POLITECNICO DI TORINO

Facoltà di Ingegneria Informatica
Corso di Laurea in Ingegneria

Tesi di Laurea

Recommender system for large-scale assortment

Relatori:
prof. Paolo Garza
Davide Iori

Candidato:
Matteo Glarey

Luglio 2019

Table of contents

Chapter 1

Introduction

1.1 Overview

In the last years, Artificial Intelligence is taking a wide part of every aspect of our life, from robotics to finance, going through healthcare, security, speech recognition, smart cities and advises on purchases. Nowadays, a lot of objects are smart, connected too each other, capable to understand human needs and acting like them. For example, in domotics, a sensor can recognise that someone is in the house and turn on the heating or a fridge which can recognise that some product is finished. This is the world of the Internet of Things (IoT). Machine Learning is part of it and its goal is to provide to computers the ability to learn without having behind an explicit program.
As Tom M. Mitchell says: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E." In other words, the program learns how to behave in a particular environment, given by the tasks it can perform. The goodness of the performance is evaluated by a metrics, the performance measure, applied on a cost function that defines the task. Once defined these relationships, the goal of the program is to minimize the difference between the predicted value and the expected one that is the cost. This is done through the cost function. In this way, machines can perform more complex tasks that otherwise are feasible only for human brain. Moreover, it helps to analyse big quantity of data, helping specialists in their job with large data in less time.
In this paper, it will be explained how a recommender system works starting from the theory of recommender systems, going through the analysis of some specific architectures and testing them to understand how performs with different datasets, in particular a sample of Kramp data and an adapted version of the MovieLens dataset.

1.2 Outline

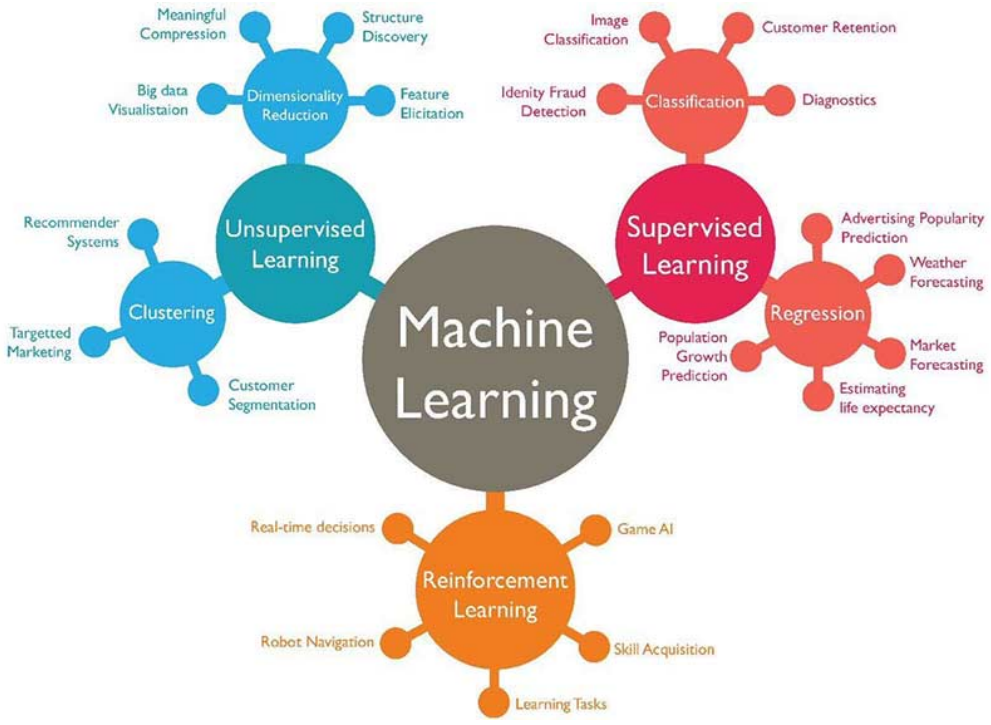
The following chapters are organized as noted below.

- Chapter 2, Theory Part. It introduces the subject with some key points useful to understand the subject and draw up the use case of the project. It explains the basic concept of Machine Learning and how recommender systems work.
- Chapter 3, Use cases. This chapter shows, after a small description of Kramp, the available options for applying machine learning in the company.
- Chapter 4, Recommender system architecture and implementation. Here it will be shortly explained the infrastructure that build up the system. Implementation. It is about explaining involved algorithms and tools.
- Chapter 5, Experimental evaluation. In this chapter, the results of the different algorithms implemented will be compared.Moreover, after a short overview on the MovieLens dataset, will compare the algorithm over this dataset with previous results with Kramp dataset.
- Chapter 6, Conclusion. Here it will be illustrated final considerations on results and future improvements of the system.

Chapter 2

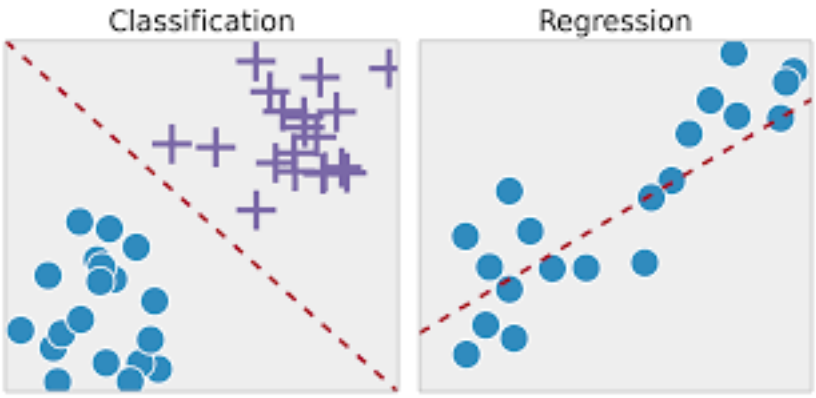
Theory Part

The design of a Machine Learning algorithm depends on the final target. For this reason many techniques can be applied.

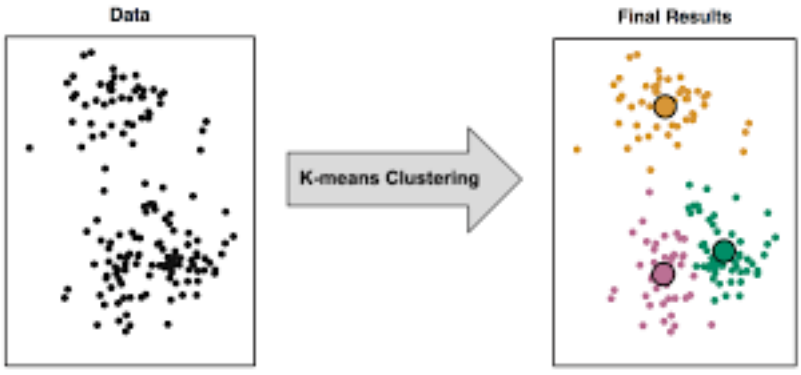


First of all, supervised learning is usually for classification, when the task is to map input to output labels, or regression, when the task is to map to a continuous output. The goal is to find a model that represents the distribution of data and produces correct output. During the training phase the predicted output is compared to the expected one to minimize the error. This is like having an oracle that knows the

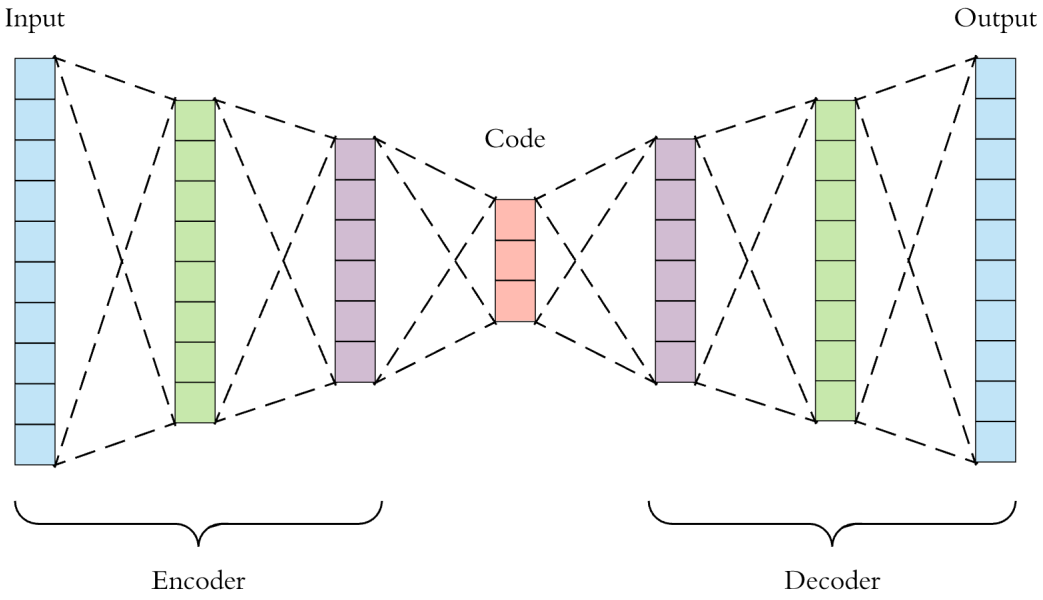
truth and teaches the machine to predict right values until the accuracy level is acceptable.



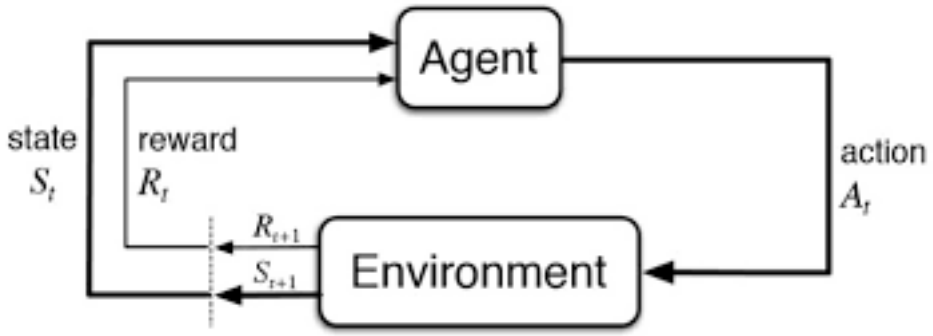
Unsupervised learning allows to find the intrinsic structure of data without using labels. The most common tasks are clustering and dimensionality reduction. It is useful also to have a first data insight that are processed in the following steps.
Clustering is a technique to group data points and to classify them in a specific class. In this way, points with similar properties are grouped and similarity can be calculated through euclidean distance or other measures.



Another technique is autoencoder for dimensionality reduction. In this way, input and output are represented by a lower number of features than the original data, saving space in memory.



Reinforcement learning allows to an agent to learn by experiencing an interactive environment. This agent has the ability to do a set of actions, each one moves the agent from one state to another one with a cost or reward. The goal is to maximize the long term reward. For example, the robot chooses actions to move from one state to another one getting a reward from the environment through a reward function. Trying to get the maximum at each time but it is not a good thing (greedy approach), possibility to get a good solution but not the best one.



The task is to learn a policy that maximizes the expected long term discounted reward by mapping states to actions, so there is the need to define a value function for each state.

