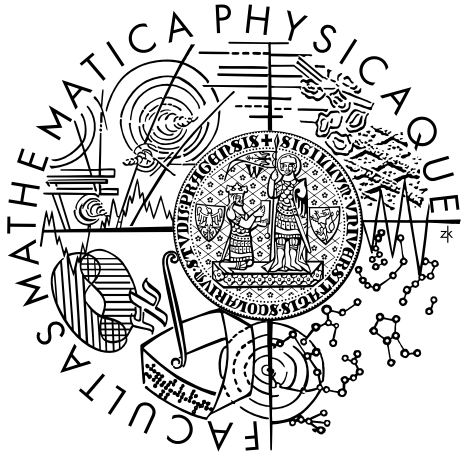
Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS



Kaan YÖŞ

Deep Learning For Implicit Feedback-based Recommender Systems

Department of Software Engineering

Supervisor of the master thesis: Mgr. Ladislav Peška, Ph.D.

Study programme: Computer Science

Study branch: Software and Data Engineering

Prague 2020

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

Title: Deep Learning For Implicit Feedback-based Recommender Systems

Author: Kaan YÖŞ

Faculty of Mathematics and Physic: Department of Software Engineering

Supervisor: Mgr. Ladislav Peška, Ph.D., Department of Software Engineering

Abstract: The research aims to focus on Recurrent Neural Networks (RNN) and its application to the session-aware recommendations empowered by implicit user feedback and content-based metadata. To investigate the promising architecture of RNN, we implement seven different models utilizing various types of implicit feedback and content information. Our results showed that using RNN with complex implicit feedback increases the next-item prediction comparing the baseline models like Cosine Similarity, Doc2Vec, and Item2Vec.

Keywords: machine-learning, recurrent-neural-networks, recommender-systems, next-item-prediction

I dedicate this work to my parents Sevcan and Turgay, and my brother Arda Ege. I could never be where I am today without their support.

Additionally, I would like to thank Agata Filipiak for her excellent support and motivation.

Finally, I kindly thank my supervisor Mgr. Ladislav Peška, Ph.D. for his great help, guidance and feedback throughout the thesis.

Contents

Introduction

Recommender Systems are playing a critical part in online shopping. Companies like Amazon or Netflix are spending many resources to build their recommendation engine. 35% of Amazon's revenue comes from the success of its recommendations [8]. In 2006, Netflix started a contest named Netflix Prize. They proposed to give 1 Million Dollars to the team, which can increase the accuracy of their recommendation algorithm by 10%. Contest took five years to complete, and the winner team was able to increase the recommendation accuracy by 10.06% [27]. Netflix Price played a crucial role in recommender systems to get attention. There is plenty of research done for Recommendation Systems, and companies start either building their recommendation framework or purchasing already implemented solutions.

When customers visit physical stores, they can often see new and popular items that the store owner pre-filtered. To buy less popular items or discover new possibilities, customers need to visit online stores or look at the printed product catalogs. While it is not possible to represent all the items in a physical store due to capacity limits, it is possible to put them into a website. However, companies like Amazon have millions of products, and it is not convenient to show all inventory to customers directly. Recommender systems take the responsibility of filtering which was done by store owners and improve it in many ways. First of all, recommender systems can recognize the customers' interest and filter the items based on individual customers. They can use only user profile, only item meta-data, or both. It is also possible to use Information Retrieval techniques such as keyword or pattern search to obtain information without having direct user feedback.

User behavior and interest differ from one domain to another. That is why, often, recommendation algorithms are adjusted specifically for the domain. According to Netflix, "typical Netflix member loses interest after perhaps 60 to 90 seconds of choosing" [8]. Mainly, Recommender Systems aim to predict user interest before the user finds it. For example, a travel agency web page would like to recommend a travel package to its users before they visit the dedicated tour page. Alternatively, Netflix would like to recommend a new series or a movie to its customers before 60 seconds. Another aspect is that the recommender systems help customers to discover new possibilities. This will lead users to make more purchases and keep their interest in the business. However, this is not an easy task to do. Recommending a product that does not reflect the initial user interest may lead customers to think the recommendation algorithm is bad and is not able to learn their preferences. Overall, building a recommender system requires not only domain knowledge but also understanding human behavior. That is why recommender systems can be concluded as a combination of art and science.

Recommender Systems require data to make better recommendations. The required data can be obtained by using information retrieval techniques. Systems can track user actions, such as time spent on a web page, clicked items, number of purchases, and so on, then ask for users' direct feedback. Even though asking customer opinion directly is the best way to understand their interests, users often do not want to interact with the system. While it is possible to have high-density

data for companies like Amazon or Netflix, this is not often true for small e-commerce businesses. One significant difference between small e-commerce businesses and big ones is user loyalty. The number of small e-commerce businesses is increasing every day, and customers have more places to make purchases. They can visit many online shops, compare prices, stock availability, trust, etc. and then make their purchase.

Additionally, users search for a specific product using search engines, end up visiting the dedicated product page instead of visiting the home page, and navigating the items they want. Those scenarios cause small e-commerce domain to suffer from the lack of data and have short user sessions to work. Furthermore, small businesses such as travel agencies, car dealers, or real estates have a lower amount of total purchases than other domains. Customers usually buy travel packages for their work holidays, which roughly corresponds to two times per year. If we consider those customers who purchase summer holidays once a year, we should consider that user interest might change next year since there is a long gap between the first and second purchases.

Small business recommendations often use collaborative filtering on the object level. They mostly suggest items similar to those that the user purchased earlier or the ones user visited. However, due to the limitations described above, using only item-based collaborative filtering might not be the best option. It is possible for small e-commerce business to access more information that can be used for recommendation. This extra information can be obtained by tracking user actions once they enter the website and having a proper meta-data regarding their company products. Keeping track of the user action and put them into sequential order by time would allow businesses to understand what were users doing during their session and it can be used as complex user feedback. Sequence Aware Recommendation Systems is a good candidate to use when the business is able to collect extensive user actions. By using recurrent neural networks, small businesses can increase the quality of their recommendations on item-based and sequence-based recommendations.

This thesis focus on deploying a recommender system for small e-commerce businesses. Proposed recommendation algorithms use a real-life travel agency dataset. While it was possible to use a collaborative filtering approach, it would cause cold start problems since there are not enough users to cover all the objects in the dataset. However, the dataset provides quite extensive implicit feedback and information about the objects. Using the available data, we have implemented recommendation algorithms by using Recurrent Neural Networks that can learn from short user sessions and create recommendations.

The thesis has four chapters. In section 1, we will describe Machine Learning, Deep Learning, Neural Networks, and give specifics about Recurrent Neural Networks. Section 2 will explain Recommender Systems, information retrieval, and filtering techniques. Then it will describe the Sequence Aware Recommender System and explain the evaluation metrics and methods used in Recommender Systems. Section 3 is dedicated to describing the domain and the dataset used for the thesis. Additionally, it will give a brief analysis of the used dataset. Lastly, Section 4 will talk about the proposed implementation of the thesis. It will describe the pre-processing the raw data, implemented algorithms, and used neural network architecture, evaluation methods, and results.

1. Machine Learning

Machine Learning is an Artificial Intelligence area that deals with the theory and implementation of intelligent computer systems able to perform tasks that would require a human presence. That is why the machine learning field has a secure connection with computer science, mathematics, engineering, psychology, statistics, and neuroscience.

The goal of machine learning studies is to find the answer and possible implementation of whether people can build an intelligent system able to perform tasks and make decisions as a human being does. However, this is not an easy task since it requires understanding how human reason and think in their lives. Any intelligence system that is proposed to perform human tasks must be accurate and fair since humans do not perform plain predictions. Human reasoning is related to individuals' feelings and emotions, which is the result of experiences. Additionally, these changes from a person to another since people are self-aware beings and have different experiences. Self-awareness is something machines do not have, and we are still very far from implementing those unique human features to machines. On the other hand, there are machine learning models currently being used, and they perform remarkable results.

Building a machine learning model is a multi-step process. Each step has its technical details and challenges. This section will cover the history of machine learning, some of the data processing steps, and different learning types for machine learning models. Then it will give brief information on Deep Learning and Recurrent Neural Networks.

1.1 History of Machine Learning

Even though term machine learning became popular recently, its origin goes back to the 1950s. Arthur Samuel (1901-1990) was a pioneer [24] of artificial intelligence research. He used a game of checkers to study how computers can learn from their experience. Arthur created a program that learns and adjusts itself to choose the right moves by using a book named Lee's Guide to Checker [22]. The learning process was no close to the artificial intelligence level we currently have. However, it was the main starting point.

Although the machine learning era started before the current century, the development was slow and close to none. The vital reason behind this was the available data and the computation power of the computers. These two factors are very close to each other since building sophisticated machine learning models, need powerful computers to process high-density data, and this was not possible before the year 2000.

The 21^{th} century was a turning point for technological content consumption. Thanks to Industry 3.0, people were able to manufacture smaller transistors and integrated circuit chips in the automated factories. This helped computer hardware to develop quickly, and more powerful machines to be built.

Social media were entered peoples' life in 2001. With the presence of social media, people start consuming more content on the internet. This consumption has increased dramatically with the new social network platforms. Myspace was

the first social media platform to reach a million monthly active users in 2014. [1]. This number increased to fifteen million in 2005. At the begging of 2019, Facebook has reported a total of 2.26 Billion active users who have logged in to the platform in the last 30 days [29].

In a short amount of time, two main blockers for machine learning started to become less problem with the 21st century. In 2020, people can build powerful hardware with the help of Industry 4.0, and digital content consumption has reached 44 Zettabytes [7]. That is why the terms Machine Learning, Artificial Intelligence, and Deep Learning have become more popular recently. With the help of bigger data sets and more powerful computers, people can create machine learning models and algorithms better than ever. As stated before, these improvements are not close to what is known as self-aware machines or high-intelligent systems capable of self-thinking and performing multiple tasks performed by a human. However, with the current improvements, researchers are still trying to find new ways for computers to "learn" from data, and the term Machine Learning is being used for this purpose.

1.2 Machine Learning Theory

Arthur Samuel described machine learning as "Field of study that gives computers the ability to learn without being explicitly programmed." Although this definition reflects the sentiment of machine learning, it does not explicitly answer the question of how this is possible. The more scientific description of machine learning was reported as "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E." by Tom Mitchel. [25]

1.2.1 Machine Learning Task

Machine learning models are being developed to perform predictions for particular problems in a short amount of time while maintaining high accuracy. To achieve that built model must learn from the provided data such that it will try to find the underlying meaning and use it to produce precise predictions for the assigned problem.

The most important and the main objective of machine learning is to have a model that can make correct predictions even for the data which the model has not seen during the training phase. There is a slight chance that the machine learning model will see the same training data ever again. Therefore, the model must correctly identify and learn the patterns of the input data that are generalizable so that the model can produce accurate predictions on the new data. For this reason, the training data must be cleaned and processed before it is given to the machine learning model. It happens that the data does not have any patterns nor relations, and in this case, the model will not be able to learn from the given data and perform weak predictions.

1.2.2 Data Processing

Data processing is the most critical part of machine learning. Input data must be carefully processed in order machine learning model to learn it. Any small error during processing can cause the model to have weak predictions. This might cause catastrophic results, especially for the sectors that machine learning is being used heavily, such as; healthcare or autonomous cars.

The data processing phase changes based on the available data and the machine learning algorithm is used. Regardless of a model or algorithm, there are general steps to be taken. This involves both **data engineering** and **feature engineering** steps as shown in Figure 1.1. They allow the machine learning model to process data efficiently and learn from it.

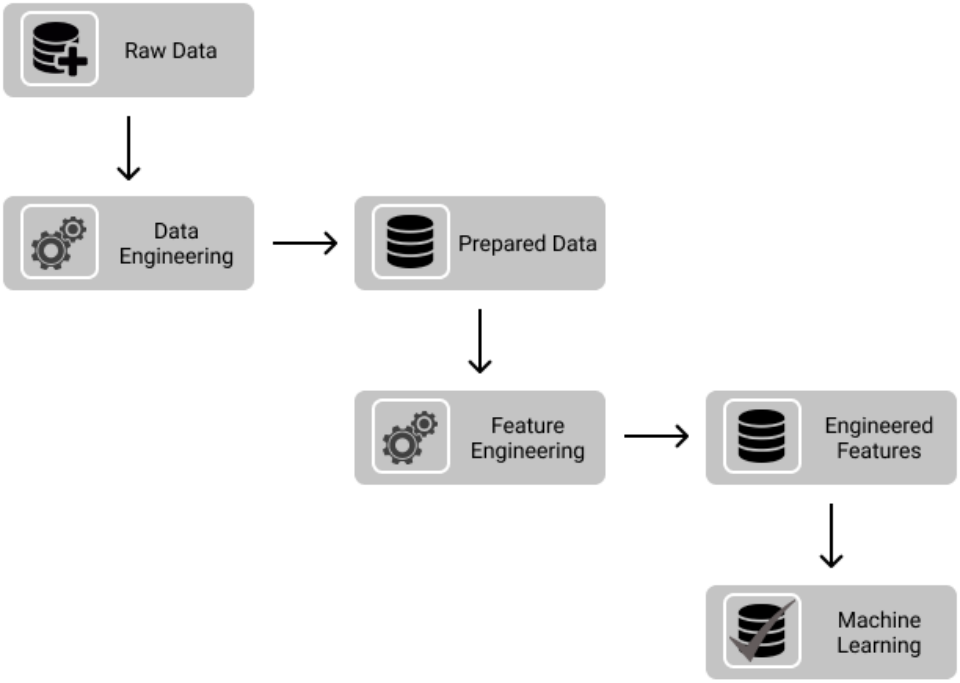


Figure 1.1: The two most important steps of data processing are Data Engineering and Feature Engineering. They require special attention due to their direct impact on learning.

1.2.3 Data and Feature Engineering

In general, the flow of creating a machine learning model starts with the raw data, which will be used to solve a particular problem. It is important to explore the source data and apply cleaning techniques such as removing/filling null or corrupted values. Once the dataset is cleaned precisely, it is possible to pick data points from the source data to create train, test, and validation splits. Processing the raw dataset and applying splits, also known as data engineering, is a prerequisite step before moving to feature engineering. In some cases, data engineering and feature engineering can be merged into the same pre-processing step.

After the raw data is processed, it must be turned into features suitable for the machine learning model. Extracting the features of the processed data is often called feature engineering. Extracted or created features are given to the machine learning model as a sequence of *observations* $((x_1, y_1), (x_2, y_2), \ldots (x_n, y_n))$ where x is known as the *features* which contain the specific representation of the given input, y is the output *label* for the given feature and n is the total number of observations. Once the machine learning algorithm is given the pair of features and labels, it knows the exact meaning of the data with the corresponding label during the training. This type of learning is called as *supervised learning*. If the model is only given features without the corresponding label and is expected to learn from the data, then the learning known as *unsupervised learning*.

1.2.3.1 Data Cleaning

Data cleaning is generally the first step of pre-processing, where unnecessary data and corrupted values are removed or corrected. Based on the domain and the available data, this step might be performed in different ways.

All the rows that contain corrupted or null data must be removed or filled with a new value to keep the meaning of the source data. The easiest way to handle this problem is to drop all corrupted or null values. However, this method reduces the information and will cause information loss, which is not the best practice for the machine learning. On the other hand, filling the null data or fixing the corrupted values requires attention. *Imputation* is beneficial when it comes to filling the null values, and there are different steps that one can choose. Imagine an e-commerce dataset that logs the products in the basket once the customer makes purchase. If somehow, the system do not have any basket data but have purchase information, it can be assumed that the user has added purchased items to the basket since the purchase was made. However, this method is not applicable for every data set and every domain since it depends on assumption.

One of the most natural imputation methods is to apply Mean-Median-Mode Imputation, which allows us to fill the missing values after performing mean, median, or mode operations. Another method is to apply Regression Imputation, which mainly using a regression line to fill missing values.

There are multiple ways to handle missing information, and there is no silver bullet. The most complicated method does not necessarily mean that it is the best one. An optimal solution can be found after analyzing the raw data and the domain.

1.2.3.2 Standardization

Real-life data is often not uniformed based on a particular value. Even the most straightforward dataset, which contains information about the people's age and weight, needs to be standardized so that the machine learning model can understand the values. The machine learning model distributes *weights* for each value, and these assigned values will be used during the training phase. If age and weight information of the person is not being scaled, the machine learning model will be giving more importance to a person weight since weight is more likely to be higher than their age. That is why standardization is a must for machine learning models. Standardization often applied by calculating **mean** and

standard deviation of the values. Then we subtract the mean and divide by the standard deviation for each value in the data set.

$$z = \frac{x_i - \mu}{\sigma} \tag{1.1}$$

Standardization is applicable by using the formula 1.1 in the cases of numeric values. However, there are other methods to be applied in order to deal with *categorical* values.

1.2.3.3 Categorical Values

Categorical data has two or more categories with no natural ordering between the categories. As an example, gender has two categories as female and male, and it is not possible to introduce ordering between these two categories. Seasons for a year can be another example of a categorical variable, and it cannot be ordered.

One way to work with categorical data is called binarization. In binarization, categories are first converted into ordinal values. Then those integers are transformed into binary for the final representation. Likewise, categorical data can be transformed into a vector of zeros whose length is the total number of categories. The location of the specified category is labeled as one. This method is also known as One-Hot-Encoding (OHE). Consider the months of a year as categorical variables. Table 1.1 shows the integer encoding, binary encoding, and one-hot-encoding of the months.

Month	Integer Encoding	Binary Encoding	One-Hot-Encoding
January	0	000	100000000000
February	1	001	010000000000
March	2	010	001000000000
April	3	011	000100000000
May	4	100	000010000000
...	...	...	...
December	11	1011	000000000001

Table 1.1: Months of a year can be encoded as integer, binary or one hot encoding as shown.

Integer encoding has an advantage since it is able to keep the ordering between the categories. However, this also causes categories to be distanced from each other, which might not necessarily be part of the original value. While it is possible to keep all the category information using One-Hot-Encoding, it might expose dimensionality problems in the case of a high number of categories. On the other hand, binary encoding converts an integer to binary digits, which generally ends up in lower dimensionality compared to OHE, but it will also cause information loss.

1.2.4 Supervised Learning

Supervised learning is a type of machine learning model in which all the input data is labeled. Meaning that the data is served into the machine learning model

in a pair of observations $((x_1, y_1), (x_2, y_2), \ldots (x_n, y_n))$ where N is the amount of data points and x_n is the input variable, also named as *independent variable* or *feature*. y_n is the *label* or *target variable*. The feature variable contains descriptive information for the object, generally in a vector form. On the other hand, the target variable is what model will be trying to predict.

The primary purpose of the Supervised Learning to perform classification or regression tasks while exposing the data labels to the model. Then, this model is being used to produce predictions for the given problem. Consider a scenario that a user visits a web site of a travel agency. The web site logs the use actions during their session so that the system aware of the actions like mouse-click, item-visit, or item-purchase. Figure 1.2 shown an example of a specific user session.

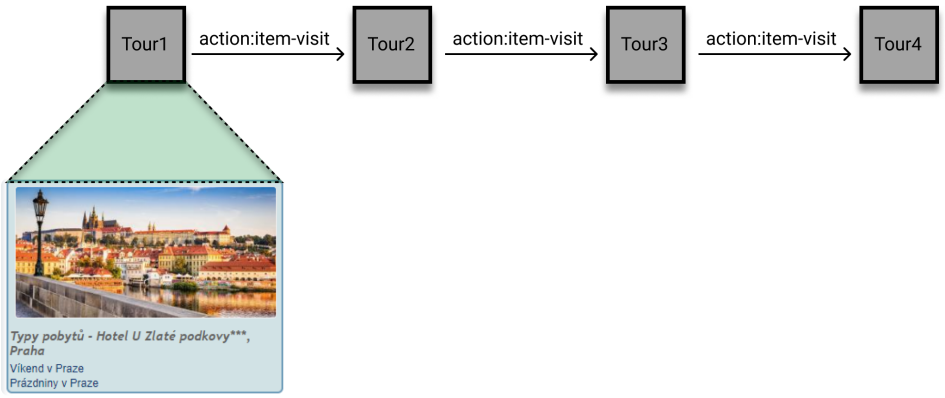


Figure 1.2: Example of a user session logged by a website. Dedicated tour pages contain content information regarding the visited tour.

The user has visited four travel destinations during the session. Each item has its own id; $Tour1$, $Tour2$, $Tour3$, $Tour4$ and its dedicated page for a description. This sequence data should not be used for Supervised Learning directly and that is why it must be labeled. One possible way to divide user sequences into features and label is shown in Figure 1.3.

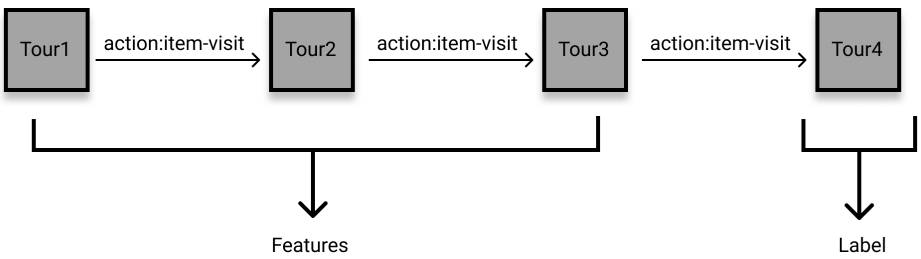


Figure 1.3: Example uses the first three visits in the user sequence as features. The last visited tour page is kept as the target variable. Using the last item in the sequence as label while keeping the previous items as features often used for next-item prediction in recommender systems.

1.2.4.1 Classification

The classification task aims to predict the correct output class of the classified data. This data can vary from identifying a cat from an image to identifying the next visited item in the user shopping session. In classification, the total number of classes is known. This allows the machine learning algorithm to be aware of the total output classes. In the case of the next item prediction, the output classes often are the items that are offered by the business. In this approach, the machine learning algorithm might use the input data described in 1.3 for the training. Once the model trained to performs classification, it will often create a ranked list of recommendations ordered by their possibilities. In this case, the output class which has the highest probability refers to the one that the model thinks is the best choice.

In classification, it is possible to have many attributes for a feature. When the number of characteristic attributes increases, the dimensionality of the input data increases. High dimensionality dramatically affects the performance of the model to fit on data with many features also known as the Curse of Dimensionality. Reducing the dimensionality often referred as *dimensionality reduction* it will bring the number of columns down to allow the model to use the given input easier.

1.2.4.2 Regression

In regression, machine learning model tries to predict a target variable y based on a given input x [46]. It assumes a linear relationship between the input and output variables. One of the best examples of a regression task is predicting future stock market prices.

1.2.5 Unsupervised Learning

It has been concluded that the Supervised Learning aims to find a functional relation between the given input features and the expected output labels. To achieve that, one must pre-process and divide the data into the feature and label pairs so that the machine learning model learns from it.

