, by increasing the complexity of our model. But the cost we pay is that it might not generalize well. Thus, because machine learning does not try to fit past data perfectly, but predict future behavioral patterns, overfitting is something that we should always take notice, when trying to solve such problems.

Figure 2-20 depicts how in-sample error correlates to out of sample error, showing perfectly the overfitting problem.

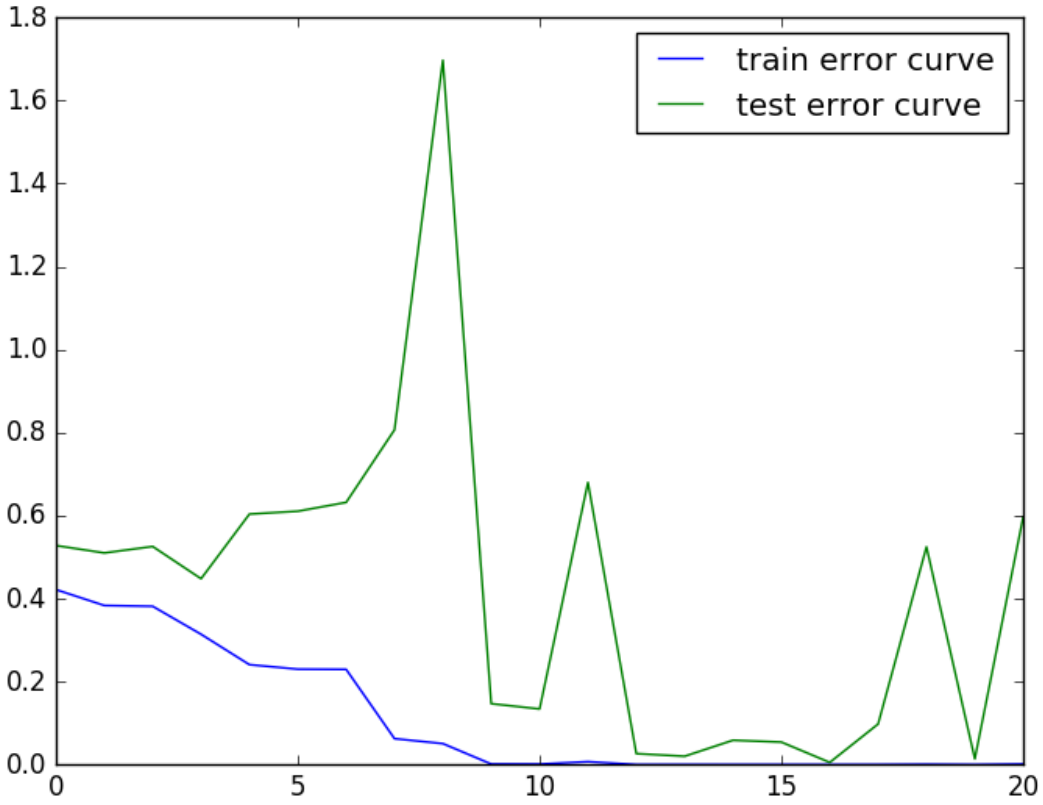


Figure 2-20 - Train error curve Vs Test error curve

2.13 Bias-Variance tradeoff

Bias-Variance tradeoff, gives us a different angle on generalization. It will be analyzed in this subchapter, along with a very helpful tool, which is called learning curves. Learning curves are mainly used to illustrate and visualize the learning process.

2.13.1 Approximation-generalization tradeoff

The purpose of learning, as discussed during the analysis of the theory, is to get a small out of sample error, $\boldsymbol{E_{out}}$, thus, generalize well and generate solid predictions. In other words, this means that the final hypothesis chosen by the system, approximates the target function (target probability distribution) well.

There is a delicate notion in the above sentence. Having small out of sample error has two components. The first one is that the final hypothesis approximates well the target function, while the second component implies that it approximates well the target function out of sample.

Thus, considering a more complex hypotheses set, we have better chance of approximating the target function. But as we saw, we might face an overfitting problem. Consequently, having less complexity gives us better chance of generalizing out of sample. So, the ideal situation would be if the target function lies within the hypothesis set; $\boldsymbol{H} = \{\boldsymbol{f}\}$.

Using the Bias-Variance analysis, we get a way of quantifying the approximation and generalization notions. It decomposes $\boldsymbol{E_{out}}$ to two pieces:

- How well $\boldsymbol{H}$ approximates $\boldsymbol{f}$
- How well can we discover a good $\boldsymbol{h} \in \boldsymbol{H}$

We are going to restrict this analysis to real-valued target function, such as the ones that linear regression tries to approximate, and use the squared error as our error measure.

Our starting point is $\boldsymbol{E_{out}}$, which we can quantify by comparing the value of the final hypothesis to the target function, and take the squared number of the result. The final hypothesis depends on several things. In mainly depends on the dataset that the system uses to learn. This is an important aspect for the Bias-Variance analysis, so, we should take special notice.

Thus, more formally:

$$\boldsymbol{E_{out}}(\boldsymbol{h^{(D)}}) = \mathbb{E}_{\boldsymbol{x}}[(\boldsymbol{h^{(D)}}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x}))^{\boldsymbol{2}}]$$

That formula describes the out of sample error of the hypothesis, derived from a specific dataset, as the expected value of the squared error. We take the expected value because we need to measure the error for the whole input space.

Now, to analyze the general behavior of the model, we need to get rid of the dependency of a specific dataset $\boldsymbol{D}$. To do that, we take the expected value of $\boldsymbol{E_{out}}(\boldsymbol{h^{(D)}})$; $\mathbb{E}_{\boldsymbol{D}}[\boldsymbol{E_{out}}(\boldsymbol{h^{(D)}})]$. So, the formula above now transforms to:

$$\mathbb{E}_{\boldsymbol{D}}[\boldsymbol{E_{out}}(\boldsymbol{h^{(D)}})] = \mathbb{E}_{\boldsymbol{D}}[\mathbb{E}_{\boldsymbol{x}}[\left(\boldsymbol{h^{(D)}}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x})\right)^{\boldsymbol{2}}]]$$

And in turn, by reversing the order of the expectations, this is equal to:

$$\mathbb{E}_{\boldsymbol{D}}[\boldsymbol{E_{out}}(\boldsymbol{h^{(D)}})] = \mathbb{E}_{\boldsymbol{x}}[\mathbb{E}_{\boldsymbol{D}}[\left(\boldsymbol{h^{(D)}}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x})\right)^{\boldsymbol{2}}]]$$

This allows us to focus on the inside quantity $\mathbb{E}_{\boldsymbol{D}}[\left(\boldsymbol{h^{(D)}}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x})\right)^{\boldsymbol{2}}]$, and when we get a clean decomposition, we can return and get the expected value with respect to $\boldsymbol{x}$.

2.13.2 The average hypothesis

In the previous subchapter, it is presented the idea of an average hypothesis, via the formula $\mathbb{E}_{\boldsymbol{D}}[\left(\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x})\right)^{\mathbf{2}}]$. This is an interesting notion, which will be analyzed next.

In order to understand this notion, think of a hypothesis set $\boldsymbol{H}$ and a dataset that the system uses to learn. We define the average hypothesis, $\bar{\boldsymbol{h}}(\boldsymbol{x})$, as follows:

$$\bar{\boldsymbol{h}}(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{D}}[\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x})]$$

In other words, we get a final hypothesis for every dataset we have. The form of the final hypothesis depends on the dataset the system uses, thus, for a different dataset we get a different hypothesis. Therefore, getting the expected value of every different hypothesis, with respect to all possible datasets, would be an ideal situation. The variable $\boldsymbol{x}$ is a fixed value and thus, we are exploiting the benefit of an infinite number of datasets on this point. Of course, in reality, we can never compute that. So, logically, in an ideal situation, where we have a great number of datasets, $\boldsymbol{D_1}, \dots, \boldsymbol{D_K}$, the perfect hypothesis would be:

$$\bar{\boldsymbol{h}}(\boldsymbol{x}) \approx \frac{\mathbf{1}}{\boldsymbol{K}} \sum_{\boldsymbol{k}=\mathbf{1}}^{\boldsymbol{K}} \boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x})$$

Next, we can use the notion of the average hypothesis to get the decomposition we are after. First, we need to transform it a little, by using a simple trick:

$$\mathbb{E}_{\boldsymbol{D}}\left[\left(\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x})\right)^{\mathbf{2}}\right] = \mathbb{E}_{\boldsymbol{D}}\left[\left(\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x}) + \bar{\boldsymbol{h}}(\boldsymbol{x}) - \bar{\boldsymbol{h}}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x})\right)^{\mathbf{2}}\right]$$

Then, we need to expand the square:

$$\mathbb{E}_{\boldsymbol{D}}\left[\left(\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x})\right)^{\mathbf{2}}\right] = \mathbb{E}_{\boldsymbol{D}}[\left(\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x}) - \bar{\boldsymbol{h}}(\boldsymbol{x})\right)^{\mathbf{2}} + \left(\bar{\boldsymbol{h}}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x})\right)^{\mathbf{2}}$$
$$+ \mathbf{2}(\left(\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x}) - \bar{\boldsymbol{h}}(\boldsymbol{x})\right)\left(\bar{\boldsymbol{h}}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x})\right)]$$

We can observe that we are asking for the expected value with respect to $\boldsymbol{D}$. This means that the quantity $\left(\bar{\boldsymbol{h}}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x})\right)$ is a constant. Thus, getting the expected value, with respect to $\boldsymbol{D}$, of $\mathbf{2}(\left(\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x}) - \bar{\boldsymbol{h}}(\boldsymbol{x})\right)\left(\bar{\boldsymbol{h}}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x})\right)$, diminishes to computing the expected value, with respect to $\boldsymbol{D}$, of $\left(\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x}) - \bar{\boldsymbol{h}}(\boldsymbol{x})\right)$.

$$\mathbb{E}_{\boldsymbol{D}}\left[\left(\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x}) - \bar{\boldsymbol{h}}(\boldsymbol{x})\right)\right] = \mathbb{E}_{\boldsymbol{D}}\left[\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x})\right] - \bar{\boldsymbol{h}}(\boldsymbol{x}) = \bar{\boldsymbol{h}}(\boldsymbol{x}) - \bar{\boldsymbol{h}}(\boldsymbol{x}) = \mathbf{0}$$

Thus, we see that our original formula diminishes to:

$$\mathbb{E}_{\boldsymbol{D}}\left[\left(\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x})\right)^{\mathbf{2}}\right] = \mathbb{E}_{\boldsymbol{D}}\left[\left(\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x}) - \bar{\boldsymbol{h}}(\boldsymbol{x})\right)^{\mathbf{2}} + \left(\bar{\boldsymbol{h}}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x})\right)^{\mathbf{2}}\right]$$

$$\mathbb{E}_{\boldsymbol{D}}\left[\left(\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x})\right)^{\mathbf{2}}\right] = \mathbb{E}_{\boldsymbol{D}}\left[\left(\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x}) - \bar{\boldsymbol{h}}(\boldsymbol{x})\right)^{\mathbf{2}}\right] + \left(\bar{\boldsymbol{h}}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x})\right)^{\mathbf{2}}$$

What we derived above can be explained by the following simple logic; the first term quantifies how much the final hypothesis differs from the target function. This is decomposed to the second term as follows; how much the final hypothesis, which was formed by exploring a specific dataset $\boldsymbol{D}$, differs from the average hypothesis, plus how much the average hypothesis differs from the target function.

The second term, $\left(\bar{\boldsymbol{h}}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x})\right)^{\mathbf{2}}$, describes the bias. This means, that even in an ideal situation, the results are biased away from the ground truth.

The first term, $\mathbb{E}_{\boldsymbol{D}}\left[\left(\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x}) - \bar{\boldsymbol{h}}(\boldsymbol{x})\right)^{\mathbf{2}}\right]$, describes the variance. This means, that if we had every information at our disposal, meaning all datasets, the best we could conclude is the average hypothesis, $\bar{\boldsymbol{h}}(\boldsymbol{x})$. But because we only work with one dataset, we come up with a hypothesis limited to this dataset, $\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x})$, thus, we are measuring how far we fall from $\bar{\boldsymbol{h}}(\boldsymbol{x})$.