Another machine learning technique is deep learning that, with artificial neural networks, emulates the structure and the function of the brain. The basic unit of a neural network is the neuron that contains the information. Like an human

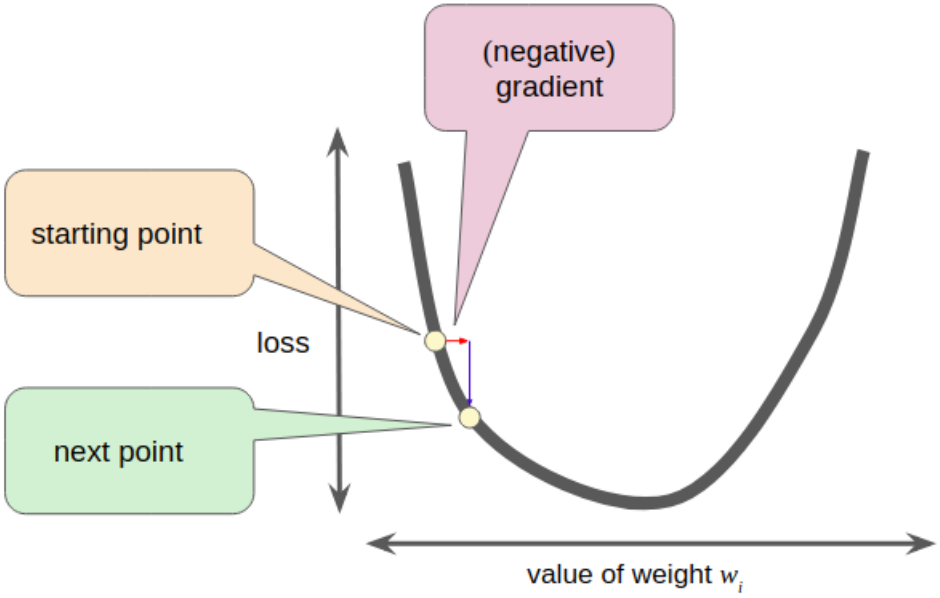
neuron, it receives an input and produces the correspondent output if it is sufficiently "stimulated". The decision is made by an activation function that lets the input go through if it is above a threshold. For example, one of the simplest activation functions is Heaviside step function. It returns 0 if the input is less than zero and 1 if it is equal or more than zero.

$$f(h) = \begin{cases} 0 & \text{if } h < 0, \\ 1 & \text{if } h \geq 0 \end{cases}$$

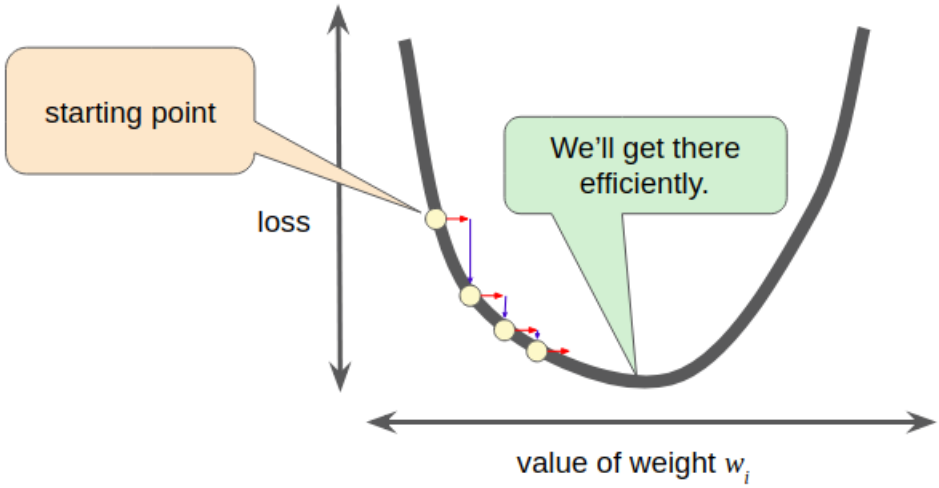
The input is multiplied by a weight to establish the importance for the neuron. These weights are initialized at random values at the beginning and they are modified or trained during the training phase of the neural network. In some cases, data can contain biases, which cause estimation to be different from the expected value. For this reason, a bias value is added to the activation function to shift up or down its curve. A single neuron, also called perceptron, is a binary classifier and it is described by:

$$y = \sum w_i x_i + b$$

where y is the output, w the weights, x the input and b the bias. This equation defines a linear equation that split the plane in two parts or two classes. More perceptrons make a layer and produce a multiclass classifier. Each neuron is connected to each other within the layer and to the next one. Usually, neural networks are composed at least by three layers: input layer where data go in, hidden layer where data are featured, output layer where the result come from. In particular, the hidden layer can include multiple layers to increase the accuracy.
The upgrade is usually done by means of **gradient descent**. Gradient descent is applied on a **loss function**. The loss function represents how costly is the actual configuration in terms of distance between the prediction and the true value. If the problem is convex, there is only a minimum for the loss function. Assuming that, with a random initialization, the starting point is on the curve in a random location. Calculating the derivative in that point, gives the slope of the curve. This is the gradient and tells the good direction to reach the minimum. In the case of multiple weights, the gradient is a vector with a direction and a magnitude. It always points to the greatest increase in the loss function. For this reason, for this algorithm is used the negative value of the gradient. After some step, it could reach the minimum.



To define the width of the step the gradient descent is multiplied by a scalar value known as **learning rate**. If it is too small, the learning experience could take ages before reaching the minimum and, in case of a non-convex function can get stuck into a local minimum giving a sub optimal solution. On the other hand, taking a too large value, the following step will continuously overtake the minimum and also make the algorithm diverging. The tuning of this parameter is very important for the success of the result.



2.1 Recommender Systems

A recommendation engine problem is to develop a mathematical model or objective function which can predict how much a user will like an item. The objective function measures the usefulness of the item for a user. The goal is to give interesting and personalized suggestions to the user, by using transaction details, product features, feedbacks to find correlation among them.
To do this, big data is really important because, otherwise, the recommender system cannot perform. For this reason, companies started to retrieve data from several sources like browser history, historical data and ratings to analyse them and provide some services.
In the next paragraphs some popular types of recommender systems will be described.

2.1.1 Collaborative filtering

This kind of recommender is one of the simplest to implement and it can be very accurate. It is part of the first generation of recommendation engines known also as neighborhood method recommenders. The basic idea is that if two users had similar tastes they will likely buy same items in the future.
There are two types of collaborative filtering:

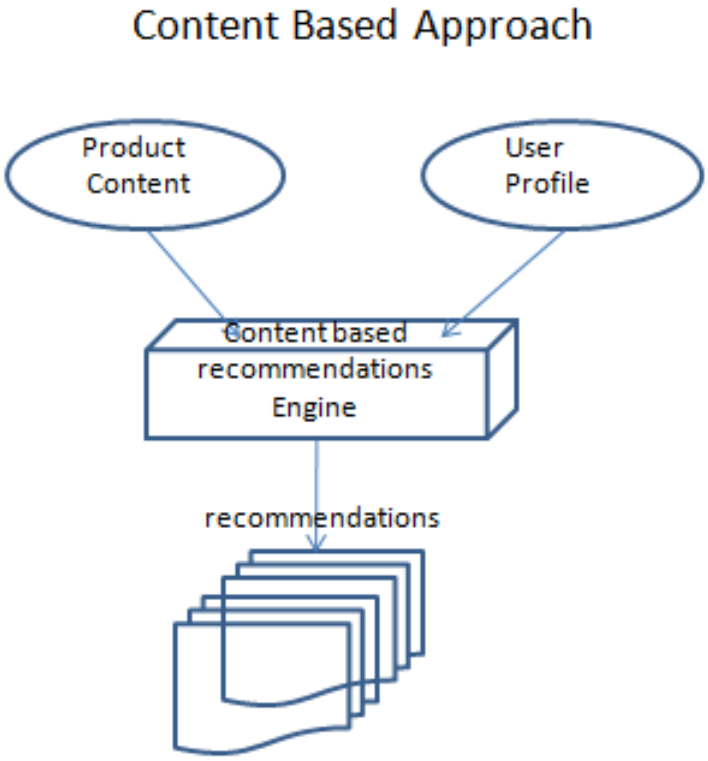
- **User-based**: recommendations are done through user's neighborhood. The neighborhood is found with some distance measure like cosine similarity or RMSE (Root Mean Square Error) and taking the closest users with respect to the target one. Previous user's history classifies the user and it can be seen as the result of its actions, giving it a position in the user space.

- **Item-based**: recommendations are done through the neighborhood of the items. In this case, the neighborhood is composed by similar items. Similarity can be calculated with the same metrics as before but this time it is measured on product features instead of user's one. The product features are the result of previous item preferences expressed by all the users.

In both cases, the system will give suggestions that are still unknown to the user.
The issue with collaborative filtering is the presence of new users because there are no informations on them in the system. This is known as *cold start* problem. This tecnique is easy to implement and does not need product information or user preferences but it is computationally expensive and needs a lot of data.

2.1.2 Content-based

In this case, the recommendation is based on user preferences and on item characteristics. For this reason the system needs to have a profile for both of them. In this way, the recommender system will advise the user of products that are close to previous preferences.
As the name says, this technique does not take into account user neighborhood to give suggestions, it simply consider past user's choices related to properties of the items. For this reason, a recommendation done with this type of engine is personalized. Moreover, it solves the cold start problem when a new user comes into the system, suggesting items similar to his tastes.



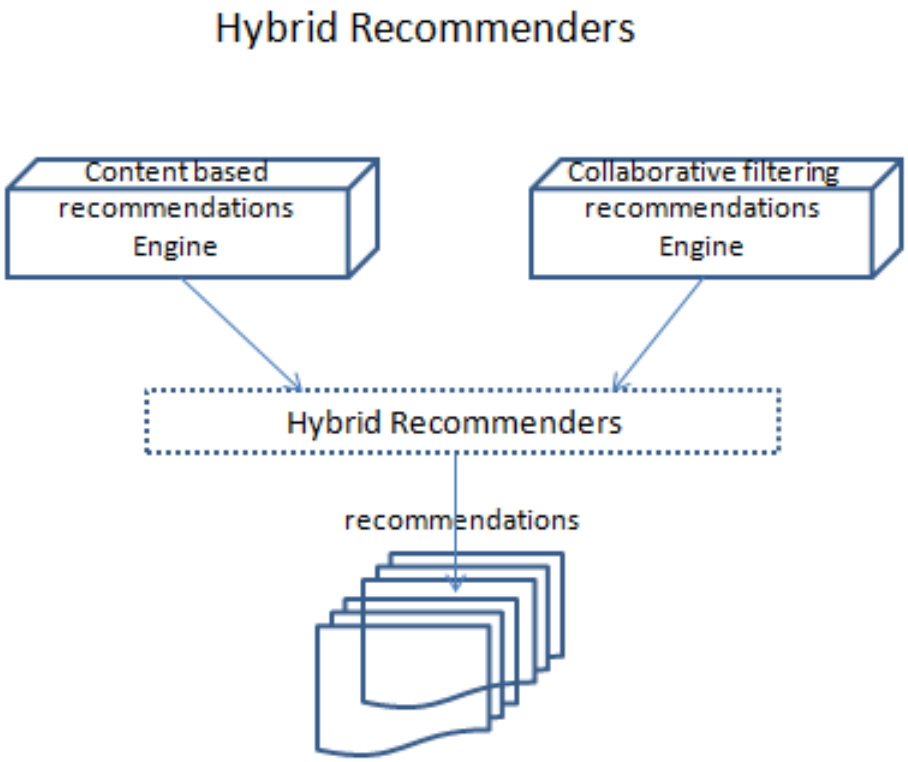
As is shown in the figure, this system needs to build item and user profiles. In these steps the content of profiles is mapped to a vector space which components are them features and compute similarity between them. Features are relevant information about the product or the user like the category of the object, the genre of a film or the user's history.
Then, with these profiles, the system is able to establish a probability with which an item is liked by the user.