The Unsupervised Learning aims to find the structure of the underlying information of the given data while no output label is given. Unsupervised learning is an important method of learning from the data. The machine learning model is only given input variables, *features*, and expected to find the underlying relation between them.

One of the most important types of Unsupervised Learning is *clustering*. Clustering is the organization of unlabeled data into similarity groups that are called clusters. A cluster is a collection of items that are similar to each other. The items which belong to the same cluster are assumed to be more similar than the ones belong to the different cluster. One of the first usages of clustering is done by John Snow, a London Physician who plotted the location of cholera deaths on a map during the outbreak in the 1850s [44].

1.2.6 Training a Machine Learning Model

The machine learning model tries to find a function f that fits best to given input features and target variables written as $y = f(x)$. However, in real life, this is not enough. The trained machine learning model should also make good predictions on the data it has never seen. Thus, the ultimate goal of training a machine learning model is to have a well-learned function f, which generalizes to data not used for the training.

One possible way to achieve generalization is to divide the entire data set into the subsets randomly; the *training set* and the *test test*. In the beginning, the machine learning algorithm will be trained on the training set to find the best possible function f. Then the model will be evaluated by using the test set. During the evaluation, the model will be tested on the data it has not seen before.

Depending on the data size, the data set can be divided into three subsets; *training set*, *test set* and *evaluation set*. With this approach, the evaluation set will be kept outside of the training and testing. After the machine learning model is ready to produce predictions, it will be evaluated with the evaluation data. This allows the model to have a more robust evaluation and get a better estimation regarding the prediction quality.

1.2.6.1 Cross-Validation

Another way to measure model performance is to split the available data more than ones. Each splitting point will contain a different group of observations in each set. During the training, model performance on each split should be logged. Then, one can take an average of the performance results to get the final metric or can pick the best model among the splits to evaluate on the evaluation data. This method is called Cross-Validation. By using Cross-Validation, it is possible to have more measures for the machine learning model, including the stability of the model.

There are multiple ways to divide the available data. One common way is to use all the available data as the training set and leave on observations out. This method will iterate all the observations in the data set and will always leave one observation out. That is why this approach is known as *leave one out*. Consider a data set that includes six observations. By applying for the leave one out method, the model will use the first five observations, from obs_1 to obs_5 for training and the last observation, obs_6 for the testing. Then, the model will assign another five sequence observations for training, including the one used for testing. This will be repeated until all the training data is being used for testing and illustrated in Figure 1.4.

Another way is to leave out a group of observations instead of one observation in each iteration as shown in Figure 1.5. Available data must be randomly partitioned into k equal-sized folds. During the training, one of the folds will be used for testing while the other folds will be used for training. Since the available data is divided into k folds, this process will be repeated k times.

There is no correct answer for dividing the available data. An important factor is to consider which validation factor will impact the bias and variance of the testing phase before choosing a validation method. Dividing the data set into one split will affect the evaluation results depending on which side the data is

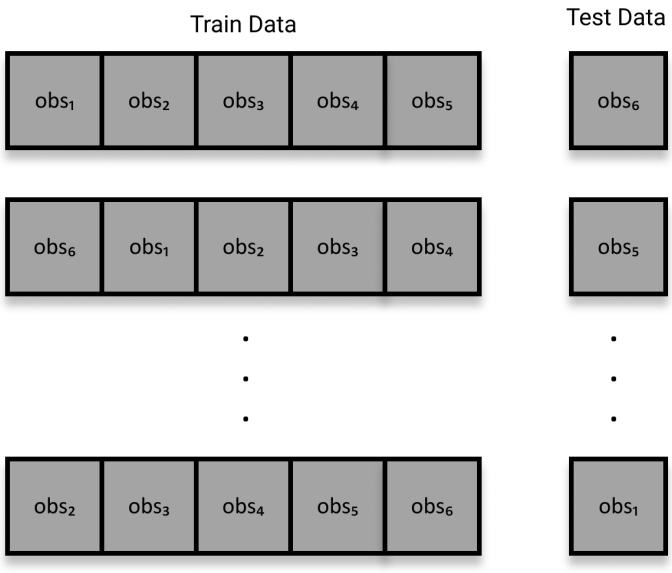


Figure 1.4: Applying Leave One Out on a training Data. Splitting the data will be repeated until all data points in the data set is being used as test data.

divided, and this adds bias to the testing phase. If the available data is split by using the *leave one out* method, then there will be many test results that will add variability in the test error.

1.2.6.2 Bias - Variance

Creating a successful machine learning model is a continuous process. As the complexity of the model increases, the model performance tends to increase. However, there are situations that the machine learning model learns wrong information from the given data, which is called *bias*. With the high bias, the machine learning algorithm will not be able to see the relation between the given features and labels correctly and produce wrong predictions. This behavior is known as *under-fitting*. A machine learning model trained on wrongly processed data or a very small number of observations, or simple algorithms or having data that does not correlate with each other is likely to perform poorly. The model will not be able to find any meaningful patterns, and it will have *high biased*.

When the machine learning algorithm has high variance, it means that the model can capture some aspects of the given training data but cannot generalize the function f very well; this is known as *over-fitting*. This generally happens when machine learning is too complex. Figure 1.6 shows the explanation of the tradeoff between underfitting and overfitting.

1.3 Neural Networks and Deep Learning

It is possible to ask why does a machine learning model needs a layered structure. When human eyes see a number, the brain can easily distinguish the number

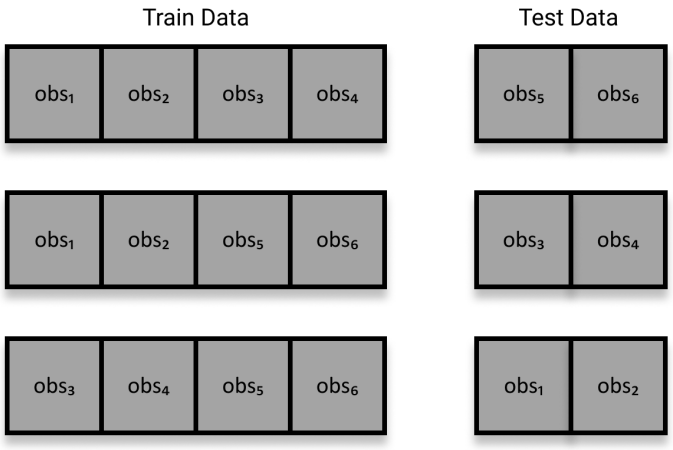


Figure 1.5: Applying K-Fold Splitting on training data. Data set will be split to equal k folds. Splitting the data will be repeated until all splits in the data set is being used as test data.

without actually thinking about the structure of the number. Consider number 9, which has a loop at the top and a curvy tail on the bottom. Number 8 also has a loop at the top as well as the second loop at the bottom. The human brain can capture the entire structure of the image so that one can understand when two loops come top of each other, it means number eight. However, this is not an easy task for a machine learning algorithm, and that is why it requires a structured network.

Structured architecture can have one or more layers in it. Each layer contains units known as *neurons*. Every neuron holds a number, known as *activation*, mostly between zero and one, and this number will let neurons lit up if the value is close or equal to one and will not otherwise. Once a neuron lits up, it can send information to another neuron on the next layer. This behavior is similar to the working principle of the neurons in the human brain. Once the network has layers and neurons, it can transmit information from the first layer to the last one and classify if the number is 8 or 9. However, the numbers in the neurons are not enough to complete this task. The network needs to divide those numbers into subproblems and make the task easier to solve. To be able to transfer information between neurons, the network creates a connection for each neuron pairs. Every connection is assigned with an adjustable parameter called *weights*. Weights are also just numbers as the neuron activations. Once neurons lit up and send information to the next one, the network sums the neurons activation values with their corresponding weights.

Shortly, the goal of the structured network is to help the machine learning model to divide the problem into subproblems in a way that neurons in one layer can lit the correct neurons on the next layer.

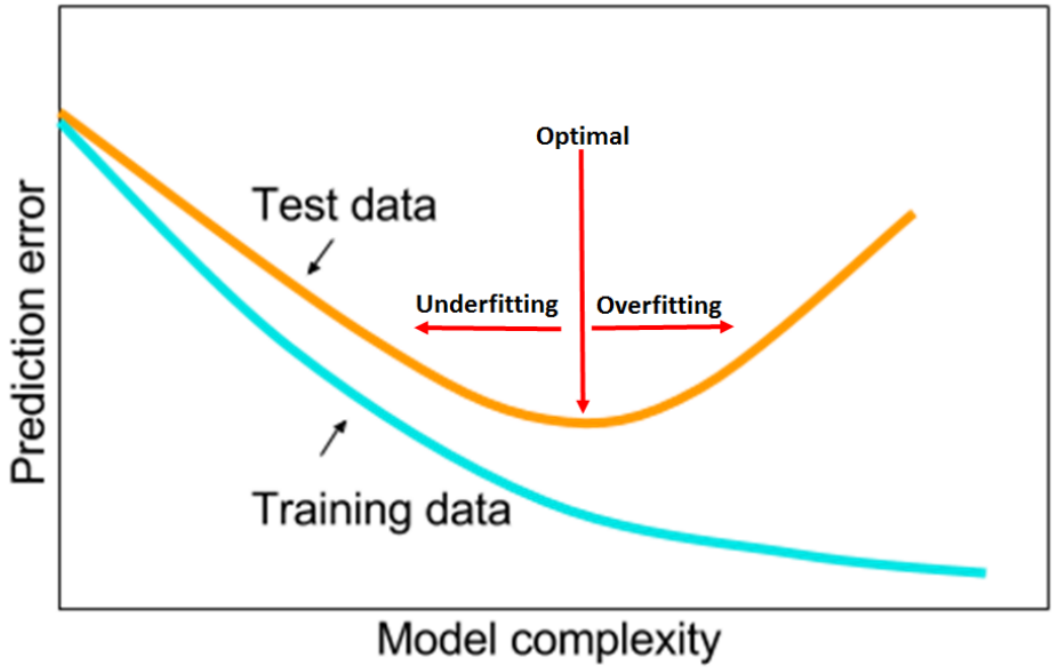


Figure 1.6: Pictorial explanation of the trade-off between underfitting and overfitting. Model complexity (the x-axis) refers to the capacity or powerfulness of the machine learning model. The figure shows the optimal capacity that falls between underfitting and overfitting. [41]

1.3.1 The Perceptron

Perceptron was introduced by Frank Rosenblatt [36] and related to a neural network unit (neuron) as a mathematical function that gets input and creates an output. The neurons in the human brain receives an input signal from the brain. This signal is processed through multiple neurons based on its density and then being performed. Figure 1.7 illustrates one of the representations of a biological perceptron.

The most straightforward neural network is called a perceptron as shown in Figure 1.8. This network includes a single input layer and an output node connected to the input layer.

Perceptron separates the incoming input into two categories and considers if the neuron should lit up and send information or not. Therefore it is also known as a Linear Binary Classifier. This behavior enables neurons to learn by themselves and processes given input in the training set one at a time.

Once the input data is sent to the network, the neurons in the first layer will be assigned with their activation number. Those neurons are connected to the neurons on the next layer by weights. One particularly important point here is to decide how to activate the neurons on the next layer. This is a curial task since it affects learning directly. What neural network does that it gets all the activation numbers of the neurons in the layer, multiply it by their corresponding weights, and sums up all the values. Then the network can decide which neurons to lit up on the next layer. However, summing all the values may add up to any

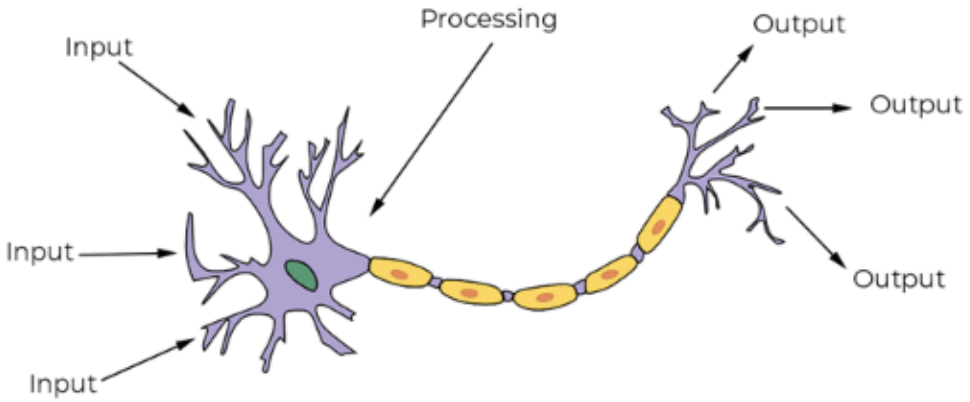


Figure 1.7: Representation of a biological neuron which consist of input, processing and output parts. [4]

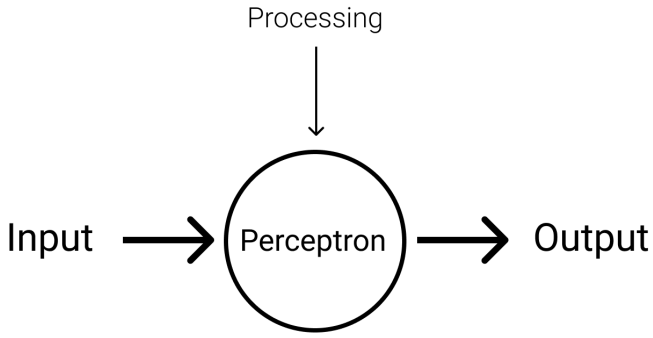


Figure 1.8: Perceptron representation that is being used in Machine Learning. Perceptron receives an input and generates an output.

value. That is why neural networks use *activation functions* to keep the weighted sum into a range. One of the common function to do this is called the sigmoid function.

The sigmoid function assigns the very negative input values to zero and very positive values to one. In the end, the activation function will let neurons have activation values between zero and one, and if the value is close to zero, the neuron will not lit up. If the value is close to one, the neuron will lit up and send information to the neuron on the next layer. So, the neurons' activation on the next layer is a measure of how positive the weighted sum of the neurons that are connected to it. This can help the network decide which neurons to fire, but it does not help with effective learning.

Neurons must send reasonable data to each other to learn from the given input. That is why the learning phase must be adjustable so that the network can find underlying relationships in the data. Moreover, calculating the weighted sum is nothing more than a matrix multiplication. This is the part where **bias** joins to the neural networks. Bias is a number that is subtracted from the weighted sum.

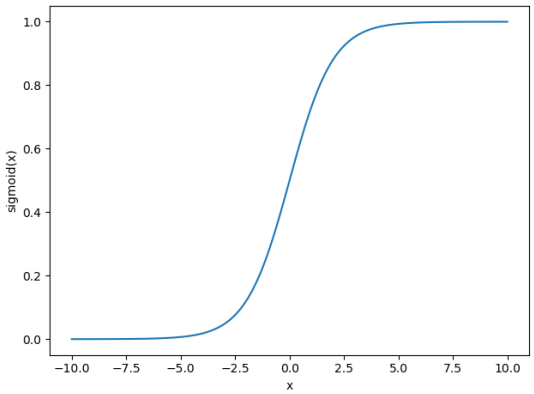


Figure 1.9: Sigmoid function converts to activation value of a neuron into range of zero to one.

This subtraction adjusts the curve of sigmoid, or any other activation function being used during the training and lets the function to fit the data better.

1.3.2 The Feed-Forward Multi-layer Neural Network

Once many perceptron come together and form a layer structure, they create a multi-layer neural network. As mentioned earlier, the perceptron tries to learn from the given data and produces an output based on that. While this is useful up to some extent, it is not enough for complex models. It is not realistic to expect one perceptron to understand all the possible relationships in the underlying data for complex tasks. That is why many neurons might be required. When many perceptrons come together, they form a layer structure, also known as the hidden layer. The neurons in the hidden layer, have their activation number, weight, and bias. They work in the same way as a single perceptron. Once the learning function is being applied, some neurons in the layer lit up and send information to the next layer. The layer structure helps the network to capture patterns in the given data. When there are more neurons in the layer, the network will have more parameters to adjust, and this will allow the network to have more possibility to send information.

1.3.3 Cost Functions and Backpropagation

When the neural network is started to be trained on the input data, it assigns weights. Weights are often randomly set numbers close to zero, which helps the learning algorithm adjust weights during the training. As stated in [9], training networks are generally iterative and challenging tasks since learning is heavily affected by the choice of weight initialization strategy. Authors also mention that modern initialization strategies are heuristic and straightforward, and designing better initialization methods is hard due to the complexity of neural network optimization. While it is possible to set weight and biases for the small tasks manually, this gets more complex for non-trivial problems. It is not realistic to rely on the output of a network that had random weights and biases. Here what network needs is an algorithm that will tell the network how it performs with the produced predictions and cost functions can provide this feedback.

Cost functions are the way to tell the network how good or bad was the produced output. After each evaluation, the cost function calculates the error rate of the predictions. This process is known as the cost of one training example. Generally, the object is to minimize the error, and that is why cost functions are often referred to as a loss function or "loss" in short. It can be interpreted as the less the cost value, the closer predictions to the correct label. On the other hand, the higher the cost, the network knows less what it is doing. Choosing the loss function is strongly related to the choice of the networks' output type and plays a crucial role for the model to learn [9]. In the end, network parameters will be adjusted after each training example to find the most suitable output based on the cost function. Additionally, once the training is over, the lost function will output a scalar value that allows the machine learning model to be ranked and compared [33].

Just grading the predictions do not help the network to learn by itself. That is why neural networks require a mechanism to adjust the initialized parameters during the training. One way is to use brute force and test all the possible ways to set the parameters and get the best model. However, this is not an optimal solution since the network parameters can grow quickly, and trying all the possible values might reach exponential time. Another way to adjust the parameters is to use the cost function.

Consider a function $y = f(x)$ where $x, y \in \mathbb{R}$. The derivative of function x gives the slope of the function at the point x. This can be interpreted as the derivative of the function also tells us how big or small change is required to get the necessary output. In short, derivative is helpful to minimize the loss function since it shows how to change x so that we can improve y. Finding the minimum loss can be imagined as a ball rolling down from a hill and the main objective is to determine which direction to take in order to reach the bottom.

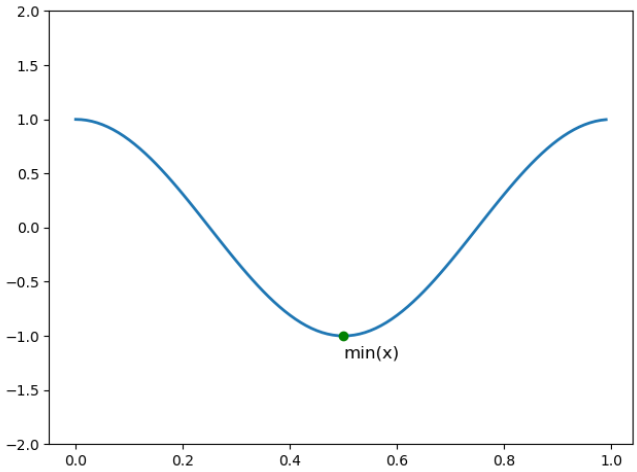


Figure 1.10: Function $f(x)$ has only one input and objective is to find the x_{min}.

The gradient of a function gives the direction of steepest ascent. It shows which direction to step to increase the function as quickly as possible. Respectively, taking the opposite sign of the gradient gives the direction of the step to decrease the function as quickly as possible. Once the gradient direction is found, one must move x in small steps down to the hill, until the minimum point where

it is impossible to decrease the f(x) any further. This point is also known as the local minimum. There can be many possible local minimums in the cost function. Depending on the starting position, one can land any of those minimums, and there is no guarantee that the landed minimum is the global minimum. The negative gradient of the cost function will then be summed up with the current weights and biases to adjust parameters for the next iteration of the training. This will let the neural network to adjust its parameters automatically and try to minimize the cost. The algorithm to compute the gradient is called backpropagation.

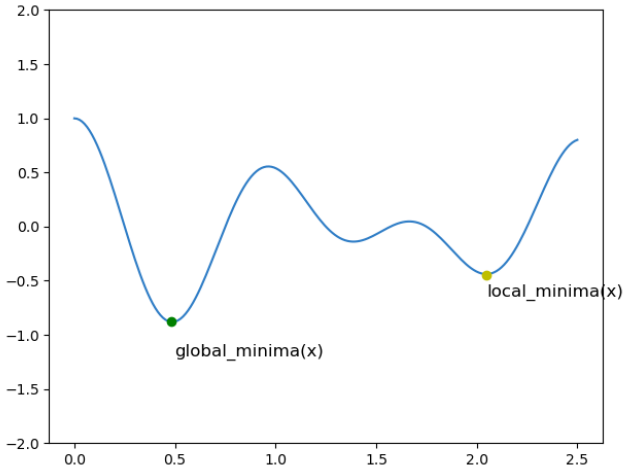


Figure 1.11: In this case function has a global and local minimum. Starting point is really important here since if the random starting point is located in the right hand side, and network is land to a local minima, it may think that it is a global minimum.

Imagine a neural network that has 12 output classes that represent the months of a year. When we want network to output the 8^{th} month for a given input, we expect the 8^{th} neuron in the output layer to have the highest activation value. However, it is not possible to directly modify the output layer since the activations of the neurons in one layer depend on the neurons' activation in previous layers. Even though it is not possible to adjust the activations of the neurons in the last layer manually, one can tweak the neurons' biases and weights on the previous layer. The network must find out the weights that have more effect on output class 8 and increase the value of those neurons while decreasing others. This could help output class 8 to has higher activation value than the other classes. Similarly, once all the critical weights are identified, the network can adjust the activation values of the corresponding neurons.

In order to have a correct prediction for class 8, weights and biases of the 8^{th} neuron in the output layer must be tweaked. However, this particular network has 11 more output classes. The main idea of Backpropagation is to apply the same strategy to find the most suitable weights for all the output classes. This process has to be done for all the training examples. Then, the network will calculate the average of all the desired adjustment values for each class throughout the training examples. Resulted vector will be the negative gradient of the cost function.

One drawback is that machine learning models generally need large training sets. When the training set grows bigger, it becomes computationally expensive

to calculate the gradient decent. **Stochastic Gradient Descent** is an extension of the gradient descent algorithm to calculate the gradient efficiently, regardless of the training data set size. At first, once must randomly divide to training data into sub batches and compute each step down to hill with respect to a mini-batch. Repeatedly going through all the mini-batches and making those adjustments network will converge one of the local minima of the cost function.

