Cross Platform Recommender Application Development with Machine Learning

By

Emre Havan

Supervised by

Dr Martin Berger

MSc Advanced Computer Science
School of Engineering and Informatics
University of Sussex
Summer 2019

Abstract

Recommender systems are automated software models that provides suggestions for contents to the users. Recommender systems are often created with an available dataset, and hence, stable once it is deployed. But the content or user data might change over time and it is vital to respond to these changes. It could be beneficial to design recommender systems to utilize user feedback for dynamization. This project aims to create recommender system application by comparing, discussing and applying different algorithms and development frameworks on four major points. First, two versions of recommender engine developments will be discussed by comparing Random Forest Classifier and Neural Networks on synthetically generated dataset. Second, an implementation of a recommender engine based on Cosine Similarity will be demonstrated on genuine, scraped dataset. Third, a set of functions will be introduced to cope with user feedback and the results of the implementation will be discussed. Finally, the project will compare two cross-platform frameworks React Native and Flutter and will demonstrate development processes of prototypes and resulting user interfaces. Findings of each point will be discussed extensively, and satisfactory results of recommender system applications will be demonstrated. The project later will be concluded by remarking possible further improvements for a dynamic recommender system application.

ACKNOWLEDGEMENTS

I would like to thank Dr Martin Berger for his expertise, guidance and assistance throughout the processes of application development and composing this dissertation.

Table of Contents

1. INTRODUCTION

Decision making is a critical and widely applied activity in human life. Humans have been making decisions for centuries, both for fundamental areas in their life such as: survival, career selection, partnership, and for less important, daily or leisure related areas such as: deciding what to wear, what to eat, what to watch, what to purchase and etc. Making a decision might not be trivial for some cases, especially if there are many options to choose from. Often people consider asking to a friend, to a relative or a mentor for recommendations or advices in order to make decisions. Getting relative recommendations might allow people to make better and confident decisions. Although it is useful to get recommendations from known people, this usually depends on a few people, whom have already gained some experience for a specific context, which might limit the sources to get recommendations from.

Recommender Systems (RS) are automated systems that are designed to provide a relevant and accurate set of recommendations for the current context of the application area. Nowadays it is very common to come across with an RS on web and mobile applications such as Amazon, Netflix and etc as shown in Figure-1 below [1]. They are usually developed and applied in e-commerce, movie, music, news, food, restaurants and in many other applications. A RS system often consists of a variety of different algorithms and approaches to determine how to make an accurate and useful recommendation for users.

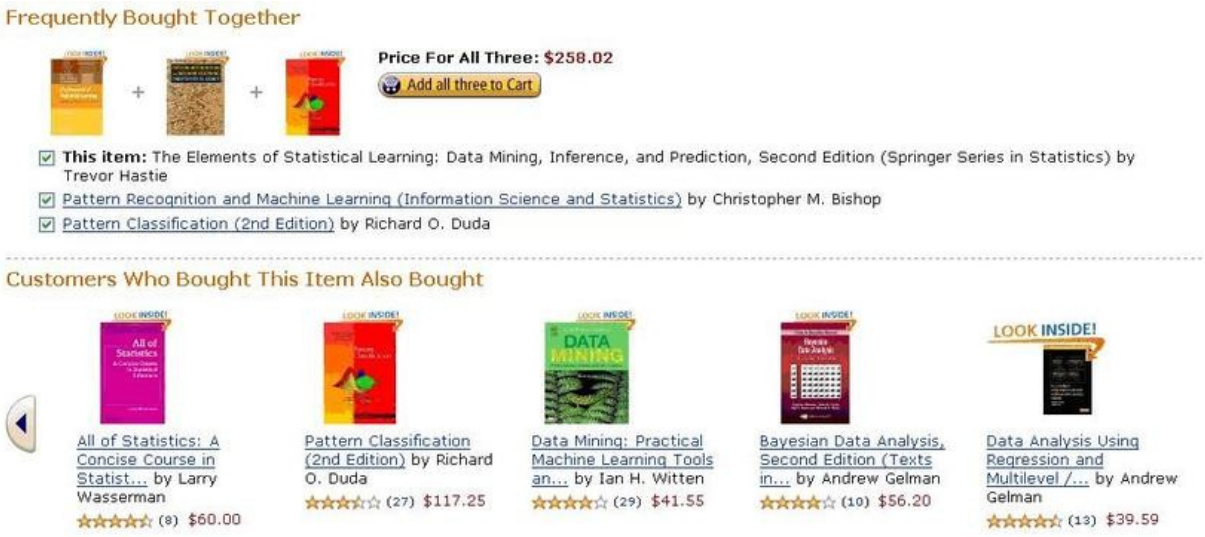

Figure 1, Content recommendation example on amazon.com [38]

RS are also instrumented in travel and tourism related applications. Travelling around the world is a popular activity for humans. It is a growing industry especially with the latest improvements and availability in world-wide transportation, which results in an increase of the destination options for anyone around the world. Therefore, RS can alleviate the process of trip planning by making contextual recommendations. RS can be used to create ad hoc recommenders within a travel or location-based applications or they could be used to create a personally tailored application, designed to deliver a complete trip package. TripAdvisor and Foursquare can be given as examples for travel recommender applications, where users can

rate and review places such as restaurants, bars and coffee shops [2,3]. Then these ratings are accumulated and fed into some algorithms to make recommendations in the future. Even though such applications can provide a set of places to visit once a travel destination is decided, it might still be an issue to make a destination decision in the first place for some users. There are a variety of possible cities/destinations to visit for travellers. A RS system may contribute to this decision in a few different ways depending on the scope of the application, and user and item data available at hand. While user demographics related data (race, gender, location and etc) can be used to provide a more general recommender, data related with user's preferences and previous opinions for a set of items, can be used to make more tailored and specific recommendations.

In today's world, it is important to develop applications for both iOS and Android platforms to make sure it will be available for anyone with a mobile device, regardless of the model or the operating system. Nowadays, there are two major different ways of developing mobile applications, native and cross-platform. Native development is the process of developing an application for a particular operating system often with the tools provided by the producer of the operating system, and hence developing two different applications for two different platforms, whereas cross-platform development simply allows developing a single application to be deployed on multiple operating systems. The details of native and cross-platform development are discussed in chapter 7.

Figure 2, Cross platform development for iOS and Android platforms [39]

1.1 Cross Platform Recommender Application Development

As mentioned above, there are a variety of different approaches to develop a recommender system for any kind of context and user platform. Usually RS are built in a static way with available dataset, and it can be a challenge to adopt user feedback to keep the system updating itself to work dynamically for a better performance after system deployment. This project has two core dimensions, the first one is to develop a dynamic destination RS by comparing different algorithmic approaches and utilizing feedback in different forms and the second one is to create a user interface (UI) for the developed RS while comparing different cross-platform frameworks.

2. INTRODUCTION TO MACHINE LEARNING

Machine Learning (ML) is a statistical algorithmic method, which is applied to computer programs to allow the program for learning from experience while working on a specific set of tasks by measuring its performance [44]. Simply put, ML is a way of acquiring insight from a dataset based on the data attributes. Most of the ML algorithms are repetitive processes, which ideally improves performance for the given task criteria after each run. ML algorithms are applied to datasets in order to understand the insight and the relationship between the features (data attributes), which is also called training. Datasets are often divided into two portions where a ML model is first trained on one portion and later makes predictions on the other one. ML algorithms are divided in two categories such as supervised learning and unsupervised learning. In supervised learning, data points have corresponding output values (often called labels) where there are no output values available on datasets in unsupervised learning. There are a variety of different algorithms and approaches for building recommending systems, but only the ones applied in the project are introduced in this chapter.

2.1 Classification

Classification in ML is a Supervised Learning technique. It is the process of identifying some data samples with respect to some characteristics which are often called features, in order to categorize data into some distinct groups [13]. Classification could be implemented by building a model consisting of a ML algorithm or a combination of different ML algorithms and techniques. These models are later trained with a labelled (class of each data sample is known) dataset to understand the relation of the features for a given data sample and acquire insight in order to make classification later on with unlabelled (class of each data sample is unknown) data. Classification can be divided to two sub categories as; binary classification (where there is only two possible category that a given data sample could be identified as) and multi-class classification (where there are more than two possible categories that a given data sample could be identified as). In this project, both binary and multi-class classifiers are implemented. There are several different algorithms and methods to use for classification such as: Support Vector Machines (SVM), k-Nearest Neighbour (kNN), Gaussian Naive Bayes (GNB), Random Forest (RF), Neural Networks and etc [14]. In this project only, RF and Neural Network classifiers are implemented for classification tasks.

2.1.1 Random Forest Classifier

It is important to understand how Decision Trees (DT) work before looking into Random Forest (RF) classifiers. DT are a supervised learning method used for classification and regression. DT are trained on a dataset to create some if-then-else decision rules with respect to feature values. Then these rules are applied on data samples in sequences for classification with respect to pre-defined classes. Each rule is based on a feature to narrow down the available class labels for the particular data sample. Eventually all these rules lead to a node where class value is determined [16].

RF classifier is a type of classifier built using a collection of tree classifiers, where each tree classifier performs predictions for some randomly chosen subset of the given dataset.

Then each tree classifier makes prediction with their own data portion and votes a predicted class for a given input. RF then aggregates these votes from each three and then makes the prediction based on the mode (most often predicted value) of all votes from the trees in the ensemble [17]. Visual representation of RF classifier is shown in Figure-3 below.

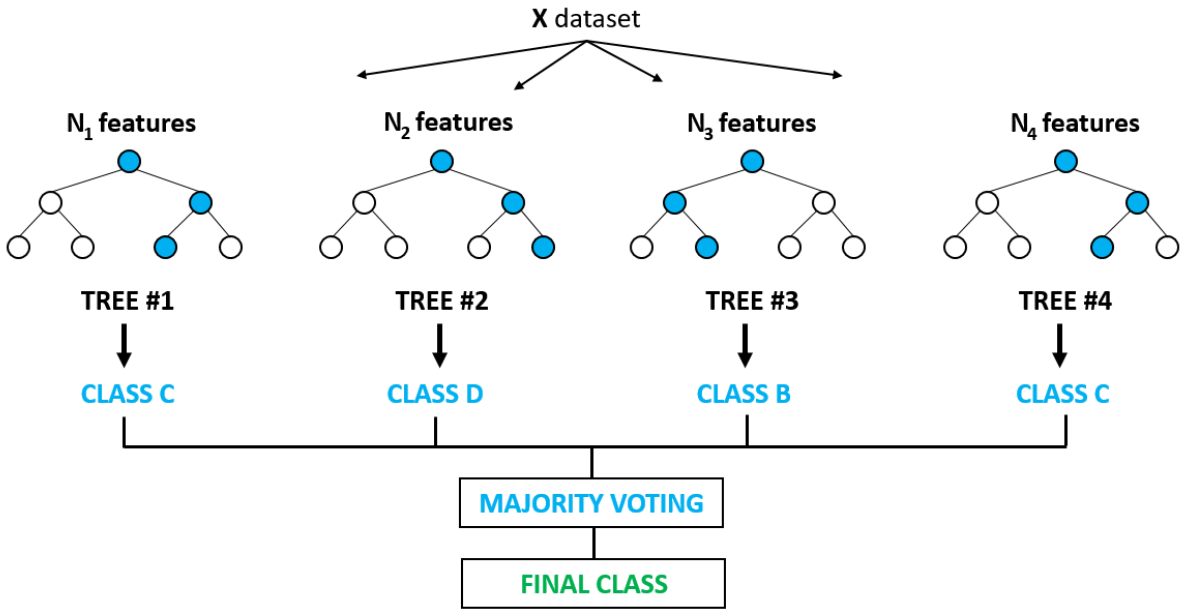

Figure 3, Random Forest Classifier example [18]

2.1.2 Neural Networks

Neural networks are computing models that are initially influenced by biological neurons, built in order to learn relationships between feature values for classification. Neural networks consist of multiple neurons which are connected to each other with weights. Weights can be interpreted as a factor which determines the importance of the connection between two neurons. Neural networks operate in layers where each layer consists of a number of neurons and connected to one layer before and after. There are three types of layers; input layers, hidden layers and output layer. Input layer feeds the data to the network and then forwards to hidden layers, where hidden layers create the body of the network where all the neurons are interconnected with weights, and finally output layer executes the classification with the aggregated data at previous layers [20]. Visual presentation of a fully connected neural network is shown in Figure-4 below.