2.1.3 Context-aware

Traditional recommender systems cannot give suggestions considering the context. A context could change the evaluation that a user gives to an item. For example, the same person in diiferent locations or situations may have different requests. This is the influence of the context, the current state of the user. It is defined by, as mentioned before, the location, the mood, the day, month and season and many others. Unlikely a user wants an icecream during a cold winter.
For this reason, the system should capture the context to refine the recommendation. The context filter can be applied to the user and product features or to the recommendations. In the first case we talk about **pre-filtering** approach, in the other case of **post-filtering** approach.

2.1.4 Hybrid systems

The fundamental concept behind this method is to combine several recommender systems to exploit the strength of each one to build a robust system while mitigating the weaknesses. For example, with the previous explained methods, content-based model can catch up new items/users issue on collaborative filtering model.



The recommendations coming from different engines can be combined in different way like a (linear) combination to give a final result.

2.1.5 Model-based

The previous approaches use calculated features to recommend items but they are very slow to respond in a real-time environment. For this reason, other algorithms such as matrix factorization and single value decomposition are applied. In this way recommendations are done through the learned weights and they are ranked before the final result.

2.1.6 Principal algorithms

- **Matrix factorization**: it will be explained in Chapter 5.6 Algoritms
- **Alternating least squares**: starting from the error minimization error

$$\min_{p*,q*} \sum_{(u,i)\in K} (r_{ui} - q_i^T p_u)^2 + \lambda(||q_i||^2 + ||p_u||^2)$$

  with r_{ui} the true label for the rating of the item i from user u, q_i the product feature vector, p_u the user feature vector and λ the regularization term. This equation has two unknowns and that introduce a non-convex problem. This method fix one of the two variables and this simplifies the problem to be quadratic and it can be solved optimally. For example, to compute user feature vector, the product feature vector to optimize for least squares and viceversa.

- **Singular value decomposition**: This method decomposes a matrix A of size m x n in

$$A = U * \sum * V^T$$

  where U is a (m x r) matrix representing user latent features, V is a (n x r) matrix representing item latent features, $\sum$ is a (r x r) diagonal matrix with all the singular values ranked for them value in decreasing order and r is the rank of the matrix.

- **Linear regression**: This technique is used for solving prediction problem and to predict future outcomes. Having historical data it can find relationships between input data. The result is a straight line following the equation

$$Y = \beta_0 + \beta_1 X + e$$

  where β_0 and β_1 are the unknowns, x the independent variables and e represents the error.

2.2 Application fields

Machine learning has a lot of application fields. It can be applied to any business, personal or useful use case. Some examples are automotive with driverless cars, customizing offers following the customer behavior, diagnosing diseases collecting medical data, detecting fraud through anomaly detection, computer vision to analyse images to recognize patterns for several applications.

2.3 Challenges

There are several challenges that a machine learning algorithm has to deal with. In fact, not every company is outfitted to manage the increasing amount of available data and some is starting to move toward a data culture to support investments and growth. Along with the lack of data, an important obstacle to good results is the quality of achievable data. As a matter of fact, data quality is crucial to get accurate results. If data are fine, the next step is to collect and clean data coming from different sources, remove noise and transform them into a suitable format for machine learning algorithm. Sometimes, data have too many features that make difficult an efficient modeling. For this reason, during the preprocessing phase, features are extracted, selected and combined to give more expressiveness to the model. However, as regards the infrastructural point of view, the challenge is to store a huge quantity and variety of data and could involve several storage platforms. Moreover, computational power has its own importance. Preprocessing and modeling phase must be refined by an human operator, so having a good computing infrastructure allows to find the best possible solution in a reasonable amount of time. Some improvements are hardware acceleration, such as SSDs and GPUs, and distributed computing to split the task across many computers. Storage and compute resources have to be scalable because the consumption can be high in certain intervals and low in others.

Chapter 3

Use cases

3.1 The company

Kramp is a company that offers spare parts, technical services and business solutions for agricultural, landscaping, forestry, earth moving and OEM (Original Equipment Manufacturer) companies. It guarantees expert advice with 2600 dedicated employees to support customers. Almost all of their products can be delivered within the following day and they have more top brands in the same home to have a wide choice in terms of price/quality. As a company operating in the agricultural sector, they know how important it is to be environmentally responsible. For this reason, since 1988, they use the famous red boxes to carry their products. In this way, due to their rectangular shape, there are no empty spaces in the truck.

During the last years, the company started to invest more in IT to grow on the global market.

3.2 Options

In Kramp there are some possible fields where machine learning can be applied:

- **Logistics**: In this case, the goal is to optimise deliveries efficency by means of organising the warehouse, transports and so on.
- **Search engine**: The objective is to create an engine to improve the research of words.
- **Product recommendation**: The goal is to create a service for the customer, helping him in the exploration of the products and improving the sales.
- **Anomaly detection**: When something goes wrong, the system should be able to find the problem and to react automatically or to request a specialist intervention.

3.3 Evaluating options

To make a choice, I analysed with the team some use case to find the best application for the company. First of all, the evaluation of feasibility of the project considering available data, thinking about how to retrieve them without interfering with other active projects. Have data for logistics was difficult and they would be not good enough, search engine was a still-in-progress project and it was early to have data. For this reason, the most suitable options were product recommendation and anomaly detection. In the case of product recommendation, the idea was to create a recommender system for the webshop to increase the sales taking data from past customer transactions. In this case, new data are available the next day from BigQuery. On the other side, retrieving data for the anomaly detection use case required to set some index in Elasticsearch to monitor events on CPU, network, bus and from all integrated systems. In such circumstance data were real-time and processed in streaming way. The goal was to predict problem conditions to react before they happen to avoid loss of data and time alerting an human operator or automatically react.

3.4 Use case suitable for Kramp

Even if all options were interesting, in the end the most appropriate and interesting option was product recommendation.

3.5 Requirements

The project is about studying how to implement the recommender system to predict the most interesting product for a customer, using data of past transaction. The study starts from data analysis in BigQuery, process them with some machine learning library and deploy the model to have a prediction with new data.

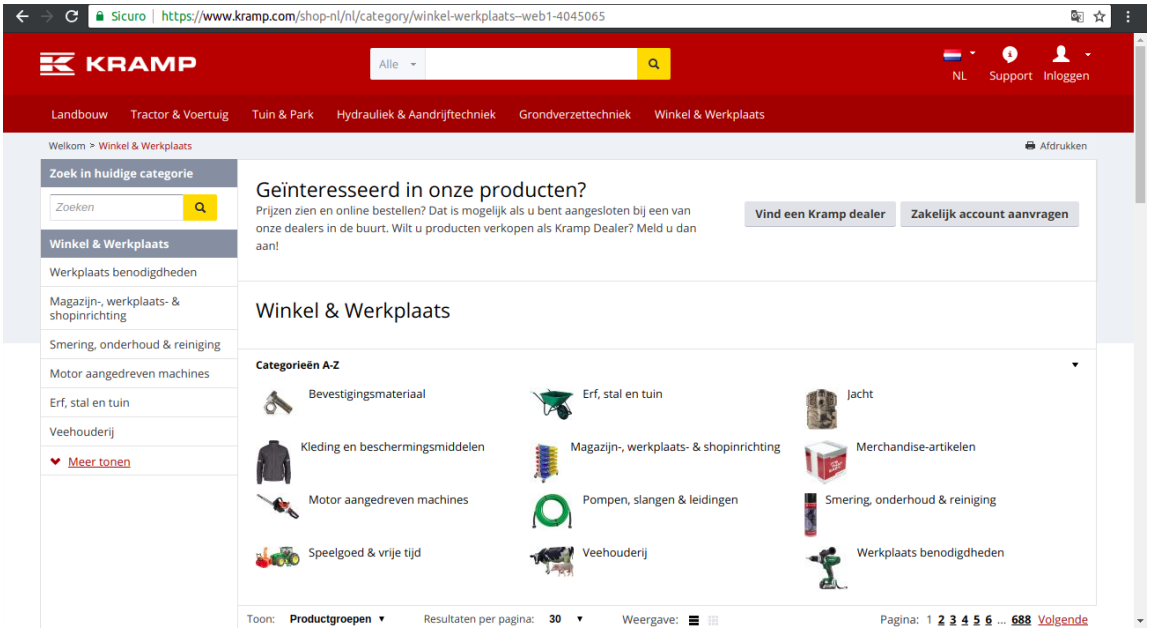
Chapter 4

Recommender system architecture and implementation

4.1 Architecture

4.1.1 System overview

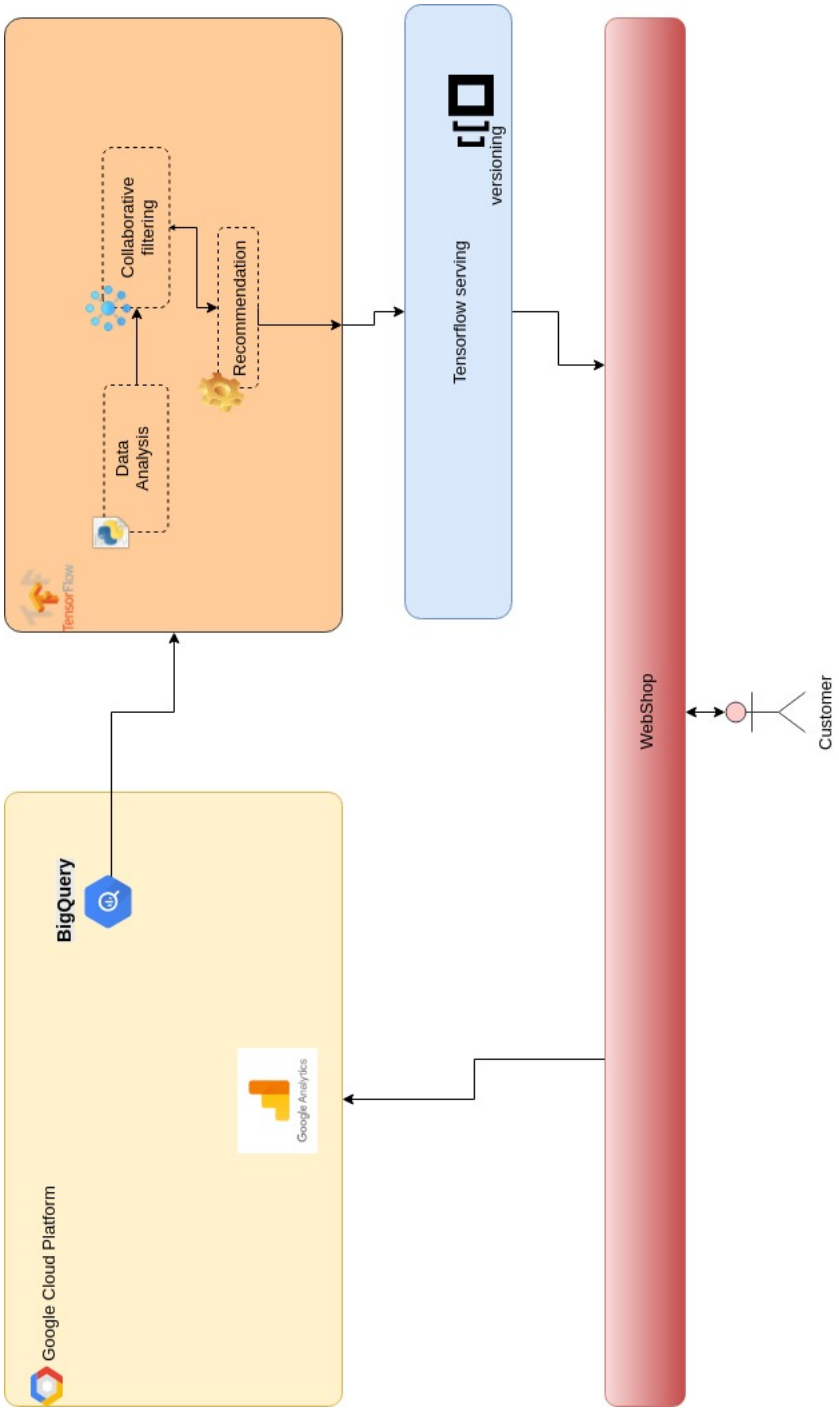
In this specific case, data are taken from the webshop. Each action taken on the webshop is registered with Google Analytics and stored in Bigquery.



