HAAGA-HELIA
University of Applied Sciences

Recommendation systems in the context of tourism

Joni Kämppä

11.12.2016

HAAGA-HELIA
University of Applied Sciences

Date

Abstract

Author(s)
Joni Kämppä

Degree programme Business Information Technology	
Report/thesis title Recommendation systems in the context of tourism	**Number of pages and appendix pages** 27

Tourism industry has grown despite recent different global issues, like economic crisis. The industry is steadily growing each year globally. Lot of travelling related information is available for the travelers in form of blogs, travelling sites and applications. The search for potentially interesting sights to visit, during the trip, might become tedious task. The information sources are extensive, and the number of different establishments and tourist attractions are noteworthy. Whilst travelling abroad, the tourists don't have good information available, data roaming costs make the use of smartphones avoidable, and access to reliable and fast wireless internet-connections might vary between countries and cities. There are mobile applications available to be used as tour guides, with offline-access to location specific information, but these usually require one application per city or country. These applications are also lacking the intuition with their recommendations, and knowledge of their user, and can offer only static, predefined content.

This thesis focuses on the research for the benefits of using machine learning techniques and recommendation systems to simplify the trip planning process for the end user. Various methods could be utilized to find patterns in given user's past behavior or to find similarities and correlations between known points of interest. These smart systems could be used to give personally tailored experience and recommendations for user. These recommendations could be dependent on their schedule, preferences and budget. Machine learning would also automate the travel planning process and make the finding of the new interesting places easier for wider audiences.

The research contains background study about the tourism industry, technological aspects and describes the general idea behind machine learning. Background study is followed by closer look at the different machine learning techniques and discusses their potentiality to be used in the given context. Suitability of different machine learning algorithms is analyzed in the empirical stage of the process

The last parts of this thesis describe the process of gathering sample data. Sample data is analyzed to find the best ways to use it as training data. Three different machine learning models were constructed to find out how well the test data could be classified. Predictions would be done in context of user's preferences for places worth of visit.

Keywords
Machine learning, recommendation systems, data science

Table of contents

1 Introduction

Despite the global instability in general safety and in overall economic situation, tourism industry has continued to grow. In 2015 it grew by 4.3% globally. Yearly number of out-bound travels reached as high as 1,024 million trips by approximately 1.2 billion international tourists arrivals worldwide (IPK International, 2015). Average tourist visits 38 travel sites 45 days prior booking, where roughly seven percent of these visits are for journey planning and reviews (Expedia Media Solutions, 2013). When users are looking for potential activity in the travelling location, they are facing large number of possibilities, even for single category of interest. For example, there were 5,658 eating establishments around Brooklyn area in 2014 (Kuusisto, 2014), which makes manual search for potential eating places time consuming.

A context-aware mobile application, for travelling purposes, would simplify users access for relevant information. The application, that would recommend points of interest for the user according his or her past interests, would automate information seeking process. Searching up-to-date information for possible activities before the trip is not as troublesome, as it becomes while being abroad. Regulations in data-roaming costs have been made for EU citizens travelling inside the Union (Herrmann, Kundisch, Nicolau, & Zimmermann, 2014), but the maximum cost for data-roaming is still relatively high, 0.20€ per megabyte (Your Europe, 2016). Taking these facts into consideration, the main functionalities of the application, would be used while being abroad and out of continuous accessibility to Internet connections.

To build an application, that would-be context aware and assistive for the user, the concept of machine learning and predictive algorithms are needed. The application would update its recommendations, when it detects that the user is in foreign country and connected to Wi-Fi-network. The current location and date would be used as a context, and automatically suggested activities would be given, based on the past interests. More the user travels, more attractive the recommendations should be for the user. The chosen machine-learning algorithm should recognize patterns in users' preferences and use these patterns, as base for the suggested points of interest.

In this thesis, there is literature review where basic concepts of machine learning is studied. To find out the most suitable machine-learning algorithms, to recognize these patterns in user's preferences, a review of different online-sources for up-to-date travel information about various activities are researched. The testing of final concept with sample data and its process is described in greater detail at the end of this research.

The research will be conducted as thesis project in HAAGA-HELIA UAS located in Pasila, Finland.

2 Objectives and scope

In the following chapter, the objectives and scope of this research are described more closely. The machine-learning field is complex field of data science; therefore, the scope of the research must be kept relevant to bachelor level research. Sorting out the information sources and analysing their suitability is major part of the research. The final objective is to research different machine learning techniques, and evaluate their convenience to be utilized in the given context.

2.1 Objectives

When the user is abroad, the access for the information is harder to reach due to the high data-roaming costs (Your Europe, 2016), therefore being dependent on public wireless networks. As the average length of stay being 7.3 nights (IPK International, 2015), user has limited time to use for information searching purposes, while travelling. For these reasons, the final production level implementation would update the recommendations, based on the user's current location. Detecting that the user is abroad there is an API to access the SIM-card's information about the service provider. Application could be set to use only Wi-Fi-connections.

One of the most important objectives, is to be able to give recommendations that user would find personally interesting. This can be achieved using effective machine-learning algorithm for pattern recognition. The patterns would be user's general areas of interest, based on his or her past interest on any specific activities abroad. To effectively find out patterns, one option could be using other users' interests as a correlation matrix. The machine-learning service would be maintained on server side, where it must be able to handle a large amount of user data. Predictions would be done in remote server, partly for heavy processing needs. After analysing the data, the service must present accurate and meaningful suggestions for the end user.

To be able to make any suggestions based on the found patterns, 3rd party service would be used to search for possibly interesting activities for the user. The objective is to find out service, which has open API for their database, consisting up-to-date information about local activities. For the suggestions to be as precise as possible, the user should be presented detailed information about the interesting sights, in the current location. For example, user might be interested in contemporary art, but especially in Andy Warhol. Then the application should see this as pattern and make the most suitable suggestions, using as many variables as possible to give specific enough results.

These suggestions should be presented in attractive way, which would be easy to access for the user, even when the Internet access is not possible. The final objective is to form a predictive model, which could be used later in the implementation of the actual application.

2.2 Scope

The scope of this research is to find out data source and web service, to be used to get test data. This test data is then analysed and pre-processed as needed, to be able to find patterns in the data. These patterns can be used to offer relevant suggestions for tourist activities. This can be later taken into use, in the production application. The performance and accuracy could be tested more easily with actual application. In this research the accuracy is evaluated with available software library metrics and empirical evaluation.

3 Background study

The next chapter describes the main concepts of the machine learning. These concepts are researched in more detail and evaluated before continuing the research process. Machine learning and recommendation systems are extensive field, and there is lot of issues to be dealt with (Brownlee, 2013) , before any the implementation phase could be followed. Decisions for the later steps in the research process will be based on this background study.