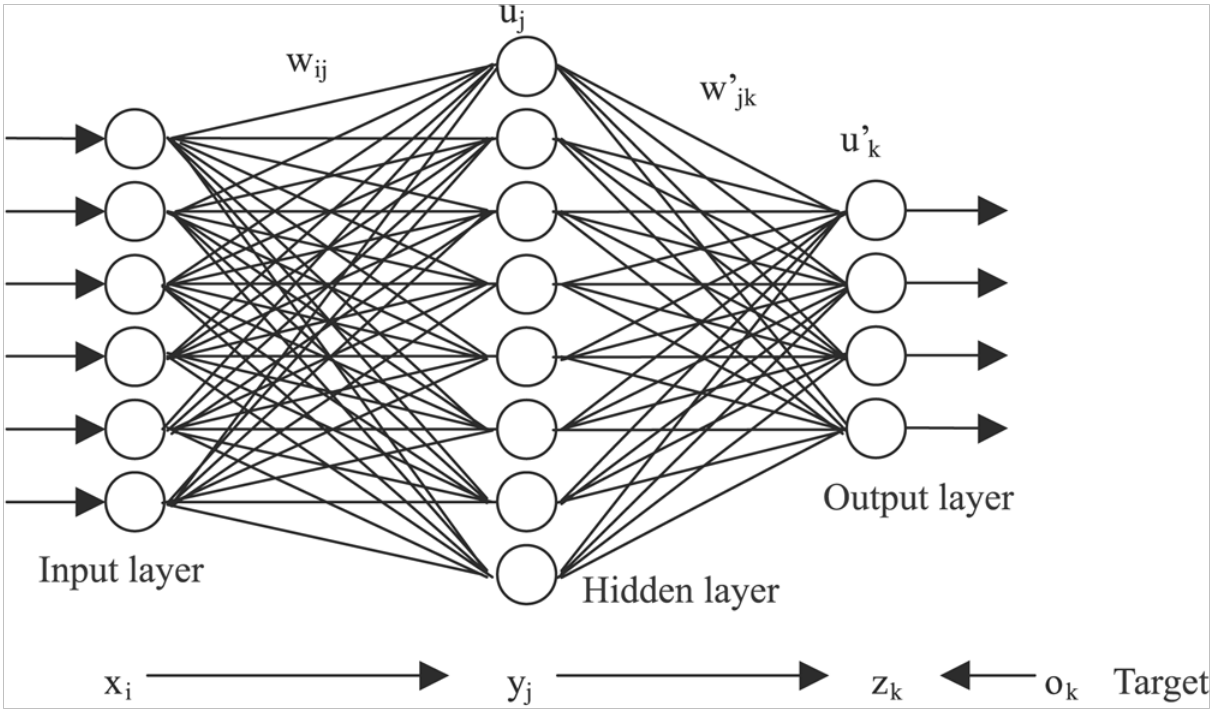

Figure 4, Fully connected Neural Network Sample [21]

Learning of a neural network is implemented with Backpropagation which is a learning algorithm that adjusts the weights for each neuron connection. All neurons in a layer creates a weighted sum value from the outputs of previous layer, adds a bias and then runs the value through an activation function to produce an output value for the next layer. Backpropagation is an iterative process which attempts to recognize patterns between input and the associated output class in a given training data to create some sort of map which could later be used to classify unseen data with similar characteristics (pattern). Inaccuracy for a class prediction of a given data sample is described in Loss function. During the iterative process of Backpropagation, the calculated loss value is fed backwards to the network and then weights are updated accordingly to minimize the loss function. Eventually the network reaches to an optimal minimum loss function for each class patterns at the end of training [22]. It is worth mentioning that, training of neural networks often takes longer than other classification methods and requires more data.

As mentioned earlier, activation functions determine how neuron outputs will behave in a neural network. Where some cases neuron outputs may behave in a binary way (active-inactive) or output can vary between a specific range and etc depending on the function type. There are two activation functions are considered and applied for the neurons of the neural network during implementation, which are Sigmoid and ReLu.

Sigmoid function is a non-linear activation function, which produces a positive output value between zero and one with regard to input parameter. Equation of Sigmoid function is given below.

$$S(x) = \frac{e^x}{e^x + 1} \quad (1.3)$$

As can be seen from the Eqn. (1.3) no matter how high or low input x is, the activation function makes sure it resides between 0 and 1, thus prevents the output variation among neurons to a minimal range but activates every neuron regardless if their output is positive or negative which increases the cost and computation time required for the training. Visual representation of Sigmoid function is given in the Figure-5 below.

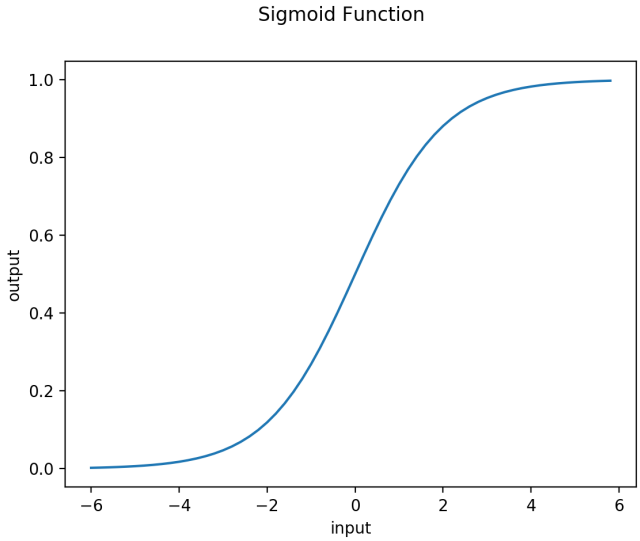

Figure 5, Sigmoid Function

ReLu makes sure the output value is greater than zero by producing the output as input itself if the input is greater than zero or as zero when the input is less than zero. Equation of ReLu is given below.

$$f(x) = \max(0, x) \quad (1.4)$$

ReLu allows neural networks to train faster by activating and de-activating some neurons for specific patterns, but it might suffer from the exploding activation problem with regard to output variations among neurons where changes of weights in the network updates the values greatly and makes neural network unable to learn. Another problem might occur is called dying ReLu problem, where gradient descent reaches to zero and stops updating the weights hence the network loses ability to learn [23, 24]. Visual representation of ReLu function is given in the Figure-6 below.

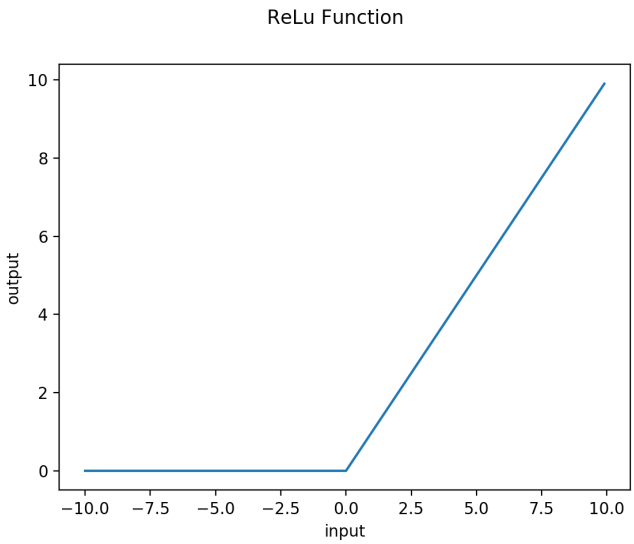

Figure 6, ReLu Function

3. RECOMMENDER SYSTEMS

There is massive amount of content created and offered to users in a variety of different industries in today's world. Often the available content is different from one another in terms of characteristics and features. Users are no different and their preferences also differ from one another and it may be difficult to access to content with their taste among all of the content available. RS is a way offering, displaying and recommending content to the users with respect to data available about users and/or contents. It is highly beneficial both for content providers and users to create systems that recommends and offers relevant content to relevant users. There are a variety of different approaches and algorithms to utilize a RS such as; classification with ML, scoring functions, collaborative filtering, content-based recommendation and etc [1]. It is possible to build an RS with any of the mentioned approaches or with combination of them (hybrid) depending on the data available. Basic form of collaborative filtering and content-based recommending for articles as example is shown in Figure-7 below.

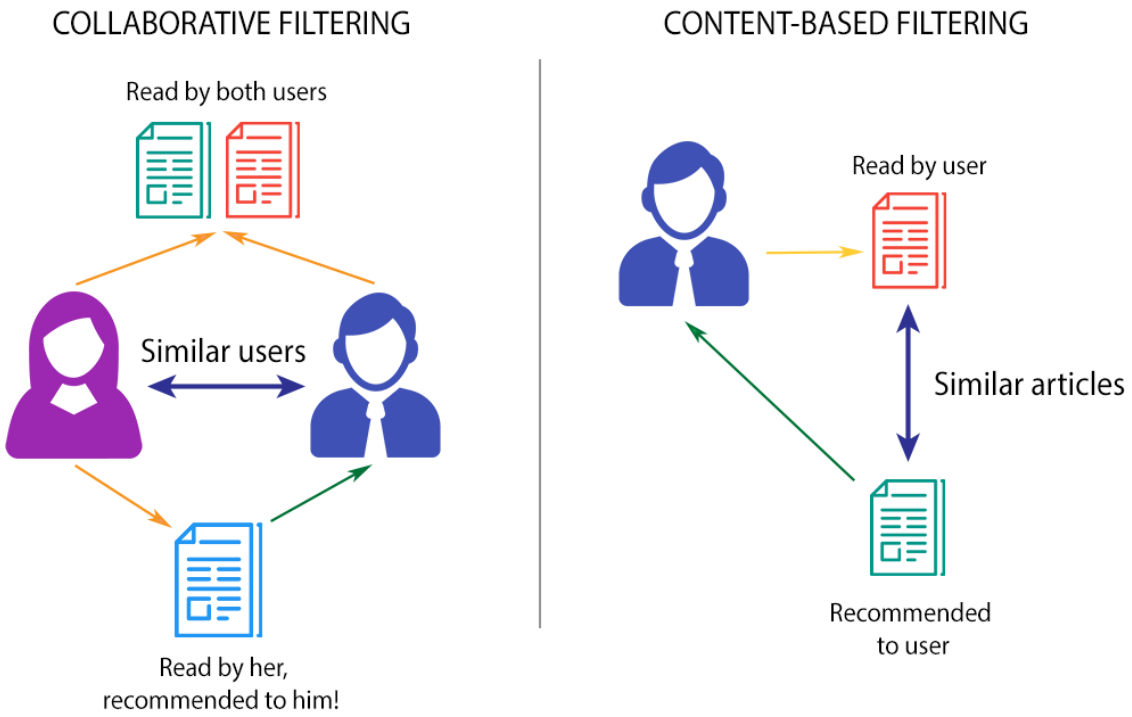

Figure 7, Collaborative Filtering and Content-Based Recommendation [28]

3.1 Collaborative Filtering

Collaborative Filtering is a personalized recommendation method that makes predictions and recommendations on a user to user basis. It is understandable for people with similar taste to like same or similar contents. For instance, a content provider might be saving user feedback to generate data to later use in a RS. Then this accumulated user feedback on contents are examined to compute user similarities. For the case there are two users, user-A and user-B, who have rated contents close to each other (giving high and low ratings to same contents), RS first computes the similarity between user-A and user-B among the contents

they both rated to identify them as similar users. After that, RS scans the contents rated higher than the average rating by user-B but not rated by user-A to create a recommendation set for user-A [27]. It is worth mentioning that Collaborative Filtering often requires a great amount of data to yield a good performance. There are a variety of ways to achieve collaborative filtering and the common two ways can be given as;

- **Neighbourhood-based approach:** This is the main approach of collaborative filtering as described above, where a set of users similar to current user are determined based on rating similarity and then top n number of users (neighbours) are selected and prediction and/or recommendation set is computed with weighted calculation of neighbouring user's ratings [29].

- **Item-based approach:** Since applying neighbourhood method to a database where there are millions of users and items exists is computationally expensive, it is easier and faster to compute recommendations by only considering similar items to user's previously rated items [29].

3.2 Content-based recommendation

Content-based recommendation is a personalized recommendation method that makes recommendations on item to item basis. Systems suited with content-based approach usually have no or small amount of data about users but have a large content database with features or keywords describing each item. Content-based approach requires previous item consumption data from users in order to find similar items for recommendation. This data could be in form of previous purchases, explicit feedback such as rating or even can be based on how long user spends on content page as implicit feedback. Although content-based approach might be useful in a sense that users are more likely to consume items related with their previous consumptions, it may not be ideal since users may wish to get recommendations from other interesting products as well and user's interests may also change, and recommendations may be irrelevant overtime [27]. Thus, relying only on content-based approach might not be ideal for a RS, instead combination with other techniques can create better performing RS as a hybrid system.

3.3 Similarity Functions

Similarity functions are a way of computing similarities among users or items, they are instrumented in collaborative filtering and content-based recommendation, in order to compute similarities and/or calculate prediction score for specific contents. As well as being a helper method in RS techniques introduced earlier, they can also act as a sole RS by themselves. There are different kind of functions used in RS such as; Cosine Similarity, Euclidean Distance, Pearson correlation coefficient and etc.