Therefore, putting it all together, we result to the formula given below:

$$\mathbb{E}_{\boldsymbol{D}}[\boldsymbol{E}_{\boldsymbol{out}}(\boldsymbol{h}^{(\boldsymbol{D})})] = \mathbb{E}_{\boldsymbol{x}}[\mathbb{E}_{\boldsymbol{D}}[\left(\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x})\right)^{\mathbf{2}}]]$$

$$\mathbb{E}_{\boldsymbol{D}}[\boldsymbol{E}_{\boldsymbol{out}}(\boldsymbol{h}^{(\boldsymbol{D})})] = \mathbb{E}_{\boldsymbol{x}}[\boldsymbol{bias}(\boldsymbol{x}) + \boldsymbol{var}(\boldsymbol{x})]$$

Last, for a higher level of abstraction we get that:

$$\boldsymbol{E}_{\boldsymbol{out}} = \boldsymbol{Bias} + \boldsymbol{Variance}$$

For example, that means that if we have a learning situation, and the final out of sample error proves to be 0.2, we could say that 0.05 is because of bias and the rest is because of variance. In other words, 0.05 means that the hypothesis approximates well the target function, but it seems to be too complex, so it has 0.15 variance.

2.13.3 The tradeoff

In the previous subchapter, are defined the notions of bias and variance. We came up with two simple formulas to describe them:

$$\boldsymbol{Bias} = \mathbb{E}_{\boldsymbol{x}}[\left(\bar{\boldsymbol{h}}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x})\right)^2]$$

$$\boldsymbol{Variance} = \mathbb{E}_{\boldsymbol{x}}[\mathbb{E}_{\boldsymbol{D}}\left[\left(\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x}) - \bar{\boldsymbol{h}}(\boldsymbol{x})\right)^2\right]]$$

Considering these two quantities, we can see that there is a tradeoff. By increasing the complexity of our model, we get better approximation. On the other hand, higher complexity brings the problem of overfitting, thus, more bias.

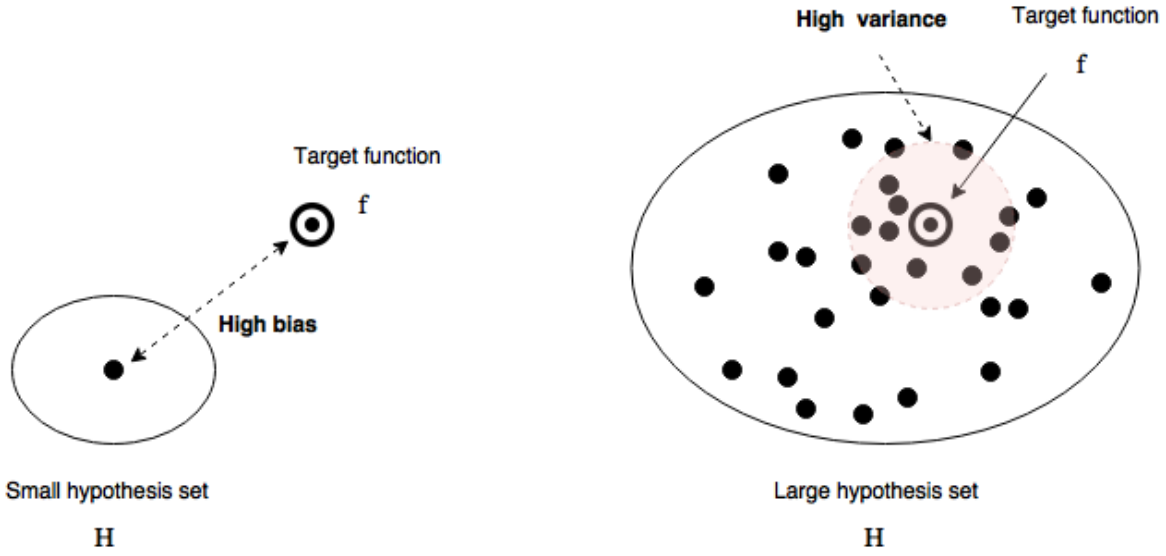
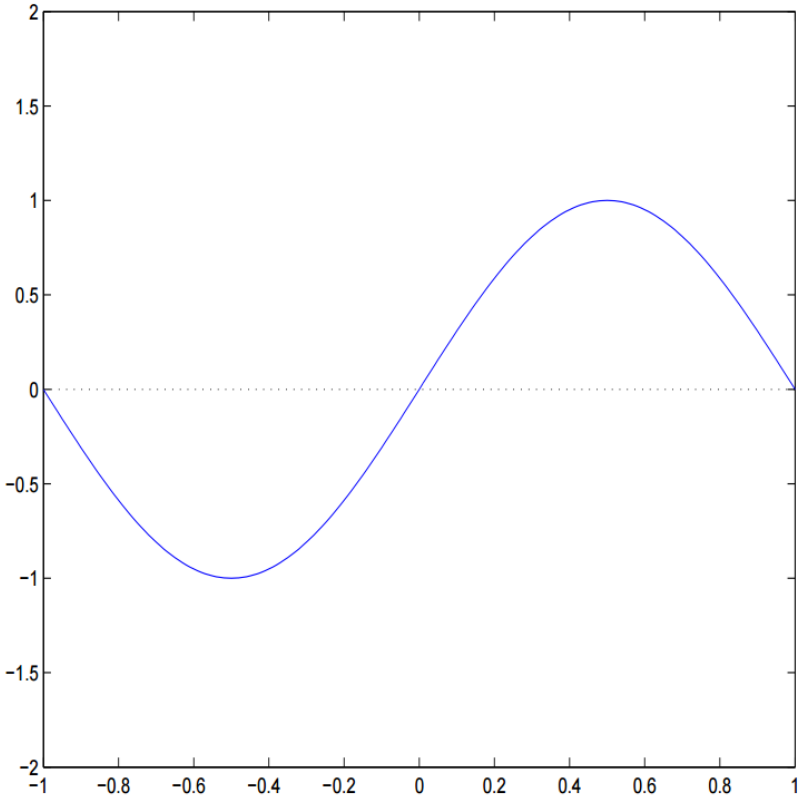


Figure 2-21 - Bias-Variance tradeoff

As we can see from Figure 2-21, having a small hypothesis set can cause high bias. This is because the final hypothesis cannot really approximate the target function well. In contrast, when we have a bigger hypothesis set of higher complexity, even in the case of a set that contains the target function, we get high variance.

For example, we will use the simple, sine function once more:



Two models used for learning:

$H_0 : h(x) = b$

$H_1 : H(x) = ax + b$

Figure 2-22 - Bias-Variance sine example

The system should try to learn the target function, using two hypothesis sets; the $\boldsymbol{H_0}$ and the $\boldsymbol{H_1}$, as show in Figure 2-22. The system will try to learn using two examples, so, $\boldsymbol{N = 2}$. Thus, concerning approximation, we expect that $\boldsymbol{H_1}$ will approximate the target function better:

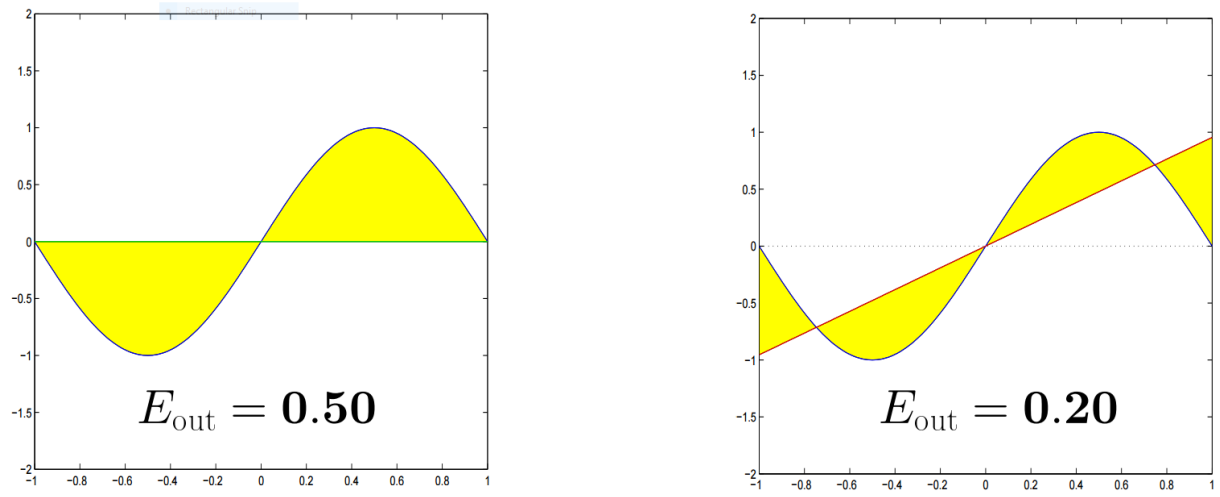


Figure 2-23 - Bias-Variance approximation

In the case of approximation, as expected we get a value of error that is higher on $\boldsymbol{H_0}$ than $\boldsymbol{H_1}$. So, in terms of approximation, knowing the target function, the more complexity we have the better it is. Let us now go through the case of learning.

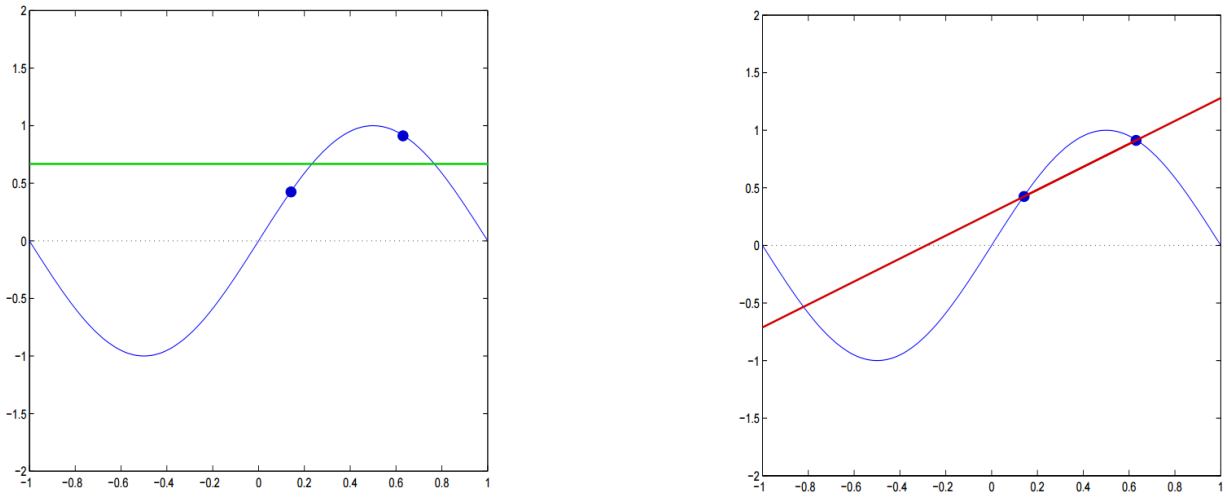


Figure 2-24 - Bias-Variance learning

In the case of learning, we have the two data points and we are trying to fit our model. So, trying to fit two data points with a line is an easy task for $\boldsymbol{H_1}$. When the process is completed, we bring back the target function and evaluate our two hypotheses. The problem is that the value of out of sample error depends on which data points we have. In other words, it depends on the dataset. Thus, we cannot be sure about how we compare $\boldsymbol{H_0}$ and $\boldsymbol{H_1}$. This is where the bias-variance analysis plays an important role. So, Figure 2-25 depicts the bias-variance decomposition for $\boldsymbol{H_0}$.

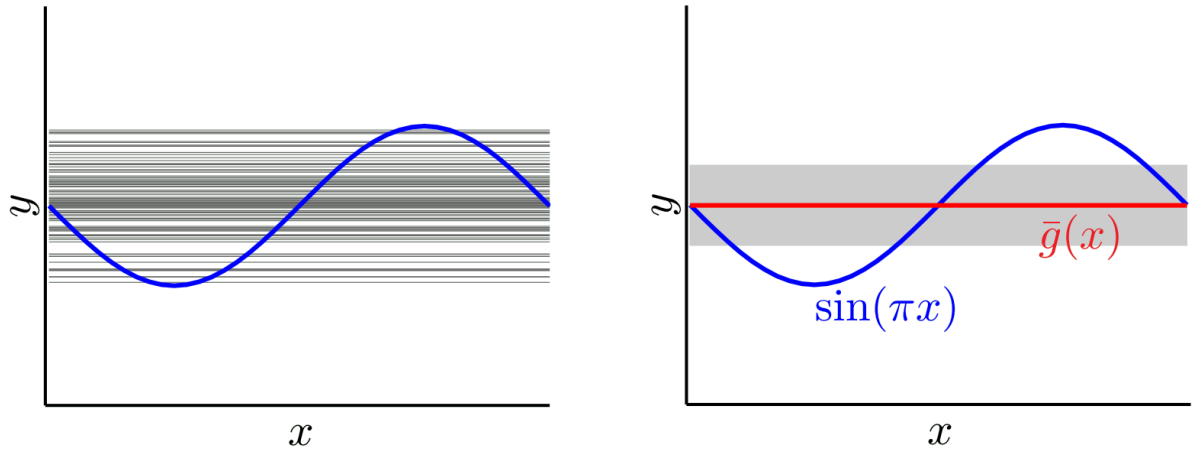


Figure 2-25 - Bias-Variance analysis for H_0

In Figure 2-25, $\bar{g}(x)$ is the average hypothesis we defined before. It is expected, by symmetry, that on average we will get a hypothesis close to zero. Thus, we see that the average hypothesis happens to be equal to the best approximation. This is not the result of learning though. The result of learning depends on the dataset and it can be any hypothesis depicted in the first schema of Figure 2-25. So, the grey area around the average hypothesis depicts the variance.