From here, all the collected data are available via SQL query. There is a record for each action and, among all, the products sold to customers. Therefore, taking the

account number, the product ID and the transaction ID, we have a basic insight in the user behavior. Then, this information is processed and used as input for the TensorFlow model training. At the end, the model is exported to TensorFlow Serving framework ready to offer recommendations to the webshop.

4.1.2 Layout

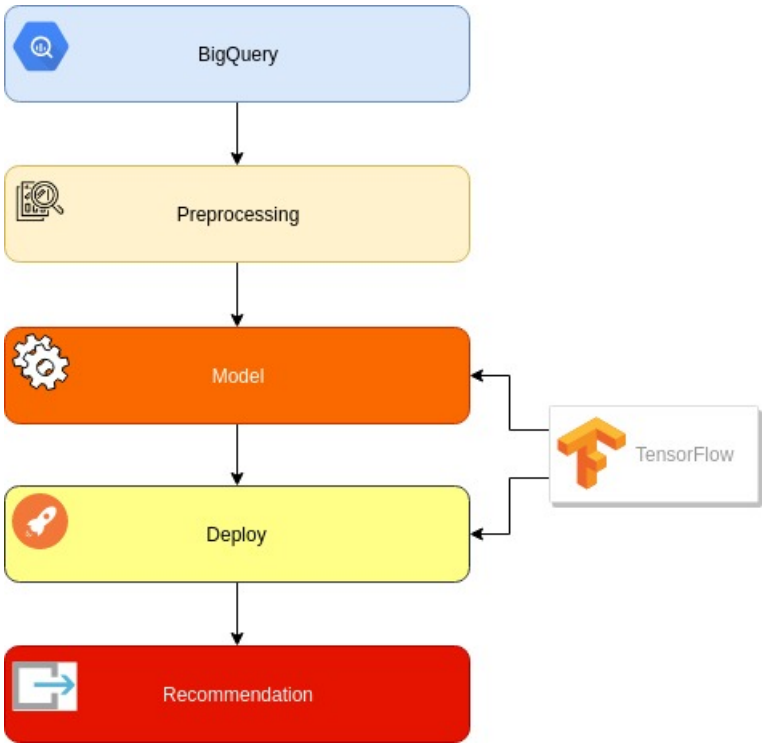

Collaborative filtering
Recommendation
Data Analysis
TensorFlow
Tensorflow serving
versioning
WebShop
Customer
BigQuery
Google Analytics
Google Cloud Platform

4.1.3 Requirements

The system should provide real-time recommendations on the webshop. The response of the recommender has to be comparable with the page load to not slow down the website, giving a bad user experience.

4.2 Implementation

4.2.1 General workflow



The idea is to use the data about transactions from BigQuery and preprocess them to have a useful input for the training phase of the model. That means to inspect them to retrieve useful information via a previous analysis with an expert. Models are almost completely interchangeable, so the input format is very similar and needs only few changes. In this way, the training phase of different models is independent from the architecture of the system. Then, the result is stored and deployed to be recalled on a recommendation request with TensorFlow Serving APIs.

4.2.2 Data sources

Data are taken from the webshop through Google Analytics that stores all the events occurred the webpage in a BigQuery database. In this way, there are information on click streams, pages visualised and item purchased for each user.

4.2.3 Issues

At the beginning, the problem was to extrapolate useful data among all the collected ones. Not all fields of the database were good for this purpose and some of them contained missing data. For example, categorization in BigQuery was specified only for some product.
For this reason, the first step was the simplest one, a collaborative filtering that uses only user and product information on transaction.
Another issue was the memory management, with more data and features the size of some models grows too much and the computation is difficult. On the other size reducing dataset and embeddings size produces a poor result.
Moreover, the amount of training data depends on RAM and CPU computation available and there is no guarantee that they are enough. The same is true for the number of features (embeddings) of the products or users. This causes a limitation in the project because the final result does not depends only on the algorithm but also on the small dataset.
Finally, this is a non-traditional recommendation problem because there are no direct feedback for users like a rating form. The information is about the purchase of a product. Also the quantity sold in a transaction is not valuable as rating. In fact, there is no knowledge about the product and the quantity sold. For example, buying 200 screws and only one wheelbarrow does not mean that the user prefers the screws to the wheelbarrow.

4.2.4 Tools

TensorFlow

TensorFlow is an open source software library for numerical computation. It was developed by Google Brain's researchers and it is a great support for machine learning and deep learning. It has a flexible architecture that allows distributed computing among different hardware like CPUs, GPUs and TPUs (Tensor Processing Units), and deploy on clusters of servers or mobile devices. The computation is based on tensors, a generalization of matrices. TensorFlow uses a dataflow graph to calculate dependencies and operations and then create a TensorFlow session to run the computation with input data.

Dataflow programming is common and effective for parallel and distributed computing. Representing the computation as a graph allows to manage dependencies and to choose hardware on which the computation is done. In this way, the efficiency is easily improved.
Moreover, TensorFlow gives an easy way to deploy the model, TensorFlow Serving. It is designed for production environments. The model is exported at the end of the training and it is available through Google, exposing customize methods to the client.
Finally, with TensorBoard, there is a graphical data insight like PCA (Principal Component Analysis), the computational graph, input values, resources allocation and customized summaries.

Google Cloud Platform

This tool offers several packages for machine learning, data analytics, data storage, security, networking that are available with a subscription. Google Cloud Platform has three main services. First of all, App Engine is a Platform-as-a-Service (PaaS) that allows to distribute code and manages big volume applications. Secondly, Compute Engine is an Infrastrucure-as-a-Service (IaaS) that offers customizable virtual machines with more flexibility with respect to App Engine. Finally, Kubernetes Engine that provides Kubernetes clusters to distribute and manage containers. Moreover, a central aspect of Google Cloud Platform are the analytics. The principal example is BigQuery, a data warehouse callable from several programming languages that provide REST APIs or client libraries. For this project purpose, BigQuery is the only package used to retrieve input batches.

Python

The choose of the programming language was between Java and Python. At the end, Python is more efficient for development because it is a scripting language and it does not need the compilation phase. This accelerates a lot the testing because in the most of cases it is about changing few parameters without the need to recompile the whole project like in Java. The syntax is easy and similar to Java, the keywords for the control flow are almost the same and it allows to focus on the problem instead of the code. Moreover, it is very powerful, in my advice, for data preprocessing also with sklearn, pandas and numpy libraries.

4.3 Machine Learning task

4.3.1 Choosing the training experience

The kind of training experience may be the borderline between the success or the failure of the learner. One decision is to provide a direct or indirect feedback. With indirect feedback there is also the problem of credit assignment, the degree to which action move in sequence determine the success or the failure. Credit assignment could be difficult, because even with initial optimal actions the final result could be spoiled by following poor actions. For this reason, it is easier to learn from direct feedback. The learner must rely on a "teacher" to understand which are the correct action to do or propose an action to the teacher to have a feedback. The last option is to learn all by itself finding minor variations to its previous experience finding the best one. Finally, training examples have to represent the complete distribution of events over which a performance P is evaluated. Sometimes training on one distribution does not guarantee that results will be good also on other distributions. For this reason, training dataset must respresent reality as much as possible.

4.3.2 Choosing the target function

This step is about deciding what type of knowledge will be learned and how it will be used, learning which actions are allowed and choosing the best one. To reach this goal there is need to define a function to do it. The function could be a numerical function that gives a score or a penality. In this way, the original problem becomes an optimization problem. Sometimes it is not possible to give a score to every action so this definition is a non operational definition. So the task of learning is reduced to find an operational description of the ideal target function that is an approximation of the real one.

4.3.3 Choosing a representation for the target function

After having specified the ideal target function, the learning program must have a representation to learn. Ideally, the representation is as close as possible to the ideal target function but, on the other hand, the more expressive the representation is, the more training data it will need. The goal is to find the tradeoff between an as close as possible representation to the ideal function and simplicity in training. It could be a linear function, a polynomial or any other kind of function.

4.3.4 Choosing a function approximation algorithm

Any training example is an ordered pair (b,V_{train}(b)) with b that is a board state and V the target function. It is possible to train by indirect experience and then working on training examples to adjust weights. There is the need to assign intermediate scores, for each intermediate board state. The last thing is to set weights defining the best hypothesis (weights) that minimizes the squared error.

$$E = \sum (V_{train}(b) - \hat{V}(b))^2$$

The minimization is done incrementally at each step by means of adjusting weights through a tecnique called gradient descent. For each training example:

- Use the current weights to calculate $\hat{V}(b)$
- Update each weight w_i as

$$w_i \leftarrow w_i + \eta(V_{train}(b) - \hat{V}(b))x_i$$

η is a small constant that controls the update.

4.4 Algorithms

4.4.1 Matrix Factorization

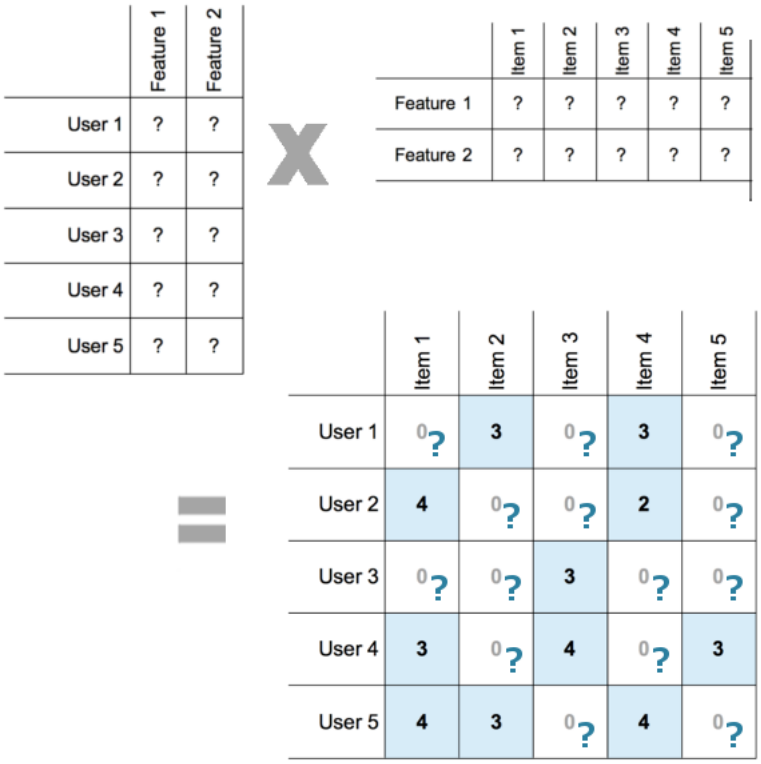
The first implementation is based on collaborative filtering, the transactions are evaluated as ratings: if a customer buy a product means that is interesting for him and likely he could buy similar products. This method allows to reduce dimensionality and to find correlation between products. The goal is to build a vector of features for each product based on transactions in this case. This vector is called embedding and contains hidden features for the product. The algorithm consists in matrix decomposition. The matrix is filled with all data, leaving some unknown cell where the interaction between user and product never occurred. These are the values to predict through the decomposed features. In fact, each occurrence defines a part of the embedding. If we call r the rating and $\hat{r}$ the predicted rating, the cost of a wrong prediction is the distance between $\hat{r}$ and r. $\hat{r}$ is defined as:

$$\hat{r} = q_i p_j$$

where q_i is the item embedding and p_j the user embedding. Since customers have different tastes, the computation has to consider to remove biases from users and from products. For example, the bias represents the preference of some user for a specific category of product. For this reason, the prediction is calculated as:

$$\hat{r} = \mu + b_i + b_j + q_i p_j$$

where μ is the average among all users and products, b_i is the bias of item i and b_j is the bias of user j.