3.1 Statistics on basis of machine learning

General knowledge on statistics is crucial in Machine learning, which is heavily based on statistics and probability theory (Lane, 2003). Statistics is the science of data-analysis, and learning from it (Davidian & Louis, 2012). Statistics can be used to analyse probabilities, and Machine learning is very probabilistic sub-field of computer science. In the heart of statistics, lies data in its various forms, which can be categorized to:

- Numerical Data – integer based discrete continuous data, e.g. how many cloudy days are in any given year?
- Categorical Data – qualitative with no inherent mathematical meaning, e.g. how long did checkout-process, in e-commerce website, take?
- Ordinal Data – mixture of numerical and categorical data, with mathematical meaning, e.g. movie ratings.

Types of data might influence on what algorithms are most suitable for Machine learning purposes. This should be considered before the data model is formatted. Familiarity of the key statistical terminology is expected to understand the concepts behind machine learning.

3.2 Machine learning

Machine learning is one of the subfields of computer science, where various methods are used to create sort of an artificial intelligence, that can ultimately learn new things independently (Ham, Dirin, & Laine, 2016). Machine learning was evolved, by using the knowledge on cognitive science (Goldberg & Holland, 1988), to utilize statistics and probability theory (Lane, 2003). Machine learning algorithms can recognize patterns in the input data and learn to make predictions for unknown future values, based on the sample data. Machine learning is based on supervised or unsupervised learning.

3.2.1 Unsupervised learning

In unsupervised Learning, the prediction model has no predefined answers to learn from. The algorithms analyse the given input data, and try to find patterns and natural clusters of data (Michie, Spiegelhalter, & Taylor, 1994). Unsupervised learning is practical, when the goal is to find regularity in the data, that has been previously unrecognized. These variables are called *latent variables* in statistics (“Wiktionary,” 2016). One use case for unsupervised learning, could be an e-commerce store, where product descriptions are analysed to find the most meaningful terms for a certain category.

3.2.2 Supervised learning

Compared to unsupervised learning, in supervised learning, the machine learning algorithms are given examples as expected outcome (Michie et al., 1994). The machine learning model, that is created, is used to predict the answer for new, unknown values. One example could be, where the price of a new car is predicted, based on historical car prices for cars with similar attributes.

Evaluating supervised learning is relatively straightforward. The training data, that is used to train the machine learning model, for predicting new values can be split into either two or more randomly assigned data-segments (Sarwar, Karypis, Konstan, & Riedl, 2001). One segment is always reserved as test data, to evaluate it against the training-set’s performance. The model is trained using training-set only. These data-segments need to be big enough, to contain representatives of all the variations and outliers in the data. In order to avoid overfitting, that is where the model captures more unnecessary outliers and inconsistencies from the data (Michie et al., 1994) that is needed, some validations methods can be used. One option is K-Fold Cross Validation.

In K-Fold Cross Validation the sample data is divided into K- randomly assigned segments. One segment is reserved as test data, and others are used to train the model. The performance of the remaining K-1 segments is measured against the test-set, and average of the K-1 R-Squared scores is taken (Brownlee, 2016b).

3.3 Machine learning Algorithms

Various algorithms can be used in machine learning. Finding out the most suitable algorithm, can be achieved only by testing different algorithms and choosing the most suitable one to any specific case. The form of data influences slightly on what methods should be used. If data is continuous in its nature, regression analysis is used, whereas categorical data is classified (Ham et al., 2016).

Regression analysis is statistical process of statistical modelling, that is used to estimate the relationships between given input variables (Prokhorov, 2012). In machine learning context, a line is fitted to a data set of observations, and this line is used to predict new values. Least-squares method can be used to minimize the squared-error between each point and the line. Regression analysis can be linear, polynomial or multivariate regression, based on how well the line should fit, straight line or curved. Multivariate regression is used, when more than one variable influences on the predicted value (Beylkin, Garcke, & Mohlenkamp, 2009).

Classification is used for nominal labels, where the data is not continuous (Ham et al., 2016). Whereas multivariate regression analysis could be used, for example to predict car prices based on its brand, model and other meaningful attributes, classification is used usually for recommendation systems, where similarity scores are computed using ordinal data.

Some of the machine learning algorithms were studied, before making initial test runs with the fake data, and they are described in more detail below.

3.3.1 Naïve Bayes

Naïve Bayes is related on the general Bayes' Theorem (Michie et al., 1994). The Bayes' Theorem, or Bayes' Law, is part of probability theory and is an example of conditional probability. Bayes' Theorem is stated mathematically as the equation presented below:

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}$$

Bayes' Theorem is used, for example in a medical study (Peter Armitage,Geoffrey Berry, 2002), where probability of x, given y, is studied. A classic example is a drug test, where probability of false positives, can be calculated using Bayes' Theorem. In Bayes' Theorem probability of A given B, is highly dependent on probability, that is known as base or prior probability, of B and A.

Naïve Bayes is approaching the probability theory, from little different perspective. Compared to Bayes Theorem, where variables are dependent on each other, Naïve Bayes algorithm assumes that all input variables are independent and, that the outcome is not product of those variables influencing on each other (Ham et al., 2016). Naïve Bayes is effective for handling unknown and/or missing values (Michie et al., 1994). Naïve Bayes is rather simple, but effective and scalable machine learning algorithm (Ham et al., 2016).

3.3.2 Decision tree learning

Decision tree learning is a classification method in machine learning, that forms a predictive model for mapping the input data into a predicted new value, based on its initial attributes (Gershman et al., 2010). The “tree” is divided into a nodes and branches, where a node represents potential decision to be made and branches are options, that might lead to it (Ham et al., 2016). Decision trees fall into supervised learning category, since it requires sample data, with the resulting classifications.

Decision trees are run through, from top to bottom, and each step is trying to minimize the data entropy on its way through the tree (Gershman et al., 2010). This makes it a greedy algorithm, where the tree picks the decision point, where it reduces the entropy the most, at that stage (Ham et al., 2016).

Decision trees suffer severely from overfitting (Michie et al., 1994), and ensemble learning techniques can effectively reduce the overfitting issue. Ensemble learning, in the context of decision trees, is achieved using multiple trees, instead of single tree. This technique is known as Random Forrest –concept, where alternative trees are formed and they “vote” on the final classification. The input data can be re-sampled randomly and subsets of the tree’s attributes are randomized for each stage. This technique is called bootstrap aggregating or bagging (Brownlee, 2016a).

3.3.3 Support Vector Machine (SVM)

Support Vector Machine is another technique of data classification. This technique is easily subject to unsatisfactory results, amongst beginners (Chih-Wei Hsu, Chih-Chung Chang, 2008a). Whereas Naïve Bayes’ rely on probability theory, SVM is a non-probabilistic binary linear classifier (Ham et al., 2016). In SVM, data is also separated into training and test sets. Training sets contain one target value each, i.e. the class labels, and several attributes, the goal being producing a model to predict the target values of the test data. The model is based on the training data and it makes the predictions with the attributes of the test data.