Next, let us do the corresponding to H_1 analysis.

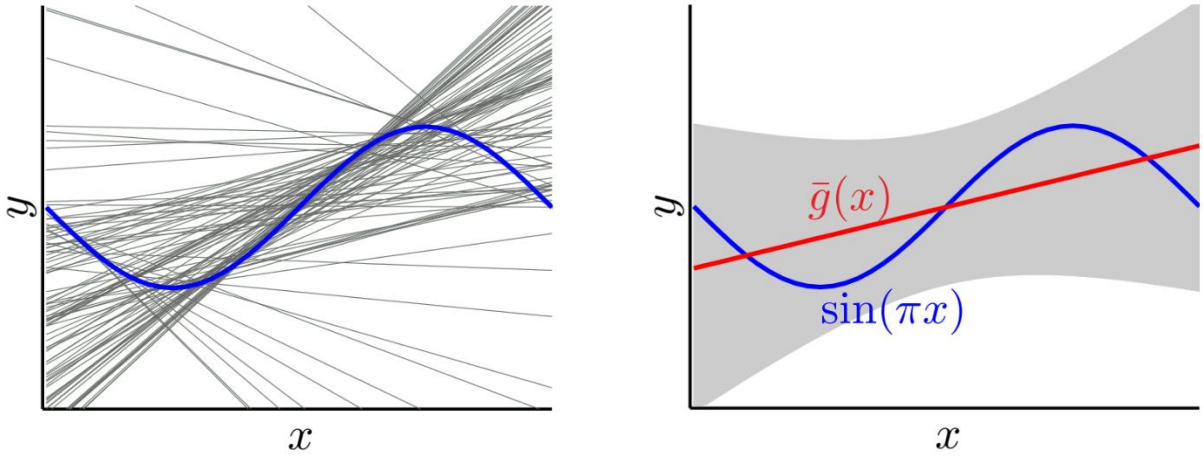


Figure 2-26 - Bias-Variance analysis for H_1

Now we can see that the variance increases. This is a great example of how a model with increased complexity has higher variance. However, the argument is that we get better approximation. Nevertheless, we can now compare.

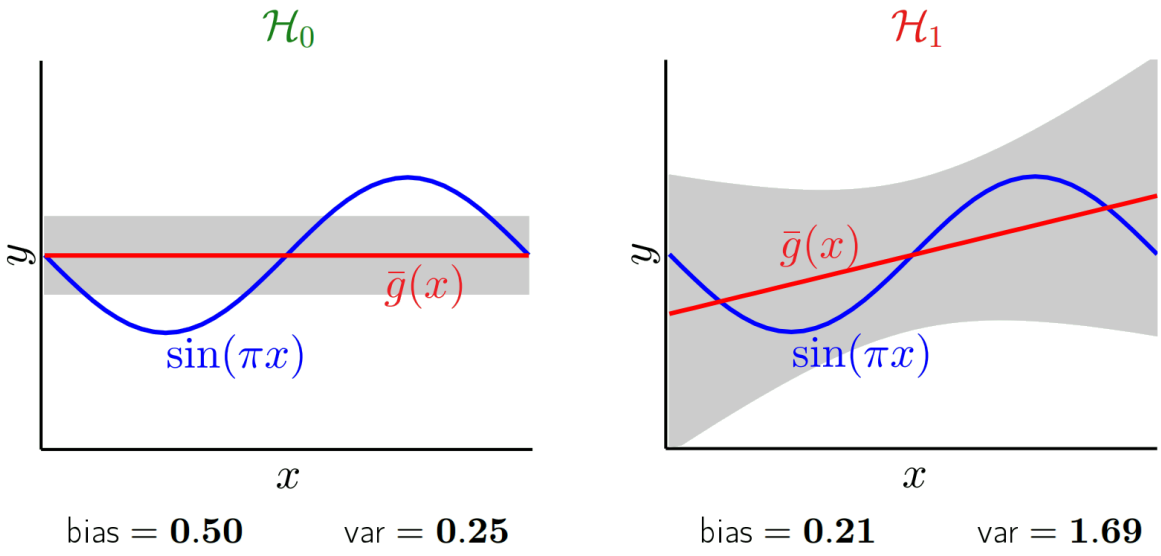


Figure 2-27 - Bias-Variance comparison

Finally, the big question is not if a constant or a line can approximate the sine function better. The big question comes from the learning perspective, and it wonders about what can we use to approximate better something that we don't know, using two examples.

Thus, in the example above, the out of sample error for a hypothesis set that uses just a constant is 0.75. Contrary, the out of sample error for a more complex hypothesis set is much higher.

The take-out from this analysis is that the model complexity should match to the data resources, not the target complexity [14].

2.14 Learning curves

So far, we looked only the out of sample error. Plotting the learning curves allows us to visualize both the in-sample and out of sample error, as a function of $\boldsymbol{N}$.

Thus, we have a dataset, $\boldsymbol{D}$ of size $\boldsymbol{N}$. The expected value of the out of sample error is computed as such; $\mathbb{E}_{\boldsymbol{D}}[\boldsymbol{E_{out}}(\boldsymbol{h}^{(\boldsymbol{D})})]$. Consequently, we have the expected value of the in-sample error; $\mathbb{E}_{\boldsymbol{D}}[\boldsymbol{E_{in}}(\boldsymbol{h}^{(\boldsymbol{D})})]$.

Below, Figure 2-28 shows the learning curves for a simple model of $\boldsymbol{N}$ data points:

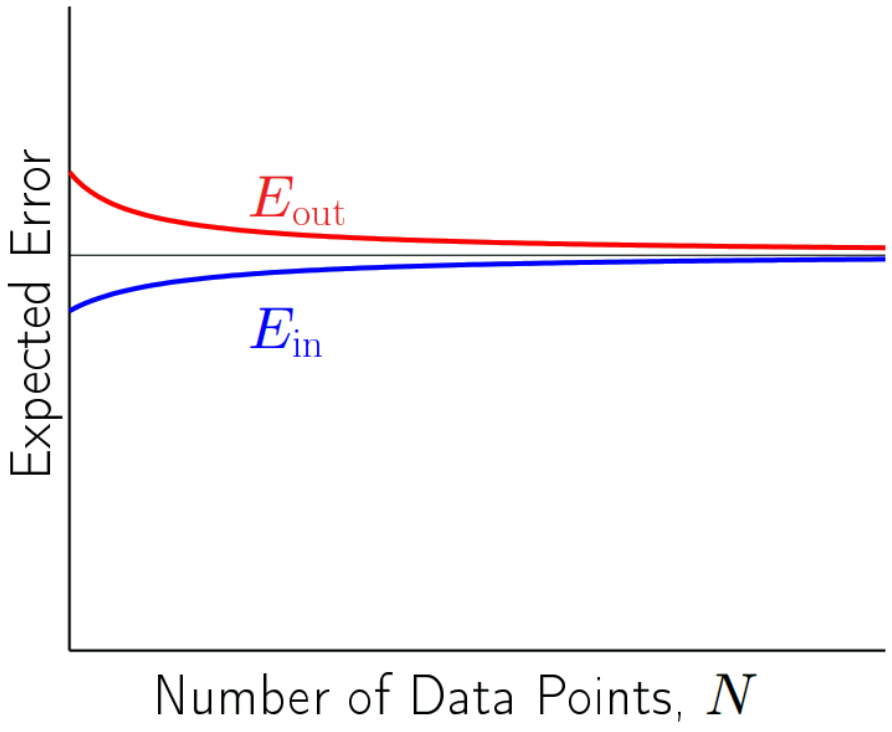


Figure 2-28 - Learning curves for a simple model

Because we have a simple model, for example a constant hypothesis like we discussed before, it does not approximate the target function well. Thus, we have a high expected error, as it is depicted by the black, horizontal line. This shows the best that our model can do.

During the process of learning, the more examples we have, the more likely it is for our hypothesis to approximate the target function. Thus, the out of sample error is decreasing. However, the in-sample error increases. At the beginning, there is few examples to fit, so this is an easy task even for a simple hypothesis set. As we increase the number of samples, the model has a hard time fitting them all, therefore, the in-sample error goes up.

The case of a complex model is similar:

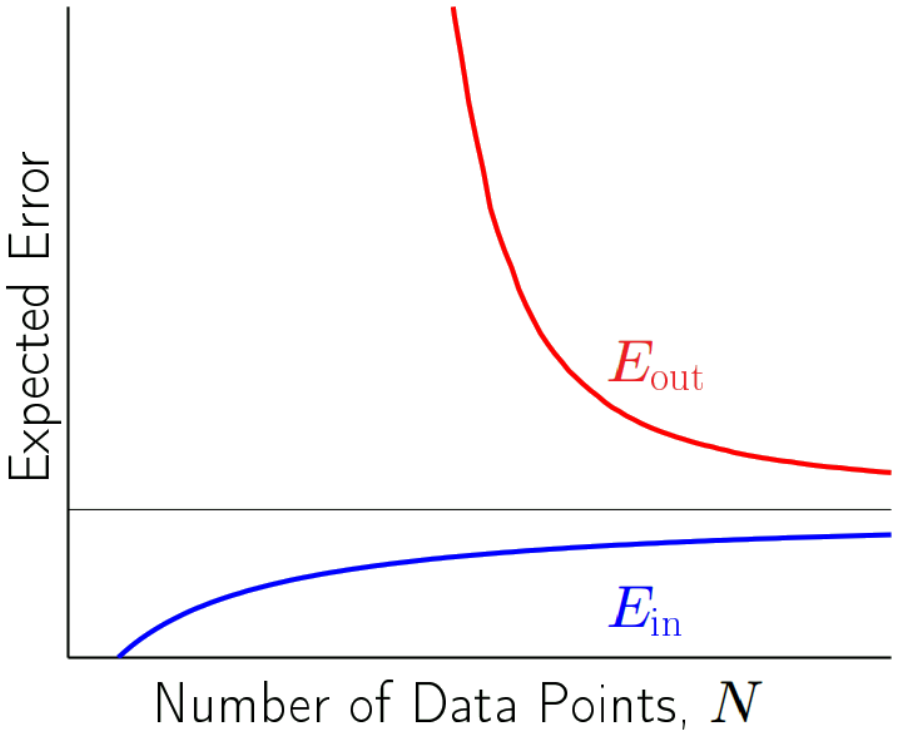


Figure 2-29 - Learning curves for a complex model

The difference is that the in-sample error is close or equal to zero at the beginning. This is because a complex model could fit every training example, and in turn produce a zero in-sample error, But the model cannot learn anything. Instead, it is as it memorized the training examples. On the other hand, because of the problem of overfitting, the out of sample error is huge at the beginning. So, we see, that even if our complex hypothesis fits every example in our training set perfectly, it does not generalize well. Going forward, we see that getting more training examples helps the case of the out of sample error, although the increase of the in-sample error.