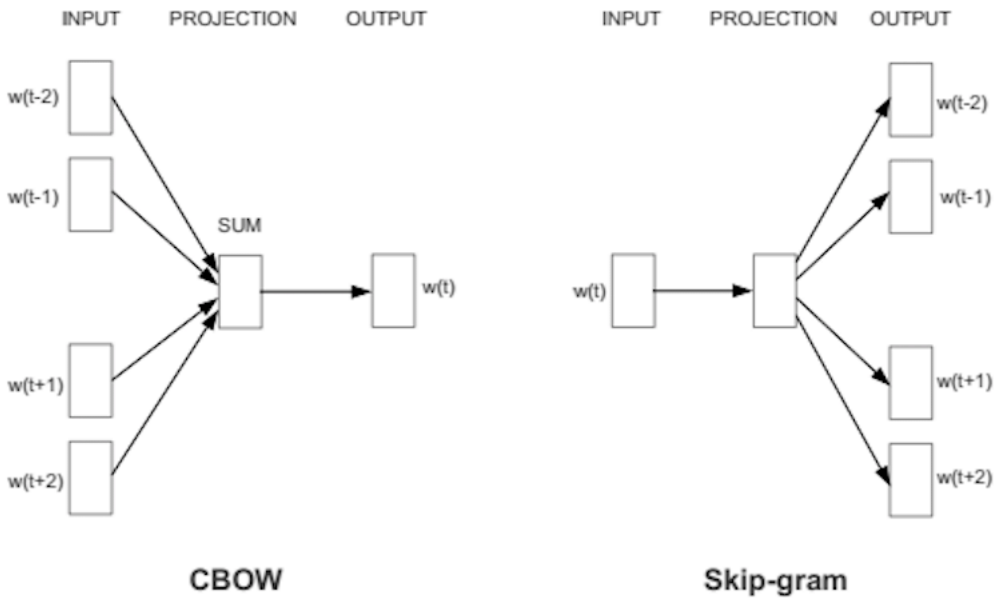
http://katbailey.github.io/post/matrix-factorization-with-tensorflow/

Once the embeddings are calculated, the recommendation can be based on user similarity or on the current item similarity.
In the first case, the user is compared to other users to find the most closer through a distance measures like top-k nearest neighbor or cosine similarity.
In the second case, the distance is calculated among products in the same way.
In this way, the system predicts interesting products in the home page, before the user starts purchasing items. The result is a set of products based on previous user interactions related to other similar user histories.

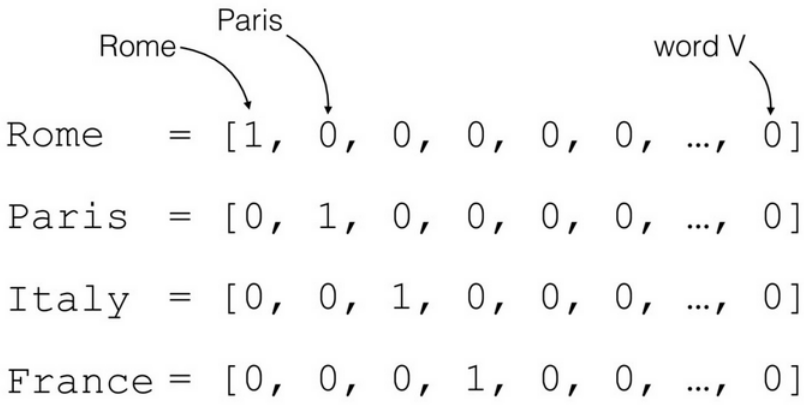
4.4.2 Word embedding

Another idea for a recommender system is the word embedding. Thinking about how the continuous bag of words (CBOW) algorithm works, the goal is to interpret

the transaction as a sentence, retrieving the context from the other products in the transaction. To do that, the system asks to the database a batch of transactions ordered by username, transaction ID and product name. For each batch, the input is separated in transactions and each product has an unique id through an associative dictionary. Then, the system takes a target product in the transaction and build a context for it. In the original algorithm, the context are the words around the target one. In the case of transaction, the order is not important and, for that reason, the transaction is shuffled each time it occurs as training sample. The context is composed by random products in the same transaction. At this point, there are two ways to proceed. The first one is CBOW model which goal is to predict a word from a context while the other one is skip-gram model that learns how to predict the most probable context from a word.

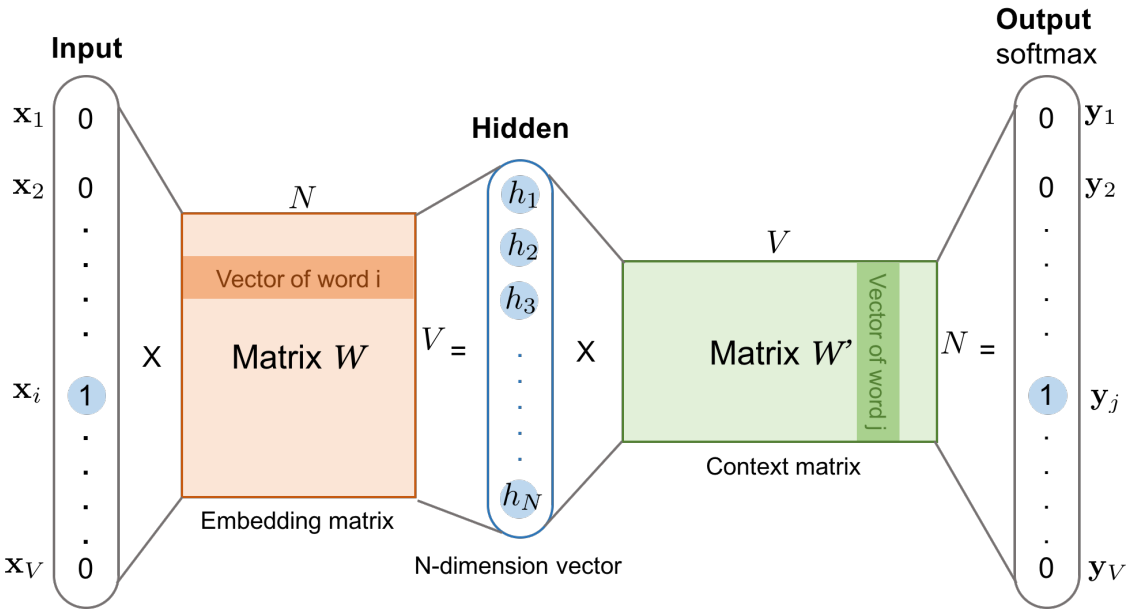


According to the dictionary entry, the input is encoded to one-hot vector that have an entry for each product. If the product is present, the correspondent entry is set to 1, otherwise is 0. For example, the product i takes the i-th position in the one-hot vector.



https://medium.com/@athif.shaffy/one-hot-encoding-of-text-b69124bef0a7

In this way, the product is associated to a position in the vector and is directly linked to its feature weights. The size of embedding depends on a parameter set by the user. A low value for the embeddings does not allow to the system to learn sufficiently about the product and a too high value slow down the computation and does not guarantee a good result.



https://lilianweng.github.io/lil-log/2017/10/15/learning-word-embedding.html

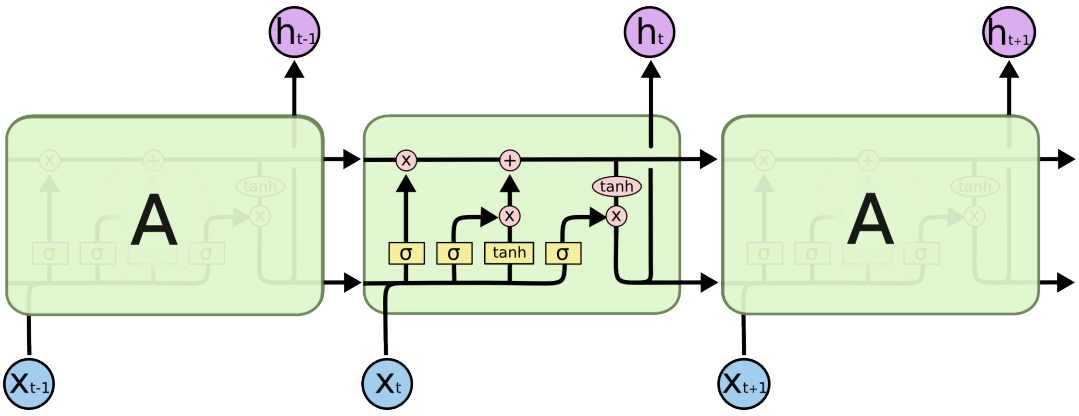
At the end, the features are compared with a top-k or cosine similarity approach like in the previous algorithm.

4.4.3 Neural network

Going on with the same idea of word embedding in mind, the next algorithm is a multi-layer neural network. The maximum depth implemented is 5 layer with different size. Starting from 2048 to 256, using power of 2, there is the possibility to choose any already implemented layer to compose the network. For example, only the network can have only one layer. The important thing is to have right dimensions between layers, in other words, they should respect the matrix multiplication rule. The input can be a single product and the result a set of predicted products, as in the Skip-gram model, or the opposite, as in the CBOW model. In this way, the input should be related to output and predict user's preferences.
This method is related to the recommendation in a product page so, starting from the current product it predicts the most correlated items.

4.4.4 LSTM

A traditional Deep Neural Network (DNN) provides a limited temporal modeling. On the contrary, a LSTM network is able to find correlation in a sequence of data considering the previous time step. LSTM stands for Long Short-Term Memory and it is a special kind of RNN network. Like all RNN, it has a chain structure. In this way, previous information influences the activation of the cell, creating a context for the next input.



The basic unit is called memory block which contains an input and an output gate. The input gate takes care of activations into the cell, the output gate to the rest of the network. The information is replicated along the whole chain with some linear changes but it can also flow unchanged. The first step is to decide if the input information is useful or not by means of the "forget gate layer", a sigmoid that allows the network to forget or keep previous information. The second step is the decision to store new input with another sigmoid, called "input gate layer", and a

tanh to create new candidate values. Then, this changes need to be propagated and to update the old state, the "forget gate layer" is multiplied and the "input gate layer" is added to the old state. At the end, the output that will be based on the cell state. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tanh (to push the values to be between 1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to. In this specific case, the longest transaction gives the length of the sequence. Shorter transactions are sized to the correct dimension by means of padding.

4.5 Analytical model

During this analysis, the four methods will be compared to have an exaustive caracterization of the problem.