- **Cosine Similarity:** Cosine similarity is the measure of similarity between two vectors, by computing the cosine of the angle between two vectors projected into multidimensional space. It can be applied to items available on a dataset to compute similarity to one another via keywords or other metrics [30]. Similarity

between two vectors (A and B) is calculated by taking the dot product of the two vectors and dividing it by the magnitude value as shown in Eqn. (1.1) below.

$$\boldsymbol{cos(\theta)} = \frac{\boldsymbol{A.B}}{\boldsymbol{||A||.||B||}} = \frac{(\sum_{i=1}^{n} \boldsymbol{Ai.Bi})}{\sqrt[2]{\sum_{i=1}^{n}(\boldsymbol{Ai})^2}\ \sqrt[2]{\sum_{i=1}^{n}(\boldsymbol{Bi})\text{^}\boldsymbol{2}}} \quad (1.1)$$

- **Euclidean Distance:** Euclidean Distance is the distance between two vectors on a multidimensional space which can also be interpreted and applied as a measure of similarity. For instance, it is possible to detect how similar two items with respect to their feature values or keywords (each might represent a dimension on the space) to each other with Euclidean Distance. The distance d between two vectors a and b in a n-dimensional space is shown in Eqn. (1.2) below [31].

$$\boldsymbol{d(a,b)} = \sqrt{\sum_{i=1}^{n}(\boldsymbol{ai} - \boldsymbol{pi})^2} \quad (1.2)$$

Both Cosine Similarity and Euclidean Distance functions are used as a part of a RS (helper function in collaborative filtering and content-based recommendation and etc) as well as being used as a sole RS. Even though Cosine Similarity and Euclidean Distance functions are similar to each other, they are not identical. The magnitude of vectors is important in Euclidean Distance whereas it does not change the similarity score in cosine similarity since a magnitude change of any vector does not affect its direction and the angle between the vectors. It is important to take this difference into account before selecting a similarity function to prevent inconsistent recommendations especially when the data or feature space is not normalized. Euclidean Distance and Cosine Similarity measurements for same vectors are displayed in Figure-8 below.

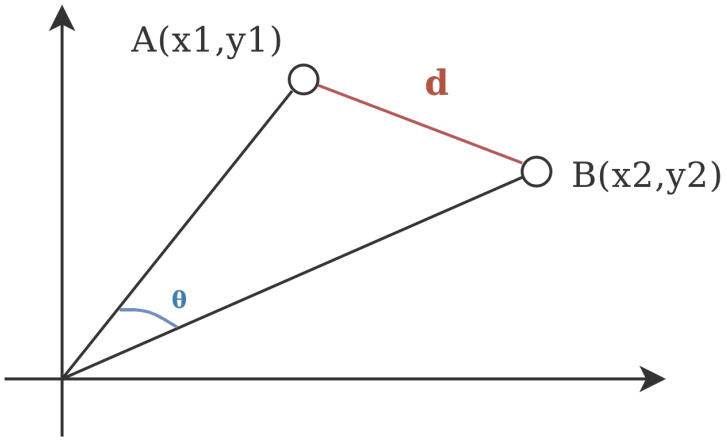

Figure 8, Cosine Similarity and Euclidean Distance [32]

As shown in Figure-8, cosine similarity is determined by θ and Euclidean distance is represented by d. Although they both can provide a good metric for calculating similarities among data points, it is important to observe that, for instance, if x and y values of vector B is multiplied by a scalar value that is greater than one, the distance between both vectors would increase, whereas it would not affect the angle between two vectors hence resulting in no

change for cosine similarity. Thus, it is crucial to identify the characteristics of data, if it is normalized, or if the scale and magnitude is important before deciding the function.

4. RECOMMENDATION WITH CLASSIFICATION

In this chapter, only Random Forest Classifiers and Neural Networks are used to train models on an auto generated dataset with some insight, to observe the performance and behaviour of the models.

4.1 Data Generation

Before proceeding with a classifier model, it is important to have a dataset ready to start with. For classification approach to a recommender system, a set of travel related user data generated with random number generators. Each data sample consists of a set of features which indicate some demographic information about the user and a label information indicating where the user has travelled to. Features are;

- Age: Indicating the age of the user (Varies between 18 and 65)
- Budget: Indicating the budget a user had for the trip (Varies between 500-5000)
- Season: Indicating the season the user has travelled (Summer, Autumn, Winter, Spring)

Assigned label for each sample;

- Destination: Indicating the city that a user with specific feature values travelled to (Possible destinations: Istanbul, London, Prague, Amsterdam, Antalya, Dubrovnik, Ibiza, Hvar, Verbier, Vogel, Bangkok, Sao Paulo)

For simplicity, there are only 12 possible destinations that are assigned to generated samples as label. As has been shown, the selected cities have some characteristics. For example, it is expected for a user to decide going to Ibiza or Hvar during summer and going to Verbier or Vogel (skiing premises) in winter and etc. Full insight behind the generated data is shown in Table-1 below.

Table 1 - Data Generation Insights

Season	Age	Budget	Destination
Summer	<= 40	<= 2500	Hvar
Summer	<= 40	> 2500	Ibiza
Summer	> 40	<= 2500	Antalya
Summer	> 40	> 2500	Dubrovnik
Autumn	<= 40	<= 2500	Prague
Autumn	<= 40	> 2500	Amsterdam
Autumn	> 40	<= 2500	Istanbul
Autumn	> 40	> 2500	London
Winter	<= 40	<= 2500	Vogel
Winter	<= 40	> 2500	Verbier
Winter	> 40	<= 2500	Sao Paulo
Winter	> 40	> 2500	Bangkok
Spring	<= 40	<= 2500	Prague
Spring	<= 40	> 2500	Amsterdam
Spring	> 40	<= 2500	Istanbul
Spring	> 40	> 2500	London

Even though Table-1 indicates that the data generated with certain boundaries, there are some alterations in the data generation code which allows the boundaries to be softer. For instance, although a sample with features of season = Winter, age < 40, budget <= 2500 would be labelled as Vogel, for some cases where if the budget is greater than 2300 it would be labelled as Verbier instead. Such alterations have been implemented for every feature in the generation code, in order to increase the confusion of the generated data samples before training the model.

Data is generated in two different ways, first one called Dataset-A is a somewhat uniform generation relying on random number generators, where each season have roughly same amount in the dataset. Even though each season have some number of samples in the dataset, labels are not equally distributed since the data samples with Spring and Autumn seasons are labelled the destination city only from the ensemble of Prague, Amsterdam, Istanbul and London. Thus, there are twice as much of data labelled with one of the cities in the ensemble than the other cities. Second generated dataset called Dataset-B, consist of more data samples generated with Summer and Winter seasons in a similar way of normal distribution. Which later allows the analysis of the models and overall RS, where data is generated non-uniformly.

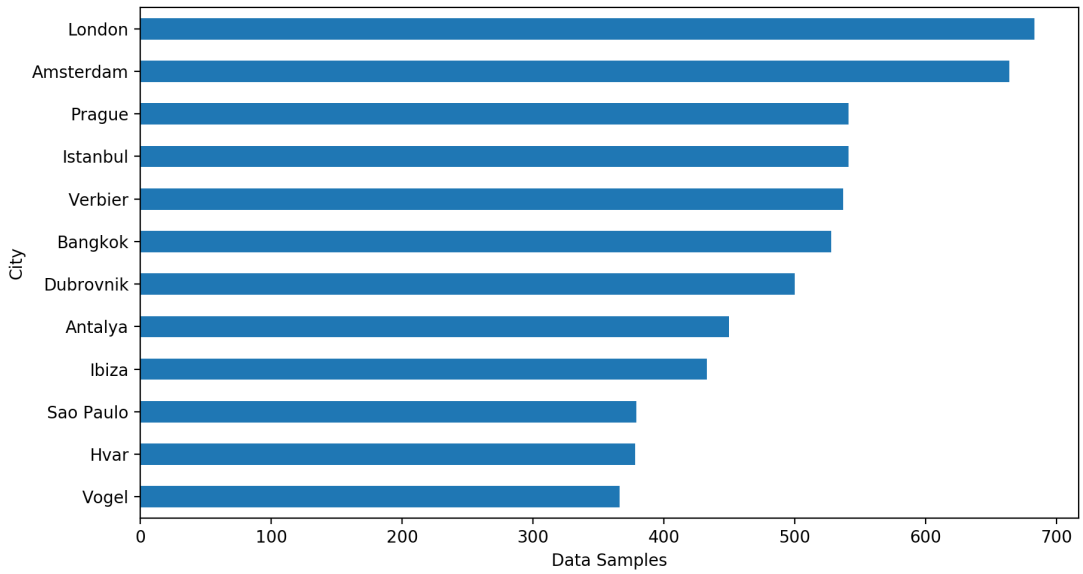

Figure 9, Destination distribution of samples (Dataset-A)

Figure-9 shows the destination distribution of 6000 generated samples. As mentioned earlier, London, Amsterdam, Prague and Istanbul are at the top since Spring and Autumn both seasons generate the same group of cities. As for the rest of the destinations, the sample amount varies between 380 and 530. This variation comes from the mentioned code alterations of the generation in order to agitate uniformity and increase confusion of the dataset. It is worth mentioning that since the models are trained with a batch of different generated datasets, the distribution for each one of them might slightly differ from the one shown in Figure-9.

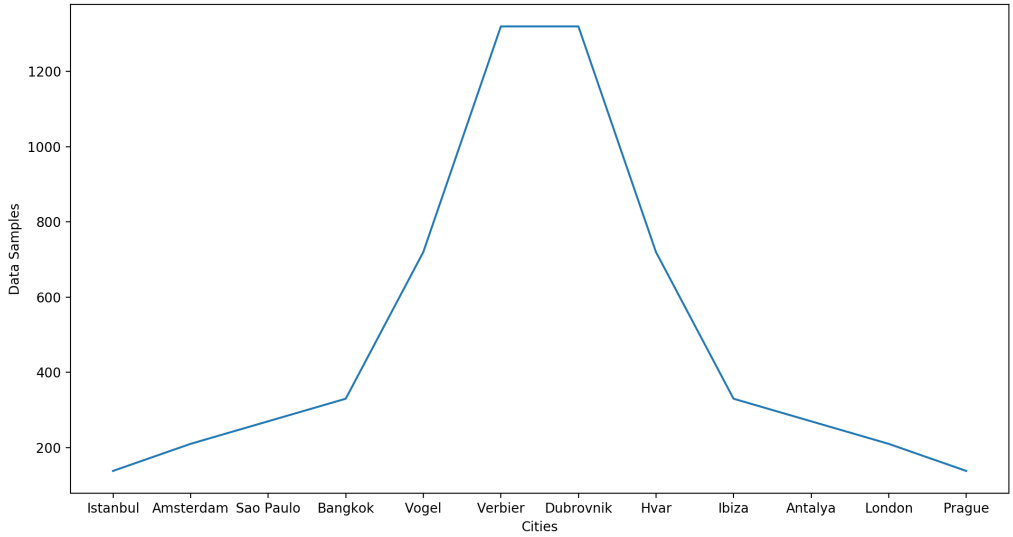

Figure 10, Normal Distribution kind distribution (Dataset-B)

Figure-10 shows the destination distribution of 6000 generated samples. As mentioned earlier, in Dataset-B, Summer and Winter cities have more data samples. The generation is implemented similar to three-sigma rule where 68 percent of the generated data consists of Vogel, Verbier, Dubrovnik and Hvar (A cities), 95 percent of the data consists of A cities plus Bangkok, Ibiza, Sao Paulo, Antalya, Amsterdam and London (B cities), and 99 percent of the data consists of A cities plus B cities and Istanbul and Prague [15]. Generation also includes some overlap of feature value ranges that determines the insight behind the generation to increase data confusion. Although this generated data is not exactly a gaussian distributed data where x axis values are not representing some numeric values, it still imitates the behaviour of normal distribution in terms of data sample amount variance among the destinations.

4.2 RF classifier implementation

A RF classifier is built using scikit-learn ML library [19]. Both generated datasets partitioned as 75% training and 25% testing. Training results of RF classifier for both datasets are discussed below.

4.2.1 RF classifier training results for Dataset-A

The model is trained with a variety of different versions of Dataset-A with different sample amounts. Data is generated 10 times per each data amount to observe the model behaviour with different distributions. Average prediction score results of RF classifier (mean of 10 different training score) on Dataset-A for each sample amount is displayed in the Table-2 below.

Table 2, Average prediction scores on Dataset-A

Sample Amount	Average Score (10 runs) %
100	85.00
250	90.20
500	94.00
1000	94.00
3000	95.10
6000	95.40
10000	95.20

Results indicate that, even with 100 data samples, RF classifier model can successfully predict correct class values for 85 percent of the data in the test portion. An increase proportionally to sample amount is observed for prediction score up to 95.4. Although it might defer from one feature space to another, RF classification results indicate that RF might be a good choice for classification for such dataset, since its prediction scores gets over 90 percent values with as low data points as 250. It is also worth mentioning that, Table-2 only includes results for up to 10.000 data samples since there were not any major prediction score increase with sample amounts higher than 10.000 such as 50.000 and 1.000.000. The reason behind the discontinuation of score increase after around 95 percent occurs due to code alterations made previously during the data generation.

4.2.2 RF classifier training results for Dataset-B

RF model is trained with a variety of different versions of Dataset-B with different sample amounts. Data is generated 10 times per each data amount to observe the model behaviour with different feature values, even though the class distribution is stationary. Average prediction score results of RF classifier (mean of 10 different training score) on Dataset-B for each sample amount is displayed in the Table-3 below.

Table 3, Average prediction scores on Dataset-B

Sample Amount	Average Score (10 runs) %
100	81.00
250	86.00
500	91.50
1.000	93.50
3.000	94.00
6.000	93.70
10.000	94.10

Results shows that, when the data distributed similar to normal distribution, RF classifier struggles to predict classes where there aren't enough data. On average it could only predict 81 percent of the test portion of the dataset correctly with 100 samples, where it could classify 84 percent correctly for Dataset-A. Even though there are more samples required for better prediction with normal distributed data, it still starts performing well just after 500 data samples as it reaches over 90 percent. RF classifier is observed to predict successfully as long

as there are enough samples with each class to understand data structure in order to build model rules, since the prediction score can rise up to 94.10 as the highest which is just 1.30 percent lower than the prediction with Dataset-A. The results are only shown up to 10.000 data samples since there were not any improvements with higher sample amounts.