Finally, visualizes the learning curves under the bias-variance analysis:

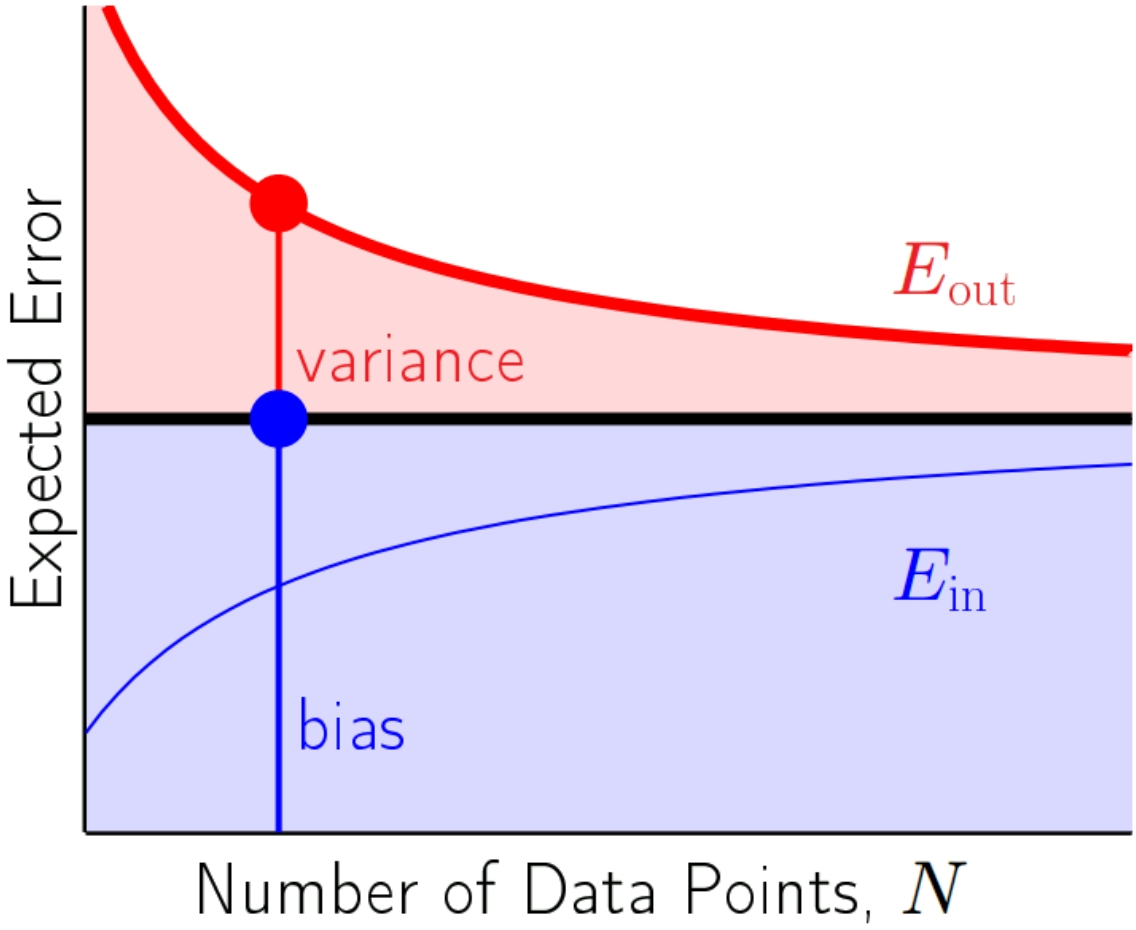


Figure 2-30 - Learning curves under Bias-Variance analysis

The bias is given by the blue region. The black horizontal line depicts the best-case scenario for this model, concerning the value of the out of sample error. This is given by considering the notion of the average hypothesis, introduced in previous subchapters, $\bar{\boldsymbol{h}}(\boldsymbol{x})$. So, the black line is the best pick in the hypothesis set, and below that is the error the system makes, by picking a different hypothesis.

The red area depicts the variance. So by taking the sum of the area of those regions, we compute the expected value of the out of sample error.

2.15 Regularization

Regularizations is the cure to the overfitting problem. Although there are a few other methods of attacking the problem of overfitting, for example increasing the number of training examples, regularization is used in any machine learning algorithm.

2.15.1 Approaches of regularization

There are two approaches in regularization in the bibliography [14]:

- Mathematical

  It mostly comes from function approximation. When we are facing the problem of approximating a function, and there are many functions that get the job done, we impose smoothness constraints on it, to be able to solve it.

- Heuristic

  In this case, we are hindering the minimization of the in-sample error. This is the most used one, as it attacks the problem in great specificity.

To better understand the notion of regularization, we are going to borrow an example from a previous subchapter. Figure 2-31 revisits the familiar example of trying to fit the sine function, using a linear model, with two training examples:

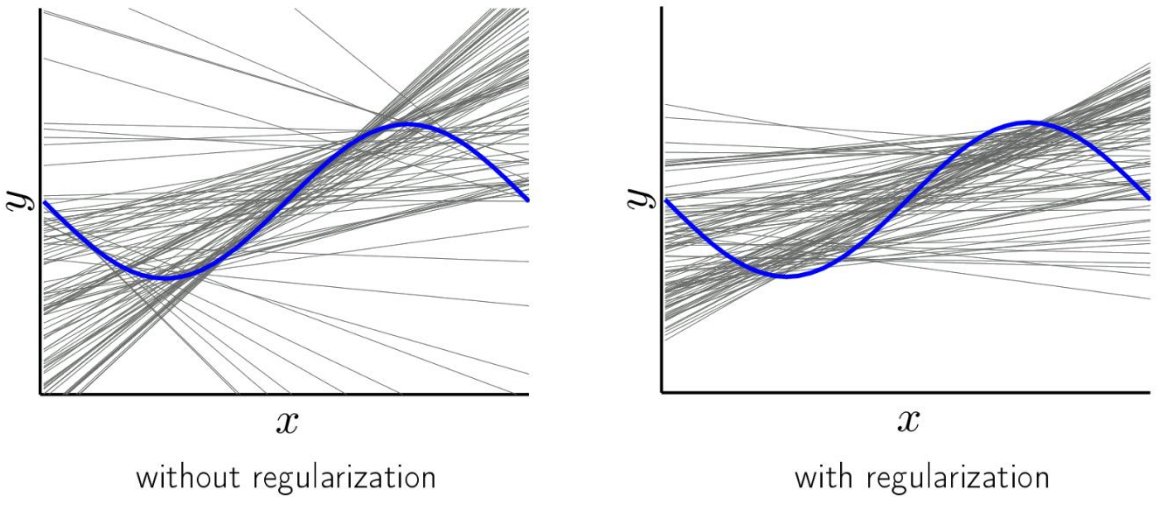


Figure 2-31 - Regularization example 1

As we know, without regularization, the main problem of this model is high variance. In this case, even a simplistic constant model performed better. By regularizing the model, we are restricting the lines (hypotheses), in terms of the offset and slope they can have. This results in sacrificing the perfect fit in-sample, but the gain in terms of variance is something to consider.

Figure 2-32 bellow depicts the final analysis of computing the out of sample error in the two different cases, with or without regularization.

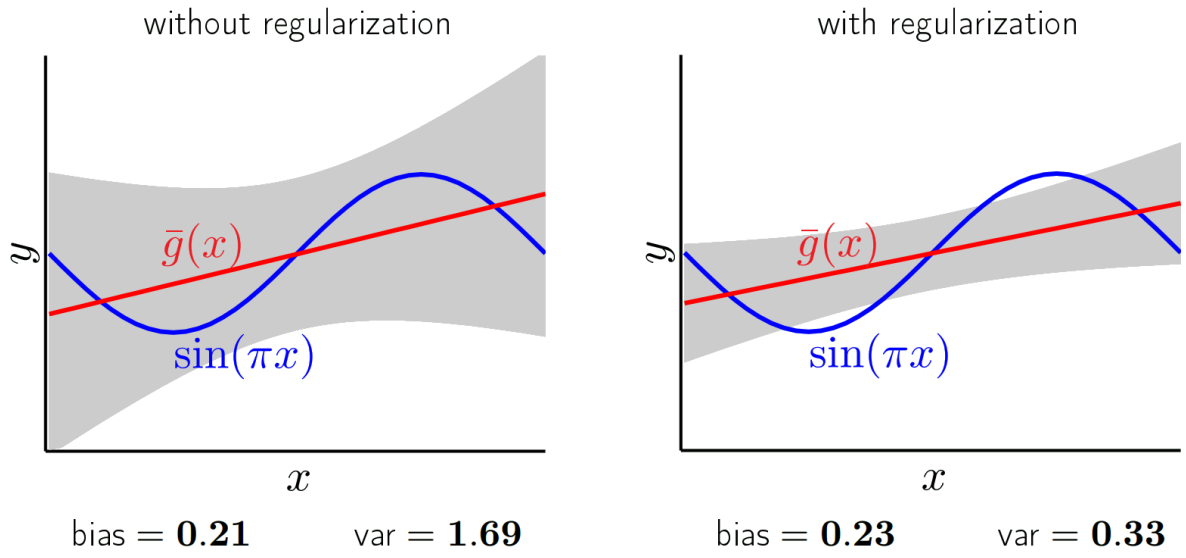


Figure 2-32 - Regularization example 2

The two obvious changes are that the grey area, which describes the value of variance, is diminished, and the bias value has slightly increased, because, now, we cannot fit the in-sample data perfectly. Thus, in general, regularization decreases the variance at the expense of slightly increasing the bias. As we said before, this is a cause of hindering the fit. Having the new values for bias and variance, after regularization, the linear model wins over the simplistic constant model.

2.15.2 Choosing a regularizer

The first and simplest regularizer is that of early stopping. This is perfectly depicted in Figure 2-33, below:

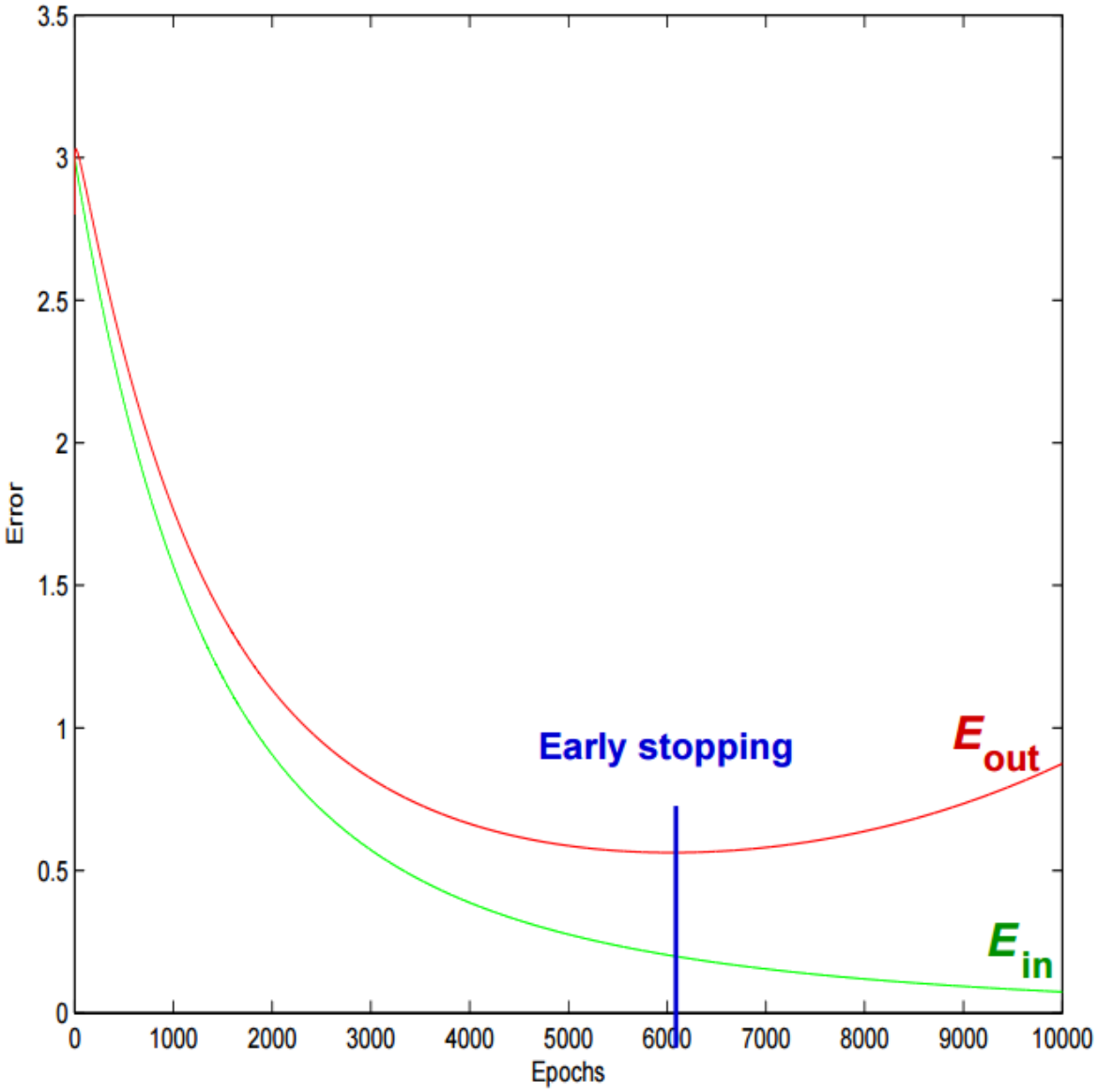


Figure 2-33 - Early-stopping regularizer

Using this approach, we are terminating the learning process, at the point of minimal out of sample error. This is the simplest approach of regularization and it does not provide a solid solution.