Chapter 5

Experimental evaluation

5.1 Testing on Kramp dataset

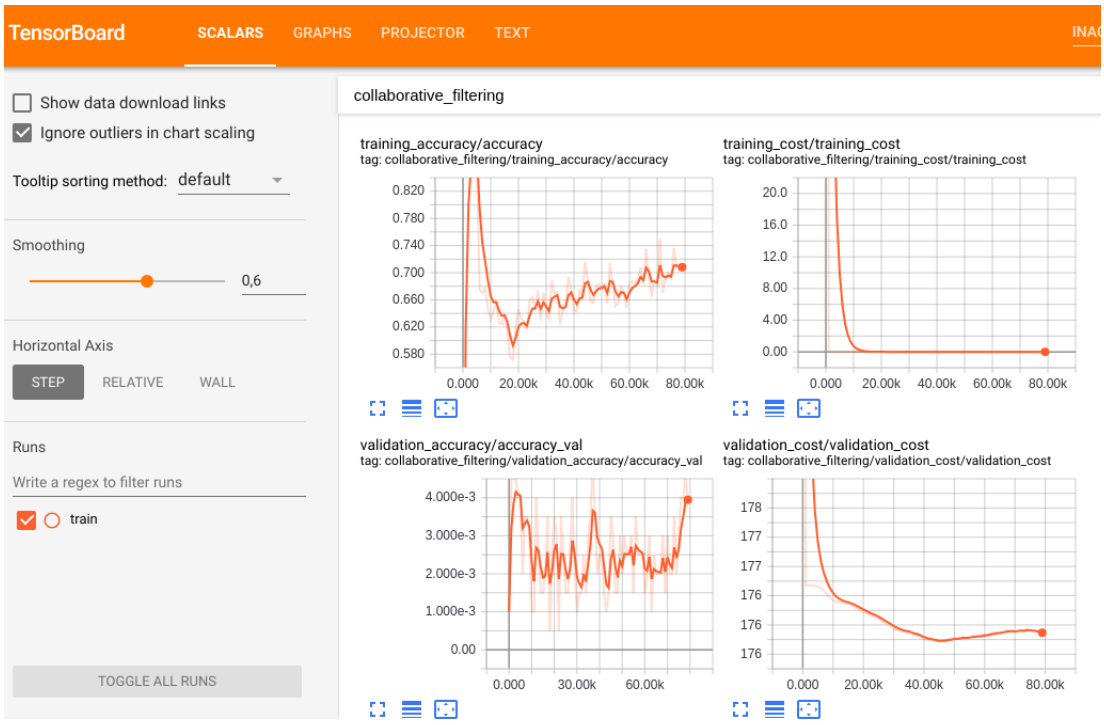
In this chapter, results will be analysed and models will be compared with the help of TensorBoard. It is the graphical framework of TensorFlow to show what happens during the computation. It is possible to see the computational graph of the model, scalar values like accuracy and loss function performances, input values, it offers data analysis tools like PCA to have a better insight of the dataset.
Test are done only on the matrix factorization algorithm and on the neural network autoencoder.

5.1.1 Dataset statistics

The dataset used for the results contains 10000 records of the transactions, each one composed by eight features as the number of the user's account number, the product id, the transaction id, the product category, the total revenue of the transaction for each product, the gross price for each product, the quantity sold and the total income. For this analysis only the first three are taken into account.
Exploring the dataset, the most sold product is 590122072KR with 21 occurrences while there are more less sold product with 1 occurrence in the dataset. In the records there are a total of 949 users, 7961 products and 2452 transactions. The longest transaction has a length of 48 products and the shortest just one product for a mean value of 4.08 products sold for each transaction.
For the analysis the entire dataset will be split into two subsets with a 80/20 proportion between training and test set.

5.1.2 Matrix Factorization

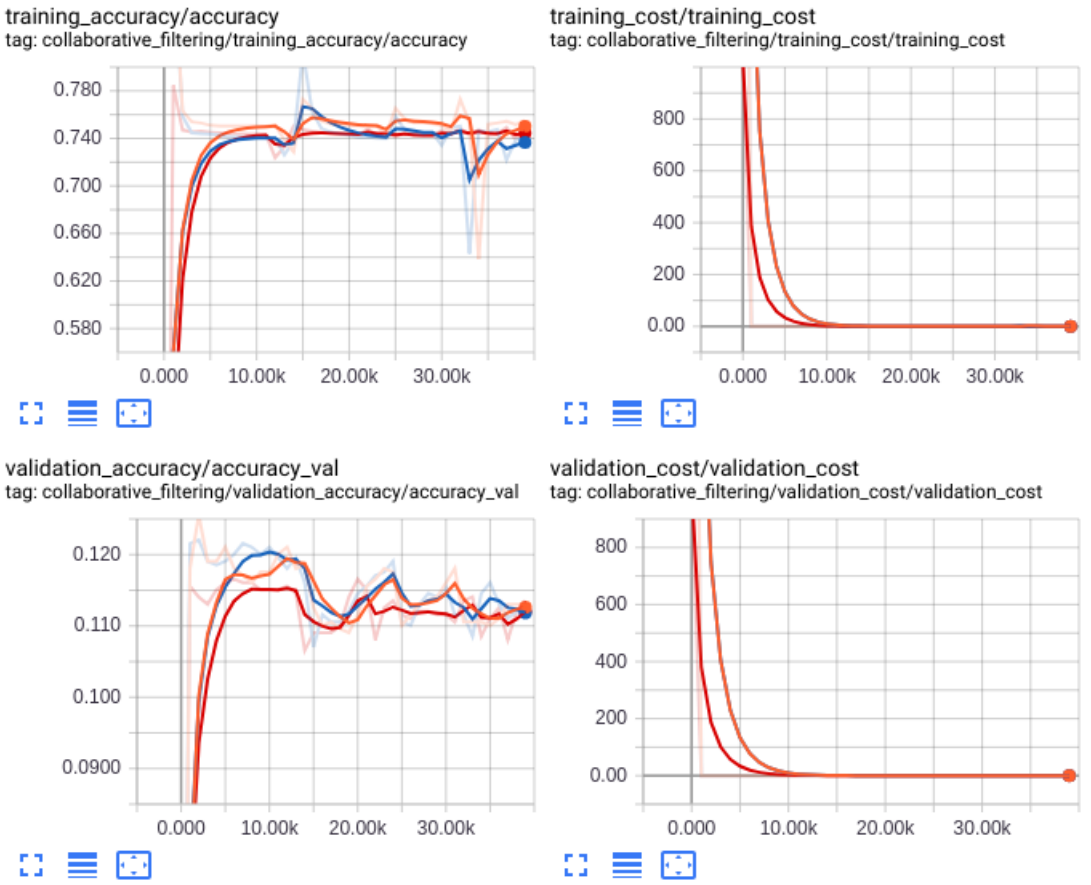
The first attempt is done with a feature vector size of 256.



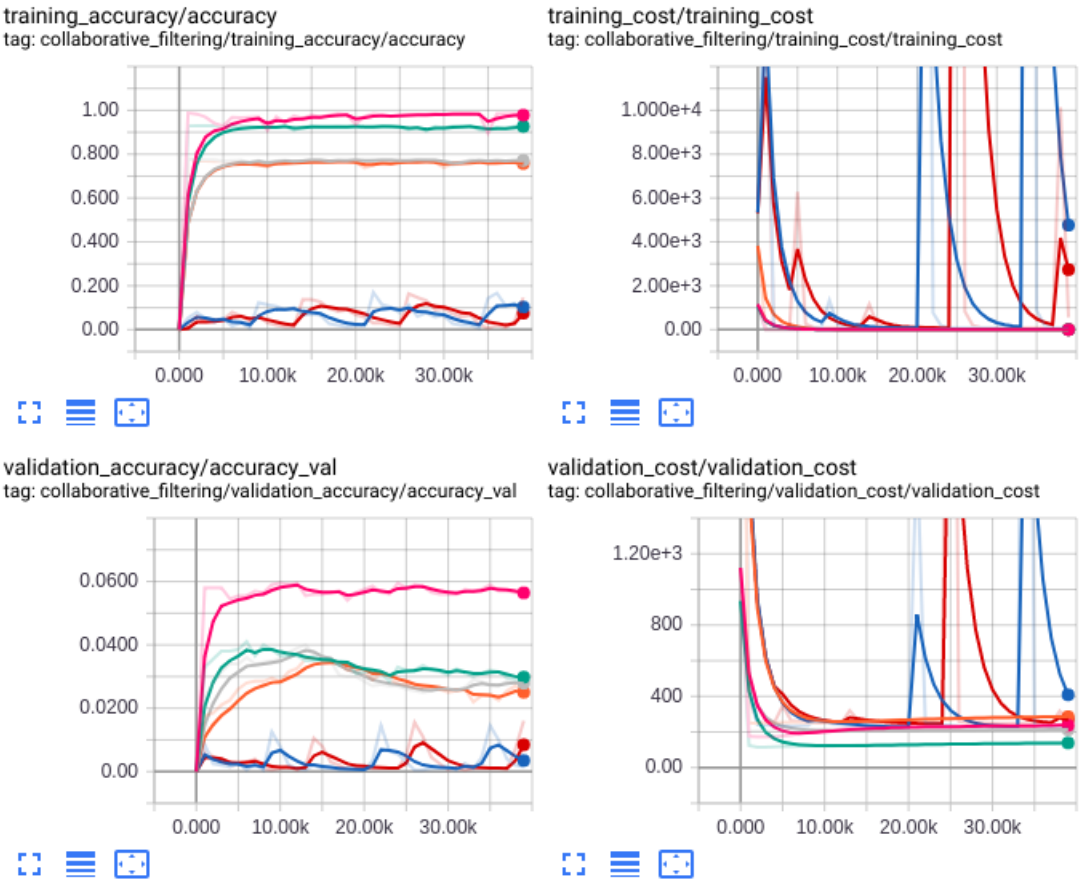
The accuracy on training set reaches a maximum of 98% in the first 1000 steps and descreases to 60% around 20000 training steps. Then it increases to reach a final accuracy of 70%. The entire training lasts 10 epochs with a duration of around six hours. The validation accuracy is very low for all the training process. This could be an overfitting problem and it can be solved with regularization. For this reason, the equation for the prediction becomes

$$\hat{r} = q_i p_j + \lambda(||q_i||^2 + ||p_j||^2)$$

In this way, noise coming from data is reduced forcing the model to learn useful features.
Using the quantity of products bought in each transaction, the system uses this values to predict how many items will be bought, considering them as ratings. The training set accuracy is good, going to an overfit without regularization. With different values of regularization the accuracy is always around 74 percent. Also changing slightly the learning rete does not improve the performance. The accuracy on the validation set is around 12 percent.



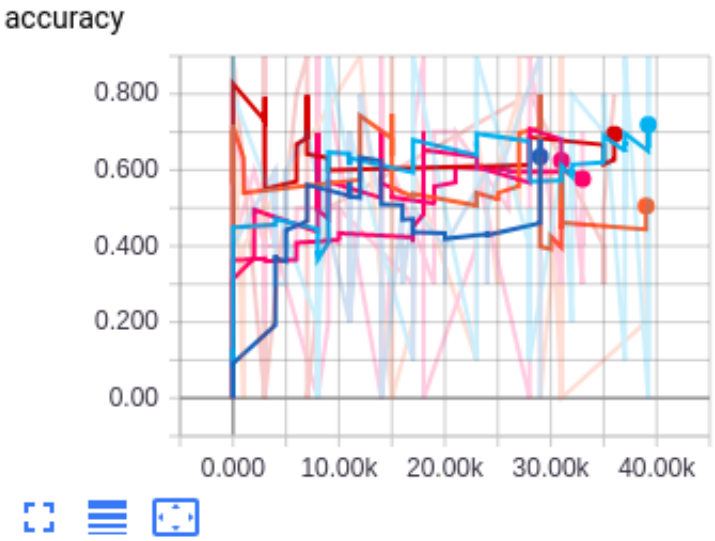
Anyway, the goal is to predict if a product is interesting or not for an user. For this reason, for each occurrence the value is set to one having in this way a binary matrix. Using a 0.5 threshold to split the yes-or-no decision, the results have an almost 100 percent accuracy on the training set and 6 percent on the validation set in the best case.