1.3.4 Vanishing and Exploding Gradients

It has been discussed that the network calculates the prediction error and by using the error, adjusts each weight (m) and bias (b) in the network so that the next iteration will be more successful. For a given cost function like Equation 1.2, it is possible to calculate the gradient by using the Equation 1.3. It is useful to have a layer structure in the neural network since it increases the capacity of the network and makes the learning phase easier for huge data sets. One problem is that, once the gradient is calculated and the network starts to tweak parameters from the last layer to first, the gradient diminishes dramatically as it is being propagated backward. Once the layer number increases the partial derivative values will become a really small value so that adjusting the new parameter for early layers will not be significant enough to improve the learning. This situation is also known as Vanishing Gradient Problem.

$$f(m,b) = \frac{1}{N}\sum_{i=1}^{n}(y_i - (mx_i + b))^2 \tag{1.2}$$

$$f'(m,b) = \begin{bmatrix} \frac{\partial f}{\partial m} \\ \\ \frac{\partial f}{\partial b} \end{bmatrix} = \begin{bmatrix} \frac{1}{N}\sum -2x_i(y_i - (mx_i + b)) \\ \\ \frac{1}{N}\sum -2(y_i - (mx_i + b)) \end{bmatrix} \tag{1.3}$$

Additionally, there might be situations where the gradient exponentially increases as it is propagated from the output layer to input. This problem is known as the Exploding Gradient.

There are many ways to can prevent the Vanishing/Exploding Gradient problem, including better random weight initialization, optimization, pre-training, or choosing a different activation function. Exploring all the possible solutions is not the scope of this paper. However, it is worth mentioning some activation functions briefly.

1.3.4.1 Hyperbolic Tangent (tanh)

tanh function is similar to sigmoid function. Only difference is that, tanh transforms the values between -1 to 1, while the range of sigmoid is between 0 to 1. Figure 1.12 shows the plot of the function.

1.3.4.2 Rectified Linear Unit (ReLU)

ReLU has a range from 0 to infinity. As it is shown in Figure 1.13 It maps all the negative values to 0 directly which affects the network ability to learn from data. Since the function is nearly linear, it protects many of the properties of linear models to optimize with gradient-based methods [9]. However, when the

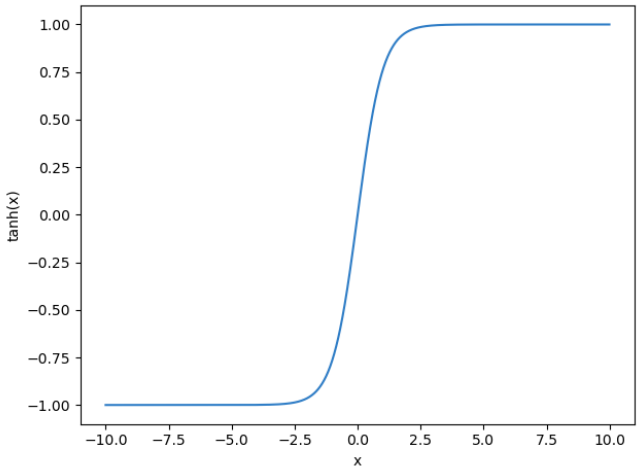


Figure 1.12: Hyperbolic Tangent will scale the input values between -1 to 1.

inputs are close to zero or negative, the gradient immediately becomes zero, and this might cause network to not learn properly.

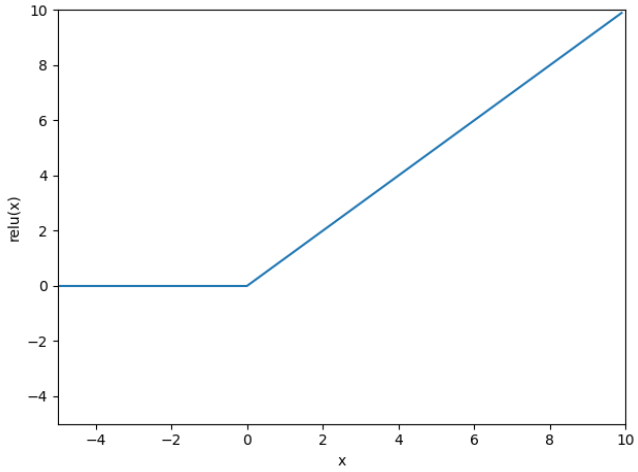


Figure 1.13: ReLU assigns zero for any negative input given to the network.

1.3.4.3 Leaky Rectified Linear Unit (Leaky ReLU)

Assigning zero for any negative values causing machine learning models to not fit the data properly. Leaky ReLU is an attempt to resolve this issue. It changes the range of function from $[0, \infty)$ to $(-\infty, \infty)$. It allows backpropagation to be done even for negative inputs and illustrated in Figure 1.14.

1.3.4.4 Softmax

Softmax is useful for output neurons in the networks and it is being used for classification purposes. It normalizes the output values for each output class

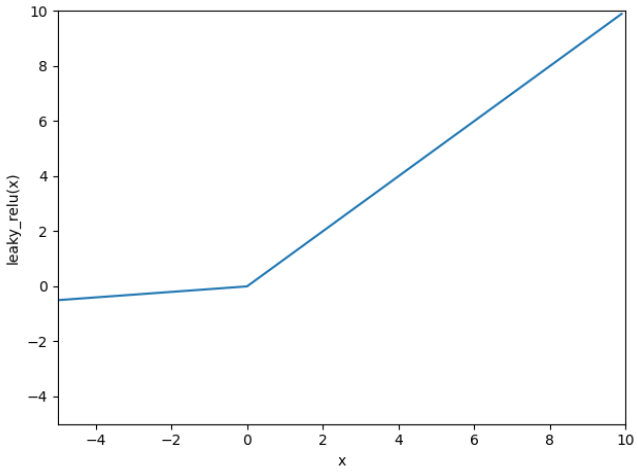


Figure 1.14: Leaky ReLU has a leaky part to increase the range of ReLU to allow more data to be fit by the network.

between 0 and 1 as shown in Figure 1.9.

$$Softmax(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{1.4}$$

1.3.5 Recurrent Neural Networks

Recurrent Neural Network (RNN) is a deep learning model that performs very well on sequential data. In general, a sequence is a sequence of data that has defined order to it. In English, the sequence could be a sequence of words in a sentence. In recommender systems, the sequence could be the items that users visited during their shopping session.

In machine learning, models map a variable of one type to variable of another type. This is also applicable for RNNs. Most importantly, RNNs can transform sequences to vector, vectors to vectors, and sequences to sequences. This allows RNNs to work on various sequence lengths. That is why RNNs are used for different machine learning tasks such as machine translation, stock prediction, recommendation engines, extracting image descriptions, and more.

1.3.5.1 Architecture

RNNs can be defined as networks with loops in them, allowing prior information to be used for predictions. Consider the neural network N which is shown in Figure 1.15. Network N gets an input x_t, and outputs a value h_t. It is possible to think RNNs are multiple copies of the same neural network, each passing information to the next one. Then, it is possibl to unroll the network in order to see detailed architecture as in Figure 1.16.

As it is mentioned, RNNs work on sequences of vectors. Figure 1.17 shows how RNNs are able to work with different input/output vectors. Working with sequences gives RNN a huge advantage comparing the networks that can only work with a fixed number of computational steps [18].

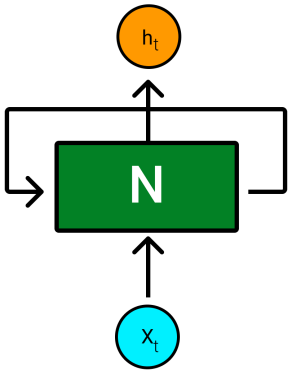


Figure 1.15: Network N looks at given input x_t which is given in time t and create an output h_t.

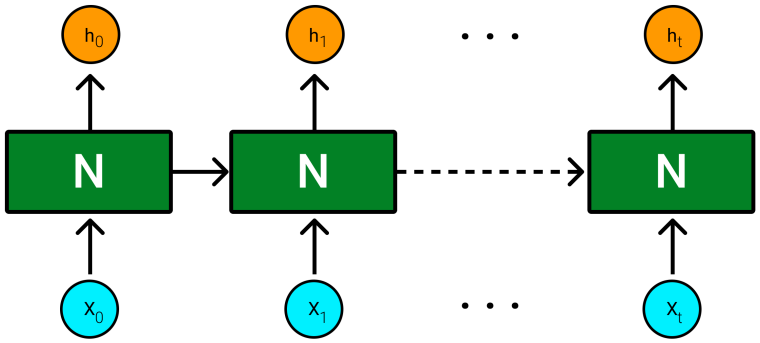


Figure 1.16: The input is given in time t is being passed throughout the sequence. The given prediction on time $t + 1$ is used the prediction information from the time t

.

Not all of the machine learning problems are based on sequential data and can often be solved using fixed input/output size. However, even for a fixed size of input/output, RNNs can be useful. Gregor at al. [10] used a recurrent neural network to generate images of digits by learning to sequentially add color to a canvas. Ba et al. [2] developed an RNN model that learns to read house numbers from left to right.

In order to explain how RNN works, let us consider an example. Imagine the RNN is given an input sentence "Sequence prediction is fun". Typically, as it is explained in Section 1.2.2, the given input data should be pre-processed and transform into a suitable vector that RNN can use. However, throughout the example, we will use the input string as-is for better readability instead of writing its vector form.

In Figure 1.18; The input node of the RNN is colored in a way that represents the information used during the prediction. Slicing of colors is randomly defined to illustrate the example, and they do not represent any significant weight distribution.

Even though the recurrent structure uses prior information, it comes with a problem. RNN has a short-term memory. As it is visible in the color distribution

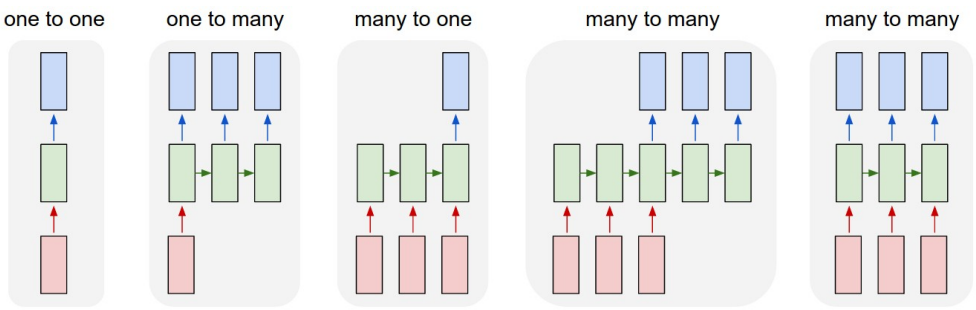


Figure 1.17: From left to right: (1) Traditional Neural Network (Fixed size of input output). (2) Sequence output (e.g. Having an image as an input and outputting the description of the image with sentences. (3) Sequence input (e.g. Sentiment Analysis; given a sentence, predicting the positive/negative sentiment. (4) Sequence input and output (e.g. Machine translation) (5) Synced sequence input and output (e.g. Video classification). [18]

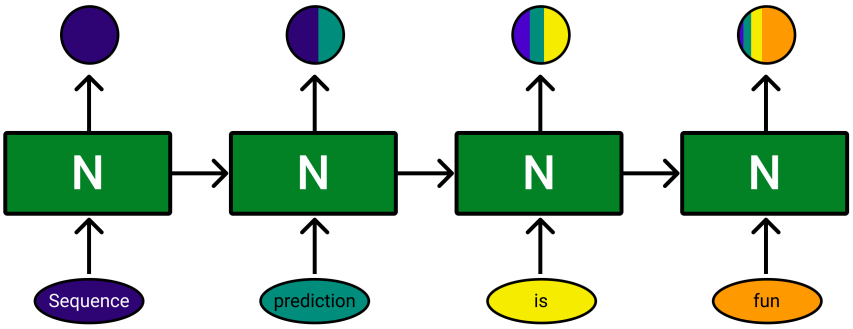


Figure 1.18: When network is given with input "Sequence", it only uses this information to make a prediction. On the next prediction, it will use the given input "prediction" as well as the "Sequence" information from the previous step.

of the final output in Figure 1.18, when the sequence gets longer, RNN will have a problem to retrieve information from previous steps. The problem of short-term memory is caused by the Vanishing Gradient Problem that was also described in section 1.3.4. In normal neural networks, the gradient is being calculated for each neuron in the network. In the case of an RNN, it is being calculated for each timestep. This is called backpropagation through time. Because of the vanishing gradient, the RNN will not be able to learn long-range relations across time stamps.

1.3.5.2 Long Short Term Memory (LSTM)

The problem of short-term memory is discovered and studied by Hochreiter under the supervision of Schmidhuber [13]. Authors come up with a solution, which is capable of learning long-term dependencies. LSTM includes three more neural network layers into the existing architecture. All layers serve for a particular task, which at the end allows RNN to ignore, forget, and filter the incoming input data while being able to remember the previous information. They are explicitly designed to avoid the long-term dependency problem by Hochreiter &

Schmidhuber [14]. Additional gates are represented in Figure 1.19

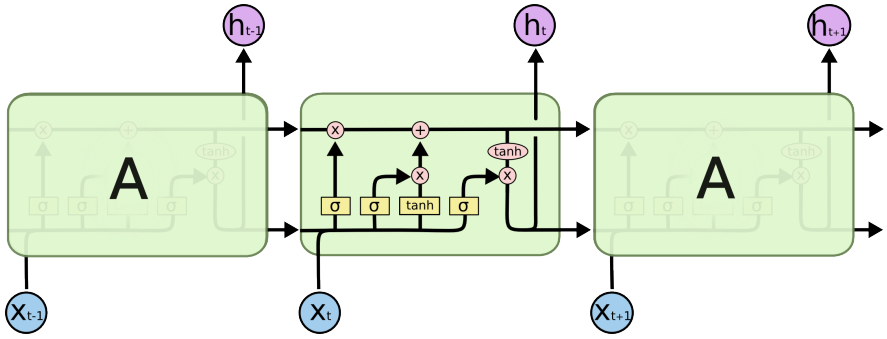


Figure 1.19: LSTM structure with additional omit, forget and filter gates [28].

LSTM has an important state called a cell state which is shown in Figure 1.20. This can be imagined as a horizontal line which lets information flow on it. The LSTM can add/remove information to the cell state by using the additional gates.

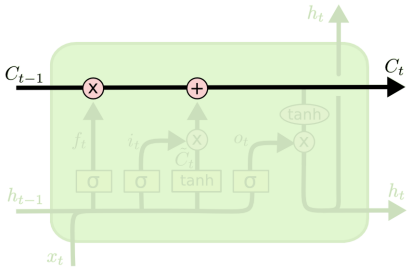


Figure 1.20: Cell State is shown on the top of the figure. It is an important state that allows information to flow and RNN can add/remove information to the cell state. [28]

The first step in the LSTM is to decide what information to throw away. Forget gate decides which data to forget by using a sigmoid function and it is one of the gates that represent RNNs' memory. Consider an input t is given to RNN. Once the input reaches to forget gate, the gate will compare the input with the previous one, $t-1$. It will apply element-wise addition and sum both input vectors. Then, it applies point wise multiplication for the vector and the threshold for gating. One can think of this as a door to let information flow. Once the input reaches here, it will be multiplied with the threshold for gating. Gating requires output to be in a range of -1 to 1 since it utilized tanh function. In the end, this multiplication will let RNN forget some information. After this point, RNN will be able to use predictions plus the memories accumulated in the forget state. Shortly, the forget state is an entirely new network that can learn which information to forget through time.

So far, RNN is able to hold information regarding past observations while choosing what to forget and remember. However, it does not know how to reset the context, meaning releasing the memory out as new predictions coming. Thus, RNN requires a filter to learn which information to keep in the cell state to use for the next prediction and which ones to release as the prediction. This

process requires another gate to be added. So far, predictions were based on the forgetting state. After adding the filter state, RNN can decide which predictions to be released as the predictions for the moment. The filter gate utilizes a tanh function, which performs element-wise addition in order to prevent values from getting bigger than 1 or smaller than -1 as shown in Figure 1.21. At this moment, RNN is able to make predictions while knowing which information to forget and which information to keep in the memory over time.

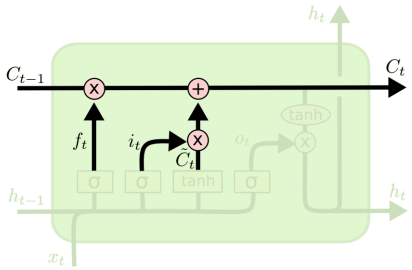


Figure 1.21: RNN decides what to output based on the information that is hold in the cell state. First, it applies a sigmoid layer to decide which part of the information on the cell state will be output. Then, the values will be transformed into a range of -1 to 1 by using tanh function. Lastly, transformed values are multiplied by the output of the sigmoid in order to decide the final output.

One problem with the above structure is that RNN does not know which possible predictions to omit. Omitting steps is important since RNN might produce the same output one after each other since it does not consider if the produced prediction was in the context in close time. This is an intention mechanism, and it lets the information that are not immediately relevant to be omitted. It has its own sigmoid function to scale the output values as well as its own gating activity.

Language translation is one of the major fields for LSTMs. Facebook has been using LSTM Network for language translation since 2017, and LSTM Network resulted in an average relative increase of 11 percent on bilingual evaluation understudy across all languages [39]. Figure 1.22 and Figure 1.23 shows the improvement of translation process.

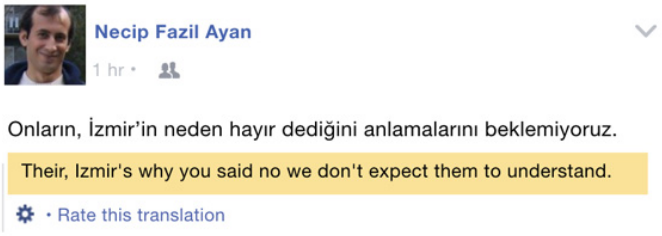


Figure 1.22: Turkish to English translation produced by a phrase-based system.

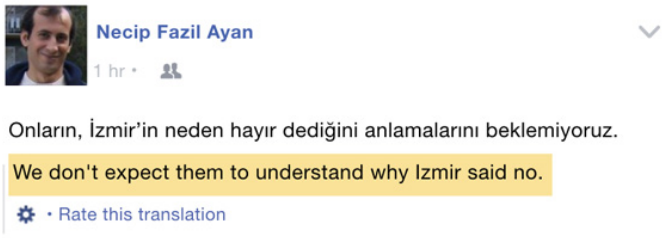


Figure 1.23: Turkish to English translation produced by a Sequence to Sequence LSTM.

Another field that LSTM is being tested and showed improvement is Flood Forecasting. A research that was conducted by Thuyloi University [19] concludes that using LSTM for one-day, two-day, and three-day flow-rate forecasting resulted in prediction ability of 99 percent, 95 percent, and 87 percent respectively.

LSTM is also actively used to create image description, recommendation system, stock prediction, and text prediction.

1.4 Evaluating a Model

Once the data pre-processing is done, the machine learning algorithm has been trained and evaluated; the next step is to decide how to quantify the model performance. Evaluation metrics and methods change based on the machine learning task.

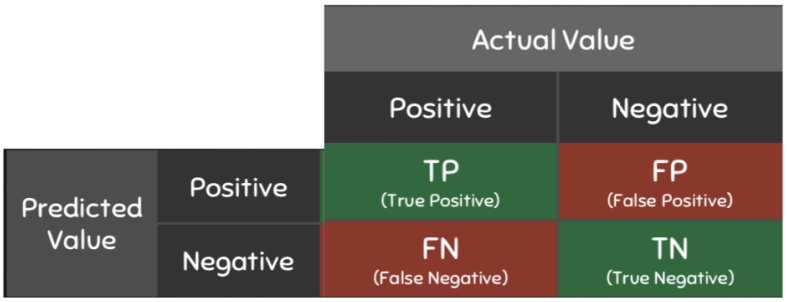
1.4.0.1 Classification

In a classification task, the performance is calculated by comparing the predicted output class with the true output class.

The final performance metric is often decided after all predictions are being compared in the entire evaluation data, One possible way to calculate the performance of the classifier is finding the percentage of correctly predicted labels over the evaluation data. This method gives the *accuracy* of the model, and it is simple to calculate. However, accuracy is not a good choice if the data is unbalanced. Consider that evaluation data contains 9 cats and 1 dog image. Even tough machine learning model classifies all images as a cat image, it will end up with 90% of accuracy which is misleading.

Traditionally, performance of a classifier is based on the counts of observations that are predicted correctly or not. Confusion matrix provides detailed information of the model not only showing the performance but also showing which classes are predicted correctly or incorrectly and what were the errors model made in case of a wrong prediction. Figure 1.24 illustrates one possible representation of a confusion matrix.

It is possible to calculate several evaluation metrics by using the confusion matrix, and each metric would describe a different perspective of the model.



- True Positive (TP) : Observation is positive, and is predicted to be positive.
- False Negative (FN) : Observation is positive, but is predicted negative.
- True Negative (TN) : Observation is negative, and is predicted to be negative.
- False Positive (FP) : Observation is negative, but is predicted positive.

Figure 1.24: The confusion matrix is useful for measuring metrics like Recall, Precision and Accuracy [23].

1.4.0.2 Accuracy

As stated, accuracy means the proportion of correct predictions made by a classifier. Calculating the accuracy is easy, and it is intuitive to understand. However, the main problem with using accuracy as the key metric is that it does not count if the data set is unbalanced, which is most of the case for real-world data.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{1.5}$$

1.4.0.3 Precision

Precision focuses on the true positive predictions and gives the proportion of positive predictions on true predictions.

$$Precision = \frac{TP}{TP + FP} \tag{1.6}$$

1.4.0.4 Recall

Recall, is also referred to as Sensitivity, is a useful measure when positive predictions are more important than others. It is possible to use the Recall to drop false-negative predictions.

$$Recall = \frac{TP}{TP + FN} \tag{1.7}$$

1.4.0.5 F1 Score

The F1 Score is a way to merge precision and recall together. It is the weighted average of the precision and recall where an F1 score reaches its best value at 1

and the worst score at 0 [40].

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{1.8}$$

1.4.0.6 Regression

The biggest difference between regression and classification is that regression works with continuous values. It means that the machine learning model, which is trained for a regression task, will produce continuous value as its prediction. Measuring the performance of a regression model is done by evaluation the difference between the predicted result ($\hat{y}$) and the actual value (y). There are also several ways to measure regression performance.

1.4.0.7 Mean Absolute Error

The mean absolute error (MAE) is the average of the absolute differences between predictions and real values. MAE tells how big of an error once can expect from the model.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{1.9}$$

1.4.0.8 Mean Squared Error

The mean squared error (MSE) is the sum of the square of the differences between predictions and real values, divided by the number of observations (n). MSE solves the issue of negative and positive values are canceling each other when they are added since it is taking the squares. However, this causes the final error measured to be squared as well.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{1.10}$$

1.4.0.9 Root Mean Squared Error

Root Mean Square Error (RMSE) solves the problem of having the final metric as square, and it converts back to its original value. RMSE gives more importance to large errors than MAE.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{1.11}$$

2. Recommender Systems

Recommender Systems (RS) is one of the most powerful machine learning algorithms used widely in E-Commerce, video-on-demand, and music stream. Recommender Systems are software tools that aim to suggest products to a user by considering their interest [34]. This suggestion relates to various decision-making processes, such as what products to buy, what music to play, or what holiday tour to select.