The next approach is called L2 regularization (ridge regression). This is used to attack outliers in out sample data. The idea is to modify the original cost function, by penalizing the large weights derived from the system.

The original cost function, using a mean squared error measure is modified as such:

$$\boldsymbol{J} = \sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} (\boldsymbol{h}(\boldsymbol{x}) - \boldsymbol{y})^{\boldsymbol{2}} + \boldsymbol{\lambda} \left|\left|\boldsymbol{w}\right|\right|^{\boldsymbol{2}}$$

Now, the new regularization term $\boldsymbol{\lambda}$ appeared, and what it tries to do is penalize the large weights $\boldsymbol{w}$, pulling their value downwards, in the process of minimizing the cost function $\boldsymbol{J}$. Thus, to compute the value of the weight vector $\boldsymbol{w}$, we get the following formula:

$$\boldsymbol{w} = (\boldsymbol{X^T X} + \boldsymbol{\lambda I})^{\boldsymbol{-1}} \boldsymbol{X^T X}$$

Another approach is called L1 regularization. This approach faces the problem when we have a small number of high dimensional training data. So, in general when $\boldsymbol{D} \gg \boldsymbol{N}$. The main notion of an L1 regularizer is that many features in are training data are contributing only noise. Thus, using L1 regularization we are setting most weights to zero.

The cost function after the modifications made from L1 regularization, transforms to:

$$\boldsymbol{J} = \sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} (\boldsymbol{h}(\boldsymbol{x}) - \boldsymbol{y})^{\boldsymbol{2}} + \boldsymbol{\lambda} ||\boldsymbol{w}||$$

In order to solve for w, we use an algorithm called gradient descent, due to singularity problems.

Chapter Three

3 Related work

Recommender systems and collaborative filtering has become an invaluable technique for reducing information overload. It is widely and successfully used on the web or web applications like Amazon.com, a retail giant, and Netflix, one of the most visited movie services today.

The most widely used approach, when it comes to recommender systems, is this of collaborative filtering. This term is coined by the developers of the first recommender system, Tapestry [15]. Tapestry was an experimental mail system, developed at Xerox Palo Alto Research Center, which supported both content-based and collaborative filtering, by introducing the human element into the information filtering process. Tapestry was intended to handle any incoming stream of electronic documents and server both as a mail filter and repository.

Since then, the area of recommender systems has been extensively researched, and the results are divided into two broad categories; neighborhood models and latent factor models.

3.1 Neighborhood models

Neighborhood models sets the base and the most common approach to collaborative filtering and recommendation systems. They, in turn, divide into two categories; user-oriented and item-oriented.

User-oriented approaches, predicted the missing ratings, by identifying similar minded users. On the other hand, item-oriented approaches strive to group similar items, and fill in the missing ratings by looking at how the same user rates similar items.

3.1.1 User-oriented neighborhood models

Information filtering systems used content-based filtering, to arrive to a filtering decision. For this to work, content-based tools gather information about each item and match them with whatever the interest of users is. Content-based techniques are successful in filtering textual documents, which might be of interest to a specific user. To achieve that, content-based tools

utilize techniques such as vector-space queries [16], intelligent agents [17] and information visualization [18].

On the other hand, user-oriented collaborative filtering approaches, collects user feedback, also known as ratings, for items within a specific domain, i.e. movies, and match together users with the same preferences or needs. This way, and automated collaborative filtering process, provides personalized recommendations to users.

Furthermore, user-oriented collaborative filtering approaches, have three main advantages over content-based information filtering systems [19]. They can:

- Filter items whose content is not easy to analyze, thus, cannot easily extract insightful information
- Filter items based on quality, taste or needs
- Provide serendipitous recommendations

First, collaborative filtering approaches can enable filtering even to items that are hard or time consuming to analyze with computers, such as movies, music or people.

Second, collaborative filtering systems can measure user preference in dimensions that are not easily captured by a simple information filtering system. Those dimensions can prove to be too vague for a human to capture, during a content-based approach, and too hard to analyze by computer systems.

Finally, collaborative filtering systems can introduce items that were unexpected, or the user have never heard of. Especially in art domain, movies or song recommendations are often serendipitous, allows the user to expand his or her preferences in unexpected ways.

The problem scope of any collaborative filtering system is to predict how a user would like an item that has not rated yet, given his historical data, on preferences and judgements. Those preferences come in the form of explicit or implicit feedback, as we saw during the introduction.

Next, the system collects all feedback and through the collaborative filtering algorithm provide its predictions. In our case, the user-oriented approach, an active user provides a list of items and the engine returns the predictions for those items. Furthermore, with a minor modification, the engine most of the times returns the top-k results [20] [21] [22].

The problem space can be visualized as a matrix, with each cell holding the rating of the user to a specific item. The missing ratings, the ratings the system is expected to predict, are empty cells in that matrix. In the case of collaborative filtering, most of the times, every user has rated only a few items, resulting to a very sparse matrix. Figure 3-1 depicts this matrix in the form of a table:

Movies/Users	*Lion*	*Arrival*	*Kubo*	*Moonlight*
John	1	3	5	?
Peter	5	1	?	4
James	?	4	3	5
Kelly	1	2	5	1

Figure 3-1 - User-Movies sparse matrix

User-oriented collaborative filtering approaches are the on-side of the neighborhood-based models in recommendation systems. In such systems, a subset of users, who are similar to an active user, are chosen and a weighted aggregate of their ratings on a specific item is calculated, in order to generate new predictions for the active user.

User-oriented neighborhood-based models use the methodology described below [19]:

1. Weight all the users with respect to similarity to an active user
2. Select a set of predictors, a set consisting of users that are most similar to the active user
3. Compute predictions from a weighted aggregate of selected neighbors' ratings

For example, using Figure 3-1, let us assume that we are trying to predict the rating for the movie Moonlight, that user one, John, would give. Seems that Kelly is the best neighbor for John, because they have rated the movies they both have seen similarly. Thus, Kelly's opinion would John influence more than Peter's.

One of the first to introduce a neighborhood-based, user-oriented collaborative filtering system was GroupLens [23] [21]. GroupLens has a few featured projects. One of them is MovieLens, that is a huge resource in the area of movie recommendations, Cyclopath, which discovers bike routes that match the way a user rides, and LensKit, an open source toolkit for building, researching, and studying recommender systems.

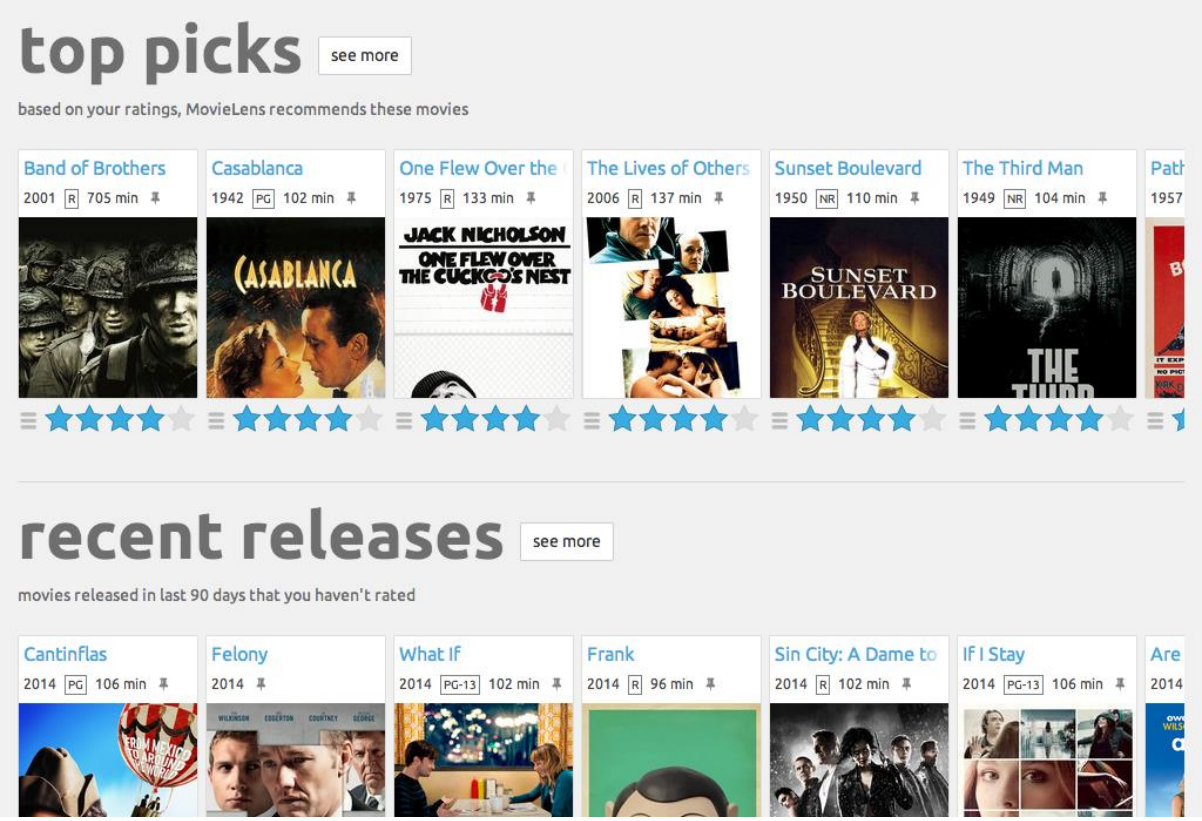


Figure 3-2 - MovieLens - A featured project of GroupLens

The original system user Pearson correlations to measure user similarity, used all available related neighbors and computed a weighted average of deviations from the neighbor's mean, as show below:

$$p_{a,i} = \overline{r_a} + \frac{\sum_{u=1}^{n}(r_{u,i} - \overline{r_u}) * w_{a,u}}{\sum_{u=1}^{n} w_{a,u}}$$

The quantity $p_{a,i}$ represents the predicted rating of an active user a for an item i. The number of neighbors is defined as n and $w_{a,u}$ is the similarity weight, between the active user and neighbor u as defined by the Pearson correlation coefficient:

$$w_{a,u} = \frac{\sum_{i=1}^{m}(r_{a,i} - \overline{r_a}) * (r_{u,i} - \overline{r_u})}{\sigma_a - \sigma_u}$$

The Ringo music recommender [22], improved the GroupLens model, using a restricted Pearson correlation coefficient:

$$w_{a,u} = \frac{\sum_{i=1}^{m}(r_{a,i} - 4) * (r_{u,i} - 4)}{\sigma_a - \sigma_u}$$

The number four was chosen because it was the median of a seven-scaled rating system. The strong point of Ringo music recommender was the greater accuracy, because of the selection of users whose correlation was greater than a fixed high threshold. On the other hand, Ringo music recommender could not make predictions for many items, because of this restriction.

Another approach, the Bellcore video recommender [20], used Pearson correlation coefficient to weight a random sample of neighbors, and consequently, selecting only the best of those. Finally, the system preformed a multiple regression on them to create a prediction.

In an empirical analysis, Breese et al. [24] found than the Pearson correlation coefficient performed better as a similarity metric than cosine vector similarity, on many of the previously mentioned algorithms.

On the other hand, B. Sarwar et al. [7] enumerates the challenges of user-oriented approaches. They argue that although the user-oriented approach has been very successful in the past, years of experience have undisclosed their various challenges.

Firstly, the main problem is that of sparsity. In big, retail datasets, active users have purchased very few products, as a fraction to the total number of products. As a result, the accuracy of recommendations may be poor or groundless.

The other important issue is that of scalability. Having a great number of customers and products, to the count of millions, can pose great problems to existing neighborhood model algorithms. This is because the computation of nearest neighbors algorithms grows with both the number of users and the number of items.

Those limitations, led researches to consider alternative methods in order to build trusted recommendation systems.

3.1.2 Item-oriented neighborhood models

User-oriented neighborhood models for collaborative filtering, quickly lost their appeal to later analogous item-oriented approaches. In such methods, the ratings are predicted using known known ratings that the user has previously given to similar items.