4.3 Neural Network Classification Implementation

Neural network used in the project is built with fully connected layers with TensorFlow keras framework [25]. Feature values of dataset are fed into the network as input layer and there are two hidden layers with 64 and 32 neurons respectively. Hidden layers are configured to get the optimal prediction score, 64-32 layer pair is selected since it performed better than single 16 and dual 32-32 and 16-32 pairs. During the activation function selection, it has been observed that, although the training was fast, the network was unable to learn when ReLu function is used as activation function for any layer in the network. This associates with the dying ReLu problem denoted earlier in chapter 2, since deactivation of some neurons prevents neurons to acquire specific patterns for each data samples, especially because there aren't many features present in the dataset (season, age, budget). Thus, all the further experiments are executed with Sigmoid as activation function. Neural network is applied for both Dataset-A and Dataset-B with a variety of different sample amounts. Epoch of the training is set to ten, meaning dataset is exposed and weights are updated 10 times during each training. The reason behind setting epoch to ten is basically because there aren't any major score improvements after ten runs regardless of the sample amount.

4.3.1 Neural Network implementation and training results with Dataset-A

Neural network is trained with a variety of different versions of Dataset-A with different sample amounts. Data is generated 10 times per each data amount to observe the model behaviour with different distributions. Minimum, maximum and average prediction score results of neural network on Dataset-A for each sample amount is displayed in the Table-4 below.

Table 4, Neural network prediction scores on Dataset-A

Sample Amount	Min. Score %	Max. Score %	Avg. Score (10 runs)
500	4.80	16.80	10.70
1.000	6.00	20.00	11.60
5.000	11.40	27.20	19.80
10.000	22.20	27.80	25.38
50.000	22.60	28.80	26.30
100.000	24.78	70.00	57.30
250.000	11.88	80.09	58.36
500.000	50.00	93.20	82.65
1.000.000	52.70	94.27	83.47
2.500.000	11.60	94.90	79.01
5.000.000	59.30	94.20	81.46

Results shown that there is some level of variation among the prediction score for different versions of the dataset. Over 100.000 data samples are required to get over 50 percent prediction score on average. Although the network can start predicting over 90 percent with 500.000 data samples and onwards, average score could not exceed any more than 83.47 percent on any sample amount because of the outlier prediction values. For instance, for 2.500.000 samples, even though the prediction score for most of the test runs varies between 87 and 94 percent, the average score is measured 79.01 due to some low prediction scores occurred such as the minimum 11.60 percent. This problem is related with the stagnation problem, where the neural network struggles to learn about the dataset after a point. Meaning that the weight updates to optimize the network does not result in any loss reduction or gain [26]. Detailed observation for such stagnation cases showed that, even though the dataset was applied for training ten times (epoch) the network fails to get any better after one or two epochs and results in the same prediction score for the rest of the training.

4.3.2 Neural Network implementation and training results with Dataset-B

Neural network is trained with a variety of different versions of Dataset-A with different sample amounts. Data is generated 10 times per each data amount to observe the model behaviour with different distributions. Minimum, maximum and average prediction score results of neural network on Dataset-B for each sample amount is displayed in the Table-5 below.

Table 5, Neural network prediction scores on Dataset-B

Sample Amount	Min. Score	Max. Score	Avg. Score (10 runs)
500	19.30	19.30	19.30
1.000	19.20	22.80	20.28
5.000	22.20	40.20	32.58
10.000	21.70	42.30	38.33
50.000	38.80	57.60	49.00
100.000	22.00	69.40	52.74
250.000	58.50	89.40	78.00
500.000	57.30	88.00	80.50
1.000.000	60.40	94.00	84.55
2.500.000	22.00	94.30	68.00
5.000.000	22.00	94.90	67.80

Results show that, even though there are some variations among the prediction score, these variations are relatively lower for the data samples lower than 2.500.000 in comparison with the results of Dataset-A. Interestingly, results also indicate that the neural network predicted higher score on average on Dataset-B and the stagnation problem occurred less for lower data amounts (500 – 1.000.000). However, as the data amount increased, the network started to stagnate more often and resulting in a dramatic decrease on average results such as 68 and 67.80 percent for 2.5 million and 5 million data respectively, whereas it scored 79 and 81 percent for Dataset-A.

As a result of overall prediction results of RF and Neural Networks for both datasets, the first version of recommendation engine is implemented using a RF classifier model since it resulted in more robust and consistent results than neural networks, even though the neural network also performed good prediction scores for some cases. The main reasons behind RF selection is that it could perform consistent accuracy, whereas neural networks stagnated for some cases of random generation, and also RF required much less data samples than neural networks, in order to provide high prediction scores.

5. RS WITH COSINE SIMILARITY

In this chapter, a sole CS approach is implemented as a recommender system. RS is built on destination data scraped from TripAdvisor.

5.1 Data Scraping

Data scraping is an important step before building a RS since each dataset usually have different characteristics and, these characteristics must be known and optimized before proceeding with an optimal RS solution. Unlike the classification approach on chapter 4, the acquired data is not synthetic but genuine real data scraped from TripAdvisor website. Acquired data consists of; city name, review and opinion count, city description and city background image URL. Data is scraped with selenium which is a browser automation tool often used for testing purposes [33]. Scraping is applied to two different pages, search result and city detail. Pages are displayed in Figure-11 and Figure-12 respectively.

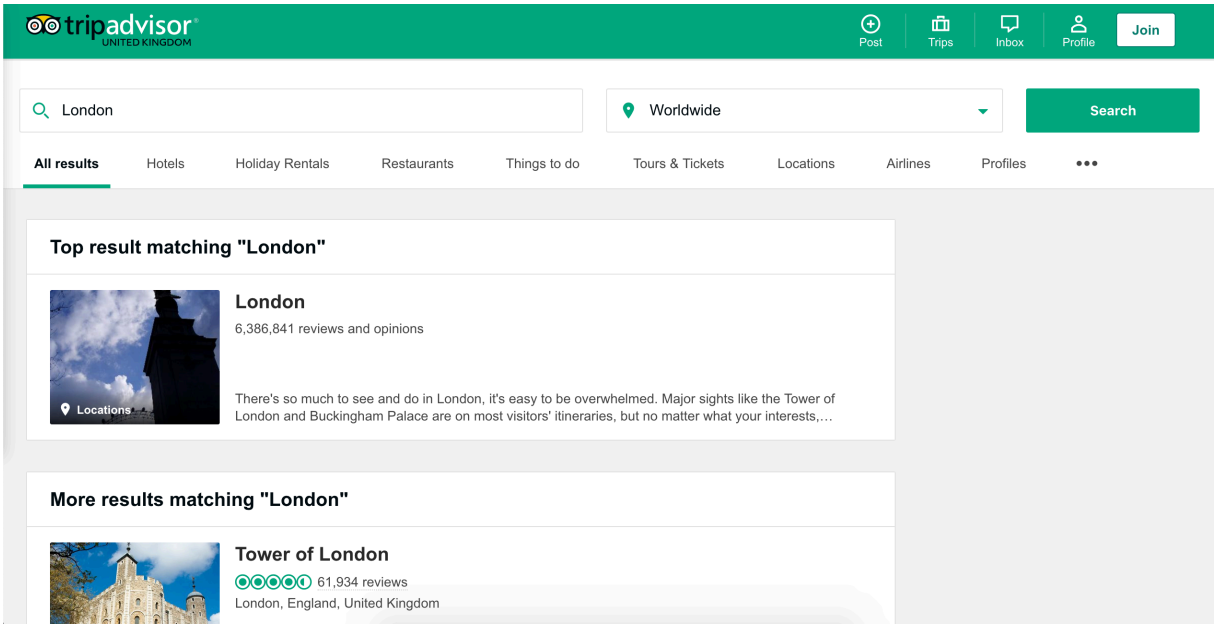

Figure 11, City search result page [34]

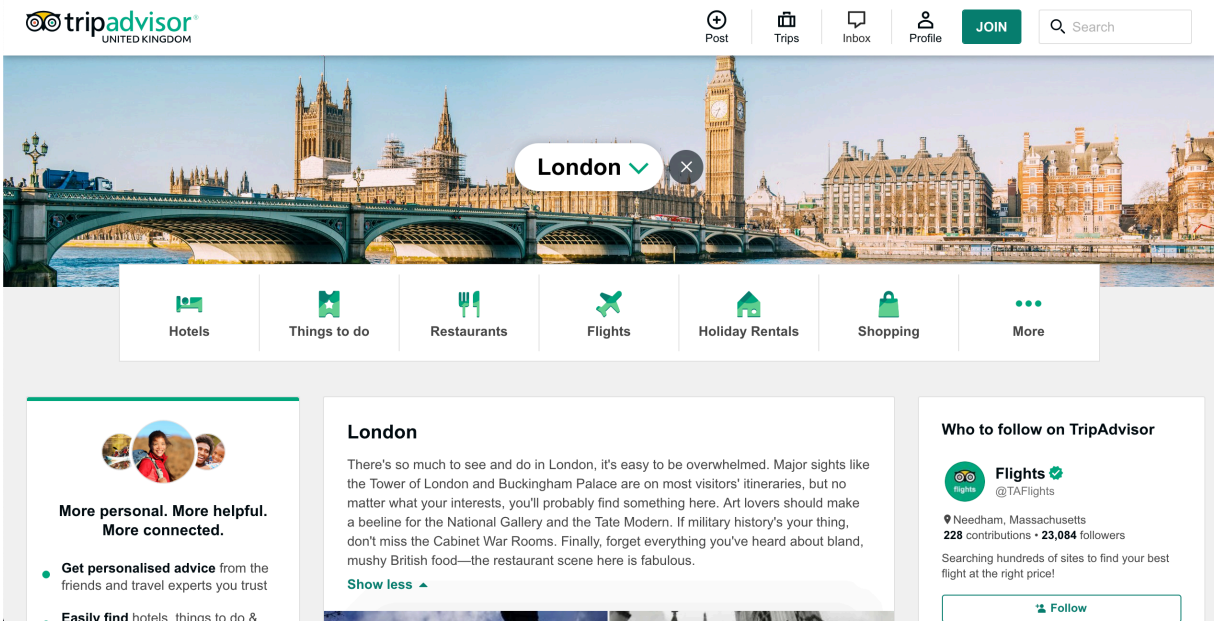

Figure 12, City detail page for London [35]

First of all, 25 popular cities are pre-determined for data collection, then a repetitive process is applied to each one of them as shown in Pseudocode-1 below to; build a search query for each city, get the city review & opinion count from the search result, then go to detail page to acquire description and image URL.

Pseudocode 1:

```
Initialize city list with 25 elements
Initialize first part of search query URL
Initialize second part of search query URL

    FOR each city in list
        Build URL with city, first URL and second URL
        Open the search query link on browser
        Get city title and review count from UI components
        Click on the result to proceed to detail page
        Get description and image URL
         Write city information to CSV file.
        Close detail page on browser
    END FOR
```

After the data acquisition is completed for all cities, data is written in a CSV file for later use. Table-6 below shows the feature and values scraped for London and Istanbul.

Table 6, Data samples for 2 out of 25 cities

City	Popularity	Description	Image URL
Istanbul	1,177,875 reviews and opinions	Europe and Asia meet in Istanbul, and throughout this vibrant city, you'll find centuries-old mosques, churches and markets happily co-existing with modern restaurants, galleries and nightclubs. And plan on visiting a hammam (traditional Turkish bath)—for about $20 your skin will be scrubbed clean. And we mean scrubbed. Your wimpy loofah has nothing on this.	"https://media-cdn.tripadvisor.com/media/photo-b/2560x500/18/2d/42/21/istanbul-coverphoto.jpg"
London	6,346,416 reviews and opinions	There's so much to see and do in London, it's easy to be overwhelmed. Major sights like the Tower of London and Buckingham Palace are on most visitors' itineraries, but no matter what your interests, you'll probably find something here. Art lovers should make a beeline for the National Gallery and the Tate Modern. If military history's your thing, don't miss the Cabinet War Rooms. Finally, forget everything you've heard about bland, mushy British food—the restaurant scene here is fabulous.	https://media-cdn.tripadvisor.com/media/photo-b/1024x250/15/33/f5/de/london.jpg

5.2 Data Pre-processing

Since the CS is going to be calculated on the description feature of the cities, some pre-processing is applied in order to get rid of unimportant conjunctive verbs and adjectives. All of the descriptions of all the cities are merged into a string and then each word count in the string is sorted in a descending order. It was observed that, the top 25 most occurring words were non-contextual and hence should have been removed. These top 25 words are saved into a vector and then all of the city descriptions are scanned through to remove whenever a word included in the vector occurs. Then the remaining words are examined to come up with some set of keywords that would describe a purpose, such as; culture, history, museum, old-town, art and etc which can be related with culturally rich cities. Three different keyword ensembles are collected from the city descriptions (nightlife, culture, summer vacation) which are later used to compute CS.