Recommender systems require certain feedback to perform recommendations. That is why they require information on users' past behavior, the behavior of other people, or the content information of the domain to produce predictions.

It is possible to define the workflow of a recommendation process as; collecting information, learning, and production recommendations. There are often three main ways for a recommender to collect information, which also known as feedback. One type of feedback is Explicit Feedback, which is the input of users regarding their interest in an item. This is the most helpful information since it directly comes from the user and shows their direct interest regarding the item. Implicit feedback is information produced after observing the users' behavior. Finally, Hybrid Feedback is a combination of explicit and implicit feedback. Then, the recommendation model tries to learn the customers interest based on the collected data and outputs a recommendation.

2.1 History of Recommender Systems

In the year 1992, emails started to become popular between corporates. In a short time, it became an official communication channel among colleagues. However, the number of incoming emails started to increase quickly. The Xerox Palo Alto Research Center invented Tapestry to address this issue and created the first recommendation systems [5]. The Tapestry was able to filter emails based on their topics. The important part is that email topics were created by the employees and reflecting their interest. Other than that, Tapestry was estimating the importance of the email by checking emails' popularity. These two concepts were the start of content base recommendation and collaborative filtering in recommender systems.

Sometime after Tapestry was released, an MIT research team called GroupLens [1] investigated the core idea of the application and developed a new recommender system that can be used in newsgroups. Newsgroups were the places that people can discuss and exchange information, and it became prevalent. One significant difference between Tapestry and GroupLens is that Tapestry was company-specific. On the other hand, GroupLens' algorithm was being used in multiple websites. As a result of this, the recommendation algorithm was able to predict user behavior more accurately.

Starting with the 21st century, the internet became more popular. Companies started to open their online shops for people to purchase products without going to actual stores. This helped customers to shop easily however, it also created a problem. Psychologists Sheena Iyengar and Mark Lepper explain this problem

[1] `https://grouplens.org/`

as The Paradox of Choice. [16] According to the theory, even though presenting multiple choices to people looks reasonable, it may cause them to feel overload and confused as shown in Figure 2.1. In physical shops, the shop owner pre-filter the products since the storage capacity is limited. This filtering often based on popular items or new productions. Online shops have unlimited capacity, and companies can put all of their products into an online store without applying any filter. However, it might not be a good idea to put a million items in front of the customers.

Figure 2.1: Iyengar & Lepper et al. performed an experiment by offering tasting of 6 or 24 flavours of jam for free. 40% of the customers who had 6 choices, tried the jams, and 30% made a purchase. When customers are presented with 26 options, 60% tried the jams. However, only 3% made a purchase [42].

Currently, customers can find popular items both in online and physical stores. Less popular items are often present online, and this creates a phenomenon called Long Tail Effect as shown in Figure 2.2. While popular items are quickly accessible, the rest of the categories require filtering. Finding less popular items would require a long time if companies like Amazon gives their entire product inventory to users directly with no filter. With the help of recommender systems, companies try to identify user interest and apply pre-filters. An example for Amazon Recommender System is shown in the Figure 2.3.

2.2 Type of Feedback for Recommendation Systems

2.2.1 Explicit Feedback

To collect explicit feedback from the user, the system must ask users to provide their ratings for items. After collecting the feedback, the system knows how relevant or similar an item is to users' preferences.

Even though this allows the recommender to learn the users exact opinion, since it requires direct participation from the user, it is often not easy to collect. That is why there are different ways to collect feedback from users. Implementing

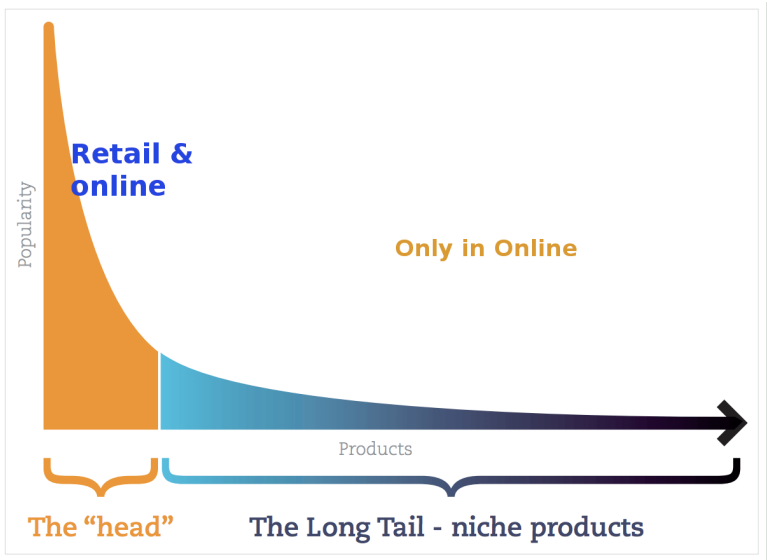


Figure 2.2: The Long Tail graph represents the items with their popularity on the X-Axis. Popularity is often measured as the amount of total purchase or the total times an item is viewed. While the most popular items will be located on the head, the less popular ones will be on the right side of the axis [31].

a like/dislike functionality into a web site, gives users to evaluate the content easily. Alternatively, the system can ask users to insert their ratings in which a discrete numeric scale represents how the user liked/disliked the content. Netflix often asks customers to rate movies as shown in Figure 2.4. Another way to collect explicit feedback is to ask users to insert their comments as text. While this is a great way to learn user opinion, it is usually not easy to obtain and evaluate.

2.2.2 Implicit Feedback

There is no user participation required to gather implicit feedback, unlike the explicit feedback. The system automatically tracks users' preferences by monitoring the performed actions, such as which item they visited, where they clicked, which items they purchased, or how long they stayed on a web page. One must find the correct actions to track based on the domain that the recommender system operates on. Another advantage of implicit feedback is that it reduces the cold start problems [37] that occur until an item is rated enough to be served as a recommendation.

2.2.3 Hybrid Feedback

Hybrid Feedback uses both explicit and implicit feedback to maximize prediction quality. To use the hybrid method, the system must be able to collect explicit and implicit feedback from users.

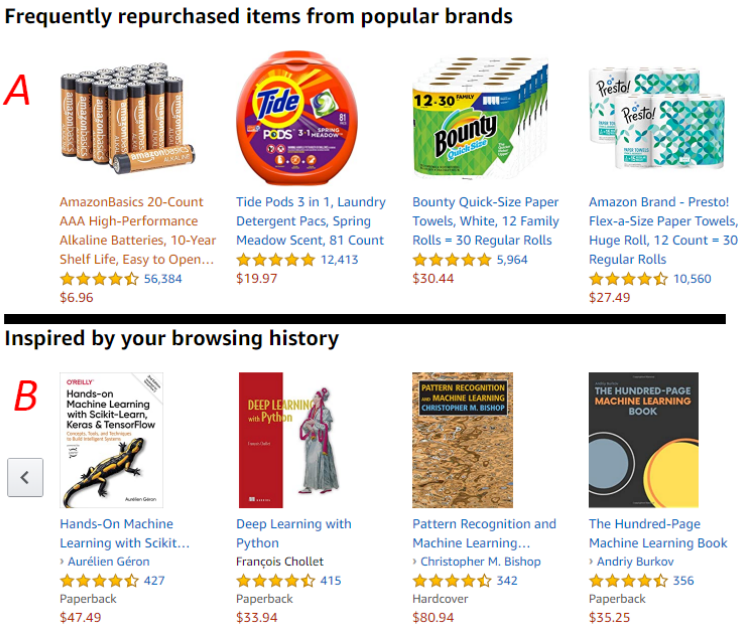


Figure 2.3: Amazon recommendation engine shows the popular items based on their repurchase counts as shown in A. If user is logged in and have a session history, recommendation engine also gives suggestions based on the users previous shopping sessions as in B.

2.3 Type of Information Filtering Techniques

Once the system collects user's actions, one must decide an efficient and accurate way to produce recommendations. Isinkaye et al. [15] shows the overview of different recommendation filtering techniques as in Figure 2.5.

2.3.1 Content-Based Filtering

Content-Base Filtering algorithm recommends similar products to those that the user already evaluated in the past. If a Netflix user liked the movie Lord of the Rings: The Fellowship of the Ring, then the recommended movies might have similar genre like fantastic-action.

Content-Base systems save user profiles automatically once there is an interaction between the user and the item. The system is aware which user liked/disliked an item, which ones made a purchase, or rate an item. This information often kept in a vector form and known as a *profile vector*. On the other hand, the system must have another vector which contains information regarding the items in the domain. This vector is usually called an *item vector*. Figure 2.6 shown an example of a movie recommended to a user, based on a movie she watched.

Once the algorithm has both non-zero vectors, it finds the cosine of the angle between the profile vector and the item vector. This calculation is also known as **cosine similarity**. Consider the profile vector as P and item vector as I. Then,

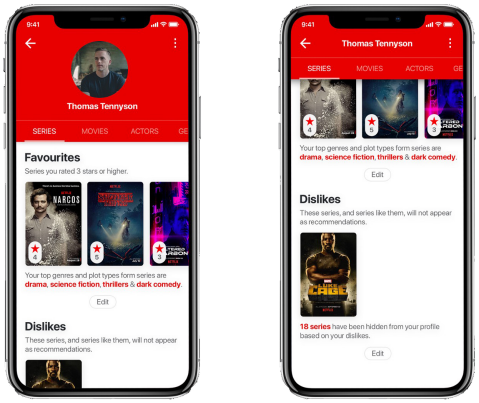


Figure 2.4: Netflix collects explicit feedback from its users by asking them to rate the movies/series they watch. Users are able to rate the items by giving integer values from 0 to 5 which indicates how much they disliked/liked the item [43].

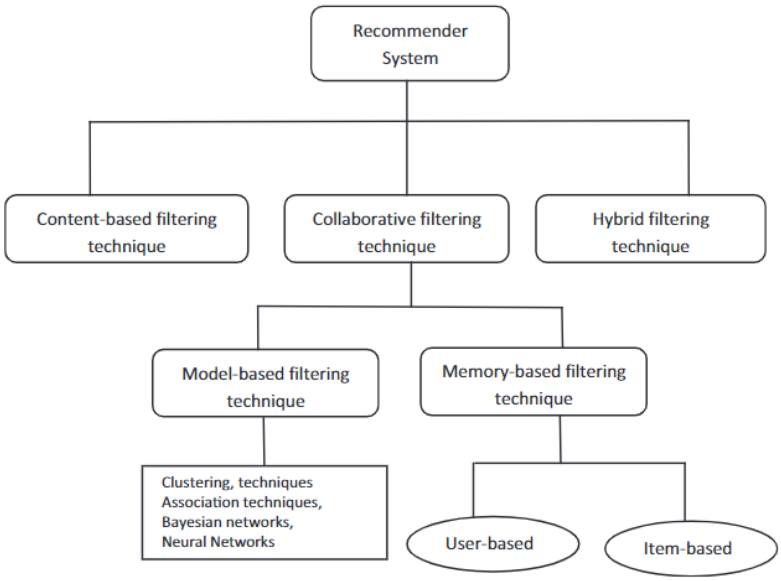


Figure 2.5: Anatomy of Recommendation filtering methods as described by [15].

one can calculate the cosine similarity by using the Formula 2.1.

$$sim(P, I) = \cos\theta = \frac{P \cdot I}{||P|| \times ||I||} \tag{2.1}$$

Calculated cosine similarity will produce a number between -1 to 1, which indicates how similar an item to the one which the user liked. While some systems prefer to put a threshold and recommend all the items above the limit, others recommend only the *best n* items where n often decided based on the domain.

Cosine Similarity is not the only way of finding similar items. When the similar items represented close to each other in multi-dimensional space, it is possible to calculate the distance between the items. This distance can be interpreted as the more the distance between two items, the more they are different. This

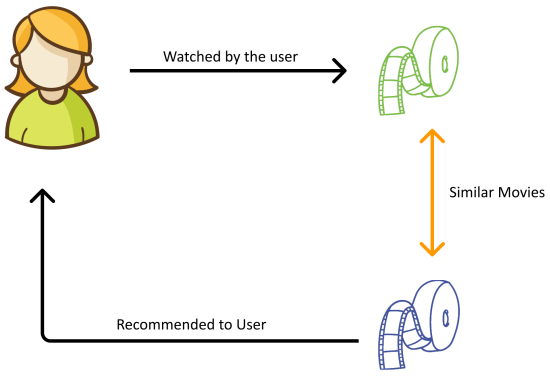


Figure 2.6: In CF, once user watches a movie, the system finds the similar movies to the one user watched. Then, the similar items is recommended to the user.

calculation is also known as **Euclidean Distance**.

$$EuclideanDistance = \sqrt{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2} \tag{2.2}$$

Another way to calculate the similarity between items to find the correlation by using the Pearson Correlation Coefficient. The higher the correlation, the more similar the items each other. Consider the *item* u and *item* v, then one can calculate the correlation between two items by using the Equation 2.3

$$sim(u, v) = \frac{\sum(r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum(r_{ui} - \bar{r}_u)^2}\sqrt{\sum(r_{vi} - \bar{r}_u)^2}} \tag{2.3}$$

Even though there are multiple ways to find similarity, there is one major drawback of using the Content-Base filtering algorithm. The algorithm only considers the content of the products, and it will only produce recommendations similar to the items user rated/liked/bought. If the user did not watch any other movie other than the Lord of the Rings: The Fellowship of the Ring, the recommendation will not be more than fantastic-action movies. Since the only source of information comes from the items, the algorithm does not know the relationships between users and will not be able to understand users' behavior. This problem is known as shallow content analysis and is described in [35].

2.3.2 Collaborative Filtering

Collaborative filtering (CF) aims to produce recommendations by using the interests of similar users. It is something people have been doing for a long time. For example, person A watches a movie and thinks that her/his friend might also like it. So, person A goes and recommends the movie to the person B. Important point here is that the more person A knows about person B, the more accurate suggestions she/he can make. Collaborative filtering is one of the most commonly used algorithms in the industry as it is not dependent on any additional information rather than user actions [21]

One major drawback of Collaborative Filtering is that it is affected by *Cold Start Problem (CSP)*. The recommendation algorithm cannot calculate the users'

interest if the user has no interaction with the system. Similarly, the CF algorithm will perform poor recommendations if there are not enough similar users in the system.

One of the advantages of collaborative filtering is that it can be used to find not only the similar users but also similar items.

2.3.2.1 User-User Collaborative Filtering

The algorithm tries to find similar users by considering past actions, such as the same movies they have rated. The system finds the K nearest users based on the similarity of past interactions where K is the maximum number of similar users to be found. Then the user is shown with recommended items that were rated by similar users. Figure 2.7 gives an example of a recommendation by using user-user collaborative filtering.

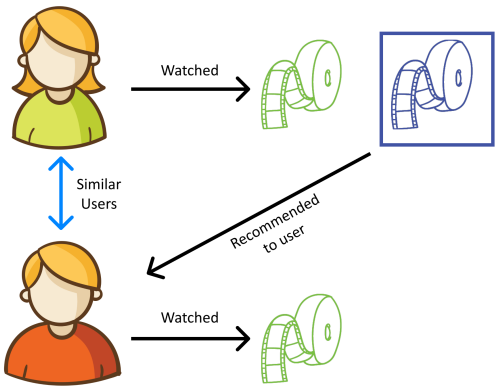


Figure 2.7: Recommending a movie to a user by using the User-User Collaborative Filtering

2.3.2.2 Item-Item Collaborative Filtering

Item-Item Collaborative Filtering is similar to User-User. The difference is that the system recommends similar items that are liked by the users.

The system first finds similar items based on the user ratings and then predicts user ratings based on the similar ones that the user liked/purchased/rated.

2.3.3 Hybrid Filtering

As it is discussed, Content-Based and Collaborative Filtering have their advantages and disadvantages. It is possible to combine both solutions as it is shown in Figure 2.8 to increase the efficiency of the recommendations. This approach is called as Hybrid Filtering. Cano et al. conduct an extensive study and reviewed 76 hybrid recommendation studies in the literature [47] and mentioned the most common problems and their possible solution researches had while using hybrid recommendation.

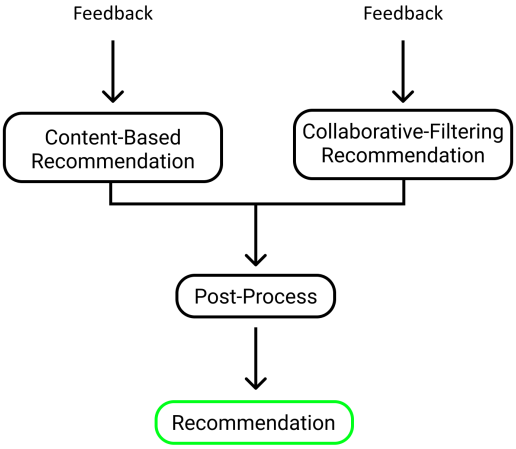


Figure 2.8: One of the possible ways to apply hybrid filtering, is to have two recommendation algorithm, one for CB and the other one for CF recommendations. Result of the both recommenders are combined and the final recommendation is being shown to the user.

2.3.4 Sequence Aware Recommendation Systems

Matrix completion is one of the most focused topics in academic researches where one tries to predict a rating that the user would give to an item. One advantage of this methodology is that it can capture the long-term interest of the user. However, when it comes to capturing the short-term interest, this method usually fails. Additionally, the user-item rating matrix does not use external information like user actions.

It can be a success factor to consider short-term user interest in real life. Since online shopping is often available for many people, it is not realistic to expect customers to have the same shopping patterns. User interest tends to change quickly, and this change does not necessarily follow a pattern. A person can order an electronic device in the first Amazon order, then buy a plant for a garden in the second purchase. While it might be a good idea to consider the first purchase of the user for the upcoming predictions, recommending electronic devices to a person who will buy a plant can be considered a bad recommendation.

Additionally, using past purchase history or user-specific data is not applicable if the user is anonymous. It is unlikely to capture the long-term interest of anonymous users since they are not logged in the system and do not share any personal information. Once the business domain has high-density of anonymous users or user interest tends to change quickly (e.g., in every shopping session), Sequence Aware Recommendation Systems (SARS) can be useful. In SARS, users are defined as their current behavior and not by their ids. That is why the system can produce recommendations for anonymous users and short user sequences.

Sequence Aware Recommenders uses ordered user actions as input. These actions might have timestamps or just follow a sequential order, representing the flow of the user session. Actions are often defined based on the business domain and might contain attributes like user id, timestamp, type of implicit feedback (item-click, item-view, etc..) and item metadata. In short, input data can be

concluded as a variation of complex user feedback and content-based information unique to a domain. Such a form of complex user feedback can be collected using web server logging, which is doable even for small e-commerce businesses.

In general, the output of SARS is a ranked list of recommended items. However, the context of the list can change. Based on the used implementation method and the input, Sequence Aware Recommenders can produce a variety of recommendations such as next user action, next item, next sequence, or next basket. The crucial point is that, since the system uses the sequence information, it aims to understand the users interest in the current session and make recommendations based on that. This strategy is useful for small e-commerce businesses to apply without considering user ratings and direct participation.

Quadrana et al. [32] identified three main algorithms to work with sequential user logs as sequence learning, sequence-aware matrix factorization, and hybrids. For the sake of the thesis, we will only focus on sequence learning and its relation with Recurrent Neural Networks. As said, SARS use ordered action logs as input. Action logs represent discrete events that often have a timestamp, also known as time-series events. The time interval between events is an essential factor in the e-commerce domain, and it carries useful information to understand the customers' behavior. Once the timestamped user actions are ordered based on the time, it is possible to simulate user behavior during the session and capture the customers intent. Understanding the user's short-term interest is significantly vital for small businesses like travel agencies where customers leave the session early.

Sequence modeling methods for sequence-aware recommendations are represented by [32] as three categories: Markov Models, Reinforcement Learning, and Recurrent Neural Networks, where the authors examined the individual methods, including their application examples. The approach which is related to this thesis is the Recurrent Neural Networks. Thus, we will talk about the recurrent neural network usage in the recommender systems.

2.3.5 Recurrent Neural Networks in Recommender Systems

There are multiple ways to use Recurrent Neural Networks for the recommender systems. One can implement an algorithm to predict the next item in the sequence by feeding only the last user action to the network. Another possibility is to use the last n user actions to predict the next item in the sequence. In short, it is possible to use n number of observations to produce a prediction for the next observation or observations. This is possible to the architectural design of RNN which is shown in Figure 1.17.

Zhang et al. [45] trained an RNN to predict the next user click based on their given last-click data in each timestamp. A different model which is proposed by Hidasi et al. [12] used Gated Recurrent Units (GRUs) [6], (a special type of RNN) to predict the next item in a sequence by using another loss functions than [45]. Hidasi et al. [11] investigated to leverage item features like pictures, text descriptions to model the user session. Their proposed solution improved recommendation accuracy.

2.4 Evaluating Recommender Systems

Earlier many of the recommendation algorithms evaluated based on their accurate predictions. However, this was not the best approach to evaluate recommender systems. Even though predicting users' exact needs is crucial, it is not enough in most of the cases. As Steve Jobs once said, "A lot of times, people don't know what they want until you show it to them" [1997], and that is why recommendation systems changed their perspective from being the most accurate engine to the best engine which allows people to discover more. Platforms like Netflix, Amazon, or Spotify use their recommendation engine to not only predict the specific interest of the user but also allows them to discover new taste. Due to its social side, humans tend to explore new things. A decade ago, this type of recommendation was coming from family, friends, or colleagues. Lately, recommendations systems take this responsibility and have been trying to show people different recommendations based on their profile to find new interests. This strategy is a win-win for both commercial applications and users. While customers expand their horizons with new interests, companies are earning more from sales.

This section will discuss the evaluation methods of recommender systems using three different strategies; offline, online, and user studies, as well as describing the evaluation metrics.

One of the easiest ways to evaluate a recommender engine is to use offline testing. Offline testing is applied to the existing data set, and the model is being evaluated by using performance metrics such as prediction accuracy. Another method is conducting an experiment on real users in live environments. Finally, it is also possible to start a user study, where a group of people is being asked to test the proposed solution and answering a questionnaire after the experiment.

While measuring the performance of the recommendation algorithms have different metrics, accuracy is mostly used for academic research. However, researches are not completely sure that if accuracy directly reflects the user intention. That is why, new researches focus on metrics like coverage, novelty, diversity, Discounted cumulative gain (DCG), Normalized DCG (nDCG) and so on.

2.4.1 Offline Evaluation

Offline evaluations are done by using historical data that contains user actions. The main idea of the offline evaluation is to simulate the user interactions with the recommender system. Performing offline evaluation has an advantage since it does not require real-time user interaction. That is why it is possible to test multiple recommender algorithms by using the same data. On the other hand, offline tests only show the user behavior when the time data was created. User behavior tends to change, and performing offline evaluation often will not give proper information on how recommendations fit users' interest in general. That is why the offline evaluation results should not be seen as the real-life performance of the recommendation algorithm.

One possible use case to perform offline evaluation is to find the best possible combination of parameters by tuning the algorithms and picking the best one.