The item-oriented approach has a few significant advantages over the analogous, user-oriented approach:

- Better scalability
- Improved accuracy
- Prominent reasoning behind predictions

Trying to create targeted advertisement and more accurate email campaigns, Amazon used recommendations throughout most of their online presence, including their high traffic homepage [25]. When a user added an item into their shopping cart, and during the process of checkout, Amazon offered recommendations based on the items placed inside the previously mentioned shopping cart. This way, Amazon digitalized the notion of impulse items placed in a supermarket checkout line, but it enhanced the idea by offering what we would might call "personalized impulse items".

Figure 3-3 - Amazon's impulse products recommendations

Using item-to-item collaborative filtering, Amazon built a recommendation algorithm that could scale to its huge catalog, covering tens of millions of customers and products. Moreover, the Amazon algorithm could produce high-quality recommendations in real time.

Rather than computing the similarity between users, Amazon's item-to-item collaborative filtering created recommendation lists consisting of items similar to what the customer had inside his shopping cart [26].

Figure 3-4 - Amazon's recommendation engine

To determine the similarity between items, the algorithm creates a table of items that the customer tends to purchase together. Figure 3-5 shows the iterative algorithm that was used to compute the similarity between a single product and every related item:

```
For each item in product catalog, I1
  For each customer C who purchased I1
    For each item I2 purchased by
       customer C
      Record that a customer purchased I1
       and I2
  For each item I2
    Compute the similarity between I1 and I2
```

Figure 3-5 - Amazon item-to-item similarity computation

B. Sarwar et al. [7] studied a class of item-base collaboration filtering algorithms, that consider a set of items previously purchased or liked by a specific user, and compute the similarity between those item and previously unseen ones. The main notion is that a user would like to be presented with items that are similar to ones that he liked before. However, a user would not care about items that are similar to others that he has previously expressed a negative opinion. This approach attacks mainly the sparsity problem, thus, improves the accuracy of such recommendation engines.

Finally, the Gravity Recommendation System (GRS), which attained RMSE 0.8743 in the Netflix Prize Contest [8], discusses also the problem of scalability. However, it is important to note, that it is easier to measure the success of an item-to-item recommendation approach. For example, we can compare two movies, based on their genres, cast or director, and conclude about the performance of the algorithm. On user-oriented approaches, comparing two users could prove to be much vaguer.

3.2 Latent factor models

Latent factor model aim to uncover latent features that could explain the ratings given by users. This approach associates every user with a weight vector $\boldsymbol{x_u}$ and each item with a corresponding feature vector $\boldsymbol{y_i}$. Examples of this approach include pLSA [27], neural networks [28], and Latent Dirichlet Allocation [29].

In this approach, every user is associated with a weigh vector $\boldsymbol{x_u}$, which encapsulates the user's preferences. On the other side, each item is defined by a feature vector $\boldsymbol{y_i}$, which, in essence, holds the features that characterizes the item. Finally, the predicted rating comes for the dot product of those vectors:

$$\boldsymbol{r_{u,i} = x_u^T y_i}$$

The cost function that we need to optimize, i.e. minimize, is using regularization to avoid the problem of overfitting:

$$\min_{x_*, y_*} \sum_{r_{u,i} \text{ is known}} \left(p_{u,i} - x_u^T y_i\right)^2 + \lambda \left(||x_u||^2 + ||y_i||^2\right)$$

where $\boldsymbol{\lambda}$ is the regularization parameter.

In this approach, we randomly initialize the users' weight vectors, as well as the items' feature vectors. Then, by keeping the feature vectors fixed, we are iterating over the training examples to get a better estimate for the weight vectors. Next, we are using the weight vectors that we learned in the previous step, to get a better estimate for the feature vectors. This happens repeatedly, and after many iterations the algorithm produces exceptional results. Figure 3-6 depicts the process.

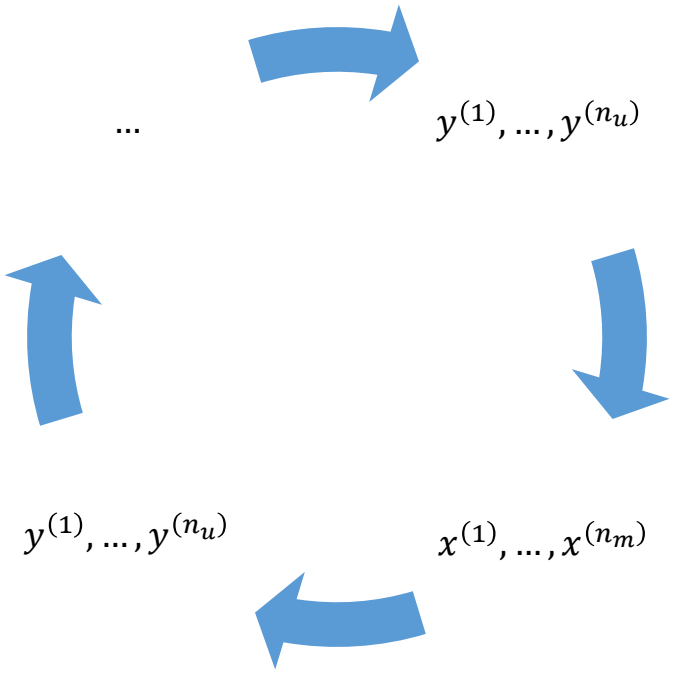


Figure 3-6 - Latent factor models process

This model yielded great results on the Netflix dataset, which was the largest available dataset to date [30].

In this work, we borrow this approach, as well as the approach presented by Yifan Hu et al. [11], which applies the latent factor model approach to implicit feedback datasets. In this case, the ratings matrix becomes a binary preference matrix, that captures whether there was an interaction between a user and an item.

$$\boldsymbol{p_{u,i}} = \begin{cases} \mathbf{1}\ \boldsymbol{r_{u,i}} > \mathbf{0} \\ \mathbf{0}\ \boldsymbol{r_{u,i}} = \mathbf{0} \end{cases}$$

In the place of ratings, $\boldsymbol{r_{u,i}}$ is filled with an implicit feedback, such as the quantity of purchased products, the minutes watch for a show etc.

In addition, there is a confidence matrix, associated with every interaction, that characterizes the confidence we have about any interaction. A simple way to capture this confidence is:

$$\boldsymbol{c_{u,i}} = \mathbf{1} + \boldsymbol{\alpha r_{u,i}}$$

where $\boldsymbol{\alpha}$ is a constant, that controls the rate of confidence increase. For example, if there is no interaction between a user and a product, the confidence takes the value one, thus, it does not contribute anything to our model.

The confidence matrix is the element that we will focus most in this work. We will try and optimize the heuristics of this function for the retail sector, where many aspects should be considered. For example, the stock quantity or the great variety in product prices.

Chapter Four

4 Proposed Model

In this section, we define the model used in this work. The goal is to build a recommendation system, tailored to fit the needs, and overcome the challenges of the retail sector. In most cases, the retail sector cannot benefit from the use of explicit user feedback. This is because the product catalogue count is in the thousands, and even if the store asks users about their opinion, the organization's analysts cannot rely on that and, of course, cannot base their findings on such sparse data.

This means that the users have not expressed any preference for the products they buy, and moreover, we cannot assume anything about the products they have not interacted with yet. Thus, we should work based on implicit feedback, that we extract observing the consumers' behavior. This approach has a few special characteristics that are described below [11]:

- We cannot extract negative feedback. By observing every consumer interaction, we can only assume that they show a preference to the products they purchase. But we do not have any knowledge about the products they do not choose to consume. They might not like them, but it is equally likely that they do not have any knowledge of their existence. This asymmetry does not appear on explicit feedback dataset. In the case of explicit feedback, the user is expected to provide a feedback in the form of a rating, that scales within a numerical range. For example, if the user rates a movie using five stars rating system, with one defining the worst-case scenario while five means brilliant, the user can express his preference or not. This leads explicit feedback recommender systems to treat the not yet rated items as missing values. Thus, their job is to fill those missing values. That could not be the case of implicit feedback recommender systems, because considering only the gathered data would leave us with only the positive feedback. That could lead to great misinterpretations of user profiles. Hence, it is important to mix the missing data into the recipe, because this is where the majority of negative feedback would lie.
- Implicit feedback is inherently noisy. This means that although we can guess that a user likes the product that he bought, we can never be too certain. A user might have purchased something and ended up not liking it. Or even he bought a specific product as a gift. And how can we take into consideration the return of a product? These questions

and challenges make the implicit feedback problem much more complex than having to deal directly with the user opinion, in the case of an explicit rating.

- We should treat the numeric value of implicit feedback as a confidence level. That means that while the explicit feedback numerical value, i.e. the rating, indicates preference, the implicit feedback equivalent indicates how confident we are that a user interaction shows preference. As we saw before, the value of an explicit rating indicates preference if it is in the upper levels of the rating scale and dislike if not. The numerical value of the implicit feedback can describe many aspects of a user-item interaction. For example, it can indicate the frequency of this interaction, how many times a customer purchased a product. However, this can be very useful. For example, if a user has purchased a product many times, we have a strong indication that this user likes the specific item.
- The evaluation metrics of implicit-feedback recommender systems has also its implications. In explicit feedback cases, a clear metric such as the mean squared error can easily measure the success of the algorithm. Using implicit feedback datasets brings other aspects into consideration though. For example, we should address the issue of out of stock products, or the returns of a product.

4.1 Preliminaries

We are considering a set of users $\boldsymbol{X}$, which consists of user vectors $\boldsymbol{x_1}, \dots, \boldsymbol{x_u}$, where every vector characterizes the user's preferences. In other words, each vector holds the weights, which define the preference of each user for a specific item feature. Thus, the user vectors' shape can be thought of $\boldsymbol{D} \times \boldsymbol{1}$, where $\boldsymbol{D}$ is the number of dimensions, i.e. the number of product features. The matrix $\boldsymbol{X}$, which holds all the users takes the form of:

$$\begin{bmatrix} x_1^{(1)} & \cdots & x_1^{(d)} \\ \vdots & \ddots & \vdots \\ x_u^{(1)} & \cdots & x_u^{(d)} \end{bmatrix}$$

where every row is a transposed user vector.

Similarly, the set of products $\boldsymbol{Y}$, which consists of item vectors $\boldsymbol{y_1}, \dots, \boldsymbol{y_i}$, where every vector characterizes the product's features. Thus, the corresponding matrix $\boldsymbol{Y}$ is:

$$\begin{bmatrix} y_1^{(1)} & \cdots & y_1^{(d)} \\ \vdots & \ddots & \vdots \\ y_i^{(1)} & \cdots & y_i^{(d)} \end{bmatrix}$$

We consider another matrix $\boldsymbol{R}$, which holds the user-item interactions. Thus, $\boldsymbol{r_{u,i}}$ holds a numerical value, that holds the interaction between user $\boldsymbol{u}$ and item $\boldsymbol{i}$. Below, Figure 4-1 depicts a visual representation of our recommendation problem:

	User 1 x_1	User 2 x_2	User 3 x_3	User 4 x_4	
Item 1	5	3	0	0	y_1
Item 2	5	0	7	0	y_2
Item 3	0	4	0	0	y_3
Item 4	0	0	5	4	y_4
Item 5	11	0	9	0	y_5

Figure 4-1 - Visual representation of the recommendation problem

Each item is characterized by a vector $\boldsymbol{y_i}$ and similarly each user by a vector $\boldsymbol{x_u}$. The numerical values, in place of an explicit rating, indicate confidence. For example, let us consider the quantity of a purchased product. Thus $\boldsymbol{r_{1,1}}$ is equal to five, indicating that the first user has purchased the first product five times. The zeros in the sparse matrix $\boldsymbol{R}$ are indicating no interaction between user $\boldsymbol{u}$ and item $\boldsymbol{i}$.

4.2 The model

First, we need to formalize the notion of confidence, which is captured by the matrix $\boldsymbol{R}$. To achieve this, we need to introduce a binary variable, that show preference. We call this variable $\boldsymbol{p_{u,i}}$, and it indicates the preference of user $\boldsymbol{u}$ for the item $\boldsymbol{i}$. Formally we write:

$$\boldsymbol{p_{u,i}} = \begin{cases} \mathbf{1}\ \boldsymbol{r_{u,i}} > \mathbf{0} \\ \mathbf{0}\ \boldsymbol{r_{u,i}} = \mathbf{0} \end{cases}$$

Thus, if the user $\boldsymbol{u}$ has interacted with the item $\boldsymbol{i}$, we assume that the user has shown a preference for this specific item. On the other hand, if the user has not yet interacted with this item, we have no indication ($\boldsymbol{p_{u,i}} = \mathbf{0}$).