Cross-validation is important in SVM, to find the most suitable parameters (Chih-Wei Hsu, Chih-Chung Chang, 2008b) and previously described K-Fold cross validation –method is one way to achieve that.

3.3.4 Principal Component Analysis (PCA)

Principal Component Analysis is a dimensionality reduction method, that mathematically transforms several possibly correlated variables into a smaller number of variables. These variables are called principal components (Jolliffe, 2005).

Human brains are bad to process more than three dimensions (Groleau, 2003) and dimensionality reduction is used to distil higher-dimensional data, down to smaller number of dimensions, while preserving as much of the variance in the data as possible.

PCA finds “eigenvectors” in the higher dimensional data, that will define hyper planes that split the data, while preserving its variance. The data is then projected to these hyper planes, that represent the lower dimensions. One of the implementations of PCA is Singular Value Decomposition (SVD). (Jolliffe, 2005)

3.4 Recommendation systems

Recommendation systems are subset of machine learning, where various statistical and neural network algorithms can be used to recommend items and services for the user, based on their past behaviour. Recommendation systems are mostly divided into two, based on the approach they make to compute similarity and how they make recommendations for users. These two main methodologies are user-based or item-based collaborative filtering. These approaches are described in further detail, below.

3.4.1 Collaborative systems

User-based collaborative filtering differs from item-based collaborative filtering, on how the similarity scores are computed. In user-based collaborative filtering, this similarity is computed based on user-user similarity, whereas in item-based it is the relationships between items (Sarwar et al., 2001).

The problems with user-based collaborative filtering are the very nature of people. Peoples’ tastes are constantly evolving, therefore it’s hard to make precise similarity computations. Another problem is performance issue (Sarwar et al., 2001), which is due to the fact, that many times there are more users than items, e.g. total number of users of Netflix service vs. number of movies in Netflix movie catalogue. User-based collaborative filtering system can also be gamed, where fake users are made to create false sense of similarity between users. This is also called as shilling attack (Chirita, Nejdl, & Zamfir, 2005).

Another approach in recommendation systems is item-based collaborative filtering. Where people's changing personalities was an issue, items and things have more static nature. *Star Wars* -movie will always remain *Star Wars* -movie, therefore the similarity is relatively easier to compute. By common sense, there are usually fewer things to recommend than users to recommend for. This leads to less computation needs. If item-based collaborative filtering were used to movie recommendation system, one way to achieve meaningful recommendations would be:

- Finding every pair of movies that were watched by the same person
- Measuring the similarity of their ratings across all users who watched both movies
- Sorting by the movie, then by the similarity

Today there are also hybrid approaches, where the recommendation system is based partly on both methods, to avoid possible limitations in both item-based and user-based collaborative filtering techniques (Adomavicius & Tuzhilin, 2005).

3.5 Things to Consider

To effectively achieve the best results, different machine learning algorithms should be tested and evaluated. The nature of the data will tell, whether it is classification or regression issue. Different methods can be used to evaluate how well the model fits to the training data, including cross-validation methods (e.g. K-Fold), R-Squared. Methods like bootstrap aggregating or boosting can be used to avoid overfitting.

4 Empirical study

In the following chapter the basic methodology to collect and analyse the test data is described. The implementation of the predictive modes will be based on this research.

4.1 Qualitative research

The machine-learning algorithm will use qualitative data of the real and/or imaginary user and his past behavior. Statistical methods like data normalization, are used to scale the continuous data to be more efficient to predict with the algorithm.

The source of the test-data is Google’s Places Web Service API. It has several different endpoints to request place related data. Test data was based on both real past visits to 25 different locations around Europe and generated randomly, by mass queries using Google Places Radar results. Google’s Radar search returns basic identification data of 200 nearest places, by given query word. More detailed information can be asked for these places individually, using their Place Detail’s endpoint. For the test purposes, around 800 individual place details were fetched from Google Places API. Fake users and their reviews were generated using normal distribution, to be used later in the item-based collaborative system.

Qualitative data was used to find trends and natural patterns in user’s past behavior. This was utilized to build a predictable model for new unseen data. Basic classification problem for this concept, was whether certain location or place of interest could be recommended to the end user, based on the data the application already knows about him.

4.1 Deductive research

Even though the test data was collected in more qualitative way, the research continued in deductive form. The experimental nature of this research required combination of different research methods.

The collected data was used to create a predictable model for any given user and his or her travel preferences. The end goal was to test out the given theory or machine-learning model with the new unseen data and real world values.

4.2 Empirical methods

Test data and initial machine-learning predictions were empirically analyzed for their validity. Machine learning is, in its very nature, very statistical science, but the results must be reviewed closely, to verify that the used model was feasible enough to produce trustworthy predictions.

4.3 Methodology implementation

The initial, smaller and more precise dataset, was collected with the information from past visits to various sightseeing locations around Europe. The data was fetched from Google Places API (Google, 2016). This initial dataset of 25 different locations, were used as basis for machine learning methods that require smaller dataset to begin with. The predictions can be accurate with less data than in some of the more complex approaches. This data was saved as comma separated value (CSV) -files. Creating a database and required database schemas were out of the scope of this research.

Second test dataset was collected programmatically, with few different simple Python applications. These applications were programmed to fetch Google Radar results from given latitude, longitude locations with query word “tourist attraction”. Google’s Radar API (Google, 2016) returns maximum 200 results from given area. 5 km radius was used, together with coordinate and query parameters.

Google Radar results contains only very basic information about the places, e.g. place id. These place ids were stored in csv-file, to be used in other Python programs. Place ids were then used, to fetch all the information, for every place that Google Radar returned from original queries. This data was stored in different csv-file, with the most meaningful features, like opening times, user rating averages, and price ranges.

There was need for a large set of user ratings, for item-based collaborative recommendation system. These ratings were generated, with another Python program, for fake user set of 500 users. The reviews were randomly either null valued or generated in range from one to five, including half-integers. The randomized reviews were generated, centered on the Google’s average of all their ratings, with standard deviation of 1.5. This was done to simulate natural distribution of the reviews, and make the original average rating stay relatively same.

4.4 Limitations with the collected data

The data, that was collected in the early stages of research, had some limitations in it. The choice for source of location and sights data, was difficult one to make. The difference in the level of detail was quite small, but accessibility and ease of use, varied.

Some sources like OpenStreetMap (OpenStreetMap, 2016) have their data available free of charge. OpenStreetMap provides the entire database as downloadable comma-separated file. The problem with OpenStreetMap was the level of detail they had about the locations. The information was very low in detail; mostly only the name of the location and the amenity. Their API is not as well documented, as Google’s equivalent.

Google’s Places API (Google, 2016) is very well documented and easy to use in general. They have different options for different platforms, but for simple Web Service, HTTP-requests were also possible. Places API has very detailed information about the places, including average rating information, as well as variable amount of individual user reviews. The problem with Google Places, place details, is ambiguous place type categorization. For potential user, there would be a big difference between e.g. contemporary art museums and classical art museums. Google knows these places only by categories museum, place of interest and establishment. Latter two types were useless, in that sense that they occurred in every place details, that were observed in the test data.