2.4.1.1 Simulating User Interaction

There are multiple ways to simulate user behavior. One can gather a particular user session and the items in it. Then for each item in the session, the algorithm produces a list of ranked recommendations. It is possible to see the predicted items and the item that were clicked by the user. In this method, the algorithm receives one item at a time and creates a recommendation. Another possibility is that dividing the number of items in the session to individual bins and feeding the algorithm for each bin to predict the items in the next bin.

Finding a way to simulate user behavior often decided based on a domain. While predicting the next item in the user session might be useful for e-commerce websites, to create a recommended playlist in Spotify, the algorithm might need to consider more than one song in the user history.

2.4.2 Online Evaluation

Online evaluation is one of the best way of seeing user interactions with the recommendation engine. The real-life performance of the recommendation system depends on a variety of factors as described by [38]. During the online evaluation, real users interact with the systems. So it is possible to understand the correct user intent and the success of the recommendation model directly. At first, subset of the total users is separated as test group. This separation is generally random so that both parties will be similar to each other. While test users are being shown the recommendation from the new algorithm, the others will continue to interact with the current recommendation system. It is crucial that there must be only one change in the new algorithm to understand the effect of the applied settings clearly. If the test case is to measure the recommendation accuracy, there should not be changes on the user interface of the web page. There must be a certain amount of time between changes so that tester can understand if the proposed algorithm improves or worsens the recommendation quality. Even though online evaluations give the most accurate success rate of the proposed recommendation system, they are risky to be applied. If the proposed algorithm produces irrelevant recommendations, it might cause test users to leave the website and eventually reduce the income of the business.

Applying online evaluation will give the direct performance of the recommendation system since real users test it. However, it might cause businesses to lose customers in case of poor recommendations. That is why online evaluation must be done after fine-tuning the algorithms by using offline testing and selecting the optimum algorithm.

2.4.3 Performance Metrics

As stated before, accuracy is not the only metric to evaluate the performance of the recommendations. When it comes the real life scenarios, algorithms must consider user needs. It is a debate that if recommendations only consider the products close to each other or also give customers some off topic choices as well. Even though it mostly depends on the domain, in order to keep users in the online shop and let them discover new possibilities recommendations should

show different items as well. When this is the case it is not completely possible to evaluate the performance by using accuracy only.

One can use other metrics like Coverage, Novelty, Diversity, MRR, DCG or nDCG to evaluate the performance of the recommendations.

2.4.3.1 Coverage

Coverage is the percentage of the possible recommendations that an implemented recommendation algorithm can produce.

Coverage is essential from the perspective of the new item being recommended. It gives an idea of how quickly a new item will appear in the recommendation list. New items will reduce the coverage metric since they need to be purchased/graded by at least a couple of people.

Based on the used recommendation technology, the list of the items that the system can produce prediction varies. If the system uses only collaborative filtering, items must have ratings higher than the decided threshold. Similarly, some systems may apply filters, and only items that satisfy the given filters can be used for prediction. One can donate the list of items that can be used for the recommendation as I_s to indicate a subset of possible items in the inventory. Then, coverage can be easily calculated by dividing the possible items by the total number of items.

$$coverage = \frac{I_s}{I}$$

2.4.3.2 Diversity

Diversity is a measure of how your recommendations are different from each other. Consider the customer finished watching the first movie of a trilogy on Netflix. Low diversity recommender would recommend only the next parts of the trilogy or the films directed by the same director. On the other hand, high diversity can be achieved by recommending items completely random.

Diversity may seem like a subjective measure, but it can be calculated by using the similarity between recommended items. One can calculate the similarity between the recommended item and get the average of the similarity score. Consider the average similarity score of the top 10 recommendations as sim_{10}. Diversity can be concluded as the opposite of the average similarity score, and it can be calculated as the following formula:

$$diversity(Top@10) = 1 - sim_{10}$$

Considering only diversity as the measure of quality can be misleading. That is why it should be used as a supplement along side other measurements.

2.4.3.3 Novelty

Novelty is a measure of how popular the items system recommending. Novelty may be similar to diversity. One can achieve high novelty by randomly recommending items since most of the items are not in the popular item list. Even

though it is possible to calculate novelty, it is hard to use as a solid metric. Customers often want to see related products in their recommendations. This bring user trust to the recommender system. If the recommender systems only recommends items that do not reflect user interest, it might damage the user trust and cause customers to leave the online store.

System should keep the balance of the items that user might like and items that user do not know but could discover in the recommendation list. It is an important topic since while familiar items will let customers to trust the recommender system, unfamiliar item will let them discover new things.

Achieving good novelty score can help with Long Tail Problem and that is why it is an important metric to keep track of.

2.4.3.4 Mean Reciprocal Rank (MRR)

MRR is a metric that returns a ranked list of predictions for a given query. For a single list, the reciprocal rank is $\frac{1}{rank}$, where rank is the position of the most correctly predicted item. Unless there exists no correct prediction, then the reciprocal rank is returned 0. When there are multi queries Q, the Mean Reciprocal Rank will be the mean of all reciprocal ranks.

$$MRR = \frac{1}{Q} \sum_{n=1}^{Q} \frac{1}{rank_i} \tag{2.4}$$

2.4.3.5 Discounted Cumulative Gain (DCG)

Recommendation algorithms produces a list of ordered recommendations. It is possible to judge the result based on the ordered recommendations. Jarvelin et al. mentions that there are two distinct cases while exploring a ranked list of recommendation results [17]:

1. Highly similar items are more valuable than marginally similar items, and
2. The greater the ranked position of a similar item (of any relevance level) the less valuable it is for the user, because the less likely it is that the user will examine the item.

It is possible to grade recommended items by their relevance. Consider relevance scores from 0 - 3 (0 is the lowest and 3 is the highest relevance). One can replace the recommended item by their relevance value each having a relevance score 0, 1, 2 or 3. Consider the top 5 recommendations produced by the system as R and corresponding relevance scores as G:

$$R = < TourID25, \quad TourID30, \quad TourID12, \quad TourID8, \quad TourID54 >$$
$$G = < 3, \quad 3, \quad 2, \quad 0, \quad 1 >$$

Cumulative Gain (CG) can be calculated by summing the relevance scores in the recommendation list:

$$\sum_{i=1}^{n} relevance_i$$

Once the cumulative gain formulate is applied to vector G, we can obtain the CG of the recommendation list. In this case. the total CG score of vector is

$$3 + 3 + 2 + 0 + 1 = 9$$

Additionally, one can create a vector of gains and receive the cumulative gain in the given position. Consider G' is a vector of cumulative gains:

$$G = < 3, 3, 2, 0, 1 >$$
$$G^{'} = < 3, 6, 8, 8, 9 >$$

It is possible to get CG at position i by using the value from the vector $G^{'}$ directly. As an example; $CG(3) = 8$

Major drawback with calculating the GC is the ordering. One can end up having same CG score by using different set of recommended items. For example;

$R_1 = < TourID25, \quad TourID30, \quad TourID12, \quad TourID8 \quad , TourID54 >$
$G_1 = < 3, \quad 3, \quad 2, \quad 0, \quad 1 >$
$G_1^{'} = 9$
$R_2 = < TourID1, \quad TourID15, \quad TourID33, \quad TourID48, \quad TourID11 >$
$G_2 = < 2, \quad 2, \quad 3, \quad 1, \quad 1 >$
$G_2^{'} = 9$

Even though R_1 is able to recommend two most relevant items in the first two recommendation it has the same CG score as R_2. If we only consider CG, we can conclude that R_1 is equally good as R_2 which is not completely correct.

In order to fix this problem, position of the recommended item must be included in the formula. Then the calculation follows dividing the relevance score of the item by the log of the respective position. This is known as Discounted Cumulative Gain and it penalizes the recommender algorithm for listing the most relevant items in lower ranks.

$$DCG = \sum_{i=1}^{n} \frac{relevance_i}{\log_2(i+1)}$$

2.4.3.6 Normalized Discounted Cumulative Gain (nDCG)

Even though DCG considers the position of the recommended item, it still does not give an lower / upper bound for averaging the scores of all recommendation lists. $nDCG$ can be calculated by $\frac{DCG}{iDCG}$ and is used to calculate the CG of the recommendation list with an ideal ordering. Consider the list of recommendation R_2. Then, $iDCG$ for R_2 would be:

$$R_2 = < TourID1, \quad TourID15, \quad TourID33, \quad TourID48, \quad TourID11 >$$
$$G_2 = < 2, \quad 2, \quad 3, \quad 1, \quad 1 >$$
$$iR_2 = < TourID33, \quad TourID1, \quad TourID15, \quad TourID48, \quad TourID11 >$$
$$iG_2 = < 3, \quad 2, \quad 2, \quad 1, \quad 1 >$$

Doing this would give a ratio between 0 - 1 which can be used as final result to evaluate the recommendation performance.

3. Dataset

This section will describe the dataset used in the thesis. The dataset belongs to a medium-size Czech Travel Agency called SLAN Tour [1]. Agency offers various types of tours from seasonal holiday destinations to spa tours or sports events. Customers can search for tours by using the search bar or navigating to the related category page. There exists a dedicated tour page per tour, which includes detailed information regarding the offer. Users can purchase the offered tour by using the purchase button located on the dedicated tour page. The website gets approximately 500 - 1000 users each day and makes several purchases daily. Figure 3.1 presents the screenshots of the website in June 2020.

Figure 3.1: Screenshots of the SLAN Tour Travel Agency, A: Website Homepage, B: Category Page with the list of tours, C: Dedicated page for a specific tour. Description of the content is highlighted by orange color.

The dataset is an updated version of the dataset used in [30]. While overall structure of the data stays same, there are few new columns in the one we used for the research. Additionally, we have used different pre-processing comparing to the old version. In order to make dataset details clear, we will only describe the columns which are used during the research.

3.1 Travel Agency Domain

Companies like Amazon sell thousands of products in a month. According to [26], on average, small and medium-sized businesses located in the US sell more than 4000 items per minute. However, this is not the case for the travel domain. Customers visit consumable goods or electronic websites more frequently than travel agencies. Buying a travel package is a rare event compared to purchasing a kitchen appliance. That is why the number of purchases in the travel domain is significantly lower than the other domains like electronics. Additionally, the time interval between the users' first visit and the second one might be very long comparing the other fields. Browser cookies to collect implicit feedback might

[1] `http://www.slantour.cz/`

have been deleted, or the user might change their device. All of this makes capturing the user intention harder.

SLAN Tour offers holiday *tours* to its customers. It is possible to have the same tour on the website more than ones. If one tour exists multiple times, that means it is a *tour series*, and each tour in the series has small changes such as price, departure, and arrival dates. That is why the tour series can be up for years, while tours will be obsolete after the departure date expires. Due to having multiple tours in a tour series, it is also possible to have different price tags for each tour. The main reason for having different price tags is the time of the tour (summer holiday destination often cheaper in winter or spring season), selected accommodation type, additional meals, discounts, if any, etc. Figure 3.2 shows the price information for a tour.

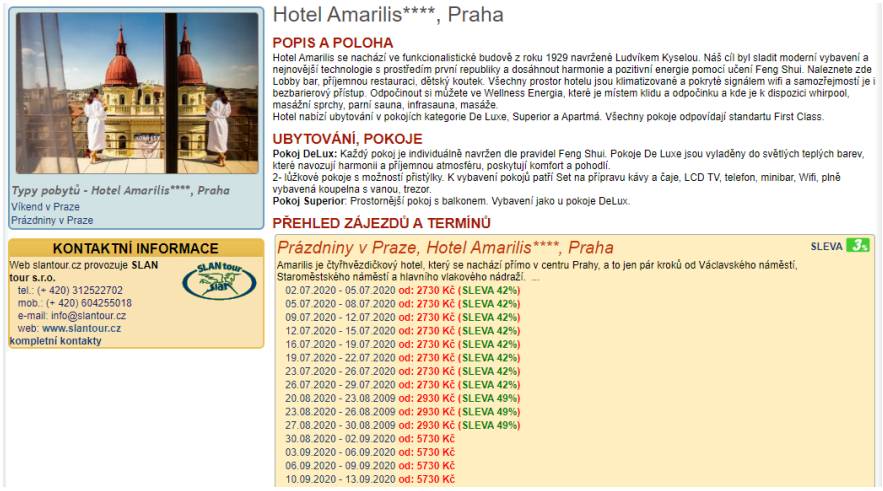


Figure 3.2: Prague Tour is offered multiple times with different price tags. Note that some of the offers has discount.

3.1.1 User Interactions

The dataset contains user interactions from September 2014 until January 2019. Each action is registered to the database one by one with a respective timestamp. Whenever a user clicks on tour, the interaction will be inserted into the database, as shown in Table 3.1. By using this information, it is possible to find entire user sessions after aggregating individual actions like represented in Table 3.2.

user_id	visited_item	session_id	action_started	action_finished
1000	0	1	2019-06-12 17:40:07	2019-06-12 17:40:11
1000	0	1	2019-06-12 17:40:12	2019-06-12 17:42:36
1000	2067	1	2019-06-12 17:42:38	2019-06-12 17:43:06
1000	387	1	2019-06-12 17:43:07	2019-06-12 17:43:11
1000	388	1	2019-06-12 17:43:12	2019-06-12 17:43:55

Table 3.1: All five pages visited by users are inserted into database with their click timestamps.

user_id	session_id	items_visited
1000	1	[0, 0, 2067, 387, 388]

Table 3.2: It is possible to aggregate items by their timestamps into a list for further analysis.

3.2 Dataset Details

3.2.1 Implicit Feedback (IF) Details

IF Table includes all the user actions. However, not all information is useful for the purpose of the thesis, thus some user actions are removed from the data. If user session contains only one tour visit, it is not possible to train and evaluate an algorithm which will predict next tour in the session. That is why, all the user session which have less than two visited pages are removed from the dataset. The rest of the explanations and analysis will cover only the part which is used to train and evaluate the proposed recommendation algorithms. Before going into more details, we will describe the IF dataset.

The dataset uses MySQL semantics and can be queried by using SQL. The name of the implicit feedback table is *implicit_user_feedback* and its columns described in Table 3.3.

ID	Attribute Name	Description
if_0	visitID	Identificaton of a user visit.
if_1	userID	Identification of a customer.
if_2	objectID	Identification of an individual tour.
if_3	sessionID	Identification of the user session.
if_4	pageID	Identification of the visited page.
if_5	pageType	Type of the visited page (e.g. Index Page, Catalog Page, Dedicated Tour Page etc).
if_6	windowSizeX	Horizontal dimension of the user screen.
if_7	windowSizeY	Vertical dimension of the user screen.
if_8	pageSizeX	Horizontal dimension of the visited page.
if_9	pageSizeY	Vertical dimension of the visited page.
if_{10}	objectsListed	Tours listed in the visited page.
if_{11}	startDatetime	Start time of the user visit on the page.
if_{12}	endDatetime	End time of the user visit on the page.
if_{13}	logFile	Log of user actions.

Table 3.3: Column names and description for the implicit_user_feedback table.

3.2.2 Description of Implicit Feedback Table

implicit_user_feedback table contains user actions logged between September 2014 to June 2019. There are 281328 distinct users who visited the website. Customers checked 3268 distinct tour pages and purchased 3555 tours in total. The most of the tours are sold during late summer - early autumn as shown in Figure 3.3.

The average user sequence in the source database contains five page visits. We found out that the sequence which has most visit action contains 841 page visits. There exists some outlier sequences and this might be caused by the agency employees to refresh the page while updating the tour information. We decided

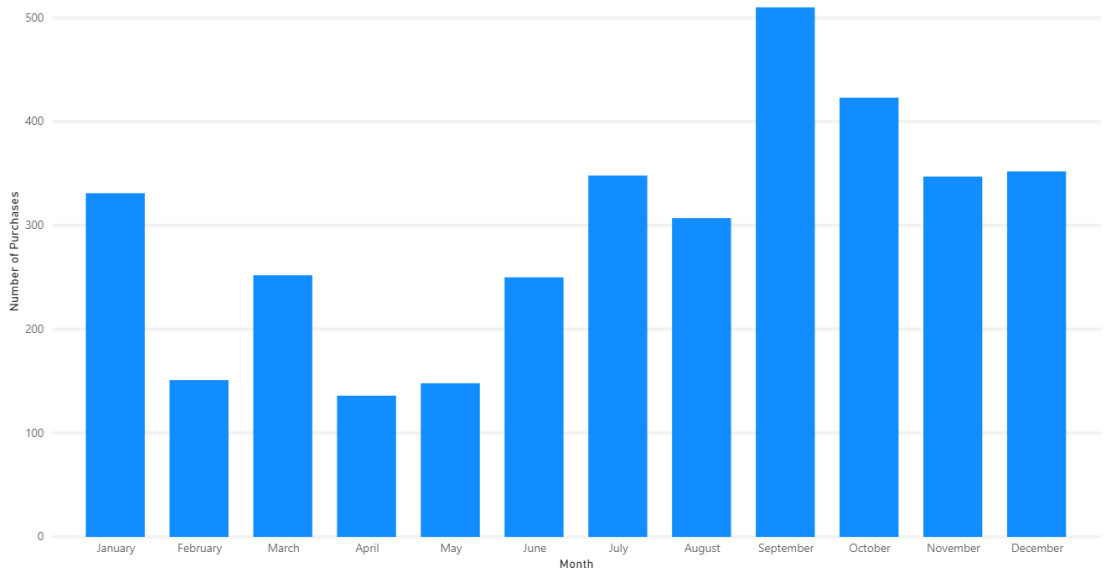


Figure 3.3: Figure shows the purchase distribution over months. Customers made the highest purchase in September.

to keep these user sequences as they represent real-life scenario. However, we limit the length of the user sessions to feed into RNN as ten. It means that if user will have more than ten visited tours in their sequence, we will only use the last ten visit information. Semantically, using the last ten visits instead of the first ones plays a crucial role here. By doing that, we can capture the latest user intention and work with that. More details using sequence data to train RNN is described in Chapter 4.

objectsListed (if_{10}) column records the items that are shown in the visited page alongside with their location. It is a textual column and it holds the information as follows:

```
tour_id:yCoordinate, xCoordinate, blockID;
```

The column aggregates the shown tour ids with the corresponding coordinates. Each item is separated by ";". As it is also described in [30], blockID column is not being used.

logFile (if_{13}) column records the user actions during a session. It is a textual information and keeps the data as follows:

```
Datetime; EventType; EventData; \n
```

Code Snippet 3.1 shows the example of a log entry.

```
1 2014-9-17 13:51:50; MouseMove, to:402,341
2 2014-9-17 13:51:50; MouseMove, to:695,662   to:783,638   to:773,503   to
     :680,470
3 2014-9-17 13:51:52; Scroll, to:0, 263
4 2014-9-17 13:51:52; MouseMove, to:624,537
5 2014-9-17 13:51:52; Scroll, to:0, 342
6 2014-9-17 13:51:53; MouseMove, to:813,552
7 2014-9-17 13:51:53; Scroll, to:0, 691
```

Code Snippet 3.1: Example of a logFile. Each user action is recorded individually. If an event triggered repeatedly, it will be stored as shown in line 2 as consecutive actions. Mouse movement and scrolling is registered in every 0.2 seconds.

Actions provided by the logFile column is useful in order to understand user behavior. While the column data does not completely reveal the direct user interest, it allow us to find which tours were visible to users screen, which tour records user hover her/his moves and clicked on. It is important that dataset also provides the screen resolution of the customer device and the web page. By using this information with the user logs, it is possible to simulate user behaviour and figure out if user stopped the mouse on a tour record or click on the tour object. Additionally, it is possible to find which tours were visible to user at a certain time. Identifying the visible items give us the opportunity of grading the importance of the item based on its visibility. It can be assumed that items were visible to user are more important than the ones were not initially visible. Therefore, they can be weighted more during the RNN training. Figure 3.4 represent this idea. Finding and grading the visible items is being used in one of the implemented recommendation algorithms and allowed us to increase recommendation accuracy as well as the other metrics we keep track of.

Figure 3.4: Tours that are visible to customers are weighted more than the ones that are not visible. In this case RNN can understand the importance of the tours visible to user. Blue colored crosses shows the starting coordinates of individual tour record.

3.2.3 Content-Based Tour Details

Travel Agency Dataset provides content base information regarding the offered tours from September 2014 to March 2020. There are 3911 [2] unique tour series and 64882 distinct tours with a total row count of 192483. Figure 3.7 shows the offered tour types distribution between 2017 - 2020.

The dataset uses MySQL semantics and can be queried by using SQL. The name of the tour metadata table is *content_base_tour_details*. Table 3.4 describes the column details.

ID	Attribute Name	Description
a_0	ID	Auto-Incremented Primary key
a_1	TourSeriesID	Identification of tour series
a_2	TourID	Identification of an individual tour
a_3	DatesFrom	Tour starting date
a_4	DatesTo	Tour ending date
a_5	TourName	Name of the tour
a_6	TourDescription	Short description of the tour series
a_7	TourTypeID	Identification of the tour type (e.g. Sightseeing)
a_8	MealType	Identification of meal served in the tour (e.g. Breakfast Only, 2 Meal etc.)
a_9	TransportationType	Identification of the transportation type (e.g. Bus, Plane, Train etc.)
a_{10}	AccommodationType	Identification of the accommodation type (e.g. Apartment, Hotel, Pension etc.)
a_{11}	AccomodationCategory	Accomodation category (Number of stars)
a_{12}	CountryList	List of destination countries (Separated by colon ",")
a_{13}	DestinationList	List of followup destination. (Separated by colon ",")
a_{14}	AveragePrice	Average service price for the selected tour in CZK
a_{15}	AveragePricePerNight	Average service price divided by TourDuration for the selected tour in CZK
a_{16}	MinimalPrice	Minimal price for selected tour in CZK
a_{17}	MinimalPricePerNight	Minimal price for services divided by TourDuration in CZK
a_{18}	MaximalDiscount	Maximal discount amount.
a_{19}	TourDuration	The duration of the tour in days
a_{20}	AdditionalInformationList	Additional information regarding the tour
a_{21}	ValidFrom	The day from tour is active
a_{22}	ValidTo	The day until tour is active.

Table 3.4: Column names and description for the content_base_tour_details table.

Figure 3.5 shows the example of a tour series that has multiple tours in it. As seen, the date and price columns tend to change between individual tours connected to a tour series.

ID	TourSeriesID	TourID	DatesFrom	DatesTo	TourName	TourDescription	TourTypeID	MealType	TransportationType	AveragePrice	AveragePricePerNight	MinimalPricePerNight
1439	22	48030	2014-12-29	2015-01-02	- Hotel Jan Šverma, Dema...	Hotel je situován v lyžařs...	7	3	1	5760	1440	1290
1440	22	48031	2015-01-02	2015-03-07	- Hotel Jan Šverma, Dema...	Hotel je situován v lyžařs...	7	3	1	1050	1050	950
1441	22	48032	2015-03-07	2015-04-02	- Hotel Jan Šverma, Dema...	Hotel je situován v lyžařs...	7	3	1	820	820	750
1442	22	48033	2015-04-07	2015-04-11	- Hotel Jan Šverma, Dema...	Hotel je situován v lyžařs...	7	3	1	3280	820	750
1443	22	48034	2015-04-02	2015-04-07	- Hotel Jan Šverma, Dema...	Hotel je situován v lyžařs...	7	3	1	5250	1050	950

Figure 3.5: There are six tours which belongs to one tour series. Even though tour name, description, meal type etc... stays same, availability dates and prices changes. Note that Figure is not showing the entire table columns.