However, we should capture the notion of how confident we are, for every user-item interaction. Somehow, we should associate low levels of preference with low confidence and work from the bottom up. This does not mean that low levels of confidence indicate dislike. We are just trying to capture the various levels of uncertainty, which exist in absence of data. Finally, the main notion is that as the numerical value of $\boldsymbol{r_{u,i}}$ raises, the levels of confidence become stronger.

Formally, we introduce a set of variables, $\boldsymbol{c_{u,i}}$, that capture the level of confidence for the interaction $\boldsymbol{r_{u,i}}$. A simple heuristic function that could take this role could be:

$$\boldsymbol{c_{u,i}} = \mathbf{1} + \boldsymbol{\alpha r_{u,i}}$$

where $\boldsymbol{\alpha}$ is a constant that controls the increase rate of the confidence.

Now, we have a minim confidence for every interaction, even if there is no interaction at all. In this case, the confidence measure takes its lower value, and it is equal to one.

The goal is to find the vectors $\boldsymbol{x_u}$ and $\boldsymbol{y_i}$, in $\mathbb{R}^{\boldsymbol{f}}$, for every user and item, where $\boldsymbol{f}$ is the number of item features and, consequently, user weights. Finally, we should have:

$$\boldsymbol{p_{u,i} = x_u^T y_i}$$

The vectors $\boldsymbol{x_u}$ and $\boldsymbol{y_i}$ are called user-factors and item-factors respectively. Thus, the technique we are using is similar to the matrix factorization techniques, used in latent factor models. There are a couple of differences though:

- We need to introduce the notion of confidence in our model
- We need to take into consideration every data point in the dataset not only the known values

Thus, the final cost function of our model is shown below:

$$\min_{x_*, y_*} \sum_{r_{u,i} \text{ is known}} c_{u,i}\left(r_{u,i} - x_u^T y_i\right)^2 + \lambda\left(||x_u||^2 + ||y_i||^2\right)$$

where $\boldsymbol{\lambda\left(||x_u||^2 + ||y_i||^2\right)}$ is performing the necessary regularization and $\boldsymbol{\lambda}$ is the regularization parameter.

Solving for $\boldsymbol{x_u}$ and $\boldsymbol{y_i}$, as we would in the case of a linear model such as linear regression, we have the two equations that calculate the user-factors and item-factors:

$$\boldsymbol{x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u)}$$

$$\boldsymbol{y_i = (X^T C^i X + \lambda I)^{-1} X^T C^i p(i)}$$

4.3 Mathematical process

The cost function contains $\boldsymbol{m \times n}$ terms, where $\boldsymbol{m}$ is the number of users and $\boldsymbol{n}$ the number of items in this case. This could easily reach numbers that are in millions or even billions in many real-world datasets. Thus, the standard techniques of stochastic gradient descent used for explicit dataset system optimization, cannot be applied light-heartedly.

In this case, we will use the alternating least squares optimization process, where we alternate between user and item factors, keeping one fixed at any point.

First, for each user $\boldsymbol{u}$ we define a diagonal $\boldsymbol{n} \times \boldsymbol{n}$ matrix $\boldsymbol{C^u}$, where $\boldsymbol{C^u_{(ii)}} = \boldsymbol{c_{u,i}}$. Also, we construct the vector $\boldsymbol{p(u)} \in \boldsymbol{R^n}$, which holds every preference of user $\boldsymbol{u}$, i.e. the $\boldsymbol{p_{u,i}}$ values. By differentiating the cost function, with respect to $\boldsymbol{x_u}$, we arrive at the statement we saw before:

$$\boldsymbol{x_u} = (\boldsymbol{Y^T C^u Y} + \boldsymbol{\lambda I})^{-1} \boldsymbol{Y^T C^u p(u)}$$

Using the same process and differentiating with respect to $\boldsymbol{y_i}$, we get the value of $\boldsymbol{y_i}$:

$$\boldsymbol{y_i} = (\boldsymbol{X^T C^i X} + \boldsymbol{\lambda I})^{-1} \boldsymbol{X^T C^i p(i)}$$

Chapter Five

5 Experimental Study

5.1 Data description

This analysis is based on retail data, of a physical store. The dataset consists of 4000 unique products, 4500 unique customers and 550000 transactions. All data was collected with user consent and in agreement with relevant privacy policies. Moreover, the dataset is fully anonymized and no personal information was collected in connection with this research.

The training data consists of users that are denoted by $\boldsymbol{u}$ and items (products), that are denoted by $\boldsymbol{i}$. The $\boldsymbol{r_{ui}}$ matrix, that captures the interaction between the user $\boldsymbol{u}$ and the item $\boldsymbol{i}$, represents how many times a user interacted (in this specific case bought) with an item, times the catalog price of that item:

$$\boldsymbol{r_{ui} = quantity * unit\ price}$$

Thus, the $\boldsymbol{r_{ui}}$ is represented by a real value. A zero $\boldsymbol{r_{ui}}$ value indicates no interaction between a specific user and a specific item.

Similarly, a test dataset is constructed. In this dataset, though, the values of $\boldsymbol{r_{ui}}$ are either one or zero, capturing what is known as the ground truth; a user has either interacted with an item or not.

In the training dataset, a small proportion of interactions has been masked. That means that the real value has been replaced by zero. This is the prominent way to test the algorithms performance in the end.

5.2 Evaluation

5.2.1 Receiver Operating Characteristic

To evaluate the algorithm, we use the Receiver Operating Characteristic Curve (ROC Curve) and the Area Under its Curve (AUC). This is a commonly used way to visualize and evaluate the

performance of a binary classifier, i.e. a classifier that has two possible output classes. In our study, those classes are denoting the interaction-or lack thereof, between a user and an item.

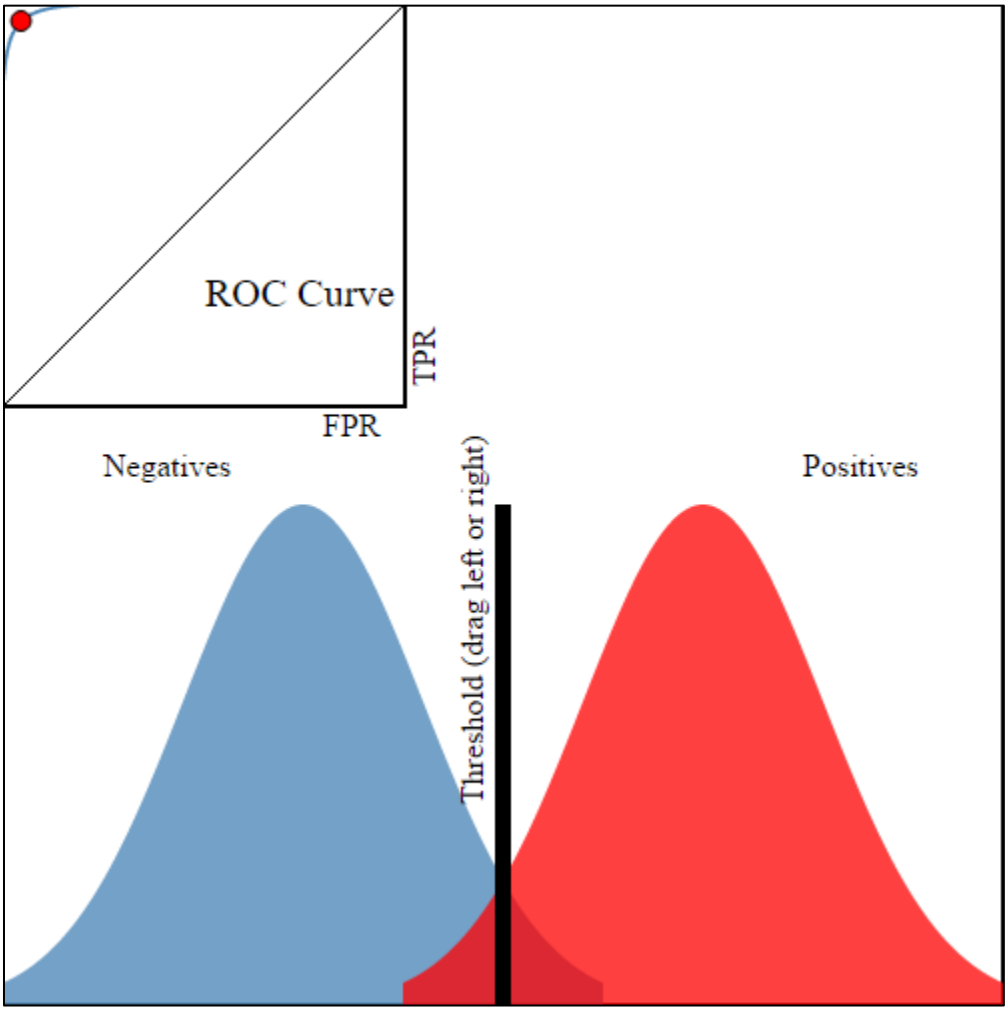


Figure 5-1 - ROC Curve

Consider the example of Figure 5-1. The red curve, denotes a positive outcome, i.e. the user interacted with an item, while the blue one denotes the opposite. To evaluate the algorithm, we want to judge how well out model is doing, by comparing the model's predictions to the true user-item interaction statuses.

The blue and red regions are visualizing the corresponding probability distributions of the predicted values. Thus, the x-axis of Figure 5-1 represents the predicted probabilities, while the y-axis represents a count of observations, much like a histogram. Setting the threshold at 0.5, we classify everything that has a probability greater or equal to 0.5 to have a positive interaction and everything below 0.5 the opposite.

With that threshold, and assuming that red and blue regions accurately depicting the predictions of our classifier, the accuracy of the model would be above 90%. Figure 5-2 depicts a classifier that has a significantly lower performance, no matter where we would set the threshold.

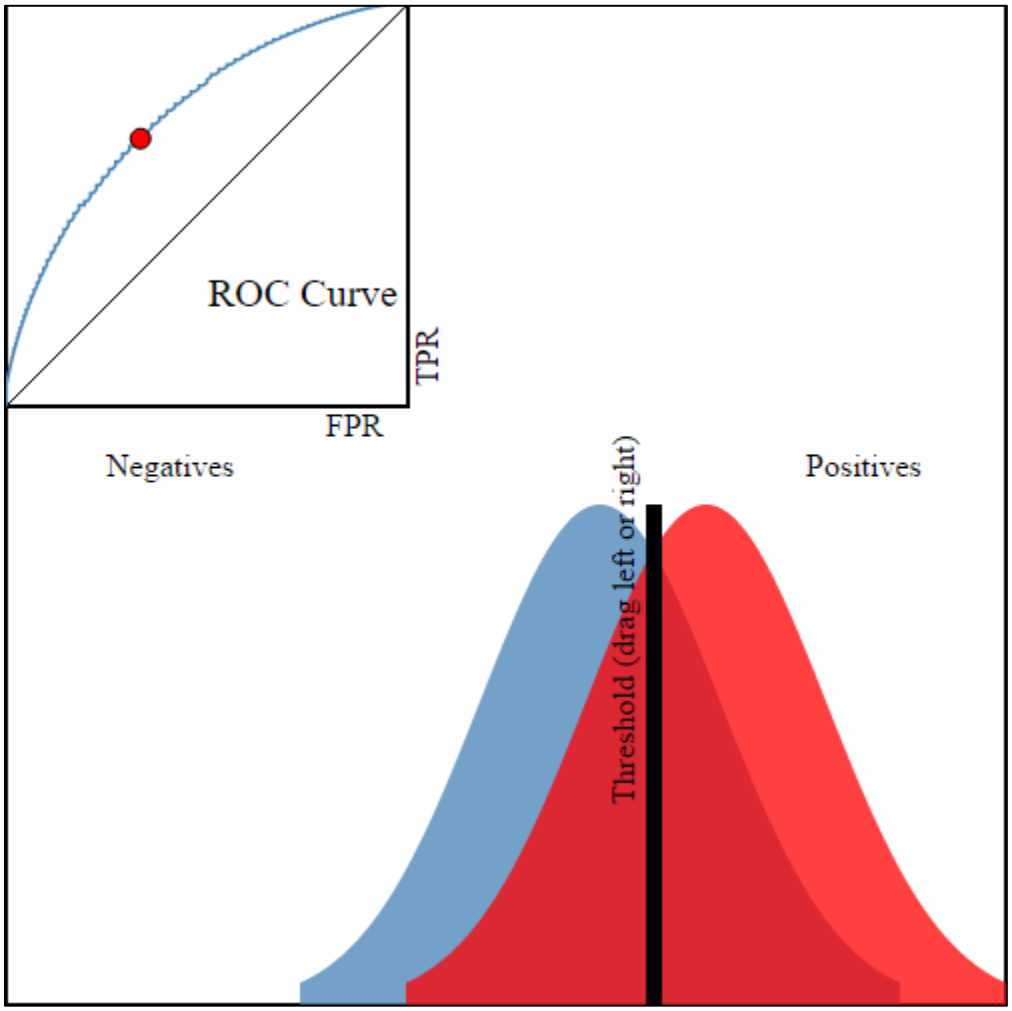


Figure 5-2 - Low-performance classifier

Figure 5-1 and Figure 5-2 also depict the ROC curve on the upper left corner. This is a plot of the True Positive Rate (TPR), on the y-axis, versus the False Positive Rate (FPR), on the x-axis, for every possible classification threshold. As a reminder, the TPR and FPR are defined as:

$$TPR = \frac{True\ Positives}{All\ Positives}$$

$$FPR = \frac{False\ Positives}{All\ Negatives}$$

Both TPR and FPR rate from zero to one.