Other options were reviewed, but there were availability problems. It was also questionable, whether they were up-to-date.

5 Design

In the following chapter the design process before the implementation process is briefly described. At this stage of the research the design is relatively simple and the solutions are not adjusted to be used in production level. The goal is to test out the machine learning algorithms, evaluate their performance and test them with new unseen data.

5.1 IPython notebooks as basis of the design

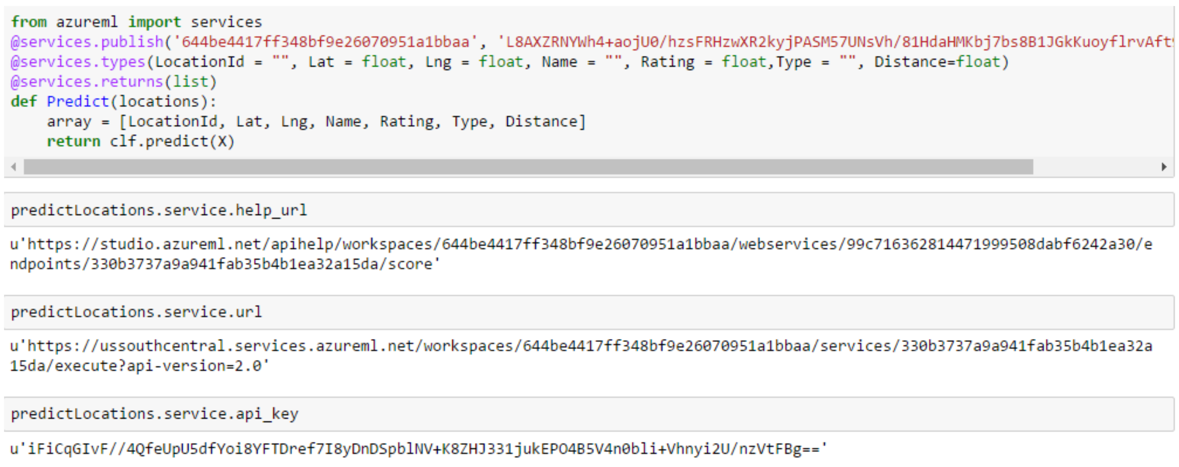
Interactive python notebooks are part of Jupyter (Jupyter, n.d.) , and they are simple way to write small Python programs that can be written and executed in the browser. They are used in some of the hosted machine learning platforms like Microsoft Azure (Mortazavi, 2015) and therefore powerful way to build initial test models.

5.2 Local notebooks

The models were originally build in local development environment before searching for potential hosting solutions. Enthought Canopy (Enthought Inc., 2016) was used to quickly create new notebooks for slightly variable solutions. Enthought Canopy has package manager which makes it easier to download necessary Python libraries.

5.3 Hosted notebooks

Some testing with hosted machine learning models were done using Microsoft Azure Machine Learning solutions. Azure has machine learning workspaces (Ericson, Gyger, Whitlatch, Nevil, & Franks, 2016) that can be used to host a model in the Azure platform. This option was tested for the Decision Tree and Bayesian -classifiers. Azure Machine Learning Workspaces support IPython Notebooks, and the process of teaching the model and making it publicly available is relatively simple. The process how the implemented Decision Tree model was published is shown below.



```
from azureml import services
@services.publish('644be4417ff348bf9e26070951a1bbaa', 'L8AXZRNYWh4+aojU0/hzsFRHzwXR2kyjPASM57UNsVh/81HdaHMKbj7bs8B1JGkKuoyflrvAft
@services.types(LocationId = "", Lat = float, Lng = float, Name = "", Rating = float,Type = "", Distance=float)
@services.returns(list)
def Predict(locations):
    array = [LocationId, Lat, Lng, Name, Rating, Type, Distance]
    return clf.predict(X)
```

```
predictLocations.service.help_url
```

```
u'https://studio.azureml.net/apihelp/workspaces/644be4417ff348bf9e26070951a1bbaa/webservices/99c716362814471999508dabf6242a30/e
ndpoints/330b3737a9a941fab35b4b1ea32a15da/score'
```

```
predictLocations.service.url
```

```
u'https://ussouthcentral.services.azureml.net/workspaces/644be4417ff348bf9e26070951a1bbaa/services/330b3737a9a941fab35b4b1ea32a
15da/execute?api-version=2.0'
```

```
predictLocations.service.api_key
```

```
u'iFiCqGIvF//4QfeUpU5dfYoi8YFTDref7I8yDnDSpblNV+K8ZHJ331jukEPO4B5V4n0bli+Vhnyi2U/nzVtFBg=='
```

Figure 1 Screenshot from Azure Machine Learning Workspace

The figure above shows the process that is done after the model is taught as it would in local IPython notebooks. The model is published with azureml-pyhton library's "services"-module methods. The module has helper methods to show how it's possible to access the service as Restful-API to predict new classifications.

5.4 Design of item-based collaborative filtering

To efficiently use the item-based collaborative filtering techniques in production environment, better solutions would be needed that with Decision Tree and Bayesian -classifiers. The amount of location and user review data would be large in production application and the need of an actual database and architecture around it would be necessary. For testing purposes, even item-based collaborative filtering used the notebooks and csv-files for data when efficiency and maintainability were not a concern.

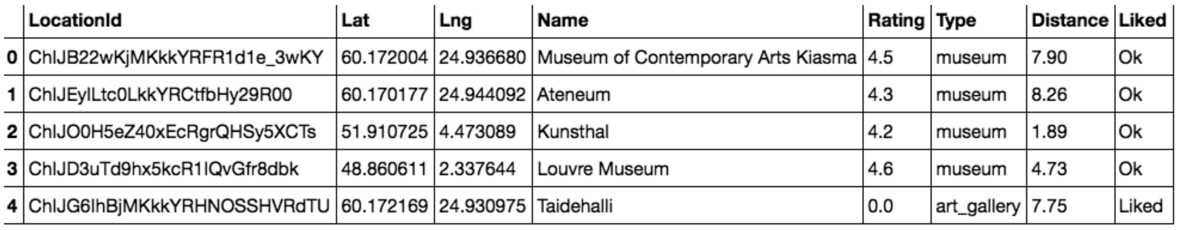
6 Implementation

In the following chapter, the three possible machine learning predictions models are described. Their performance and main problem areas are analysed and discussed. For every model, there are excerpts from the code, to show together with the description of the model.

6.1 Naïve Bayesian Classifier model

Bayes Theorem and Naïve Bayes are simple, but efficient classification methods in machine learning. In some cases, it has outperformed more complex algorithms. One example, where Bayes Theorem is used, is spam-mail filtering, where pre-classified collection of emails is labeled as spam or ham. Bayes Theorem is used to calculate probability of an appearance of a word in either one of them. In the case of this research, few different versions of Bayesian classifier were tested.