Dataset also contains a column for a discount. Columns a_8 - a_{11} can be consider as ordered variable and they indicate the chosen meal plan, transportation,

[2] Although there exist 3911 distinct tour series, *implicit_user_feedback* table contains user visits to only 3486 tours series.

accommodation type and category for the tour. Higher values corresponds to better service (e.g. for $a_8 = 0$ means *no meal*, $a_8 = 1$ means *only breakfast* etc.)

Additionally, there are columns that order does not exist. Tour, tour series and type (a_1,a_7 columns have categorical values and do not have ordering. Country and Destination List columns (a_{12}, a_{13}) contain mostly real geographical locations such as countries, cities, towns, etc. However, there are cases that location columns also contain event names. Figure 3.6 shows an example of a sport event and sight seen tours offered by Slan Tour.

ID	TourSeriesID	TourID	DatesFrom	DatesTo	TourName
1748	352	48407	2015-03-03	2015-03-04	- FC Liverpool, Premier League - Liverpool FC - ...
1503	2458	48101	2015-02-28	2015-03-31	Hotel Vltava, Mariánské Lázně - Minikůra -

Figure 3.6: There is a possibility that tour name refers to a sport event rather than a geographical location.

The tour attributes in the travel domain tend to change frequently. Even though the tour main characteristics will stay the same, there is a high chance that the price and date interval will change. In the case of this change, the dataset has two relevant columns to use. ValidFrom column shows the first occurrence of the tour in the travel agency catalog. ValidTo column refers to the last available date of the particular tour with the offered attributes. If the ValidTo attribute is NULL, then the tour is still valid up that time and can be purchased by customers.

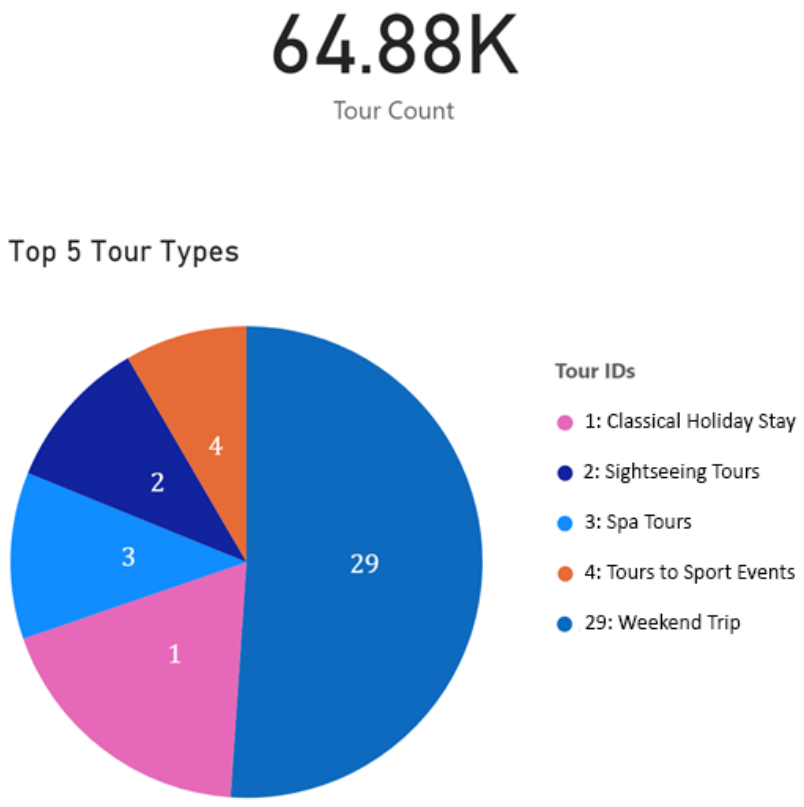


Figure 3.7: Top 5 tour types out of 64.88K tours that are offered by Slan Tour.

4. Implemented Solution

Using the travel agency dataset and Recurrent Neural Network, we aim to produce recommendations not only for long user sessions but also for short ones. It is mentioned that the visitors of small e-commerce websites often do not log in to the system and it is particularly hard to create recommendations for anonymous users. The proposed solution is able to recommend items to anonymous users due to its training methodology. Besides, we intend to increase the recommendation performance by using different pre-processing methods and using auxiliary data. We have considered four main metrics to measure the success of the recommendation algorithms: accuracy, recall, mean reciprocal rank (MRR), and normalized discounted cumulative gain (nDCG).

The dataset gives us the possibility to simulate real-user behavior and contains quite extensive implicit information as well as content metadata. Due to the business domain, content information is generally repeating (updating only the price and time of the tour, which belongs to a tour series) and typically requires processing. We experiment on using the implicit information as is, by applying basic and advanced encoding techniques. It is worth to mention that, proposed models can work on the new dataset as long as the dataset satisfies certain conditions such as captured user action logs with timestamps and content-meta data. The requirements depend on a specific model and the data it is using.

Implemented solutions use Recurrent Neural Network with LSTM extension to predict the next user item in the sequence. We primarily used user page visits to train the RNNs. Due to the structure of the dataset, we perform pre-processing to transform the raw data into a form that RNN can process. After the training, each created RNN model is tested and evaluated by using specific performance metrics. To show the improvements, we implemented baseline models such as Item2Vec [3], Doc2Vec [20], and Cosine Similarity, then compared those models with the proposed solutions.

Overall, we observed that using RNN increased the performance of the next item recommendations comparing the baseline models. Furthermore, we found some critical factors between implemented RNNs. While some models that were using none or basic pre-processing methods were performing poorly, the RNN models with complex pre-processing offered higher performance. We saw that the models trained by the all implicit feedback data outperformed the ones that use partial implicit feedback. During our experiment with implementing multi-input RNN, which uses user sequences and auxiliary data (content base information), we could not see performance increase comparing the one without the auxiliary input. While this might happened due to the quality of the content-based data, it is also possible that used pre-processing method is not enough for the model to learn the pattern of the data.

In total, there are seven RNN LSTM variations implemented and tested. While there are common methods used by all models, there are also unique methods for specific models. That is why source code follows object-oriented fashion with inheritance, and we will explain all models in their respective subsections.

The following sections will discuss the workflow of the source code by showing the pre-processing, model creation, and evaluation steps with partial code snip-

pets. Although there exist multiple methods and classes, we will only focus on the most important ones. The rest of the information and the functionalities can be found in the Github Repository [1].

4.1 Workflow

The source code follows three main parts, particularly pre-processing, model creation, and evaluation. First, we prepare the data, then create a model and evaluate it. This process can be imagined as a continuous cycle, as shown in Figure 4.1. While some implemented models use the same data as input, some require additional pre-processing or information.

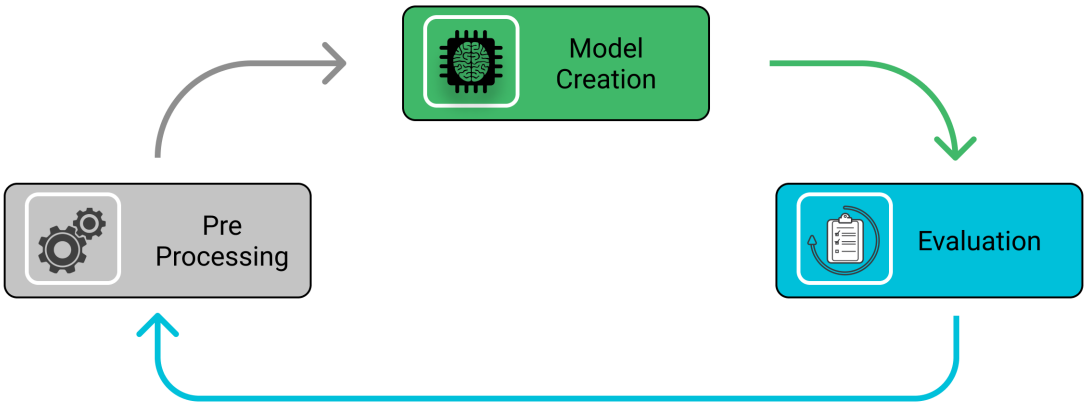


Figure 4.1: Implementation consist of three main stages. We first perform data processing, then create LSTM model and finally evaluated the results.

4.2 Data Pre-Processing

The dataset contains 3486 distinct tour series objects visited by the users. Every object has its unique id. Before we start any encoding, we first created an item dictionary contains all the visited objects. After ordering the items in ascending order, we inserted them into a Python dictionary to represent each item as the dictionary index instead of their objectIDs. This allows us to reduce the vector size for the encodings. One important factor is that once LSTM models create next item recommendations, the recommendation list will consist of object ids based on the dictionary mapping. Thus, recommended object ids must be converted back to their unique objectIDs using the item dictionary.

4.2.1 Simulating user actions

Travel agency web page logs the tour page users visited during the shopping session, as shown in Table 3.3. If a user visits a dedicated tour page, the id of the tour will be registered as the visited item. However, when the user visits the

[1] `https://github.com/Rinqt/charles_uni_master_thesis`

homepage or a category page, instead of inserting the objectID, the system inserts zero. This indicates that the user was visiting a category page that contains multiple tours. We treat both information separately to allow RNN to distinguish between a category page and a dedicated tour page. We used userID and sessionID columns to aggregate the objectIDs that customers visit. This gives us the ability to work with individual user sessions and encode catalog or index pages differently. An example of a user session is given in Table 3.2.

Source dataset contains 131647 sessions which are shorter than two visits. It means that users left the website after checking one particular item. We removed those sequences, which corresponds to %59 of the entire dataset since there is no possible way to use them for next-item prediction.

4.2.1.1 Encoding User Sequences

Working with User Visits to Tour Pages

We use two different methodologies to encode the tour series objects. First, all travel objects are transformed into a vector by using One-Hot-Encoding (OHE) strategy. With this approach, we insert 1 for the location of the visited travel object in the vector. Second, we create and train an Item2Vec model to encode each objectID as Item2Vec vector.

Working with User Visits to Catalog Objects

When a user visits a catalog or index page, the system logs all the presented tours into the objectsListed column. It is possible to understand if the user interacted with a tour record by using the $if_{6:10}$ and if_{13} columns.

if_{10} column contains the objectIDs of the tours in the catalog page and their starting location. We implement a method to understand if the user interacted with an object in the catalog page. For this, we had to pre-process the if_{13} column and split user actions by their type. We used three different action types (MouseMove, MouseClick, and Scroll) to analyze the logs. Figure 4.2 shows the difference between raw and processed information. Next, we used $if_{6:9}$ columns to identify if the catalog objects were visible to users screens during the visit. Since the raw data only contains the starting x and y coordinates of the tour image, we had to assume the total size of the image. After considering some tour series images, we decided on fixed image size and work based on it. One important thing is that due to assumptions, we applied buffer for the scroll events. It is often possible that the scroll event might end up when half of an image is visible, and the rest is not. That is why we decided to use $default\_image\_size\_y - int((y\_scroll\_amount \mod default\_image\_size\_y))$ amount of pixel as a buffer. Using the buffer, we were able to check the image visibility based on the scroll event dynamically. We classified the visible items as 1 and not visible items as 0 by creating a tuple of (objectID, itemVisibility). In case there were no logs located, we inserted "No Log Found" or "No Item Found" string for better readability. Lastly, we aggregated all the processed information, including event timestamps, as shown in Figure 4.3.

Although there are multiple tour objects in the catalog page, it must be represented as one vector to keep the user sequence order and length as-is. To

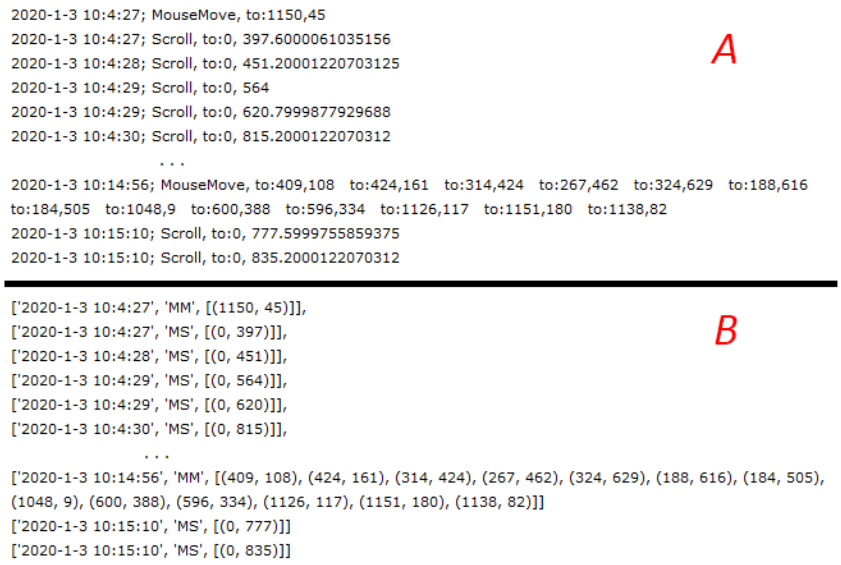


Figure 4.2: Comparison of raw and processed information of logFile column. (A) Raw Data (B) Processed Data.

user_id	715904
session_id	94
item_sequence	[3280, 0, 0, 0, 1920]
session_start_time	[datetime.datetime(2019, 3, 4, 14, 51, 28), datetime.datetime(2019, 3, 4, 15, 5, 16), . . .
catalog_item_list	['No Catalog Item', [(6542, 'C'), (6574, 'C'), (6521, 'C'), (6527, 'C'), (6518, 'C')], . . .
user_log_list	[[['2019-3-4 15:5:19', 'MM', [(928, 5)]], ['2019-3-4 15:5:19', 'MM', [(946, 112), . . .
good_catalog_items	['No Item Found', [(6521, 0), (6574, 0), (6527, 0), (6542, 0)], 'No Item Found'] . . .

Figure 4.3: Representation of the pre-processed user logs. We created a Pandas Data frame to keep the processed logs. From top to bottom each row contains information regarding user id, user session, objects (tours/catalog pages) visited during the session, timestamps of the user actions, found catalog objects (labeled as 'C', mouse events and catalog items with their classified visibility labels.

describe various tours in one vector, we used K-Hot Encoding, which allows us to represent multiple tour data in one vector. Once we collected all the catalog objects, we applied l2-norm scaling to all vector information. In the end, the resulting vector contains data for all the tours that were in the catalog page.

4.2.1.2 Encoding Auxiliary Data

The source database also provides content metadata, which is described in Section 3.2.3. Due to the travel agency domain, tours are mostly repeated throughout the year. For example, a summer holiday tour can be offered to customers during autumn with lower price tags. In the case of the repeating tours, the price and tour date often change while the rest of the content data usually stays the same. The validation date of a tour is located in column a_{21}, and it is possible to identify which particular tour user visited by comparing the ValidFrom value with the time of the user visit. In case of multiple same tours available for a tour series, we compare the session time with the ValidFrom and ValidTo columns to find the particular tour. Once we have more than one tour found, we average the tour

vectors to represent them in one vector while trying to keep metadata as much as possible.

There are many possible ways to represent content data in vector form. We identified columns a_3, a_7, a_{10}, $a_{12:13}$, $a_{15:19}$ to work with. All columns are encoded by using the OHE strategy. Then, all created encoding vectors are concatenated to represent all tour data in one vector. This method is also known as K-Hot-Encoding since multiple ones are representing different types of information in one vector. Table 4.1 illustrates this idea.

DatesFrom	TourTypeID	Accommodation Type	K-Hot-Encoding
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]	[1, 0, 0, 0]	[0, 0, 1, 0, 0]	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0]

Table 4.1: Once all the content metadata one hot encoded, we concatenate all the vector. Resulting vector will be K-Hot-Encoded representation of a tour.

4.3 Proposed Recommendation Algorithms

We proposed seven RNN versions. All implemented models perform next-item predictions to predict the tour series while having different characteristics. Before moving forward with model descriptions, we will describe the training strategies since all models share the same training pattern.

4.3.1 Model Training

All implemented RNN models created by using TensorFlow Keras. While six models use the standard Keras Sequential API, one particular model is implemented by Keras Functional API. We conducted multiple test scenarios to adjust model hyperparameters. Then, we trained each model by using different epochs. Additionally, we tried to manipulate the model architecture and decided on the optimal possible architecture that best fits the data with the given parameters.

Train Test Validation Split

All models are trained and evaluated on the same training and evaluation data. First, 25% of the entire dataset is randomly selected for evaluation. Evaluation data is not visible anyhow to the RNN models during the training phase. It allows all models to be evaluated on an input that they have not seen before. The leftover 75% of the data randomly split into 4-Folds by using the Cross-Validation strategy mentioned in 1.2.6.1. 75% of each fold data is used for training, while the remaining is used for testing. All information is saved in the file system for the future training and evaluation. Each model which uses catalog data is trained on four-fold contains roughly 51K train and 17K test samples. The rest of the models contain 30K train and 15K test samples since the catalog visits are removed from the user sequences. Then, the best-performed model is used for the evaluation.

Hyper Parameter Tuning

Keras Tuner Library is used to tune model parameters. It uses random search to find the best possible combination of hyperparameters for the candidate model. To do that, it needs a range of configurable parameters. Code snippet 4.1 shows the given parameters and their range.

```
hyper_parameters =
    {
        'objective_function'    : 'val_accuracy',
        'lstm_units'            : {'min': 32, 'max': 128, 'step': 32},
        'dropout'               : [0.1, 0.3, 0.5],
        'lstm_layer_activation' : 'relu',
        'lstm_layer_dropout'    : 0.1,
        'dense_activation'      : 'softmax',
        'learning_rate'         : [0.01, 0.001, 0.0001],
        'metric'                : 'accuracy',
        'loss'                  : 'categorical_crossentropy',
    }
```

Code Snippet 4.1: We tried multiple ranges of parameters (neurons in the layer, learning rate, dropout etc...) and received similar results with the parameters given in the snippet. As the objective function, tuner is given validation accuracy meaning that it will try to increase the validation accuracy instead of training accuracy. Presented parameters are used to train all the models.

Keras Tuner is adjusted to apply three trials and fit three different models in each trial. In the end, Tuner trains three different RNN models and tries to increase validation accuracy. All given parameters are adjustable for future usage, and it is shown in Snippet 4.2. All created models are saved in the file system, including their plots, performance metrics, and architectures. Figure 4.4 shows an example of one of the best possible model with its structure and selected hyperparameters.

```
    tuner = RandomSearch(self.build_model,
                         objective=self.hyper_parameters['objective_function'],
                         max_trials=self.max_trials,
                         executions_per_trial=self.executions_per_trial,
                         seed=self.random_state,
                         project_name=f'split_{str(split_number)}',
                         directory=os.path.normpath(self.path_tuner_directory))
```

Code Snippet 4.2: Keras Tuner uses RandomSearch and finds the best possible hyper parameters for the training.

Furthermore, each model is trained by using eight and sixteen epochs to see model behavior in longer epochs. Figure 4.5 shows an example of two different training phases. As the batch sizes, we tried 64, 128, 256, and 512. 512 bath size performed the best and selected as the default value for all the models. Based on the test results, the best-performed models were the ones trained with eight epochs. All in all, by using Keras Tuner and cross-validation, we identified the best models with the highest validation accuracy to test on the evaluation data.

4.3.1.1 Creating Training Data

Overall training data preparation is separated into four main steps as follows:

1. Creating features and labels
2. Encoding features

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
masking (Masking)            (None, 10, 3486)          0
_________________________________________________________________
lstm (LSTM)                  (None, 10, 96)            1375872
_________________________________________________________________
batch_normalization (BatchNo (None, 10, 96)            384
_________________________________________________________________
dropout (Dropout)            (None, 10, 96)            0
_________________________________________________________________
lstm_1 (LSTM)                (None, 64)                41216
_________________________________________________________________
batch_normalization_1 (Batch (None, 64)                256
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 3486)              226590
=================================================================
Total params: 1,644,318
Trainable params: 1,643,998
Non-trainable params: 320
_________________________________________________________________

Trial ID: 10eafdc7901d0a152885e9d883b1f854 [BEST TRIAL]
_________________________________________________________________


Selected Hyper Parameters:
|    first_layer: 96
|    dropout_one: 0.1
|    second_layer: 64
|    learning_rate: 0.001
_________
| --> Trial Score: 0.2820856074492137
_________
```

Figure 4.4: Selected parameters and architecture information is saved in the file system. Best trials are labeled as [BEST TRIAL].

3. Applying masking
4. Encoding labels

Creating features and labels

Once the raw data is pre-processed, user sequences divided into two parts for the training. While the last visited object in the sequence used as a label, the rest of the sequences selected as features. Code snippet 4.3 shows the feature/label selection process.

```
        self.all_features = np.full(len(self.user_sequence_list), fill_value=
    self.mask_value, dtype=object)
        self.all_label = np.full(len(self.user_sequence_list), fill_value=self.
    mask_value, dtype='int32')

        for index, sequence in enumerate(self.user_sequence_list):
            new_sequence = np.full((len(sequence) - 1), fill_value=self.
    mask_value, dtype='int32')
            for idx, item in enumerate(sequence[:-1]):
                new_sequence[idx] = item

            self.all_features[index] = new_sequence
            self.all_label[index] = sequence[-1]
```

Code Snippet 4.3: Code iterates each user sequences. The last visited item kept as a label while the previous visits are assigned as features.

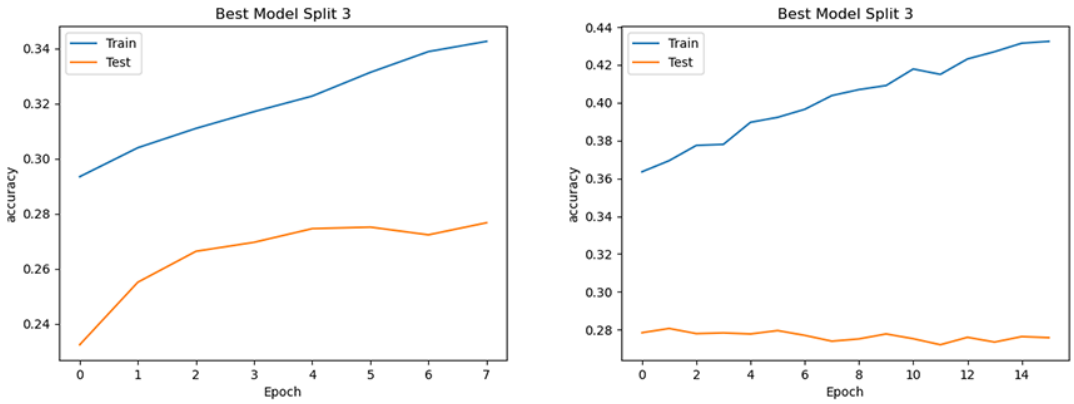


Figure 4.5: 8 and 16 epochs training plots of Model 2. Both plots shows the accuracy of the model on Fold 3. Training accuracy increases when the number of epochs increase. However, test accuracy start decreases which often leads to over-fitting. To stay in the safe zone and not over-fit the models, we used the models with 8 epochs.

Encoding Features

All models expect the Model 1, perform feature encoding. This process will transform a sequence of numerical objectIDs into a sequence of vector representations. Snippet 4.4 shows one of the encoding approaches used during the feature encoding.