Finally, the area under the ROC curve, of a classifier that performs exceptionally well, is close to one. On the other hand, when we have an AUC close to 0.5, the classifier does no better job than random guessing.

5.2.2 Popularity metric

As a baseline, we constructed a recommender system based on popularity. In other words, we take the most popular items in the store, we exclude the ones that a customer might have already bought, sort them and finally recommend those items. Beating popularity is difficult task for any algorithm that tries to make personalized recommendations. The test set remains the same as before, as well as the evaluation method, described in the previous subchapter.

5.3 Proposed approach

The approach that was followed during this analysis is described on chapter four. There are some slight variations concerning the confidence matrix $\boldsymbol{c_{ui}}$.

In chapter four we thought of the confidence matrix as:

$$\boldsymbol{c_{ui} = 1 + \alpha r_{ui}}$$

Due to high variance on the metric we used in place of implicit rating (quantity * unit price), the confidence matrix took the shape of:

$$\boldsymbol{c_{ui} = 1 + \log(1 + \frac{r_{ui}}{\varepsilon})}$$

For this experiment, we found that the values of $\boldsymbol{\alpha = 15}$ and $\boldsymbol{\varepsilon = 10^{-2}}$ work best**.**

5.4 Results

We compared the model to popularity metric, as we mentioned before, and evaluated it using the Receiver Operating Characteristic Curve (ROC Curve).

Figure 5-3 depicts the score of the algorithm (ALS) versus the recommender system based on popularity:

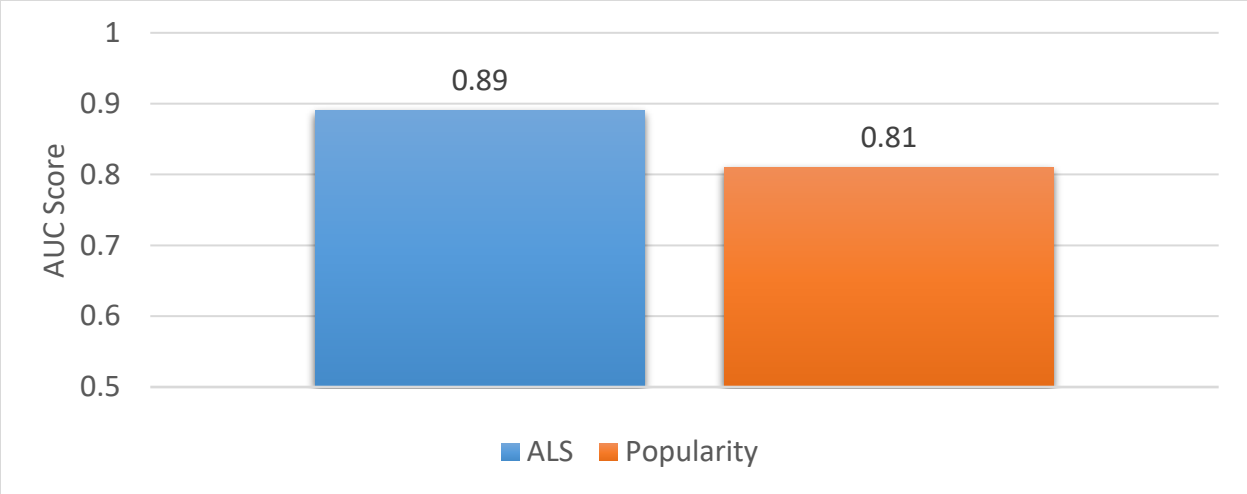


Figure 5-3 - Model Vs Popularity Score

We can see that our model (Alternating Least Squares – ALS) performs better than the popularity recommender system. Moreover, our model offers personalized recommendations instead of recommending just the most popular items.

Furthermore, using an algorithm like ALS we can uncover user preferences and item features, in a vector format. These discoveries can be used in other scenarios, like reverse top-k queries [31].

Chapter Six

6 Summary and Future Research

In this work, it was presented a study of collaborative filtering algorithm using implicit feedback datasets. We studied the broad subject of machine learning and laid the foundations of learning theory, and learning feasibility on chapter two.

Next, we distinguished a subset of machine learning, that includes recommender systems. We presented the notable previous work done on the subject and separated the case of implicit vs explicit feedback datasets.

We, then, proceed to propose a model for recommendations, using collaborative filtering for implicit feedback datasets, and, in particular, we studied the case of retail.

Finally, a corresponding experiment, its results and evaluation were presented on chapter five of this dissertation.

6.1 Summary

When talking about user feedback, it is necessary to distinguish the feedback that we take directly from the users, to the one that we ourselves extract from the users' behavior. Thus, we have the explicit feedback datasets and the implicit feedback datasets.

The first kind of feedback, explicit feedback, indicates preference. The second one indicates confidence. In other words, when a user directly shows an interest or dislike towards an item, then we get explicit feedback, which contains both positive and negative feedback. On the other hand, implicit feedback derives from our observations on users' behavior. This sets us with a confidence level about whether a user likes an item. Implicit feedback does not directly contain negative feedback.

In our model, we applied a latent factor algorithm, and expand it to use an implicit feedback dataset, considering the confidence notion. The algorithm should take every user-item interaction as input, including the situation where no user-item interaction has been recorder. In this case, the cell in the user-item interaction matrix takes the value zero. This is crucial for our

model because if we skip those values, our model will be heavily biased towards a positive preference, ignoring any negative feedback that is most likely to be discovered wherever no interaction has been occurred. In such a case, the algorithm would not represent correctly the user profile.

6.2 Future research

Next steps include ways to improve the accuracy and performance of the algorithm. When we are talking about accuracy, there are a few elements of the algorithm that will be revisited. The confidence matrix function is one of them. Another part of that process is the evaluation and analysis of the zero values in the user-item interaction matrix. Since those values form the majority of that matrix, we need to analyze and reevaluate their properties. One simple example is how to treat those zeros when an item is out of stock.

A major problem of collaborative filtering algorithms is that of high sparsity. When the user-item interaction matrix consists mainly of zeros, then the sparsity is high and the algorithms yields poor results. Thus, the next steps will try to alleviate this problem, using techniques such as associative retrieval techniques.

On the side of the performance, this algorithm will be transformed in order to be able to work in parallel and scale linearly. Moreover, there will be a study to make this algorithm work iteratively, and be able to learn using only some delta differences that would occur, instead of going through the whole dataset from the beginning.

References

[1] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development,* vol. 3, pp. 211 - 229, 1959.

[2] F. Ricci, L. Rokach and B. Shapira, Introduction to Recommender Systems Handbook, US: Springer, 2011.

[3] C. Anderson, The Long Tail: Why the Future of Business Is Selling Less of More, US: Hyperion, 2006.

[4] B. Schwartz, The Paradox of Choice, US: Harper Perennial, 2005.

[5] J. M. O'Brien, "The race to create a 'smart' Google," *Fortune,* November 2006.

[6] R. M. Bell and Y. Koren, "Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights," in *IEEE International Conference on Data Mining (ICDM'07)*, 2007.

[7] B. Sarwar, G. Karypis, J. Konstan and J. Riedl, "Item-based Collaborative Filtering Algorithms," in *Proc. 10th International Conference on the World Wide Web*, 2001.

[8] G. Takacs, I. Pilaszy, B. Nemeth and D. Tikk, "Major Components of the Gravity Recommendation System," *SIGKDD Explorations,* vol. 9, no. 1, pp. 80-84, 2007.

[9] M. Morita and Y. Shinoda, "Information filtering based on user behavior analysis and best match text retrieval," in *Proc. 17th ACM SIGIR Conference on research and development in information retrieval*, 1994.

[10] L. Terveen, W. Hill, B. Amento, D. McDonald and J. Creter, "PHOAKS: A system for sharing recommendations," *Communications of the ACM,* vol. 40, pp. 59-62, 1997.

[11] H. Yifan, K. Yehuda and V. Chris, "Collaborative Filtering for Implicit Feedback Datasets," in *Proc. IEEE Int'l Conf. Data Mining (ICMD'08)*, 2008.

[12] C. M. Bishop, Pattern recognition and machine learning, US: Springer, 2006.

[13] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, MIT Press, 1998.

[14] Y. S. Abu-Mostafa, M. Magdon-Ismail and H.-T. Lin, Learning from data, AML, 2012.

[15] D. Goldberg, D. Nichols, B. M. Oki and T. Douglas, "Using collaborative filtering to weave an information Tapestry," *Communications of the ACM,* vol. 35, pp. 61-70, 1992.

[16] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information processing & management,* vol. 24, pp. 513-523, 1988.

[17] P. Maes, "Agents that reduce work and information overload," *Communications of the ACM,* vol. 37, pp. 30-40, 1994.

[18] J. A. Wise, J. J. Thomas, K. Pennock, D. Lantrip, M. Pottier, A. Schur and V. Crow, "Visualizing the non-visual: Spatial analysis and interaction with information from text documents," *IEEE Computer Soc. Press,* vol. 30, pp. 51-58, 1995.

[19] J. L. Herlocker, J. A. Konstan, A. Borchers and J. Riedl, "An algorithmic framework for performing collaborative filtering," in *Proc 22nd ACM SIGIR Conference on Information Retrieval*, 1999.

[20] W. Hill, L. Stead, M. Rosenstein and G. Furnas, "Recommending and evaluating choices in a virtual community of use," in *Proc. ACM'95 conference on human factors in computer systems*, 1995.

[21] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom and J. Riedl, "An open architecture for collaborative filtering on netnews," in *Proc ACM CSCW'94 conference on Computer supported cooperative work*, 1994.

[22] U. Shardanand and P. Maes, "Algorithms for automating "word of mouth"," in *Proc. ACM CHI'95 conference on human factors in computer systems*, 1995.

[23] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon and J. Riedl, "GroupLens: Applying collaborative filtering to Usenet news," *Communications of the ACM,* vol. 40, pp. 77-87, 1997.

[24] J. S. Breese, D. Heckerman and K. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *Proc. 14th conference on uncertainty in artificial intelligence*, San Francisco, 1998.

[25] G. Linden, B. Smith and J. York, "Amazon.com recommendations: Item to itemcollaborative filtering," *IEEE Internet computing,* no. 7, pp. 76-80, 2003.

[26] G. D. Linden, J. A. Jacobi and E. A. Benson, "Collaborative recommendations using item-to-item similarity mappings". US Patent US6266649 B1, 24 July 2001.

[27] T. Hofmann, "Latent semantic models for collaborative filtering," *ACM transactions of information systems,* vol. 22, pp. 89-115, 2004.

[28] R. Salakhutdinov, A. Mnih and G. Hinton, "Restricted Boltzmann Machines for collaborative filtering," in *Proc. 24th Annual International Conference on Machine Learning*, 2007.

[29] D. Blei, A. Ng and M. I. Jordan, "Latent Dirichlet Allocation," *Journal of Machine Learning Research,* vol. 3, pp. 993-1022, 2003.

[30] J. Bennet and S. Lanning, "The Netflix Prize," in *KDD Cup and Workshop*, 2007.

[31] A. Vlachou, C. Doulkeridis, Y. Kotidis and K. Norvag, "Reverse Top-k Queries," in *Data Engineering (ICDE), 2010 IEEE 26th Internation Conference on Data Engineering*, 2010.

[32] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer and R. Harshman, "Indexing by latent symantic analysis," *Journal of the American society for information science,* vol. 41, p. 1990, 391-407.