In the dataset, that was collected for this research, the calculated word counts were place types, e.g. museum, bar or clothing store. Multinomial Bayesian classifier can count the probability of a certain place category to belonging to certain classification. Other parameters for the model were distance, from the original location of the user to the visited location, and average rating score in Google's service, from where the data was fetched. The picture below show the structure of the test data that was used.



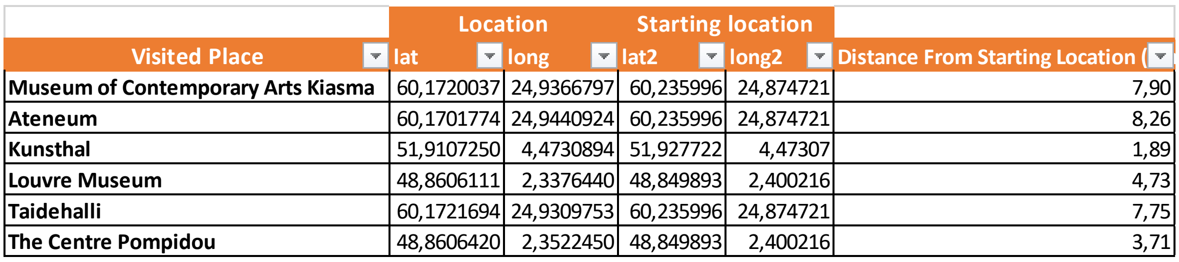
	LocationId	Lat	Lng	Name	Rating	Type	Distance	Liked
0	ChIJB22wKjMKkkYRFR1d1e_3wKY	60.172004	24.936680	Museum of Contemporary Arts Kiasma	4.5	museum	7.90	Ok
1	ChIJEyILtc0LkkYRCtfbHy29R00	60.170177	24.944092	Ateneum	4.3	museum	8.26	Ok
2	ChIJO0H5eZ40xEcRgrQHSy5XCTs	51.910725	4.473089	Kunsthal	4.2	museum	1.89	Ok
3	ChIJD3uTd9hx5kcR1lQvGfr8dbk	48.860611	2.337644	Louvre Museum	4.6	museum	4.73	Ok
4	ChIJG6lhBjMKkkYRHNOSSHVRdTU	60.172169	24.930975	Taidehalli	0.0	art_gallery	7.75	Liked

Figure 2 Test dataset for Multinomial Bayesian Classifier

As with any other Scientific Python's machine-learning algorithm, the Bayesian classifiers does not work directly with the categorical data, here the type field, and it must be converted to meaningful numerical form. One of options, would be using the textual feature extraction method, from Sklearn-library, called CountVectorizer. It calculates each word and converts the word count, into a Python dictionary of key-value pairs. In the final model the DictVectorizer-method, from same feature extraction library, was used to turn the type-feature into dictionary, where textual names were mapped as zeros and ones.

Rating-feature data came from Google Places API. It is an average of all the individual reviews, that are added to the Google. Every place does not have the rating information. In the dataset of 25 place details, their overall average was quite high, being 3.768. Therefore, variance was quite low in the test dataset.

The distance-feature was manually counted for the test dataset. The idea was, to store the distance from the user's original location to the visited location. Hypothesis was, that there could be some correlation between the distance user was ready to go to visit some locations during his or her trip, and whether the same place was liked by that person. Given the fact that average trips are rather short, as found in the literature overview, this could be a significant factor, when user is planning his or her stay abroad. Below here, is a table from the original excel, where distance vectors were calculated from original location from where the visit to the location started.



	Location		Starting location		
Visited Place	lat	long	lat2	long2	Distance From Starting Location (
Museum of Contemporary Arts Kiasma	60,1720037	24,9366797	60,235996	24,874721	7,90
Ateneum	60,1701774	24,9440924	60,235996	24,874721	8,26
Kunsthal	51,9107250	4,4730894	51,927722	4,47307	1,89
Louvre Museum	48,8606111	2,3376440	48,849893	2,400216	4,73
Taidehalli	60,1721694	24,9309753	60,235996	24,874721	7,75
The Centre Pompidou	48,8606420	2,3522450	48,849893	2,400216	3,71

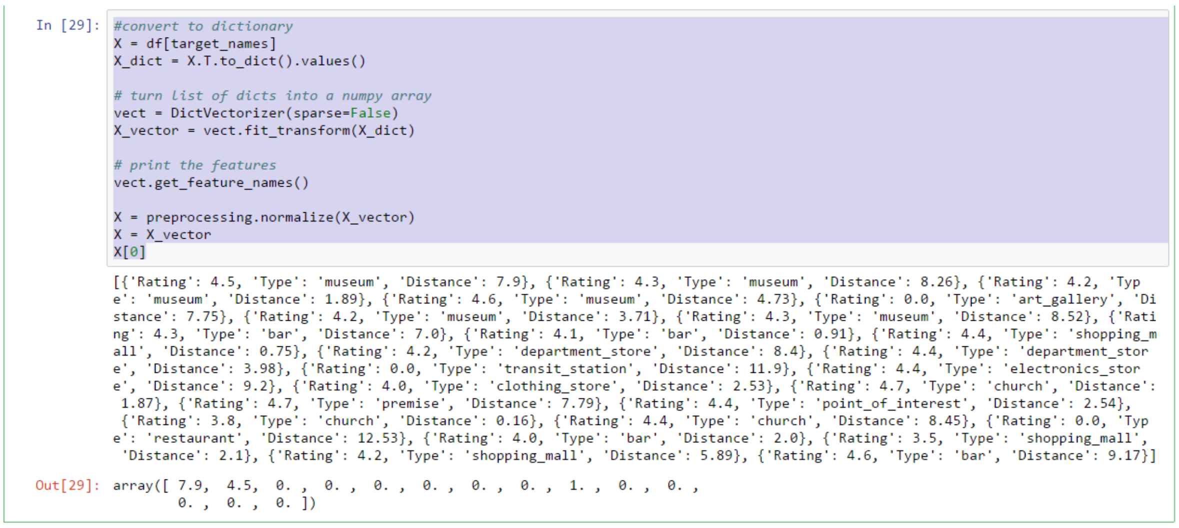
Figure 3 Calculating distance vectors

In machine learning, there is sometimes issue with numbers with different weight or scale. In this dataset, there was a scale difference between rating, ranging from one to five, and distance which was continuous with no upper limit. Sklearn- libraries have functions for data preprocessing, like normalizing data in the same range, e.g. -1 to 1. The data frame including number encoded type features, was normalized and the train/test scores were compared before and after normalizing the fields. With the data that was used for training and testing the model, there was no significant difference between normalized and unnormalized data. With the normalize function there was also unwanted effect of encoded type fields becoming normalized in the same scale, if the normalization function was applied in the wrong time. Normalization also adds complexity to predicting new values, where the scale must match the original normalized scale. The following table shows the variance in the distance values in test dataset.