5.3 Cosine Similarity Model Implementation

Recommender System is built to operate on computing similarities among city descriptions with given keywords vectors. But since there was no user related data such as previous ratings or previous travelled destinations or anything at all, it was not possible to give recommendations right away. This is also called Cold Start problem, where there are no or very few data about users' past behaviours or consumptions to create a reliable set of

recommendations from [36]. In order to cope with this problem, there are three options created consisting of an option title, keywords (hidden from users) and an image. The options are:

- Culture, Art and History (history historical art architecture city culture)
- Beach and Sun (beach beaches park nature holiday sea seaside sand sunshine sun sunny)
- Nightlife and Party (nightclub nightclubs nightlife bar bars pub pubs party beer restaurants dinner)

The cosine similarity of each option keywords is then calculated with all the cities to determine top 5 city for each category with highest similarity score to recommend a set of cities depending on the user option selection.

Each word in a city description, represents a different dimension and the count of words represent the value of the word vector for that specific dimension. For instance, given a string of “one one two one one” the dimension would be two since there are only two words (one and two), and the respective values would be (4,1). In this RS implementation, the system is designed to increase similarity of an option and the city description, as long as that specific word occurs. Although the word count still affects the outcome of similarity with CS, it is less affected by scalar changes in comparison with Euclidean Distance, since it does not take scalar factors into account and CS normalizes the score by dividing the magnitude of the vectors as explained in Chapter 2. Thus, for the recommender application, the similarity is computed with CS.

Due to the nature of dot product, the summation only includes the multiplication with respect to word count only for the words which exists both in the option and city description. Firstly, the common words in both the option and the city description is determined in order to find dimensions to be multiplied. Then the dot product is summed up with the multiplication of each common word’s count in both option and description. After that the magnitude is calculated with all of the words included in option and description respectively. Finally, the similarity is found by dividing the dot product by the magnitude. It is worth mentioning that score varies between 0.0 and 1.0 since it is normalized. Top three similarity results for options “Culture, Art and History”, “Beach and Sun” and “Nightlife and Party” is shown in Table-7 below.

Table 7, Top 3 cities with highest similarity score for three different options

	City	Score
Culture, Art and History		
1	St. Petersburg	0,12285
2	Madrid	0,12247
3	Athens	0,11846
Beach and Sun		
1	Brighton	0,10192
2	Marmaris	0,08362
3	Miami	0,05698
Nightlife and Party		
1	Ibiza	0,15075
2	Miami	0,12598
3	Hvar	0,11111

Results show that the overall similarity score is low for all the cities since the comparison of keywords are executed on city descriptions, where city descriptions are written in a narrative way and often includes many words. However, rankings for options shows that the cities are relatively scored well, since when the ranked cities are examined, they fit to the recommended category contextually. It is presumable that cities like St. Petersburg, Madrid and Athens are culturally and historically rich and old cities, whereas Ibiza, Miami and Hvar are known for their nightlife and parties, even without reading the city description. Thus, CS might be a good way of creating a recommendation set especially when there is no user data available (cold start) by only comparing the content features with a set of keywords. It is also convenient to offer users a few options to gather some knowledge about their intention without asking too many overwhelming questions before making a recommendation. Similar approaches can be applied to many different industries as well, for instance, MovieLens (a movie rating website) has changed their way of coping with cold start problem for new users, from asking for specific ratings for some movies to distributing some interest points among some category of movies [37].

6. RS Dynamization with Feedback

Recommender systems are often created with the user and content data available and does not change the way it operates. But in reality, both users and content change overtime, and it is important to collect user feedback to act on it for future recommendations. For instance, even though a user might like some certain group of items for a period of time, this may change in the future due to aging, change of interests, experiencing new ideas and etc. An RS should be designed to observe these changes overtime via explicit or even implicit feedbacks, where available. Moreover, for an RS based on similarity, it is also important to observe feedback results for the content to make optimal recommendations for the users. Even though some set of items might be very relevant to a specific category and might seem to perform well in the beginning, the system should cope with possible changes via feedback as users might dislike the item regardless the relevance.

It is important to collect feedback one way or another, there are two kinds of feedback available, explicit and implicit. The basic distinction between them can be explained as, users willingly express their feedback about the content for explicit feedback, whereas for implicit feedback, the system determines/generates some feedback from the user's interactions within the system or more specifically while visiting or analysing a specific content. Explicit feedback can be in forms of rating on a scale, binary (liked, disliked), detailed text review and etc. For implicit feedback, there can be more variations depending on the context of the service such as; if a user bought a product, how long did a user spend on a specific page, how long a user played the product (movies and music) and many others.

This chapter introduces a several different approaches to cope with user feedback for dynamization on second version of the recommender system discussed in chapter 5.

6.1 Recommendation Score with CS and rating

Recommender engine is updated to make recommendations by computing a final score from the CS score and the rating of the destination. In this initial implementation, rating quantity is omitted, the score contribution of rating is only calculated from the given average rating regardless how many users rated the destination. The engine first calculates the CS and then passes the CS and the rating value to a function to have a final recommendation score. Where another function is called with two parameters r and Q from the first function, where r is the rating and Q is the importance parameter determining what percentage range the CS score should be affected to create the final result. Function is designed to compute a final score by increasing or decreasing the CS score depending on, if the rating is greater or less than 5 proportionally to the given Q parameter (assuming destinations with average rating greater than 5 is liked and should be encouraged whereas destinations less than 5 is disliked and should be discouraged by the system). The rating range is between 0 and 10 and rating effect range is between -Q and Q where the default value for Q is set to 10, meaning that maximum rating increases CS by 10 percent and the minimum rating decreases CS by 10 percent for final recommendation score. Visual representation of the function for Q equals to 10 is shown in Figure-13 below.

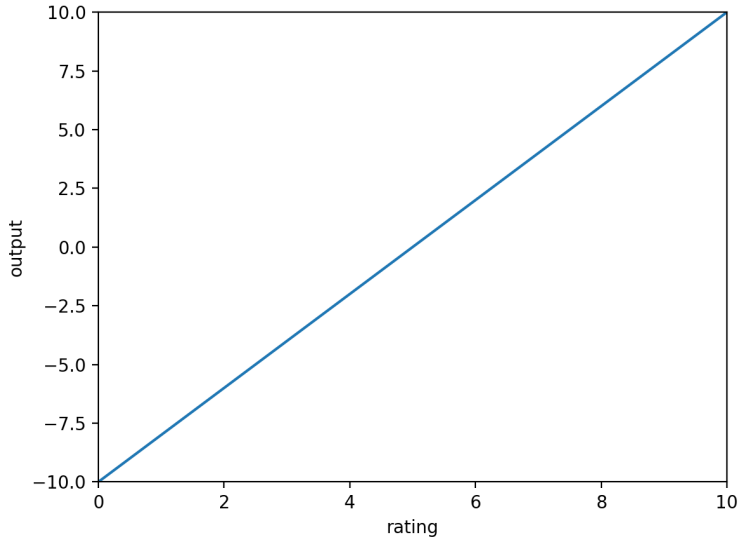

Figure 13, Rating contribution computer function for Q=10

Since there is no rating information available in the scraped data, a set of synthetic rating values are assigned to each destination in order to observe updated model performance. Top 5 destinations for "Nightclub and party" option with Q values of 5, 10 and 100 is shown in the Table-8 below.

Table 8, Recommendation score with different ratings and parameters

Destination	Rating	CS Score	Recommendation score for Q=5	Recommendation score for Q=10	Recommendation score for Q=100
Ibiza	8.4	0,151	0,156	0,161	0,256
Miami	3.4	0,126	0,124	0,122	0,086
Hvar	9.5	0,111	0,116	0,121	0,211
Prague	2.3	0,075	0,073	0,071	0,034
Madrid	7.0	0,61	0,62	0,063	0,085

Results indicate that there is a score increase from sole CS score to recommendation score, for ratings greater than 5.0 and a score decrease for ratings less than 5.0. It is also observed that the decrease or increase of the score depends on the absolute distance of the rating from 5.0. Even though Hvar and Miami had similar CS scores, Hvar's recommendation score is increased more than Miami's score is decreased since Hvar's rating distance to 5.0 is greater than Miami. Moreover, the recommendation score does not shift much from the CS score where Q is 5 or 10 regardless of the rating. However, as the ratings get closer to the highest or lowest ranking values with Q equal to 100, the final score diverts further away from the CS score, since the importance of rating increases.

This implementation displayed a way of implementing a final score with respect to CS score and rating value, it allows configuration to determine how greatly rating and CS score should affect the end result with the parameter Q. However, this implementation does not consider the rating quantity, for example a destination with a very few amounts of high ratings

could achieve a high recommendation score whereas another destination with same similarity, slightly lower average rating but with a great quantity could achieve less recommendation score. It could be improved with a threshold value for rating quantity to discard destinations with lower rating quantity. Nevertheless, updating the function to cope with ranking quantities is expected to result in more robust RS.

6.2 Recommendation Score with CS and rating (including quantity)

As mentioned previously, the recommendation score function is missing an important feature, the rating quantity. It is important to take rating quantity into account as it is not ideal to have lower or higher scores for a destination, based on a few ratings. There is an additional parameter T is introduced, which indicates the rating quantity required for a destination, in order to contribute 50 percent to the main output range. For example, given a destination with rating of 10 with 100 user ratings, and function parameters; Q equals 10 and T equals 100, the function returns 5 percent increase, whereas it would return 10 percent with the previous implementation. In addition, for any rating quantity lower than T, multiplier approaches to 0 depending on the quantity and for any rating quantity greater than T, multiplier approaches to 1. Thus, the contribution multiplier value M, ranges between 0 and 1.

$$\boldsymbol{M} = \boldsymbol{e}^{-\boldsymbol{T}*\boldsymbol{0{,}68}/\boldsymbol{c}} \quad (1.5)$$

As shown in the Eqn. (1.5), the contribution multiplier (M) is calculated by exponentiation with base e (Euler's number) and exponent; negative T times 0,68 divided by c, where c is the rating count. For any given T, M approaches to 1 as c increases greater than T and M approaches to 0 as c decreases lower than T. It is worth mentioning that 0,68 is included in the equation since e to the power of negative 0,68 gives 0,50 which means any value multiplied with it would be halved. After M is calculated, it is multiplied with the value generated depending on the rating and then the resulting value is applied on the similarity score as before. Overall recommendation score (RC) is then calculated as shown in Eqn. (1.6) where;

- RC = Recommendation Score
- CS = Cosine Similarity of the destination
- Q = Range to indicate the importance of ratings
- r = Rating of the destination
- T = Rating count reference where the multiplier would halve the rating contribution
- c = Rating count of the destination

$$RC = CS + \frac{CS}{100}\left(\frac{2Q}{10} * r - Q\right) e^{-T*0{,}68/c} \quad (1.6)$$

Recommendation scores for destination St. Petersburg with same rating but different rating quantities and T values are shown in the Table-9 below.

Table 9, Recommendation Scoring result with rating counts

City: St. Petersburg			
Rating: 8.5			
CS Score (for Culture, Art and History option): 0.12286			
Chapter 6.1 Score (rating applied without rating quantity, Q = 10): 0,13146			
ID	**Rating Count**	**T**	**RC**
1	1.000	50	0.13117
2	1.000	100	0.13089
3	1.000	200	0.13036
4	1.000	1.000	0.12721
5	1.000	20.000	0.12286
6	100	1.000	0.12287
7	1.000	1.000	0.12721
8	10.000	1.000	0.13089
9	1.000.000	1.000	0.13145

Results shows that, when rating count is greater than threshold value, recommendation score is higher since the rating is above 5. As T increases the score starts to decrease where rating contribution score is halved when T is equal to given rating count as shown in Eqn. (1.7) below.

$$0.13146 - 0.12721 \cong 0.12721 - 0.12286 \quad (1.7)$$

Rating contribution to the final score is quite low where rating count is way below the threshold value T and the contribution gets higher as the rating count increases over the threshold as can be observed from the tests ID 5 to 9.

Rating contribution multiplier introduced in this chapter, performs quite well as expected for the given threshold value. Since it is normalized, it does not boost the rating contribution by itself and the importance on the rating contribution in general is configured by Q parameter. Selection of these parameters might depend on the context and similarity value range of the system, but parameters are set as; Q=10 and T=500 for the destination recommender application of this project.

6.3 Binary Feedback Acquisition from review texts (positive or negative)

Reviews are an important type of explicit feedback, where users often have a chance to criticise, praise and express their feelings for an item in detail rather than just giving a rating. Reviews are a popular feedback method as they are available on content provider applications in many industries such as Amazon, IMDB, TripAdvisor and etc [40, 41]. Depending on the length and details of a review, it is possible extract useful information for different aspects. Information may consist of some specific features of a content, overall satisfaction and even about the application itself.

In this project, review functionality was implemented in order to extract binary feedback, namely, to understand if the feedback is positive or negative from users depending on their review. A Random Forest classifier model is built and trained on the publicly available "515 K Hotel Reviews Data in Europe" data [42]. Even though the project application is based on destinations rather than hotel reviews, the dataset provided sufficient context, since hotel reviews and destinations are somewhat related and also the classifier was able to understand the core features of reviews to predict unseen reviews for given destination related reviews.