```
    def encode_features(self):
        count = 0
        for user_sequence in self.all_features:
            encoded_sequence = np.full(shape=(len(user_sequence), self.
encoding_vector_size), fill_value=self.mask_value, dtype='float32')
            for idx, item in enumerate(user_sequence):
                if item != 0:
                    encoding = list(self.item_dictionary.keys())[list(self.
item_dictionary.values()).index(item)]
                    vector = np.zeros(shape=self.encoding_vector_size, dtype='
float32')
                    vector[int(encoding)] = 1
                    encoded_sequence[idx] = vector

            self.all_features[count] = encoded_sequence
            count += 1
```

Code Snippet 4.4: Python method used for One-Hot-Encoding

Applying Masking

There are various ways to feed different lengths of input data to RNN. We used the masking ability of Keras Library to work with varying lengths of sequences. We set each user sequence to have precisely ten visit actions in it. For the sequences longer than ten visits, we used the last ten items to represent users' latest interest. However, if the user sequence has less than ten page visits, we fill the user sequence with masked value (-9.0) and then append the objects to the end of the sequence. Once RNN sees masking value during the training, it will not assign any weights and omit them. By applying masking, we were able to

feed sequences with different lengths to RNN and train the models on the exact user sequences.

RNN requires 3D vectors as input data. So far, all the pre-processing used only a one-dimensional vector space. Thus, we applied shape transformation to the feature data so that it can be fit to the network. Snippet 4.5 shows the masking and shape transformation process.

```
    def apply_mask(self, all_features, number_of_features):
        masked_features = np.full((len(all_features), self.max_sequence_size,
    number_of_features), fill_value=self.mask_value, dtype='float32')

        for index, feature in enumerate(all_features):
            seq_len = feature.shape[0]
            difference = seq_len - self.max_sequence_size

            if difference > 0:
                feature = feature[difference:]
                seq_len = seq_len - difference

            if masked_features.shape[2] == 1:
                masked_features[index, -seq_len:, :] = feature[:seq_len].reshape
    (seq_len, 1)
            else:
                masked_features[index, -seq_len:, :] = feature[:seq_len]
```

Code Snippet 4.5: apply_mask method, creates an empty 3D numpy array. Then it transforms processed features to the 3 dimensional arrays and assigns them to the newly created vector. Additionally, method controls the user sequences and makes sure they are in the selected sequence length (in our case it is 10)

Encoding Labels

All implemented RNN models encode the labels before the training. Code Snippet 4.4 shows one of the ways we used to encode labels.

4.3.2 Model Characteristics

Each RNN model uses a different type of features and labels. Four models do not use catalog page information and only utilize item visit events. Those models can only perform recommendations for individual tour pages and cannot work with catalog page data. The rest of the models use different levels of implicit feedback. They utilize catalog page information and capable of recommending objects even for catalog pages. Table 4.2 shows the raw and encoded features for all the models. Since there exist seven different models, we will describe every model in its dedicated subsection. While six models trained to perform classification task, one model is performing regression.

Before the training, we insert model metadata into the database as shown in Snippet 4.6.

```
        insert_model_data(model_type=model_name,
                          training_type=option_alias,
                          description=model_description,
                          hyper_parameters=json.dumps(hyper_parameters),
                          meta_data=json.dumps(training_options))
```

Code Snippet 4.6: Each model and their corresponding metadata (model description, training and hyper parameters) is inserted into the database

Model	Raw Feature	Encoded Feature	Encoded Label
Model 1	[objID, objID, objID . . .]	[objID, objID, objID . . .] (No encoding applied)	[<ohe_vector>]
Model 2	[objID, objID, objID . . .]	[<ohe_vector>, <ohe_vector>, <ohe_vector>. . .]	[<ohe_vector>]
Model 3	[objID, objID, objID . . .]	[<i2v_vector>, <i2v_vector>, <i2v_vector>. . .]	[<i2v_vector>]
Model 4	[objID, objID, objID . . .]	[<i2v_vector>, <i2v_vector>, <i2v_vector>. . .]	[<ohe_vector>]
Model 5	[catID, catID, objID . . .]	[<khe_vector>, <ohe_vector>, <khe_vector>. . .]	[<ohe_vector>]
Model 6	[catID, catID, objID . . .] [objData, objData, objData . . .]	[<khe_vector>, <ohe_vector>, <khe_vector>. . .] [<khe_vector>, <khe_vector>, <khe_vector>. . .]	[<ohe_vector>]
Model 7	[catID, catID, objID . . .]	[<khe_vector>, <khe_vector>, <khe_vector>. . .]	[<ohe_vector>]

Table 4.2: Table shows the encoding methodologies used for each model. objID: ID of the tour object, ohe_vector: One-Hot-Encoded Vector, i2v_vector:Item2Vec Vector, cat_id: Catalog Page ID, khe_vector: K-Hot Encoded Vector, objData: Tour object metadata.

4.3.2.1 Model 1

Model 1 is the basic RNN model which does not utilize encoding for the features. Visited objectIDs located in the user sequence kept as-is. On the other hand, labels are One-Hot-Encoded for RNN to classify items. The model is trained by using only dedicated item-visit feedback, and it cannot make predictions for the catalog pages. As the model features are not vectorized or scaled, the implemented model could not understand the data as the other models. This model performed worst comparing all the other implemented solutions which perform the classification task.

Model Architecture

We trained the network by using *categorical_crossentropy* loss function. Model parameters and architecture is shown in Figure 4.6.

4.3.2.2 Model 2

Model features and labels are transformed into OHE vectors for the classification task. The input data only contains dedicated tour-visit actions. The model does not use catalog information, it cannot produce recommendations for the catalog pages. Since both features and labels are one-hot-encoded, the model was able to capture some sort of pattern in the data and performed better than Model 1. Additionally, the model is the best-performing implementation between the models that do not utilize catalog information.

Model Architecture

Model 2 is trained with *categorical_crossentropy* loss. Best selected parameters and the model architecture is shown in Figure 4.7.

4.3.2.3 Model 3

Model 3 is utilizing Item2Vec vectors for the objectIDs. We transformed both features and labels into Item2Vec vectors before feeding into the network. All catalog visits are removed from the user sequences. That is why the model cannot create recommendations for the catalog pages.

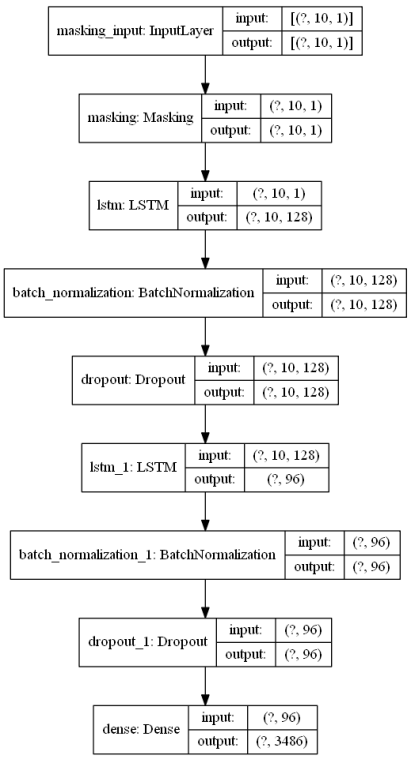


Figure 4.6: Model 1 Architecture consists of Input Layer, Masking Layer, First LSTM Layer with 128 neurons and second LSTM Layer with 96 neurons. Each LSTM layer has batch normalization and dropout layers. At the end, there is a dense layer that outputs one class.

Model Architecture

Model 3 is the only model trained with *mean_absolute_error* loss function due to its Item2Vec encoded labels. Item2Vec transforms objectIDs into numerical vectors, which changes the classification task to a regression task. Keras Tuner is given with *val_mean_absolute_error* objective function particularly for this model. Figure 4.8 show the selected parameters and the model architecture.

4.3.2.4 Model 4

Features of the Model 4 is transformed into Item2Vec vectors like in Model 3. However, we encoded labels into OHE vectors. By using this approach, we were able to compare Model 3 and Model 4 directly with each other and see how both models performed by using Item2Vec and OHE labels. Like the previous models, we removed the catalog visits from the user sequences, and the model is only able to create recommendations for dedicated item page visits.

Model Architecture

Model 4 is trained with *categorical_crossentropy* loss function since its labels are encoded to OHE vectors. The best parameters and network architecture is shown in Figure 4.9.

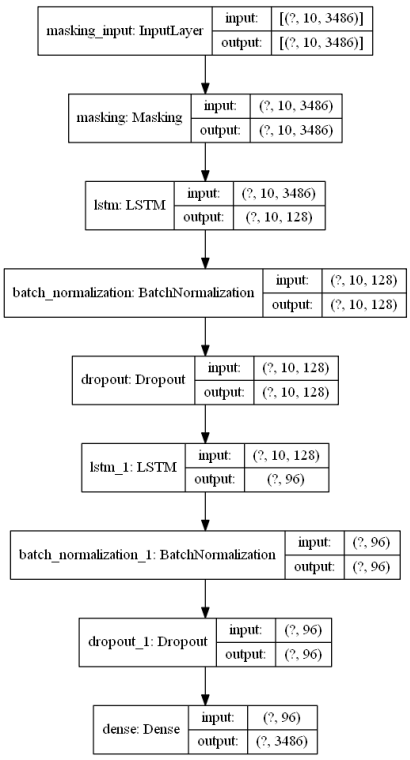


Figure 4.7: Model 2 Architecture follows the same design with Model 1. There are Input Layer, Masking Layer, two LSTM Layers with 128 and 96 neurons respectively. Each LSTM layer has batch normalization and dropout layers. At the end, there is a dense layer that outputs one class.

4.3.2.5 Model 5

Model 5 is the first model capable of making recommendations for catalog pages. During the pre-processing phase, We keep the catalog visits in the user sequences. Then, for each catalog visit, we located the tours that were shown to the user. Once the tours are found, they transformed into OHE vectors and scaled based on their visibility to the user. Then, we assigned different weights for each different catalog objects. We chanced the encoding value of the objects that were in the catalog page to 2. The catalog items that were visible to users are assigned with 5. Finally, we gave 15 for the objects that the user clicked. Table 4.3 illustrates the process of assigning different weights to catalog pages that users interacted differently. The idea of giving different weights is to represent the difference between the catalog objects and how the customer treats them. It is possible to say that the user is more interested in the items that she/he clicked, comparing the once that are visible to the user. Similarly, we can think that the user is not interested in the tours that were not located on the user screen. Once the catalog items are given extra weights, we used l2 normalization to describe all the catalog items into one vector so that user sequence length could be preserved.

Model 5 is the best performing model between all the implemented solutions.

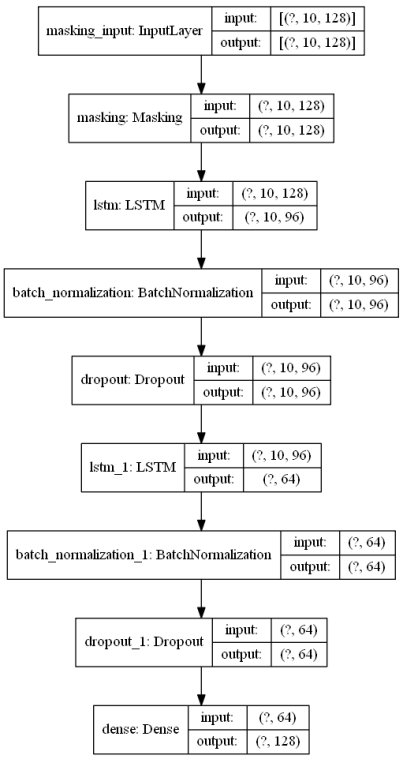


Figure 4.8: Model 3 has different size Input and Masking Layer since its input is item2vec vectors. There are 2 LSTM layer which has 128 and 96 neurons. Both LSTM layers is followed by Batch Normalization and Dropout Layers. In model there the Dense layer has lower size due to Item2Vec vectors.

Catalog Items	Clicked Item	Visible Item	Not Visible Item	Weighted Catalog Items
[1920, 2006, 6957]	2006	1920	6957	[5, 15, 2]

Table 4.3: Consider that there are three tours shown in the catalog page. It is important for RNN to understand which object is more important to user. That is why, we applied custom weights for the objects that user clicked, see or not see during the catalog visit.

Model Architecture

Model 5 is trained by using *categorical_crossentropy* loss function. The best parameters and network architecture are presented in Figure 4.10.

4.3.2.6 Model 6

Model 6 is the most complex RNN implementation among all the others. It uses Keras Functional API, which allows us to feed two different inputs to the network by using two input branches. The first branch contains the user sequence with catalog visits and utilizes the same encoding as Model 5. The second branch of the network contains only metadata about the tours. *content_base_tour_details* table is used to gather tour information. When a user visits a page, we located the possible tours from the *content_base_tour_details* table. Then, tour information encoded by using K-Hot-Encoding, which is described in Section 4.2.1.2. Once the encoding is completed, we average the vectors to keep general tour information

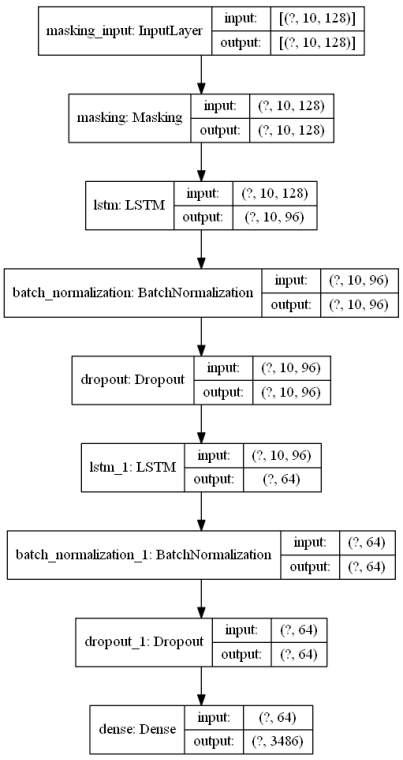


Figure 4.9: Model 4 architecture is exactly same as Model 3 except the Dense Layer. Since Model 4 is using One-Hot-Encoded Labels its Dense Layer has more output classes.

between the similar tours and represent it as one vector.

During the training, RNN was given two different inputs so that once it processes the user sequence information, it was also processing the corresponding tour metadata. However, this model was not able to perform any better than Model 5. Results show that the network was not able to understand the underlying pattern in the given data. We believe that this might be caused by the quality of the content metadata and the way we initially encode it. As stated earlier, we used specific columns from the *content_base_tour_details* table for encoding. It is possible to have different results by using more or fewer columns during the pre-processing. Additionally, implemented RNN architecture highly affects the learning and the overall performance of the network. Further testing could be applied by manipulating the existing model to allow RNN to fit data better.

Model Architecture

Like Model 5, Model 6 is also trained by using *categorical_crossentropy* loss function. The best parameters and network architecture are presented in Figure 4.11. Once the branches of the network are built, they concatenated to have a one vector which contains information from both input branch. Code Snippet 4.7 show the code to build the model.

```
    def build_model(self, hp):
        sequences_model = self.create_sequence_branch(hp)
        auxiliary_model = self.create_sequential(hp)

```

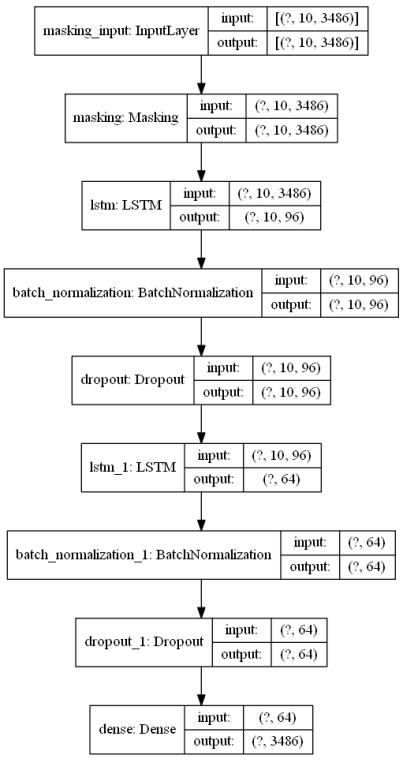


Figure 4.10: Different than the other models, Model 5 has lower neurons assigned by Keras Tuner. Similar to other models, Model 5 also contains Input Layer, Masking Layer, 2 LSTM Layers with 96 and 64 neurons respectively and a Dense Layer. Each LSTM Layer is followed by Batch Normalization and Dropout Layers.

```
5         concat_layer = keras.layers.Concatenate()([sequences_model.output,
    auxiliary_model.output])
6
7         tour_pred = keras.layers.Dense(self.number_of_distinct_items, activation
    =self.hyper_parameters['dense_activation'], name='sequence_prediction')(
    concat_layer)
8
9         model = keras.Model(inputs=[sequences_model.input, auxiliary_model.input
    ], outputs=tour_pred)
10
11        opt = Adam(hp.Choice('learning_rate', values=self.hyper_parameters['
    learning_rate']))
12
13        model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=
    self.hyper_parameters['metric'])
14
15        return model
```

Code Snippet 4.7: First, we created the branch contains the user sequences by using Keras Functional API. Then Sequential API used to create the second branch which process the content metadata. Once both models are created we used Concatenate() Method to merge output of two models into a single vector and compile the model.

4.3.2.7 Model 7

Model 7 only uses content metadata to perform predictions. This model is specifically developed to see the performance of using content metadata for the rec-

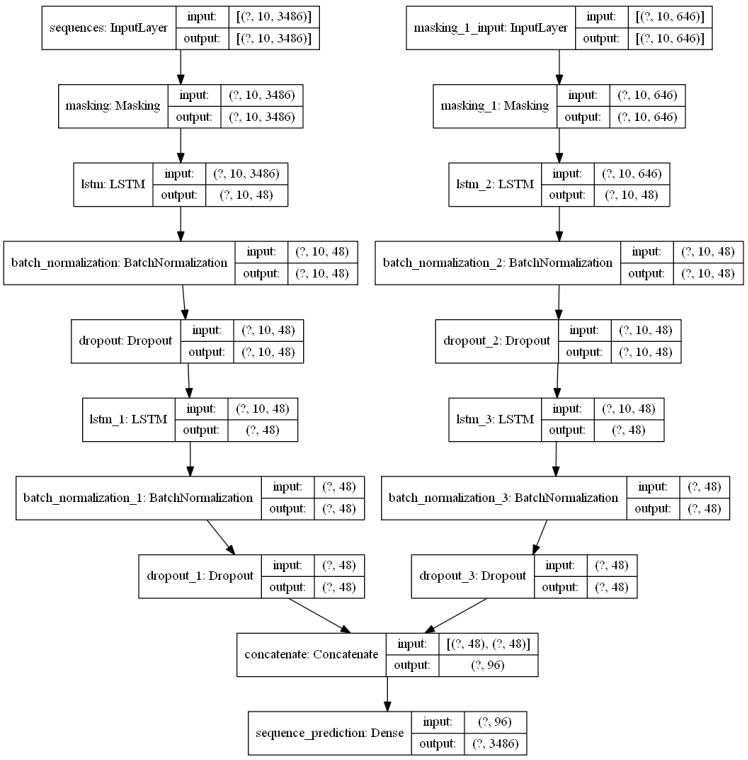


Figure 4.11: Model 6 has a unique architecture comparing the other models. It is implemented by Keras Functional API and it is able to get two input at the same time. Model 6 has two branch of layers. Each layer has its own Input, Masking, LSTM, Batch Normalization, Dropout and Dense Layers.

ommendations. We keep the catalog feedback in the user sequences so that the model can make recommendations for catalog pages.

Model Architecture

The model is trained by using *categorical_crossentropy* loss. Figure 4.12 describes the model architecture with its selected training parameters.

4.4 Model Evaluation

As specified earlier, we used a separate evaluation data for measuring the model performance. During the evaluation, we consider four metrics; accuracy, precision, recall, and nDCG. Between all the performance metrics, nDCG is selected as the success criteria as it was the evaluation metric we tried to increase.

We implement a method to run the evaluation in multi-threads. Method calculates a bucket size by dividing the size of the evaluation data to thread number. The thread size is adjustable however we did not see any performance approve after creating 128 threads. Each bucket will be assigned to some portion of the evaluation data. Then buckets will be distributed over the threads so that they can be evaluated and their results can be inserted to the database. Code Snippet 4.8 shows the process of creating the evaluation buckets.

```
1     def evaluation_creator(self, sequences_to_evaluate, evaluation_class):
```

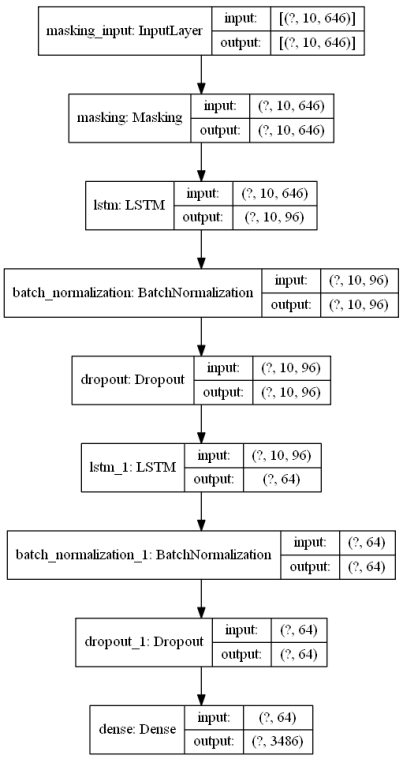


Figure 4.12: Model 7 utilizes object metadata. Its Input Layer size is different than the other models. Other than that, it follows the same architectural pattern.