Average of distances	Maximum distance	Minimum distance	Standard deviation
5,596783833	12,52983285	0,160706477	3,552431351

Table 1 Data variation figures of test data

For the classification, there was tests for Boolean and multi-class classifications. The Boolean classification was “Liked/Not Liked” classification for user’s likelihood for being interested about the location. The multi-class alternative included third option, falling between two extremes, “Not Liked/Ok/Liked”. The classification features had to be encoded with sklearn- method LabelEncoder. Below is a screenshot of the data, that was preprocessed in the form where the used Python libraries would work.

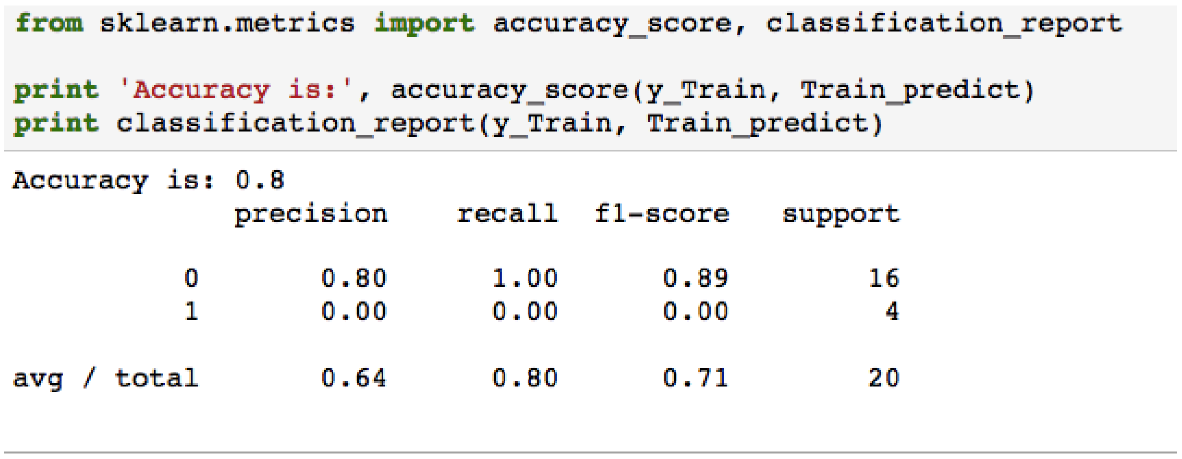


```
In [29]: #convert to dictionary
         X = df[target_names]
         X_dict = X.T.to_dict().values()

         # turn list of dicts into a numpy array
         vect = DictVectorizer(sparse=False)
         X_vector = vect.fit_transform(X_dict)

         # print the features
         vect.get_feature_names()

         X = preprocessing.normalize(X_vector)
         X = X_vector
         X[0]

         [{'Rating': 4.5, 'Type': 'museum', 'Distance': 7.9}, {'Rating': 4.3, 'Type': 'museum', 'Distance': 8.26}, {'Rating': 4.2, 'Typ
         e': 'museum', 'Distance': 1.89}, {'Rating': 4.6, 'Type': 'museum', 'Distance': 4.73}, {'Rating': 0.0, 'Type': 'art_gallery', 'Di
         stance': 7.75}, {'Rating': 4.2, 'Type': 'museum', 'Distance': 3.71}, {'Rating': 4.3, 'Type': 'museum', 'Distance': 8.52}, {'Rati
         ng': 4.3, 'Type': 'bar', 'Distance': 7.0}, {'Rating': 4.1, 'Type': 'bar', 'Distance': 0.91}, {'Rating': 4.4, 'Type': 'shopping_m
         all', 'Distance': 0.75}, {'Rating': 4.2, 'Type': 'department_store', 'Distance': 8.4}, {'Rating': 4.4, 'Type': 'department_stor
         e', 'Distance': 3.98}, {'Rating': 0.0, 'Type': 'transit_station', 'Distance': 11.9}, {'Rating': 4.4, 'Type': 'electronics_stor
         e', 'Distance': 9.2}, {'Rating': 4.0, 'Type': 'clothing_store', 'Distance': 2.53}, {'Rating': 4.7, 'Type': 'church', 'Distance':
          1.87}, {'Rating': 4.7, 'Type': 'premise', 'Distance': 7.79}, {'Rating': 4.4, 'Type': 'point_of_interest', 'Distance': 2.54},
          {'Rating': 3.8, 'Type': 'church', 'Distance': 0.16}, {'Rating': 4.4, 'Type': 'church', 'Distance': 8.45}, {'Rating': 0.0, 'Typ
         e': 'restaurant', 'Distance': 12.53}, {'Rating': 4.0, 'Type': 'bar', 'Distance': 2.0}, {'Rating': 3.5, 'Type': 'shopping_mall',
          'Distance': 2.1}, {'Rating': 4.2, 'Type': 'shopping_mall', 'Distance': 5.89}, {'Rating': 4.6, 'Type': 'bar', 'Distance': 9.17}]

Out[29]: array([ 7.9,  4.5,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  1. ,  0. ,  0. ,
                 0. ,  0. ,  0. ])
```

Figure 4 Preprocessed sample data

As seen above, categorical textual data was mapped into zeros and ones, where number zero meant absence of the corresponding place type value in that index of the array. Looping the dictionary would be the only way to find out the correct index, if the model was used as a Web Service.

The multinomial Bayesian classifier resulted in a relatively good result, having accuracy of 80%. The accuracy was counted with Sklearn metrics-library. Results are presented in the figure below.



```python
from sklearn.metrics import accuracy_score, classification_report

print 'Accuracy is:', accuracy_score(y_Train, Train_predict)
print classification_report(y_Train, Train_predict)
```

```
Accuracy is: 0.8
             precision    recall  f1-score   support

          0       0.80      1.00      0.89        16
          1       0.00      0.00      0.00         4

avg / total       0.64      0.80      0.71        20
```

Figure 5 Bayesian classifier's accuracy metrics

Precision here, is so called positive predictive value indicating the fraction of the instances that were relevant and recall is sensitivity or fraction of relevant instances retrieved (“Precision and recall,” 2016).

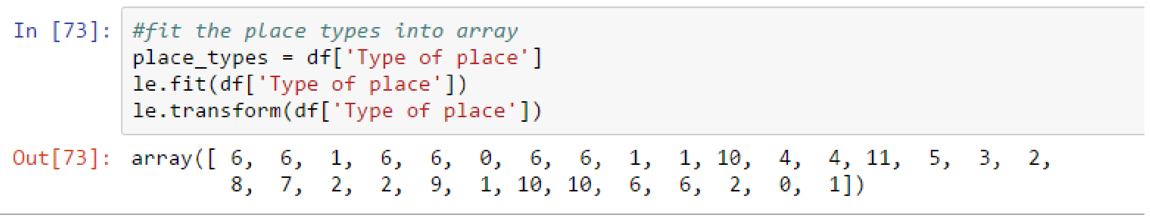
The accuracy of 80% could be good, but the model failed empirical tests with new data. Classifications looked mostly correct, but there was strange phenomenon, where place types that were the most liked in the test set, were classified as “Not Liked”. At the same time, the new place type with distance of 120km could be classified as “Liked”. The reason behind this, could be the fact that there was too little variance in the test data, especially in average rating scores.