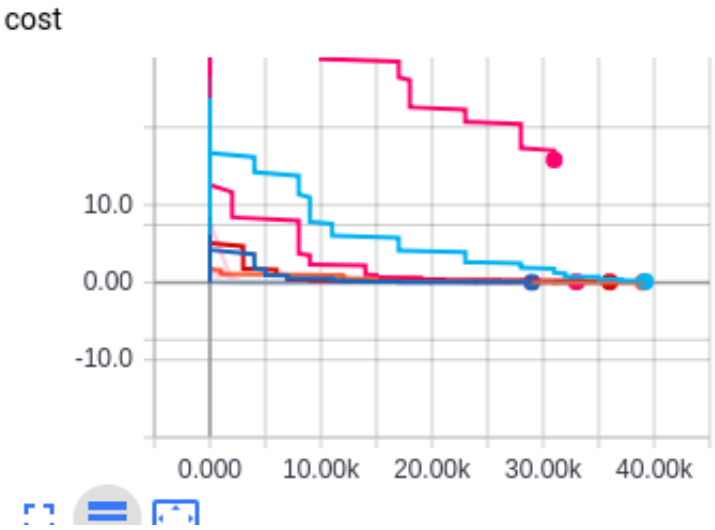
5.1.3 Neural network

In this testing phase, the same network is evaluated with 0.01, 0.03, 0.1 as learning rate values. The first case is the 1-layer autoencoder, with an hidden layer of 2048 elements and a dropout layer before the output. The dropout is a kind of regularization for this network that avoid the neurons relying on themselves and overfit on the dataset. It works keeping only a part of the nodes in the layer for the training phase to compute the prediction, while in the testing phase the probability to keep the neuron is 1 so every node is used to evaluate the output.
With a two-layers network, the first layer is the same as before and the second is composed by 1024 elements. As is shown in the figures below, the performance is lower with respect to the previous method. The training accuracy, despite the

small error cost, is between the 40% and the 80% but very irregular. The training cost converges near to zero at different times. With both network, 0.01 and 0.03 as learning rate give the fastest convergence.



cost



With the test dataset, results on the accuracy are higher with respect to the previous algorithm. In fact, in all the tests give performances between 5% and 40% with best results with the two layers network around 30%.

5.2 Testing on MovieLens dataset

5.2.1 Dataset overview

This dataset (ml-20m) describes 5-star rating and free-text tagging activity from MovieLens, a movie recommendation service. It contains 20000263 ratings and 465564 tag applications across 27278 movies. These data were created by 138493 users between January 09, 1995 and March 31, 2015. This dataset was generated on October 17, 2016.
The MovieLens datasets are widely used in education, research, and industry. They are downloaded hundreds of thousands of times each year, reflecting their use in popular press programming books, traditional and online courses, and software. These datasets are a product of member activity in the MovieLens movie recommendation system, an active research platform that has hosted many experiments since its launch in 1997. This article documents the history of MovieLens and the MovieLens datasets.
All ratings are contained in the file ratings.csv. Each line of this file after the header row represents one rating of one movie by one user, and has the following format:

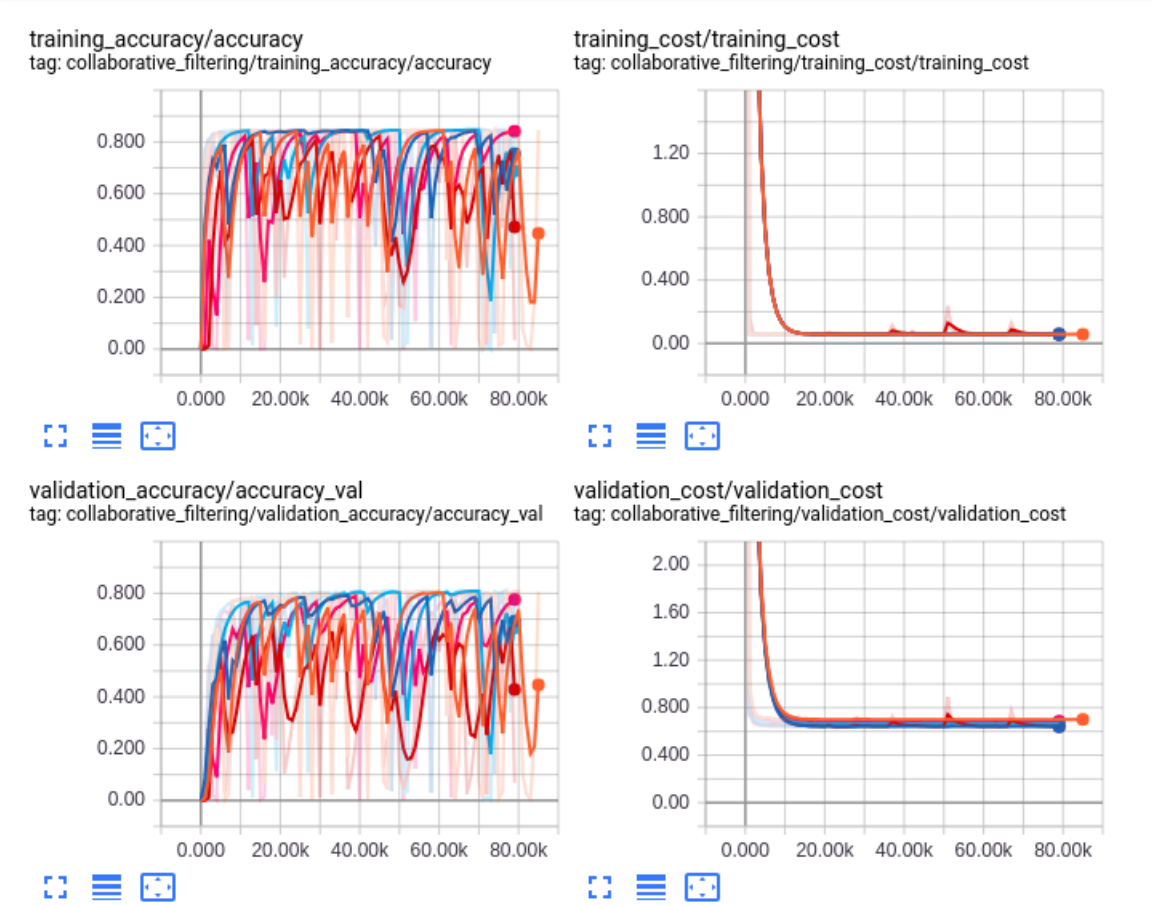
userId,movieId,rating,timestamp

The lines within this file are ordered first by userId, then by movieId. Whereas for the movieId there is the *movies.csv* which specify the relationship between the id and movie itself, the user is identified only by its id for privacy reasons. Ratings are

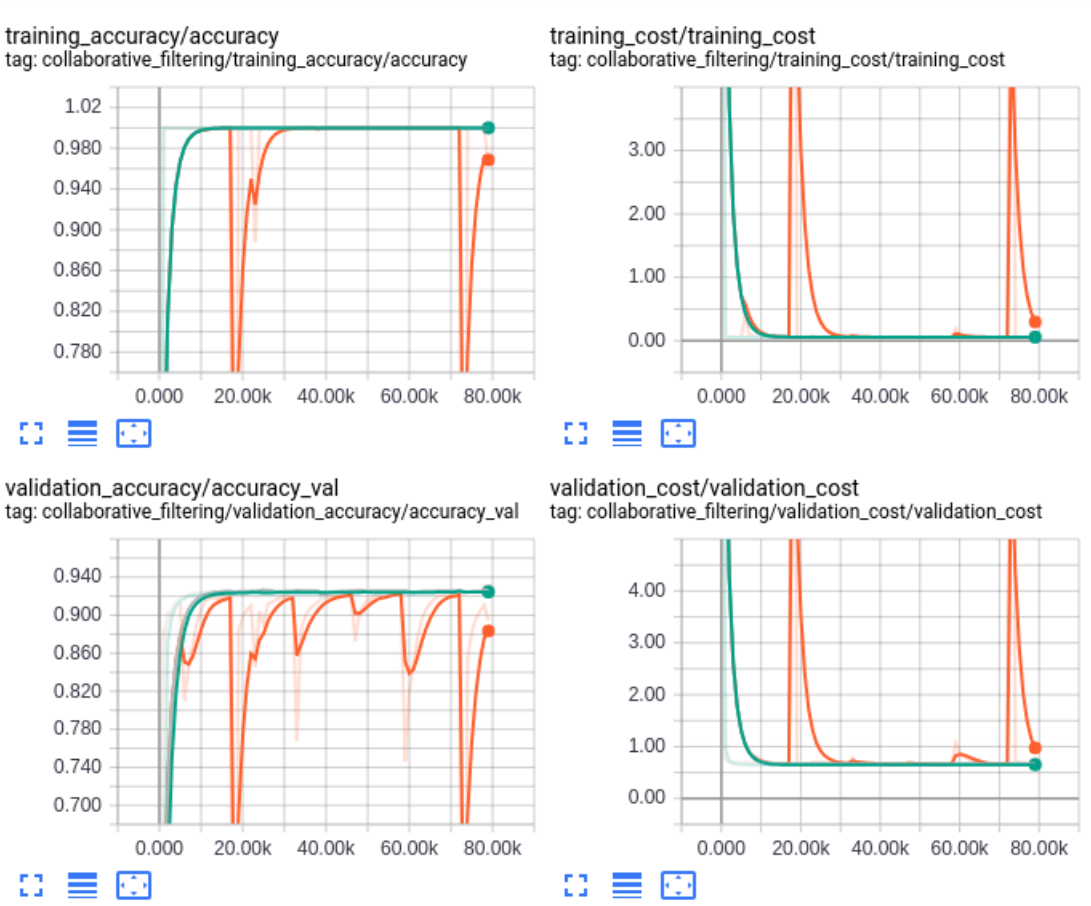
made on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars).
For this analysis, timestamp will not be considered and in some cases ratings will be set to 1 to match the previous cases done with the original dataset.
For computational reasons, only the first 100000 records are considered.