```
        bucket_size = int(len(sequences_to_evaluate) / self.thread_number)
        lower_limit = 0
        upper_limit = bucket_size

        self.log_event(f'Sequence evaluation has started with {self.
    thread_number} Threads with {bucket_size} Buckets ..')

        threads = []
        for bucket in range(self.thread_number):
            individual_thread = threading.Thread(target=self.
    multi_thread_sequence_evaluation, args=(sequences_to_evaluate[lower_limit:
    upper_limit], evaluation_class))
            lower_limit = upper_limit
            upper_limit += bucket_size

            if bucket == self.thread_number - 2:
                upper_limit = len(sequences_to_evaluate)

            threads.append(individual_thread)
            individual_thread.start()

        for thread in threads:
            thread.join()
```

Code Snippet 4.8: *evaluation_creator* method is responsible for applying evaluation by using multi-threading. First, it will calculate the portion of the evaluation data and assign as individual buckets. Then each bucket will be given to a separate thread and perform evaluation.

4.4.1 Evaluation User Sequences

Each model is evaluated on 23082 user sequences. Catalog information was removed from Model 1, 2, 3, and 4 since they did not use the catalog page data. For Model 5, 6, and 7, we keep the catalog pages in the user sequences due to their ability to incorporate information from the visited catalog pages. During the evaluation, each visited object in the user sequence is evaluated one by one. By this way, we simulate the user page visit individually and produce next-item recommendations. For example, if the model is evaluated on a user sequence containing five visits, we performed five individual evaluations. During each step, RNN is given one more input on top of the previous one. Table 4.4 gives an example of an evaluation scenario. Once the user sequence contains catalog information, instead of directly creating a recommendation for the catalog visit, we inserted catalog information into the RNN input stack until we see an object visit. Consider a user sequence of four visits, and two of the visits are catalog pages. We add those two catalog page information to the input stack of RNN until the next object in the sequence is a dedicated page visit as illustrated in Table 4.5. In the end, for each evaluation, we gathered the top 10 recommendations and calculate their performance.

User Sequence	RNN Input	Ground Truth
[7024, 6708, 6667, 1024]	[7024]	[6708]
[7024, 6708, 6667, 1024]	[7024, 6708]	[6667]
[7024, 6708, 6667, 1024]	[7024, 6708, 6667]	[1024]

Table 4.4: RNN gets the first visited item as the only input and produce next-item recommendations. Then it gets the first and second item as input and creates next-item predictions. In short, RNN has stacked input data thought the user session, and each recommendation is created by the previous visit information.

User Sequence	RNN Input	Ground Truth
[0, 0, 6667, 1024]	[0, 0]	[6667]
[0, 0, 6667, 1024]	[0, 0, 6667]	[1024]

Table 4.5: User sequence contains two catalog page visits (labeled as zero). Once RNN sees a catalog page, instead of making a prediction, it adds the sequence visit data into its input stack. At the moment it finds a object visit, it will produce recommendation while utilizing the catalog data.

Some user sequences only contain repeated tour page visits. Instead of removing those sequences, we keep them since they represent real-life behavior. To distinguish those sequences, we introduced an indicator called *trivial_prediction*. Once we evaluated a recommendation for given objectID, we checked if the rest of the sequence contains the same objectID again. If yes, we labeled those predictions as *trivial_prediction*. By doing that, we tried to understand if the prediction was affected by a repeated objectIDs or not.

Additionally, since some models utilize catalog page information and some not, we create another label named *catalog_prediction* to represent the recommendations that use only catalog page visits. In the end, we can compare all models regarding their capabilities. It is possible to query for the evaluations which are not labeled as *catalog_prediction* and this gives the possibility of comparing how each model performed on the sequences that contain only dedicated tour page visits.

Each evaluated recommendation is inserted into the *evaluation_table* with its performance metrics. As a supplementary data, we also insert information like; *model_name*, *user_sequence*, *trivial_prediction*, *catalog_prediction*, *catalog_count*, *input_sequence*, *ground_truth*, *predictions* and *sequence_length*. Figure 4.13 shows the evaluation table.

user_id	session_id	precision	recal	mrr	ndcg	predictor_name	trivial_prediction	catalog_prediction	catalog_count
1326034	12	0.1	1	0.125	0.315465	model_05_8_epoch_512_batch	0	0	0
1326034	12	0.1	1	0.333333	0.5	model_05_8_epoch_512_batch	0	0	0
1326034	12	0.1	1	0.1	0.289065	model_05_8_epoch_512_batch	0	0	0
1326034	12	0	0	0	0	model_05_8_epoch_512_batch	0	0	0
1326034	12	0.1	1	0.5	0.63093	model_05_8_epoch_512_batch	0	0	0

ground_truth	sequence	input_sequence	input_sequence_length	user_sequence_length	predictions
6258	6289 6258 6290 6282 5932 6404	6289	1	6	6289 6369 6290 ···
6290	6289 6258 6290 6282 5932 6404	6289 6258	2	6	6257 6258 6290 ···
6282	6289 6258 6290 6282 5932 6404	6289 6258 6290	3	6	6258 6257 6290 ···
5932	6289 6258 6290 6282 5932 6404	6289 6258 6290 6282	4	6	6258 6257 6290 ···
6404	6289 6258 6290 6282 5932 6404	6289 6258 6290 6282 5932	5	6	6258 6404 5932 ···

Figure 4.13: Each evaluation result is inserted to the database along with additional data as shown in the Figure.

4.4.2 Evaluation Results

Implemented RNN models compared with some of the recommendation algorithms used in practice such as Cosine Similarity, Doc2Vec and Item2Vec. Those models are also evaluated on the same evaluation and do not utilize catalog information. Figure 4.14 shows the overall comparison of the implemented models including the baselines.

When we compare the results of the implemented models, we observed that utilizing complex implicit feedback (in the case of catalog information) significantly increases the nDCG score comparing the models trained on only visited item feedback. However, once we tried to use complex implicit feedback together with metadata information (in the case of Model 6), we could not obtain better performance. We also compared the performance of Model 5, 6, and 7 on catalog and non-catalog predictions as shown in Figure 4.15.

Model Name	Precision	Recall	MRR	nDCG
model_05_8_epoch_512_batch	6.222%	62.222%	32.237%	39.331%
model_07_8_epoch_512_batch	6.169%	61.691%	30.694%	38.005%
model_06_8_epoch_512_batch	5.692%	56.917%	28.160%	34.945%
model_04_8_epoch_512_batch	5.165%	51.655%	29.400%	34.642%
model_02_8_epoch_512_batch	4.885%	48.850%	27.840%	32.793%
base_item2vec_model	2.440%	24.399%	9.517%	12.981%
cosine_similarity	2.044%	20.442%	9.928%	12.426%
model_01_8_epoch_512_batch	0.869%	8.687%	3.451%	4.663%
base_doc2vec_model	0.170%	1.701%	0.602%	0.854%
model_03_8_epoch_512_batch	0.051%	0.510%	0.154%	0.234%

Model Name	Precision	Recall	MRR	nDCG
model_05_8_epoch_512_batch	5.648%	56.482%	26.374%	33.484%
model_07_8_epoch_512_batch	5.579%	55.789%	25.097%	32.321%
model_06_8_epoch_512_batch	4.933%	49.335%	21.240%	27.842%
model_04_8_epoch_512_batch	3.895%	38.948%	15.339%	20.873%
model_02_8_epoch_512_batch	3.586%	35.858%	14.640%	19.616%
base_item2vec_model	2.206%	22.061%	8.593%	11.726%
cosine_similarity	1.239%	12.389%	4.781%	6.559%
model_01_8_epoch_512_batch	0.403%	4.031%	1.192%	1.844%
base_doc2vec_model	0.159%	1.593%	0.559%	0.796%
model_03_8_epoch_512_batch	0.038%	0.376%	0.120%	0.178%

Figure 4.14: Evaluation results for the sequences. Overall, Model 5 is the best-performing model and Model 3 is the worst one. Left: The results do not have any filter like *catalog_prediction* or *trivial_prediction*. Right: Evaluation results for the recommendations that are labeled as non-trivial. While the model raking stayed the same, we see that all evaluation metrics decreased.

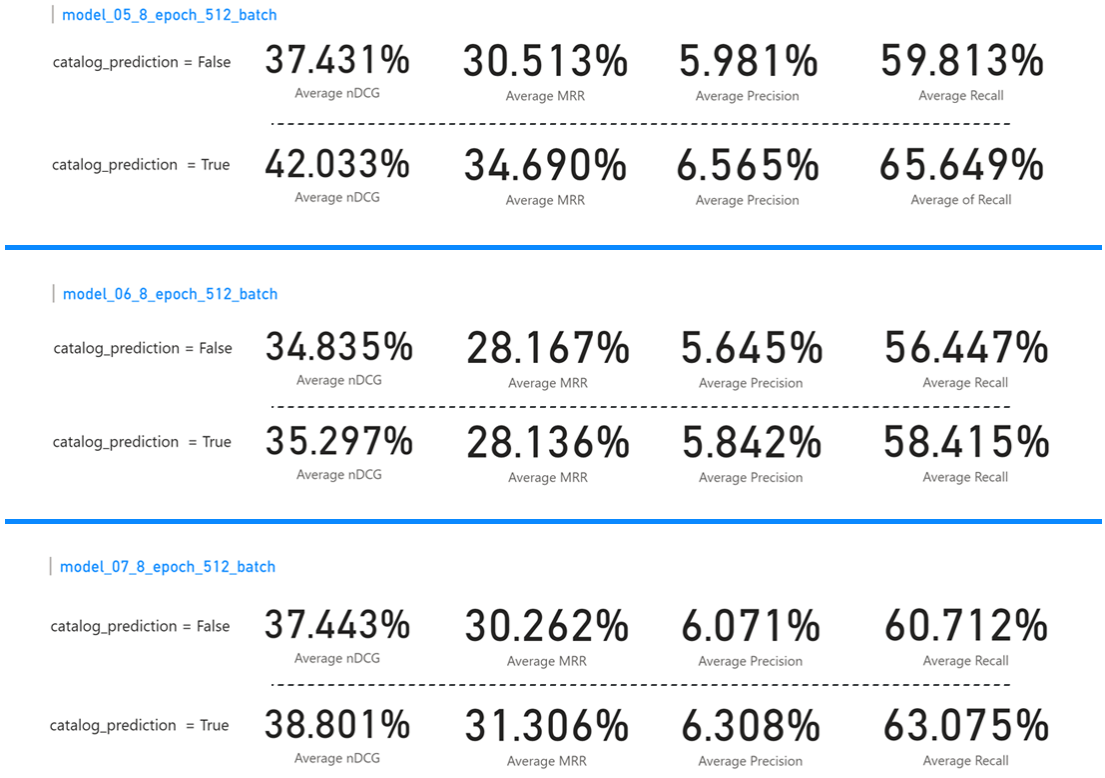


Figure 4.15: Evaluation result of Model 5, 6 and 7 on catalog and non-catalog predictions.

Additionally, we applied some limitations to understand how each model performed on specific sequences. Figure 4.16 presents the model performance on the user sequences which contains 2, 5, 7 and 10 object visits. Once we segmented the model performance based on the sequence length, we see that models are generally performing better once the user sequence is short. Finally, as it is

shown in Figure 4.17 we compared Model 2 with Model 5 on the user sequences which contains only dedicated tour visits to understand the performance of a model trained with additional catalog page information. Since Model 2 is not able to perform predictions for the catalog pages, the comparison only contains non-catalog predictions.

Model Name	2	4	6	8	10
model_02_8_epoch_512_batch					
Precision	6.486%	4.743%	3.705%	3.912%	3.473%
Recall	64.863%	47.429%	37.047%	39.119%	34.726%
MRR	45.759%	23.198%	16.927%	17.211%	15.315%
nDCG	50.343%	28.885%	21.622%	22.340%	19.852%
model_05_8_epoch_512_batch					
Precision	7.194%	6.670%	6.034%	5.747%	5.560%
Recall	71.940%	66.705%	60.342%	57.469%	55.604%
MRR	47.099%	34.967%	27.829%	24.906%	24.003%
nDCG	53.066%	42.501%	35.491%	32.570%	31.397%
model_06_8_epoch_512_batch					
Precision	6.514%	6.378%	5.541%	5.197%	4.866%
Recall	65.143%	63.777%	55.406%	51.974%	48.656%
MRR	37.133%	32.479%	24.795%	22.566%	20.871%
nDCG	43.806%	39.879%	31.979%	29.448%	27.404%
model_07_8_epoch_512_batch					
Precision	7.071%	6.629%	6.003%	5.723%	5.486%
Recall	70.707%	66.289%	60.030%	57.232%	54.863%
MRR	44.611%	32.357%	26.904%	23.770%	23.437%
nDCG	50.843%	40.367%	34.699%	31.610%	30.788%

Model Name	2	4	6	8	10
model_05_8_epoch_512_batch					
Precision	6.958%	6.777%	6.167%	6.057%	5.525%
Recall	69.581%	67.770%	61.670%	60.573%	55.254%
MRR	39.722%	35.872%	29.189%	29.172%	28.028%
nDCG	46.865%	43.425%	36.880%	36.603%	34.495%
model_06_8_epoch_512_batch					
Precision	5.964%	6.159%	5.185%	6.000%	2.500%
Recall	59.644%	61.587%	51.852%	60.000%	25.000%
MRR	30.007%	30.280%	21.321%	30.708%	4.167%
nDCG	37.044%	37.689%	28.467%	37.612%	8.905%
model_07_8_epoch_512_batch					
Precision	6.600%	6.548%	5.894%	5.839%	5.366%
Recall	65.997%	65.480%	58.936%	58.394%	53.659%
MRR	36.480%	32.643%	26.774%	25.260%	23.559%
nDCG	43.497%	40.384%	34.342%	33.036%	30.671%

Figure 4.16: Performance comparison of Model 2, 5, 6 and 7 on the different lengths of user sequences. Left: Overall results without any filters. Right: Evaluation results for recommendations labeled as *catalog_prediction*. Note that since Model 2 is not able to make recommendations for catalog pages, it is not included to the right table.

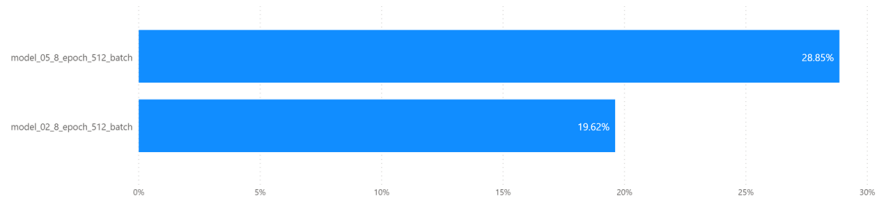


Figure 4.17: nDCG Score of Model 2 and Model 5 on the non-catalog predictions. It is possible to see that the utilizing catalog data during the training allows Model 5 to perform better than the Model 2.

Conclusion

In this thesis, we focus on using Recurrent Neural Network with Long Short Term Memory and try to improve the performance of session-aware recommendations using real-life user implicit feedback and content metadata. One of our works' contributions is to help increasing the recommendation quality on small e-commerce domains where the density of anonymous users is high, and the length of a user session is short.

Different types of RNN models that use various encoding strategies have been implemented and evaluated on a real-life dataset. It has been shown that usage of implicit feedback affect recommendation quality directly. Throughout the thesis, four evaluation metric is used and between all the metrics nDCG is given the most priority due to its success in evaluating the position of ranked recommendations.

We addressed the problem of data sparsity in small e-commerce domain. Our experiments show that utilizing specific implicit feedback increases the quality of the recommendations. It is possible that our experiment can be applied for all the other data sets that satisfy certain criteria mentioned in Section 4.

Regarding future work, there exist two main ways to improve. As experimented, encoding the implicit feedback plays a crucial role in the predictions. We believe that further research on feature/label encoding could increase the presented results. Additionally, it is possible to investigate further improvements to the implemented RNN models. Changing the model architecture and its parameters may lead to performance increase. Other than that, following is the some ideas that can be tested:

1. Custom weight initialization is an important topic of machine learning. Instead of using default weight initialization, it is possible to try a different kernel and recurrent initializers such as GlorotNormal, He Initializer, etc..

2. Implementing a custom loss function for the classification task. We trained the classifiers by using *categorical_crossentropy* loss. One further improvement could be implementing a custom loss function that can utilize evaluation metrics to calculate the loss.

3. It could be useful to use different methods like Fuzzy Logic while segmenting the numerical values like price and duration before encoding the content metadata.

Bibliography

[1] Anonymous. History of advertising: Early digital marketing, 2000-2010. URL https://www.needls.com/university/blog/history-of-advertising-digital-marketing-2000-2010/.

[2] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple object recognition with visual attention.

[3] Oren Barkan and Noam Koenigstein. Item2vec: Neural item embedding for collaborative filtering.

[4] Adarsh Bhadauria. What is a neural network? visualising & understanding a neural network in-depth. Online, August 2019. URL https://articles.adxy.in/2019/08/23/introduction-to-neural-networks-and-visualisation/.

[5] Jan Boehmer, Yumi Jung, and Rick Wash. Recommender systems.

[6] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling.

[7] Jeff Desjardins. Theory of point estimation. Online, April 2019. URL https://www.weforum.org/agenda/2019/04/how-much-data-is-generated-each-day-cf4bddf29f/.

[8] Carlos A. Gomez-Uribe and Neil Hunt. The netflix recommender system. *ACM Transactions on Management Information Systems*, 6(4):1–19, jan 2016. doi: 10.1145/2843948.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[10] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation.

[11] Balázs Hidasi, Massimo Quadrana, Alexandros Karatzoglou, and Domonkos Tikk. Parallel recurrent neural network architectures for feature-rich session-based recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, sep 2016. doi: 10.1145/2959100.2959167.

[12] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks.

[13] Josef Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. Master's thesis, Institut fur Informatik Technische Universit at Munchen, June 1991. URL http://people.idsia.ch/~juergen/SeppHochreiter1991ThesisAdvisorSchmidhuber.pdf.

[14] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, nov 1997. doi: 10.1162/neco.1997.9.8.1735.

[15] F.O. Isinkaye, Y.O. Folajimi, and B.A. Ojokoh. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, 16(3): 261–273, nov 2015. doi: 10.1016/j.eij.2015.06.005.

[16] Sheena Iyengar and Mark Lepper. When choice is demotivating: Can one desire too much of a good thing? *Journal of personality and social psychology*, 79:995–1006, 01 2001. doi: 10.1037/0022-3514.79.6.995.

[17] Kalervo Järvelin and Jaana Kekäläinen. IR evaluation methods for retrieving highly relevant documents. *ACM SIGIR Forum*, 51(2):243–250, aug 2017. doi: 10.1145/3130348.3130374.

[18] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. Online, May 2015.

[19] Le, Ho, Lee, and Jung. Application of long short-term memory (lstm) neural network for flood forecasting. *Water*, 11(7):1387, Jul 2019. ISSN 2073-4441. doi: 10.3390/w11071387. URL `http://dx.doi.org/10.3390/w11071387`.

[20] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents.

[21] SeungGwan Lee and DaeHo Lee. A personalized channel recommendation and scheduling system considering both section video clips and full video clips. *PLOS ONE*, 13(7):e0199748, jul 2018. doi: 10.1371/journal.pone.0199748.

[22] J. Lees. *Lees' Guide to the Game of Draughts Or Checkers*. David McKay Company, 1931. URL `https://books.google.cz/books?id=499JMwEACAAJ`.

[23] Clare Liu. More performance evaluation metrics for classification problems you should know. Online, April 2020. URL `https://www.kdnuggets.com/2020/04/performance-evaluation-metrics-classification.html`.

[24] John McCarhy. Arthur samuel: Pioneer in machine learning. URL `http://infolab.stanford.edu/pub/voy/museum/samuel.html`.

[25] Tom M. Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/-Math; (March 1, 1997), 1997.

[26] Maryam Mohsin. 10 amazon statistics you need to know in 2020 [infographic], April 2020. URL `https://www.oberlo.com/blog/amazon-statistics`.

[27] Netflix. Online. URL `https://www.netflixprize.com/index.html`.

[28] Christopher Olah. Understanding lstm networks. Online, August 2015. URL `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[29] Esteban Ortiz-Ospina. The rise of social media. Online, September 2019. URL `https://ourworldindata.org/rise-of-social-media`.

[30] Ladislav Peska. *Recommender Systems in E-Commerce - Methods, Models, Experiments*. PhD thesis, Charles University in Prague, April 2016.

[31] Saimadhu Polamuri. Collaborative filtering recommendation engine implementation in python. Online, May 2015. URL https://dataaspirant.com/collaborative-filtering-recommendation-engine-implementation-in-python.

[32] Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. Sequence-aware recommender systems.

[33] R. Reed and R.J. MarksII. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. A Bradford Book. MIT Press, 1999. ISBN 9780262181907. URL https://books.google.cz/books?id=sreDQgAACAAJ.

[34] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender Systems Handbook*, pages 1–35. Springer US, oct 2010. doi: 10.1007/978-0-387-85820-3_1.

[35] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors. *Recommender Systems Handbook*. Springer US, 2011. doi: 10.1007/978-0-387-85820-3.

[36] Frank Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, 1962. URL http://csis.pace.edu/~ctappert/srd2011/rosenblatt-contributions.htm.

[37] Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM Press, 2002. doi: 10.1145/564376.564421.

[38] Guy Shani and Asela Gunawardana. Evaluating recommender systems. Technical Report MSR-TR-2009-159, November 2009. URL https://www.microsoft.com/en-us/research/publication/evaluating-recommender-systems/.

[39] Alexander Sidorov. Transitioning entirely to neural machine translation. Online, August 2017. URL https://engineering.fb.com/ml-applications/transitioning-entirely-to-neural-machine-translation/.

[40] sklearn. sklearn.metrics.f1_score. Online. URL https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html.

[41] Leslie N. Smith. A disciplined approach to neural networkhyper-parameters: Part 1 – learning rate,batch size, momentum, and weight decay. *US Naval Research Laboratory Technical Report 5510-026*, April 2018. URL https://arxiv.org/pdf/1803.09820.pdf.

[42] Przemek Vincent Szustak. Less is more? the paradox of choice. Online, March 2020. URL https://medium.com/@przemekszustak/less-is-more-the-paradox-of-choice-behavioural-economics-in-ux-318849b2d70.

[43] Thomas Tennyson. Online. URL https://www.ediflo.tv/blog/how-netflix-should-improve-recommendations.

[44] Shimon Ullman, Tomaso Poggio, Danny Harari, Daneil Zysman, and Darren Seibert. Unsupervised learning. Online, September 2014. URL http://www.mit.edu/~9.54/fall14/slides/Class13.pdf.

[45] Yuyu Zhang, Hanjun Dai, Chang Xu, Jun Feng, Taifeng Wang, Jiang Bian, Bin Wang, and Tie-Yan Liu. Sequential click prediction for sponsored search with recurrent neural networks.

[46] Yangchang Zhao. Chapter 5 - regression. In Yangchang Zhao, editor, *R and Data Mining*, pages 41 – 50. Academic Press, 2013. ISBN 978-0-12-396963-7. doi: https://doi.org/10.1016/B978-0-12-396963-7.00005-2. URL http://www.sciencedirect.com/science/article/pii/B9780123969637000052.

[47] Erion Çano and Maurizio Morisio. Hybrid recommender systems: A systematic literature review. *Intelligent Data Analysis*, 21(6):1487–1524, Nov 2017. ISSN 1088-467X. doi: 10.3233/IDA-163209.

List of Figures

List of Tables

List of Code Snippets