Data Pre-Processing: Hotel Review dataset is constructed on a hotel-user basis, where each user has written both positive and negative feedback for their time in a hotel (there are some cases where one of the two is missing). Pre-processing is applied as following:

- Since there are 515.000 reviews available in the dataset, the dataset is refactored to include only one percent of its actual size in order to make computation faster.
- There were many features available in the dataset apart from negative and positive feedback, such as: review date, review score, hotel average score, reviewer nationality and etc. All other features except negative and positive reviews were discarded. Then a different dataset is created by looping through all the negative and positive reviews. New dataset consists of two features; review and is_positive. For is_positive feature, negative reviews labelled as 0 whereas positive ones labelled as 1.
- Every word in reviews are lowercased and words with low lexical-meaning such as; "the, at, for, on, which, is" and any punctuations and numbers are removed from all reviews.
- Each word is refactored down to its roots by using the stemming algorithm with Natural Language Processing toolkit [43] in order to reduce word dimensionality. For instance, "lover, lovely, loved," all refactored to love.

Building the Model: After the pre-processing and the reviews are cleaned, a count vectorizer matrix is initialized with the most frequent 1500 words of all reviews. The matrix is (review count by 1500), meaning that each review has a corresponding value for all these 1500 words. The value for any given word for a review is determined by how many times that word occurs in the review, if the word does not occur, the value is set to 0. Which also resulted in a very sparse matrix. After the matrix is created, a RF classifier is built and trained with the 75 percent of the dataset where the feature set per data point was the 1500 words, meaning, each word is fed into the classifier as a different feature with the corresponding values per data point. Then the training model is used to predict labels for remaining 25 percent of the data.

The matrix creation and RF classifier training processes applied to 10 different fractions of the data set where each time the word matrix and RF classifier were different depending on the reviews in the chosen fraction. Prediction result of the classifier for 10 different fractions are shown in Table-10 below.

Table 10, RF classifier prediction scores on review dataset

Percentage of the Dataset (%)	Prediction Score (%)
1	89,80
1	90,20
1	90,10
1	89,10
1	89,20
2	88,60
2	90,30
2	89,60
2	90,10
2	89,95

Results show that RF classifier can predict if an unseen hotel review is positive or negative around successfully around 90 percent of the time regardless the dataset size as long as it is trained with a fraction larger than 1 percent (around 6000 samples) of the dataset. It is also worth mentioning that for the lower fraction amounts the score variance is resulted larger depending on what kind of words included in reviews, whereas the variance is minimal as the fraction size gets larger than 1 percent as can be seen from the results in Table-10.

Testing with a review other than hotel review dataset: After the model is trained and ready for use, it was tested with manually written destination related reviews. The reviews were written in text format and hence a pre-processing was required. Since the model is trained on 1500 different features, the review data had to be adjusted to fit in to the feature space of the classifier. Each review is scanned through and a list initialized consisting of every word in the review. Then each word in the list is checked with the feature space if the word exists, if so, the feature index for that particular word is found and then the value is set to how many times that word occurs in the review, and if the word does not exist in the feature space, then it is simply discarded. The value for all other words that exist in the feature space but not present in the review is set to 0.

The model is tested with 10 positive and 10 negative feedback and the results shown that the classifier could predict the outcome successfully 8 times out of 10 for both positive and negative reviews. Only 4 of these results, including correctly and incorrectly classified reviews for both originally positive and negative feedback are shown in Table-11 below.

Table 11, Prediction results for manual review entries

Review	Actual Outcome	Predicted Outcome
It was a pleasant experience we enjoyed our stay in London a great city with a lot of activities	Positive	Positive
Even though the places were expensive overall, we were satisfied and happy with our trip	Positive	Negative
It was an unpleasant experience we disliked our stay it was boring	Negative	Negative
Food was nice, but the city was boring it was below average for us	Negative	Positive

Results shown that the classifier can correctly label the reviews where the context is rather clearer and strongly positive or negative, whereas it misclassifies where the feeling behind the review is not so certain due to usage of some words that are opposite of the actual feeling behind the review.

This approach resulted rather successfully even though the manual entry review success results were slightly lower than hotel reviews. Such a model could be trained and installed in a destination recommender system and generate binary feedback from the incoming reviews per destination. Moreover, by the time as the reviews increases, the model can be re-configured and include both hotel reviews and actual destination reviews for training. It might result in better predictions for the application as the system goes on, dynamically.

6.4 Recommendation Score with CS and different feedback combinations

In both chapter 6.2 and 6.3, two different types of explicit feedback mechanisms are demonstrated. As mentioned earlier, one challenge of dynamic RS is to incorporate feedback in different forms. As working with rating (0-10 scale) and binary feedback data already introduced earlier, this chapter will present a way of incorporating both feedback forms to improve the RS.

Incorporation for two different feedback forms can possibly be configured in different ways, such as;

- Creating a third dimension and adjusting feedback from both forms to fit in to the new dimension and then provide new feedback dimension to the RS.
- Selecting a feedback form as a base and then adjusting the data from other form to fit in to the base form and then feed updated base feedback form to the RS.

In this project, later of the two is implemented where the binary feedback is fit in to the rating feedback and at the end updated rating feedback is used within the RS.

A basic approach of doing so would be to select two rating values for both negative and positive feedback, and then consider each review feedback as a rating and recalculate the average rating of an item. But this approach would not be ideal. For example, if the ratings are selected as 0 and 10 for negative and positive reviews, then the effect of reviews would be more than actual rating data especially when the rating of an item is either closer to 0 or 10. It could be possible to select different rating values to alleviate the review effect on the rating. For example, if the ratings are selected as 2.5 and 7.5, then another problem would occur where a positive feedback for an item with average rating greater than 7.5 would still contribute as lowering the rating, whereas a negative feedback for an item with average rating less than 2.5 would contribute as increasing the rating. Hence, another approach was needed in order to incorporate both forms better.

A different approach is applied in order to address the issues mentioned above as following;

- For any positive review, a new rating value is added to the ratings by calculating the distance between the average rating and maximum rating value and then adding half of the calculated distance to the average rating to determine rating value for review.
- For any negative review, a new rating value is added to the ratings by calculating the distance between the average rating and minimum rating value and then subtracting half of the calculated distance from the average rating to determine rating value for review.

Positive and negative rating calculations Rp and Rn, for both positive and negative review for any given average rating r on rating scale 0 to 10 is shown respectively in Eqn. 1.7 and Eqn. 1.8 below.

$$Rp = r + (10 - r)/2 \quad (1.7)$$

$$Rn = r - (r - 0)/2 \quad (1.8)$$

For instance, for an item with average rating of 6 (range 0-10), for every negative review, a new rating with value of 3 is added to ratings, whereas for every positive review, a new rating with value of 8 is added to ratings. Then the average rating is calculated again with new ratings before fed into the scoring function. Review feedback conversion into rating feedback results are shown in Table-12 below for different ratings, rating counts and review counts.

Table 12, Review and rating calculation results

Rating	Rating Count	Positive Review Count	Negative Review Count	Resulting Rating
7,2	1500	140	80	7,15
7,2	10000	550	150	7,22
7,2	10000	5000	5000	6,65
6	1500	140	80	6,02
6	10000	550	150	6,06
6	10000	5000	5000	5,75
4	1500	80	140	3,98
4	10000	150	550	3,94
4	10000	5000	5000	4,25
2,8	1500	80	140	2,86
2,8	10000	150	550	2,78
2,8	10000	5000	5000	3,35

Final rating results suggest that with the current implementation, when rating value gets closer to either maximum or minimum value, calculated rating tends to favour the opposite rating value. For example, for the case where rating is 7,2 and positive and negative review counts are equal, the result is 6,65 as it goes closer to minimum rating from the original value. This is because, the distance between 0 and 7,2, is greater than the distance between 7,2 and 10, similar behaviour is also observed when the original rating is 2,8 with equal positive and negative rating values as well. Moreover, the effect of the reviews is minimal since the test cases included more rating values than review which is often case for the commercial recommender systems. The effect could have been increased by introducing an additional parameter to change the importance of reviews to make this approach more efficient.

7. RS APPLICATION IN PRACTICE WITH CROSS PLATFORM

In this project, there are two versions of RS applications developed. First one is based on the RS implementation with classification as discussed in chapter 4, and the second one is based on the RS implementation with CS and feedback data as discussed in chapter 5 and chapter 6. Both versions are developed using a cross-platform application development framework Flutter [10] which will be discussed later in this chapter. First of all, two cross platform frameworks (Flutter and React Native) is compared and then both recommender versions are developed with the selected framework. This chapter will first discuss the comparison of the cross-platform frameworks and then continue with demonstrations of the application development for both RS versions.

There are two dominant and used mobile application platforms, iOS and Android. Traditionally, applications are developed separately per platform by using the frameworks and other development tools provided by the owner of the operating systems (Apple for iOS and Google for Android), which is also called native development. For native development, applications are developed with Swift/Objective-C and Java/Kotlin programming languages for iOS and Android respectively, with two different code bases [4, 5].

Cross platform is an alternative way of application development, where there is only one code base for development for multiple platforms. Applications are developed using a single framework and a dedicated programming language, then the framework adjusts the developed application for both platforms internally. Cross platform development may have some advantages and disadvantages as shown in Table-13 below.

Table 13, Advantages and Disadvantages of Cross Platform over Native Development

Advantages	Disadvantages
1. Cost efficient, less developers needed for development, where there are more developers needed in native development for both platforms. 2. Better reusability and easier maintenance, since one code base is developed. 3. It may provide faster development since the whole application is developed and automatically configured for deployment on both platforms.	1. Lower performance, especially if the device model is older. 2. Framework dependent, a framework might stop further development at any given point and it might be impossible to cope with further OS changes, whereas its highly unlikely for native development frameworks to cease operation as long as the OS is active. 3. Lack of user experience, it may be harder or impossible to develop some certain UI that could be automatically implemented for both platforms.

Even though there are some disadvantages, cross platform development might be highly beneficial for some cases and it is an important, growing concept in application development world. There are a variety of different cross-platform development frameworks, but only React Native and Flutter are discussed and compared in this project [7, 10].

7.1 React Native and Flutter

Both React Native and Flutter, provides services to develop cross platform applications for mobile platforms by using programming languages JavaScript and Dart respectively. Both development frameworks are compared in this section in terms of; getting started, interface development, networking, testing and popularity. The summary of the comparison is shown in Figure-14 below where stars indicate the strength of the framework for given area.

	Flutter	React Native
Getting Started	★★★★	★★★★★
Interface Development	★★★★★	★★★★
Networking	★★★★★	★★★★
Testing	★★★★★	★★★★★
Popularity	★★★★	★★★★★

Figure 14, Flutter and React Native comparison summary

7.1.1 Getting Started

It is straight forward to make necessary installations to start development with React Native. Documentation is well written to guide developers through the process, and troubleshooting pages exist in case any problem occurs. It is also easy to get started with Flutter, although there might be additional steps to complete configurations depending on the operating systems. Even though Flutter may require additional steps, it has a good control system called doctor, which checks the operating system to determine if it is ready for development and provide information on additional requirements that needed to be installed before starting development.

Both React Native and Flutter requires additional configurations with iOS (Xcode, Command Line tools, Simulator) and Android (Android SDK, Emulator) to get started, whereas it is easier to start development natively. But these necessary configurations are fairly easy and well guided through documentation for both cross platform frameworks.

7.1.2 Interface Development

In React Native, user interface is built in a similar way to HTML (although view names are different), all styling and layout configurations are provided with style sheets similar to CSS. In Flutter, user interface is built in, in a nested way, somewhat similar to HTML as well but by providing child UI components for each component in view hierarchy. In addition to initializing and configuring the interface structure, providing methods such as event handlers and functions during view creation, might make the code less readable. But for both platforms it is possible to create separate interface components to increase reusability and readability. It is worth mentioning that, for React Native it is required to import each UI component to use, where there is no need to make additional imports for most of the UI components in Flutter.

7.1.3 Networking

For both platforms it is very easy to make networking requests. Both Flutter and React Native supports REST API which is a software design pattern that identifies certain characteristics for web services in order to facilitate uniform communication among all other web and mobile applications to each other [45]. For communication, data is transmitted from one service to another in a human readable file format consists of text, called JSON [46]. Applications decode and encode these files in order to read and process data attributes for later use. Even though it was not implemented in this project, it is worth mentioning that, it is possible to interact with GraphQL instead of REST APIs using both frameworks. GraphQL is a language developed by Facebook which allows fast and direct data retrieval from servers [47]. Developers interact with the networking libraries to make request and get or send JSON components in a very similar way in both platforms. It is slightly syntactically clearer to write networking code in Flutter due to programming language differences, but this might differ depending on the bias and style of developers.

7.1.4 Testing

Testing is a crucial part of software development and its applied on both platforms. React Native uses a JavaScript testing framework called Jest for writing unit tests and Flutter has a testing library called flutter test for writing unit tests. It is fairly easy to write and run tests on both platforms. There are good informative messages for test failures as well. Also, both frameworks have very similar approach for writing unit tests and there is only some subtle difference due to syntax difference.