With bigger training dataset and good test set, the performance could be more accurately evaluated and the strange misclassifications could become rarer.

6.2 Decision Tree Classifier

Another option for small initial dataset, and quicker learning curve for the machine learning system, was Decision Tree Classifier. In the case when R-language would be used as programming language, decision trees would naturally support categorical data, like here the place categories, but in Scientific Python -libraries, these had to be converted to numerical values.

Simple conversion to numbers could not be done, since the Scientific Python’s decision tree classifier treats numbers as continuous data. If the mapping of place types were done as converting them into an array from 0 to the total count of different place type categories, would this lead to a model, where some numerical value in the range would be treated as dividing line for classification. Which means if museum type was present the most and it had value of 63, numbers smaller than 63 could be treated as they belong for “Not Liked” -classification. This issue was faced in the first tests, where place categories were mapped like this. Scientific Python has its own preprocessing library for these conversions, where the numbers are converted into arrays or dictionaries, that will not lead to same behavior in the model. Below is the figure showing the conversion functions used to preprocess the data.



```
In [73]:  #fit the place types into array
          place_types = df['Type of place']
          le.fit(df['Type of place'])
          le.transform(df['Type of place'])

Out[73]:  array([ 6,  6,  1,  6,  6,  0,  6,  6,  1,  1, 10,  4,  4, 11,  5,  3,  2,
                  8,  7,  2,  2,  9,  1, 10, 10,  6,  6,  2,  0,  1])
```

Figure 6 Encoding the place type feature into numerical values

The decision tree classifier used same dataset for training as the Bayesian classifier. The decision tree tried to classify given feature array using an entropy as its evaluation criterion. Below is the figure showing the picturized tree-model.

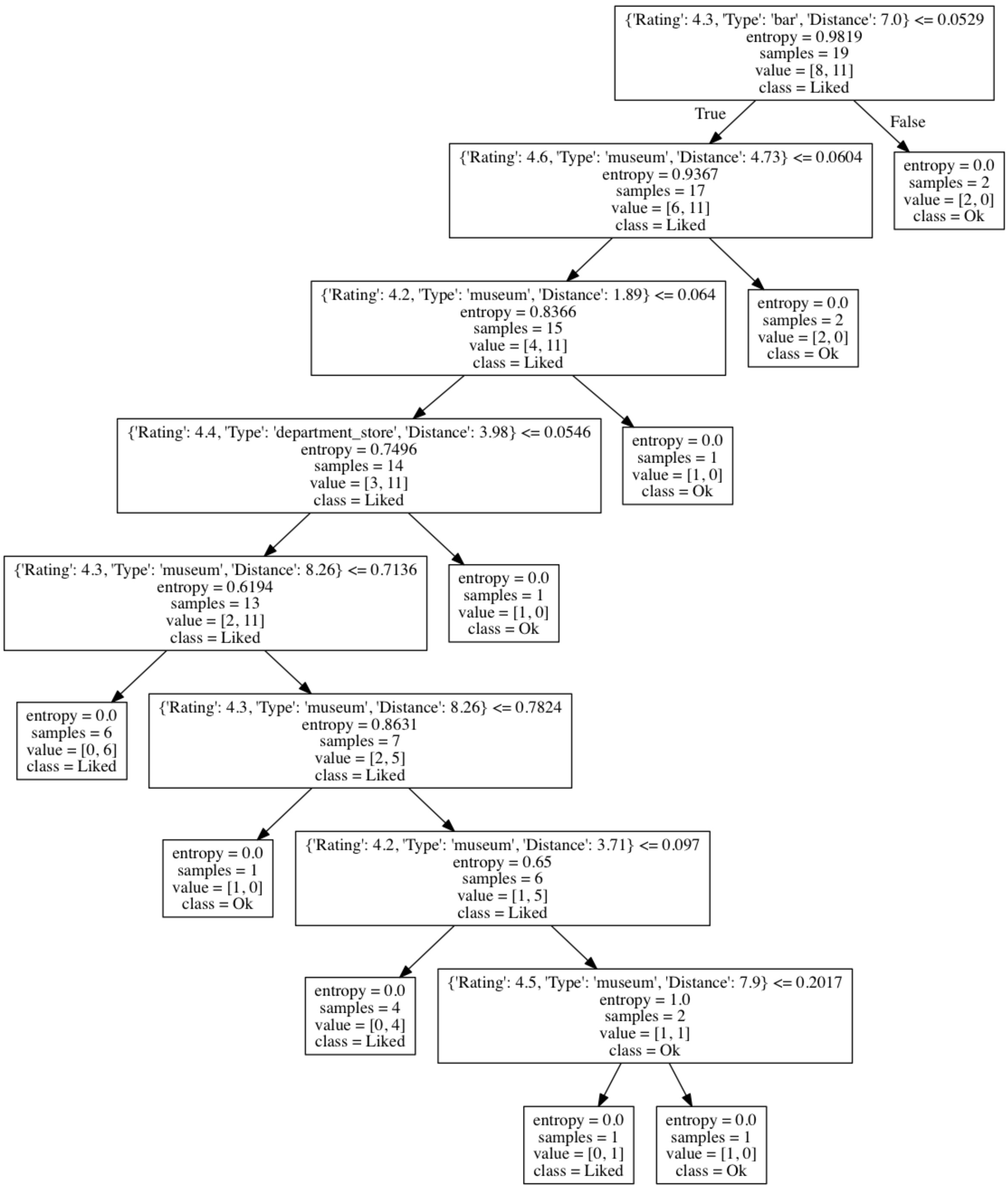
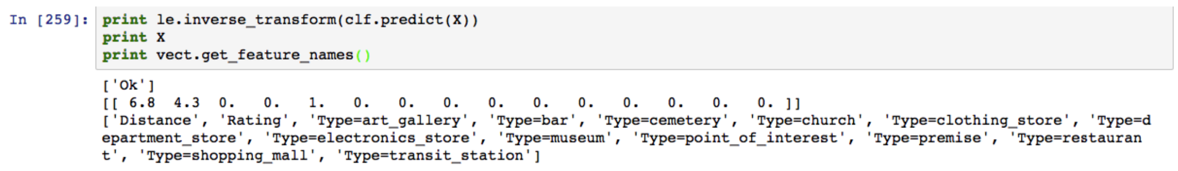