5.2.2 Matrix Factorization

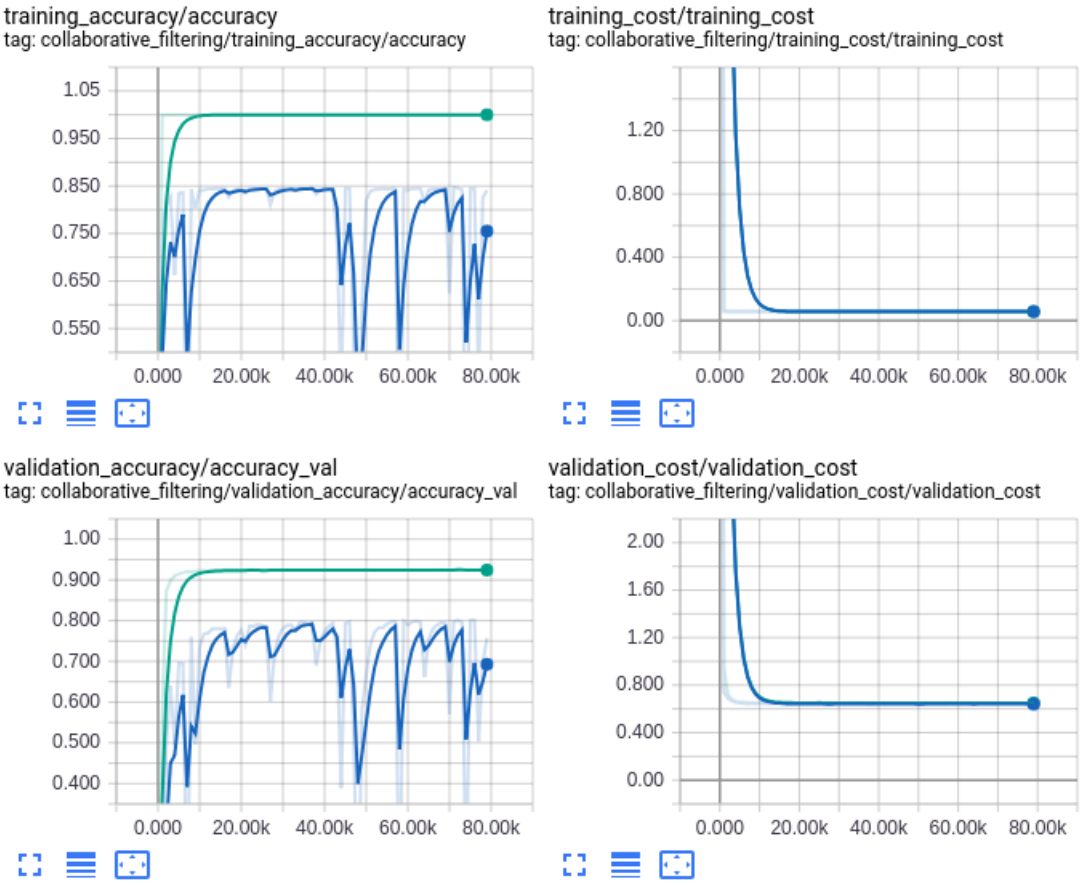
The first testing steps are related to a threshold of 0.01, that means that, if the predicted value is closer than 0.01 to the testing value, the prediction is good. The graphs below show how the results are improved simply using a larger dataset, even if the trend is more noisy with respect to the test with the Kramp dataset. The fluctuation is about the 20% between 50% and 80% of the accuracy in the validation set. The test is done with learning rates of 0.01, 0.03 and 0.1. In all of this tests, the computational time is around 12 hours with an embedded layer of 512 elements and about 24 hours for 1024 over 80000 training steps. Despite the bigger amount of time the increase of the performance on the accuracy is not so valuable.



With a threshold of 0.5, the graphs have less spikes except the one (in red in the figure below) with a learning rate of 0.1. In this case, the algorithm cannot reach the same accuracy values in the validation set as the 0.01 and 0.03 cases. The graph, in the last two cases, reaches the 100% of accuracy in the training set and 92% for the test set. The cost of both sets is very low, around 0.



The figure below shows the difference between best performance with 0.5 and 0.01 threshold which is about 15% for both sets.

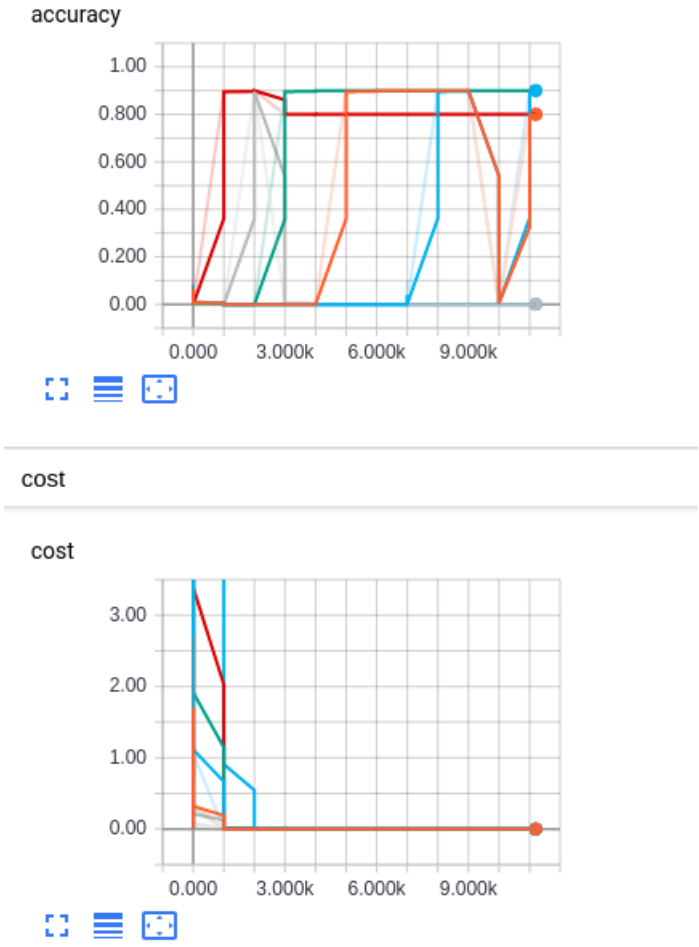


5.2.3 Neural network

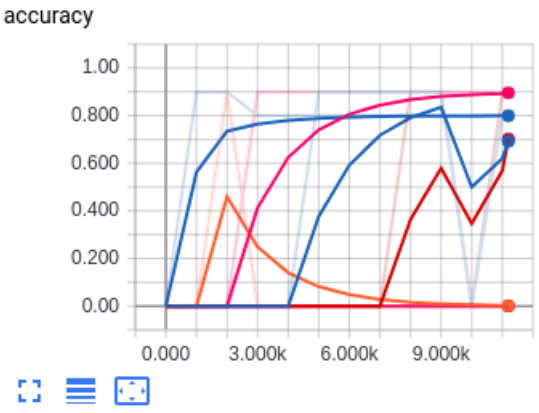
For this part, due to the nature of the dataset, the entire history of the user is used to create random transactions to have a similar situation with respect to the Kramp dataset:

```
def random_transactions(batches, length, num_transactions):
    transactions = []
    for batch in batches:
        for i in range(num_transactions):
            transactions.append(random.sample(batch, length))
    return transactions
```

Firstly, for the training set, the accuracy is around 90% with every configuration except the 2-layer configuration with 0.01 and 0.1 learning rate which is not able to perform well. The cost, instead, is near to zero for each configuration.



On the other side, as regards the test set, the accuracy is around the 70% and the 90% except for 0.01 and 0.1 values in the 2-layer configuration that does not perform like with the training set. The best performance is given by the 1-layer with 0.1 as learning rate parameter which gives 90%.



Chapter 6

Conclusion

6.1 Results

The results described in the previous chapter are, in the first case, not optimal due to the small dataset and the use of few features. Instead, with a larger dataset, the performance is improved.

6.2 Benefits

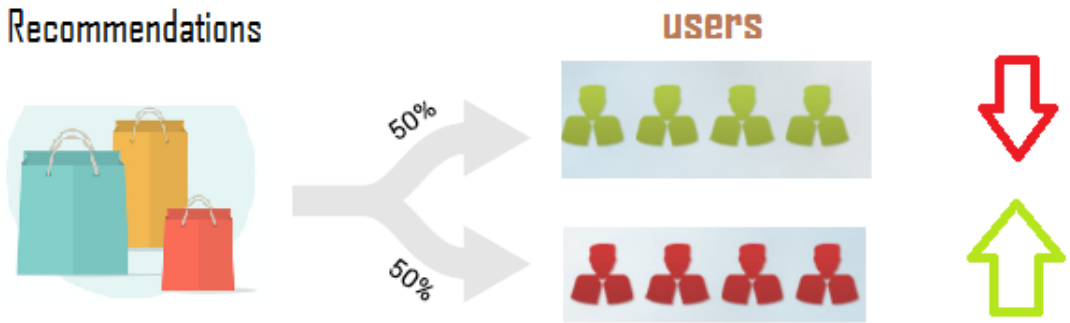
A recommender system on a website has the goal to increase sales by advising the customer of new products relied to his interests. It is a matter of making the user conscious of a wider part of the assortment, the part that could be valuable from him. This may lead to new purchases and changing relationships among products. In this way, the customer can find easily products he may be interested in with less effort and more inclined to buy other items because of his increasing satisfaction towards the system.

6.3 Business opportunities

Beyond increasing sales, a good recommender creates loyalty to the company. If the service is good, the consumer will likely reuse it again, because he feels that his needs are understood and thereby he creates an habit. In that way, he could become a long-term business customer. Moreover, data insight coming from recommendation can tell which products are the most trending in certain periods of the year, giving the possibility to be ready to the incoming requests.

6.4 Future improvements

The actual system and testing is done with a small batch of data. Training the system with more data could improve the behaviour and the performance of the recommender. On the other hand, a larger variety of data allows to have a better insight of the customer interests producing more accurate recommendations. For example, using information like age, country of the customer or category and alternatives for the product increases the possibility that the suggestion is appropriate.
Moreover, the training phase becomes heavy with a large number of data. For this reason, it is good to have parallelized computation among different CPUs or distributed computing among different PCs. This is possible with TensorFlow API, creating a cluster of TensorFlow server or with the support of Hadoop.
Furthermore, another improvement could be the merge of different models with a technique called model ensemble. For example, same input data train different classifiers and the results of each one go as input for a learner that tries to minimize the error.
In addition, the recommender system should consider temporal aspect giving more importance to most recent actions. The interests of an user may change over time due to different life situations.
Finally, another important thing is to find a good way to test the machine learning model. Instead performing only usual metrics as RMSE and precision-recall, the performance is evaluated at real time by splitting users in two parts to see which one performs the best.



In this way there is a real feedback done on the present period, it is even better if the recommender system receives the user interactions on itself and adjust weights to fine-tune the prediction.

Chapter 7

Acknowledgments

I would first like to thank my thesis advisor prof. Paolo Garza of the DAUIN - Dipartimento di Automatica e Informatica at Politecnico di Torino. The door to Prof. Garza office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this project to be my own work, but steered me in the right direction whenever he thought I needed it.
Then, I would thank my tutor Davide Iori, he helped me during my internship at Kramp, pointing out some possible outcome, giving me feedbacks and organising meetings with the team and some other experts.
I would also like to thank my collegues who were involved in decisions and helped me in this research project: Jaap Stramrood, Marcel van den Berg, Michiel Buevink, Aris van Ommeren, Bas Heijdra, Rob Sessink, Anna Adeney and Matthias Meerhof from Integration Team. Oliver Meisch, who helped me with data insights and with the understanding of Machine Learning concepts. Without their passionate participation and input, the project could not have been successfully conducted.
Finally, I must express my very profound gratitude to my parents and to my friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Bibliography

[1] T. M. Mitchell, *Machine Learning* McGraw-Hill Science/Engineering/Math, 1997.

[2] [Online]: https://developers.google.com/machine-learning/crash-course/reducing-loss/gradient-descent

[3] S. K. Gorakala, *Building Recommendation Engines* Packt, 2016.

[4] [Online]: https://www.tensorflow.org

[5] [Online]: https://cloud.google.com/gcp/

[6] [Online]: https://www.geeksforgeeks.org/best-python-libraries-for-machine-learning/

[7] [Online]: https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0

[8] F. B. Hasim Sak, Andrew Senior, in "Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling"

[9] [Online]: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[10] F. M. Harper, J. A. Konstan, "The MovieLens Datasets: History and Context" in *ACM Transactions on Interactive Intelligent Systems (TiiS) - Regular Articles and Special issue on New Directions in Eye Gaze for Interactive Intelligent Systems*, v. 5, p. 19, December 2015.

[11] [Online]: https://www.tensorflow.org/deploy/distributed