7.1.5 Popularity

React Native is claimed to be started as an internal tool in Facebook in 2013 [6]. However, it was made public with initial release on March 2015 [7]. React Native has around 77.750 stars on GitHub (development platform for developers) [8] and around 68.000 questions asked on Stack Overflow (Q&A platform for developers) [9]. On the other hand, Flutter was released on May 2017 [10]. Flutter has around 65.900 stars on GitHub [11] and around 35.700 questions asked on Stack Overflow [12].

Both platforms are popular in mobile development world and seems to be getting more popular thanks to cross-platform development functionality. React Native has the major market share for cross platform since it came out earlier than Flutter and more stable with more years of development and optimization. Nonetheless, Flutter is also providing a good framework for cross-platform application development and seems to be gaining more popularity as well.

7.2 Framework Selection

It has been observed that both frameworks were good enough for developing cross platform applications as a result of the comparison. It is not possible to declare either one of them as a superior framework with the current scope of the comparison. An exhaustive comparison for many different aspects of the frameworks might result in better indicators to make a decision. Overall both platforms are robust and easy to interact with. Although it is easy to write code for building user interfaces, both platforms are missing a visual editor to alleviate implementation of user interfaces which is possible in native development environments (Interface builder of iOS and Layout Editor of Android). Both platforms provide Hot-Reloading during development which makes development much faster since it does not require for a new build every time code base changes. It seems like initially React Native was built with the main focus of allowing web developers to develop applications for mobile devices rather than emphasising the importance of cross-platform. Whereas Flutter, was built with the idea of cross-platform application development in mind rather than focusing mainly on web developers, since it seems easier for mobile developers to interact with rather than React Native. Moreover, it was observed that some errors related with build configurations of React Native affected native development for iOS platform of the current operating system, whereas there were no negative effects of Flutter build configurations on native iOS development.

Finally, Flutter is selected as the development framework for the project. Even though there were not many differences among the two platforms, Flutter is selected for development, thanks to it's easy to interact with programming language, robust and effective foundation frameworks, working with native development configurations (if necessary) seamlessly and rapidly growing popularity among the mobile application development world.

7.3 Version 1 RS with RF classifier in practice

A prototype recommender application is implemented for first version of RS, by creating a backend service on the local host and a cross platform mobile application. RS is implemented using a RF classifier on the dataset discussed in chapter 4. The application consists of two parts, backend and the mobile application. The model is installed on the backend service, and UI consists of two pages, first one is for entering input and submission, second one is for displaying recommendation. Visual representation of the application is given in Figure-15 below.

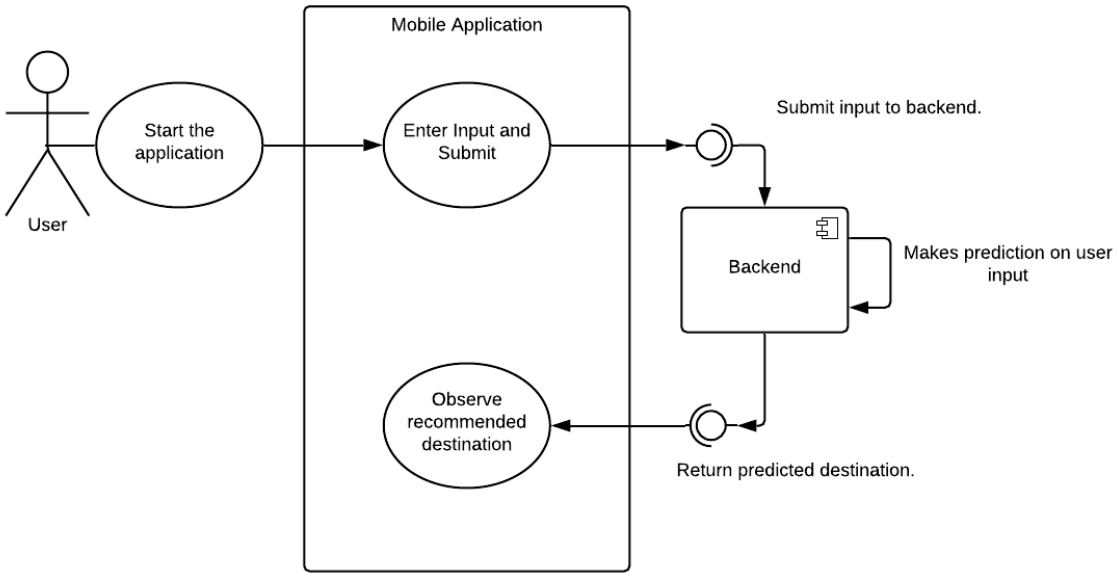

Figure 15, Recommender application with RF classification lifecycle

User input is converted to JSON format and submitted to the backend via HTTP post request (a part of REST API discussed earlier), where backend decodes the JSON and creates the feature vector for prediction. Then after the prediction, the result is converted back to JSON and sent back to the mobile application, where the resulting page is displayed according to the received prediction value. Both user input and result pages are shown in Figure-16 below on an iOS Simulator.

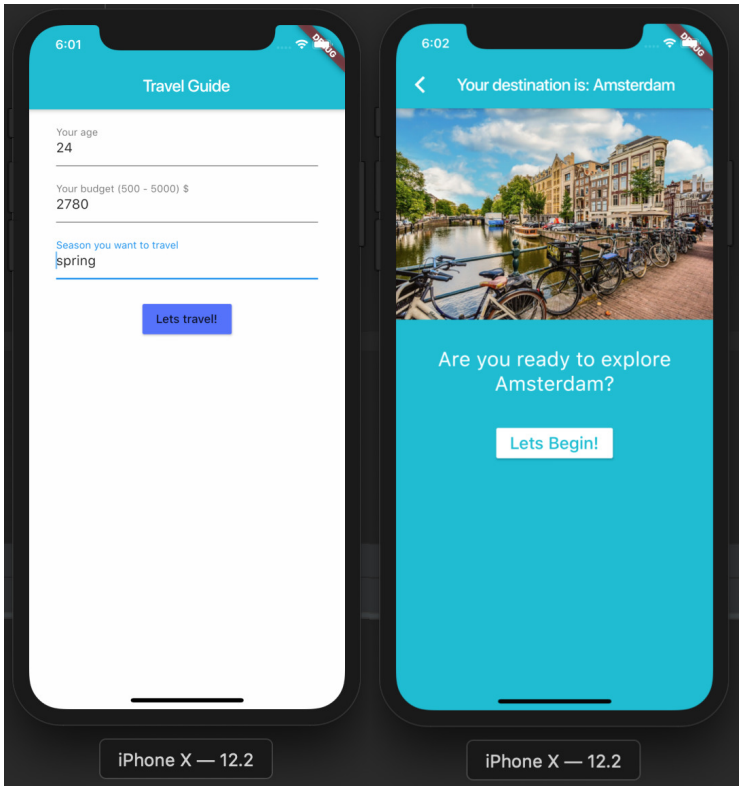

Figure 16, Input and Result pages

Although the application is represented in Figure-7 above is on iOS Simulator, the application is available for both platforms iOS and Android since it was developed using Flutter framework. Overall the application performed well and efficiently on both simulators. Since the pre-trained model was deployed on the server there were no major latency observed during the request and execution of the prediction. Moreover, the prediction results were mainly accurate as expected from the earlier training test runs. Application implementation denoted that classification algorithms are quite powerful and could be used as a part of or as a whole RS.

7.4 Version 2 RS with CS in practice

A prototype recommendation application is implemented for second version of RS, with backend server on localhost and mobile application as front-end developed with Flutter. CS model and all data manipulation code is installed on the server and UI consists of several pages, for selecting an option, displaying recommendations, displaying details of a destination, writing a review and a review classifier result. Even though it may not make much sense to implement a review classification result page for a production level application, it was implemented for classification performance demonstration purpose. Visual representation of the application is given in Figure-17 below.

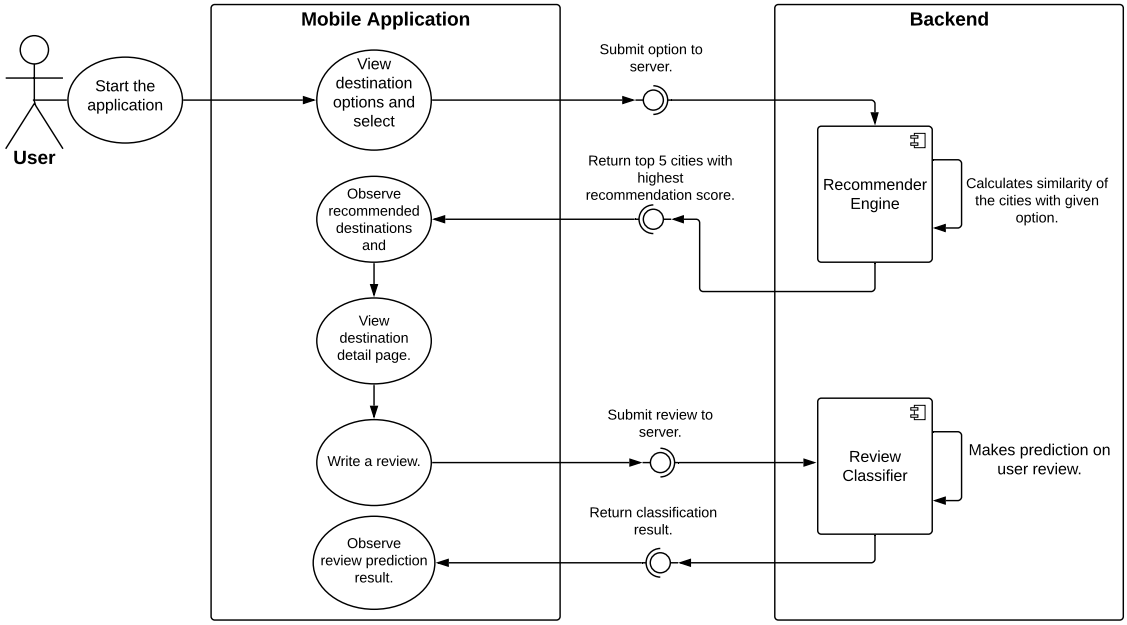

Figure 17, Recommender application with CS lifecycle

On the server side first of all, all the city description data is merged into a single String, each word is separated with regular expressions and a map object is created with words and counts. Then the map is sorted in descending order and top 25 words are identified and a vector with top 25 words is created. Upon non-contextual word identification, identified words are mapped through each description and any word occurred in descriptions is removed. Then CS is calculated for every city using the keywords of the option submitted by user and sorted by the similarity score. The top 5 city with highest similarity is then selected,

all feature values (city name, description, review count and background image) for those cities are retrieved from the dataset and returned back to the mobile application in JSON format. Users can display the recommendations depending on the selected category and can tap on a recommendation to view details about the recommended city and may write a review as well. Option selection, recommendation set selection and destination detail pages are shown in Figure-18 below.

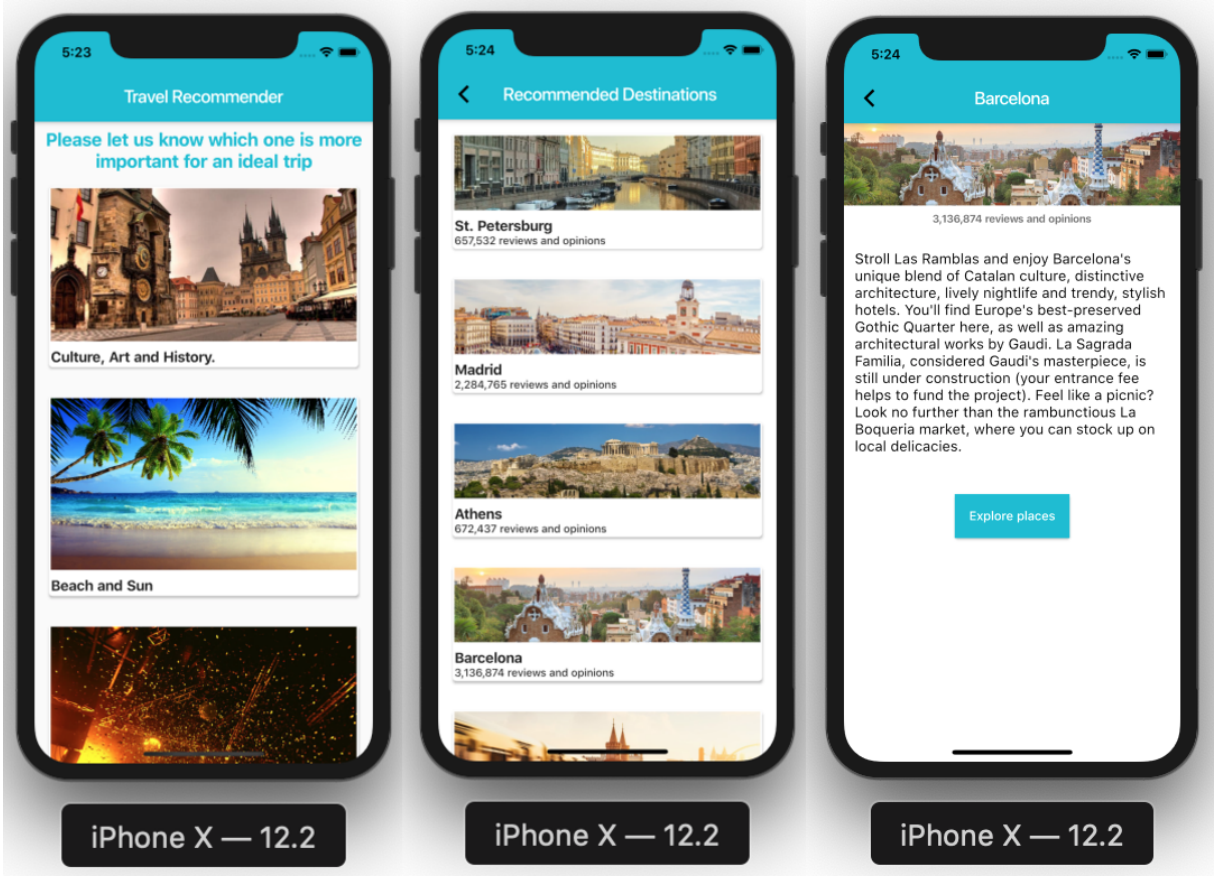

Figure 18, CS Recommender Application pages

8. FUTURE WORK

This project introduced two different approaches for a RS application with RF classification, neural network and with cosine similarity function respectively. It has also proposed and applied different methods to facilitate user feedback to address the dynamization problem of RS's. Although both RS versions could effectively function and make recommendations for the given contexts, there are still opportunities for further enhancements for a better recommendation engine. The next steps could include an implementation of a more hybrid RS, for example, a RS comprises two or more ML and/or scoring functions and working with multiple datasets could be implemented. Such a RS could then possibly apply different approaches for particular stages of an application, a RS perhaps could first apply classification to distinguish user based on some demographics or any other relevant data, then use collaborative filtering or cosine similarity to create a recommendation set. Hybrid system implementation could vary, since there are a variety of algorithms as discussed earlier.

As explained in chapter 6.3, the current RS implementation trains on a hotel review dataset which is contextually similar to destinations for feedback classification. Even though the trained model performed good accuracy for destination review trials, it could have been improved with domain adaptation. Domain adaptation can be described as applying some methods to identify a subset of features on a dataset within a certain domain, that could conduce to better prediction results for classification in another domain [48]. Hence, future work includes research and implementation for possible domain adaptation approaches for RS's. Moreover, currently, the feedback data generated by reviews are converted to rating feedback form in order to incorporate feedback uniformly. Although it displays a possible incorporation approach for the problem, it could be possibly enhanced by two approaches. First one could be introducing a parameter to determine importance of different feedback types since one type of feedback may generate more insight than another. Second one could be generating a new feedback form or dimension from available feedback types, where each feedback type could have a role during the generation with associated weights for importance and relevance determination.

User experience could be improved for application part of the system by introducing further functionality, such as implementing a new page consisting of interesting places for a given destination (by scraping more data or using publicly available data), displaying flights from a particular location and possibly create a social networking functionality to increase user engagement. Furthermore, although Flutter provided a good development tools and environment, next steps could include a more extensive comparison of cross platform frameworks to determine best performing framework for the application needs. It could also be beneficial to compare not only React Native and Flutter but also other available cross platform development frameworks as well.

9. CONCLUSION

Recommender System applications exists on many industries and provides an automated way of recommendation for the application context. Different algorithmic approaches for a recommender application design has been applied both on synthetic and existing data for travel industry by developing two different versions of recommender applications. Experiments were focused on two different RS versions in order to explore various ways of implementation with a comparative approach, depending on the contextual characteristics of a given dataset. Moreover, a variety of functions were introduced to utilize feedback in different forms, in order to deal with dynamization problem of RS's. Although the applied methods resulted in a satisfactory recommender engine which is responsive to aggregated feedback, and hence dynamic, there are still possible actions for improvements as discussed in chapter 8.

This project also developed a mobile application compatible for both iOS and Android platforms, by using a cross-platform framework Flutter, for development. Flutter and React Native frameworks firstly compared and discussed, and then two prototype applications were developed for two different versions of RS. Development process and the applications demonstrated how effectively an application can be developed for multiple platforms from a single code base. Mobile applications shown impressive performance and elegance, as good as natively developed applications.

Overall, the project developed a cross platform mobile application for destination and trip recommendation, powered by a recommender engine dynamized by feedback data. Even though the applications were not developed to a production ready stage, the prototypes could demonstrate how a dynamic RS application could be implemented. As available content keeps increasing in today's world, it is expected for RS's to be more vital and present in the applications.

REFERENCES

[1] D. Jannach, Recommender systems. New York: Cambridge University Press, 2011, p. 1.

[2] https://foursquare.com/city-guide

[3] https://www.tripadvisor.co.uk

[4] https://developer.apple.com/develop/

[5] https://developer.android.com/

[6] "A brief history of React Native", Medium, 2019. [Online]. Available: https://medium.com/react-native-development/a-brief-history-of-react-native-aae11f4ca39. [Accessed: 02- Jun- 2019]

[7] "React Native", En.wikipedia.org, 2019. [Online]. Available: https://en.wikipedia.org/wiki/React_Native. [Accessed: 02- Jun- 2019]

[8] "facebook/react-native", GitHub, 2019. [Online]. Available: https://github.com/facebook/react-native. [Accessed: 03- Jun- 2019]

[9] "React Native Questions", Stack Overflow, 2019. [Online]. Available: https://stackoverflow.com/search?q=react+native. [Accessed: 02- Jun- 2019]

[10] "Flutter (software)", En.wikipedia.org, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Flutter_(software). [Accessed: 02- Jun- 2019]

[11] "flutter/flutter", GitHub, 2019. [Online]. Available: https://github.com/flutter/flutter. [Accessed: 02- Jun- 2019]

[12] "Flutter Questions", Stack Overflow, 2019. [Online]. Available: https://stackoverflow.com/search?q=flutter. [Accessed: 02- Jun- 2019]

[13] R. Boyne, "Classification", Theory, Culture & Society, vol. 23, no. 2-3, pp. 21-30, 2006. Available: 10.1177/0263276406062529.

[14] F. Pereira, T. Mitchell and M. Botvinick, "Machine learning classifiers and fMRI: A tutorial overview", NeuroImage, vol. 45, no. 1, pp. S199-S209, 2009. Available: 10.1016/j.neuroimage.2008.11.007.

[15] M. Blanco and M. Ginovart, "How to introduce historically the normal distribution in engineering education: a classroom experiment", International Journal of Mathematical Education in Science and Technology, vol. 41, no. 1, pp. 19-30, 2010. Available: 10.1080/00207390903060628.

[16] C. Agarwal and A. Sharma, "Image understanding using decision tree based machine learning", ICIMU 2011: Proceedings of the 5th international Conference on Information Technology & Multimedia, 2011. Available: 10.1109/ICIMU.2011.6122757

[17] L. Breiman, "Random Forests", Machine Learning, vol. 45, no. 1, pp. 5-32, 2001.

[18] https://www.globalsoftwaresupport.com/random-forest-classifier-bagging-machine-learning/. [Accessed: 02- Aug- 2019].

[19] Random Forest Classifier scikit-learn", scikit-learn.org. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html [Accessed: 02- Aug- 2019]

[20] H. Abdi, D. Valentin and B. Edelman, Neural networks. Thousand Oaks (Cal.): Sage, 1999, pp. 1-3.

[21] G. Templeton, "Artificial neural networks are changing the world. What are they? - ExtremeTech", ExtremeTech, 2019. [Online]. Available: https://www.extremetech.com/extreme/215170-artificial-neural-networks-are-changing-the-world-what-are-they. [Accessed: 12- Jul- 2019].

[22] A. Lapedes and R. Farber, "How Neural Nets Work", Evolution, Learning and Cognition, pp. 331-346, 1989. Available: 10.1142/9789814434102_0012

[23] N. Slater, "What is the "dying ReLU" problem in neural networks?", Data Science Stack Exchange, 2019. [Online]. Available: https://datascience.stackexchange.com/a/5734. [Accessed: 04- Aug- 2019].

[24] R. Pascanu, T. Mikolov, Y. Bengjo, “On the difficulty of training recurrent neural networks”, ICML'13 Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28, pp. 1310-1318. Available: http://proceedings.mlr.press/v28/pascanu13.pdf

[25] "Keras | TensorFlow Core | TensorFlow", TensorFlow, 2019. [Online]. Available: https://www.tensorflow.org/guide/keras. [Accessed: 04- Aug- 2019].

[26] A. Piotrowski, "Differential Evolution algorithms applied to Neural Network training suffer from stagnation", Applied Soft Computing, vol. 21, pp. 382-406, 2014. Available: 10.1016/j.asoc.2014.03.039.

[27] L. Terveen and W. Hill, "Beyond Recommender Systems: Helping People Help Each Other", AT&T Labs - Research, pp. 7-14, 2001. Available: http://files.grouplens.org/papers/rec-sys-overview.pdf. [Accessed 6 August 2019].

[28] "An Overview of Recommendation Systems - Data Meets Media", Data Meets Media, 2019. [Online]. Available: http://datameetsmedia.com/an-overview-of-recommendation-systems/. [Accessed: 10- Jul- 2019].

[29] P. Melville and V. Sindhwani, Recommender Systems, Encyclopedia of machine learning. New York, NY: Springer, 2010, Chapter 38, pp. 2-3.

[30] C. Akbaş, O. Günay, K. Taşdemir and A. Çetin, "Energy efficient cosine similarity measures according to a convex cost function", Signal, Image and Video Processing, vol. 11, no. 2, pp. 349-356, 2016. Available: 10.1007/s11760-016-0949-7.

[31] D. Tapucu, S. Kasap and F. Tekbacak, "Performance Comparison of Combined Collaborative Filtering Algorithms for Recommender Systems", 2012 IEEE 36th Annual Computer Software and Applications Conference Workshops, 2012. Available: 10.1109/compsacw.2012.59

[32] "Euclidean vs. Cosine Distance", Cmry.github.io, 2019. [Online]. Available: https://cmry.github.io/notes/euclidean-v-cosine. [Accessed: 07- Aug- 2019].

[33] "Selenium - Web Browser Automation", Seleniumhq.org, 2019. [Online]. Available: https://www.seleniumhq.org/. [Accessed: 09- Aug- 2019].

[34] "Search results: London - TripAdvisor", Tripadvisor.co.uk, 2019. [Online]. Available: https://www.tripadvisor.co.uk/Search?q=London. [Accessed: 09- Aug- 2019].

[35] "TripAdvisor: London Detail Page", TripAdvisor, 2019. [Online]. Available: https://www.tripadvisor.co.uk/Home-g186338. [Accessed: 09- Aug- 2019].

[36] J. Bobadilla, F. Ortega, A. Hernando and J. Bernal, "A collaborative filtering approach to mitigate the new user cold start problem", Knowledge-Based Systems, vol. 26, pp. 225-238, 2012. Available: 10.1016/j.knosys.2011.07.021.

[37] S. Chang, "Onboarding New Users in Recommender Systems", GroupLens, 2019. [Online]. Available: https://grouplens.org/blog/onboarding-new-users-in-recommender-systems/. [Accessed: 10- Aug- 2019].

[38] C. Experience and S. Arora, "Recommendation Engines: How Amazon and Netflix Are Winning the Personalization Battle", Martechadvisor.com, 2019. [Online]. Available: https://www.martechadvisor.com/articles/customer-experience-2/recommendation-engines-how-amazon-and-netflix-are-winning-the-personalization-battle/. [Accessed: 11- Aug- 2019].

[39] "Cross-Platform Framework for Mobile App Development", icoderzsolutions, 2019. [Online]. Available: https://www.icoderzsolutions.com/blog/flutter-cross-platform-app-development/. [Accessed: 11- Aug- 2019].

[40] "Amazon.com: Online Shopping for Electronics, Apparel, Computers, Books, DVDs & more", Amazon, 2019. [Online]. Available: https://www.amazon.com/. [Accessed: 15- Aug- 2019].

[41] "Ratings and Reviews for New Movies and TV Shows - IMDb", IMDb, 2019. [Online]. Available: https://www.imdb.com/. [Accessed: 15- Aug- 2019].

[42] "515K Hotel Reviews Data in Europe", Kaggle.com, 2019. [Online]. Available: https://www.kaggle.com/jiashenliu/515k-hotel-reviews-data-in-europe. [Accessed: 15- Aug- 2019].

[43] S. Bird, E. Klein and E. Loper, Natural language processing with Python. O'Reilly, 2009, Chapter 3.6.

[44] A. Samuel, "Some Studies in Machine Learning Using the Game of Checkers", IBM Journal of Research and Development, vol. 3, no. 3, pp. 210-229, 1959. Available: 10.1147/rd.33.0210.

[45] "Web Services Architecture", 2004. Chapter 3.1.3 [Online]. Available: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211.

[46] "JSON", Json.org. [Online]. Available: https://www.json.org/. [Accessed: 19- Aug- 2019].

[47] "The GraphQL Foundation | An open and neutral home for the GraphQL community", Foundation.graphql.org. [Online]. Available: https://foundation.graphql.org/. [Accessed: 19- Aug- 2019].

[48] W. Kouw and M. Loog, "An introduction to domain adaptation and transfer learning", Delft University of Technology, 2019, pp. 2-3. Available: https://arxiv.org/pdf/1812.11806.pdf