{'Rating': 4.3, 'Type': 'bar', 'Distance': 7.0} <= 0.0529 entropy = 0.9819 samples = 19 value = [8, 11] class = Liked
True
False
{'Rating': 4.6, 'Type': 'museum', 'Distance': 4.73} <= 0.0604 entropy = 0.9367 samples = 17 value = [6, 11] class = Liked
entropy = 0.0 samples = 2 value = [2, 0] class = Ok
{'Rating': 4.2, 'Type': 'museum', 'Distance': 1.89} <= 0.064 entropy = 0.8366 samples = 15 value = [4, 11] class = Liked
entropy = 0.0 samples = 2 value = [2, 0] class = Ok
{'Rating': 4.4, 'Type': 'department_store', 'Distance': 3.98} <= 0.0546 entropy = 0.7496 samples = 14 value = [3, 11] class = Liked
entropy = 0.0 samples = 1 value = [1, 0] class = Ok
{'Rating': 4.3, 'Type': 'museum', 'Distance': 8.26} <= 0.7136 entropy = 0.6194 samples = 13 value = [2, 11] class = Liked
entropy = 0.0 samples = 1 value = [1, 0] class = Ok
entropy = 0.0 samples = 6 value = [0, 6] class = Liked
{'Rating': 4.3, 'Type': 'museum', 'Distance': 8.26} <= 0.7824 entropy = 0.8631 samples = 7 value = [2, 5] class = Liked
entropy = 0.0 samples = 1 value = [1, 0] class = Ok
{'Rating': 4.2, 'Type': 'museum', 'Distance': 3.71} <= 0.097 entropy = 0.65 samples = 6 value = [1, 5] class = Liked
entropy = 0.0 samples = 4 value = [0, 4] class = Liked
{'Rating': 4.5, 'Type': 'museum', 'Distance': 7.9} <= 0.2017 entropy = 1.0 samples = 2 value = [1, 1] class = Ok
entropy = 0.0 samples = 1 value = [0, 1] class = Liked
entropy = 0.0 samples = 1 value = [1, 0] class = Ok

Figure 7 Decision Tree classifier outputted as image

The used model was suffering from severe overfitting, where the accuracy score was inaccurately 100%. This could have been due to too small dataset, which could not be divided into meaningful enough train/test -sets.

Even though the accuracy score was 100%, the model was giving classifications for new data, that made more sense that with the Bayesian classifier. With the tree model, candidates for places to visit, that were like previously liked places, were classified more correctly, as the figure below shows.



```
In [259]: print le.inverse_transform(clf.predict(X))
          print X
          print vect.get_feature_names()

          ['Ok']
          [[ 6.8  4.3  0.   0.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0. ]]
          ['Distance', 'Rating', 'Type=art_gallery', 'Type=bar', 'Type=cemetery', 'Type=church', 'Type=clothing_store', 'Type=d
          epartment_store', 'Type=electronics_store', 'Type=museum', 'Type=point_of_interest', 'Type=premise', 'Type=restauran
          t', 'Type=shopping_mall', 'Type=transit_station']
```

Figure 8 Predictions for place candidates with Decision Tree-classifier

In the above case place candidate was almost seven kilometers away from original location, had average rating of 4.3 and was type of cemetery.

With a different programming language, like statistical programming language R, decision tree classifier would be ideal solution if the most meaningful feature is categorical data type. Google's Places data had the largest number of potential candidates to be used as features for the model, but the problem was the low variance in ratings, missing data and inconsistence in the results. For example, in bigger cities, place details included price-range information, but it appeared in the test data so rarely, that it would not be good feature. Other data sources were missing ratings and didn't have any other good feature candidates. Place type on its own, would have been too little for the model to be based on. It wasn't clear, whether the accuracy score was due to too small number of features to base the tree on, or small size of the dataset, having only 25 rows of visit data.

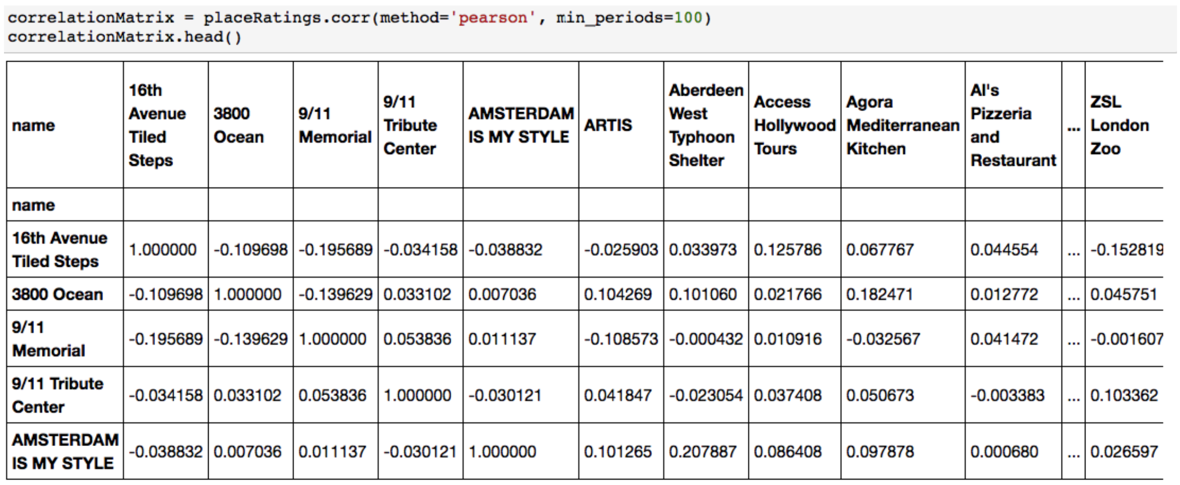
With the decision tree classifier, the random forest classifier was also tested, but the problem with unrealistic accuracy score didn't go away. Ideally random forest, with *n*-number of trees, would be better option than using a single tree. Since the accuracy score didn't change to more realistic one, random forest classifier was not used.

6.3 Item-Based Collaborative System

As the research was done for recommendation system type of use case, item-based and user-based collaborative systems would be the most ideal ones for good recommendations. Collaborative filtering algorithms are known for example from Netflix Prize ("Netflix Prize," 2016).

To get the best results out of the collaborative filtering technique, this solution required larger dataset to test the concept. The used technique calculated correlation scores for sparse matrix of every place and rating pair in the dataset. This solution, and item-based collaborative filtering in general, require more data to give meaningful results. Total of almost 800 place details, were programmatically fetched from Google Places API. Since the solution required also *n*-number of fake users, and their ratings for another *n*-number of random places, these ratings were also programmatically generated. To simulate realistic behavior, the average rating from Google Places API was used to center the randomized user ratings around it, with standard deviation of 1.5. All ratings were rounded to closest half integer, and randomly some ratings were skipped completely. Range from 0 to 500, was used as user IDs. Every user either “gave” rating to any single place in the places data or not. Overall, the test dataset had 150 000 user ratings for each of the roughly 800 places.

For these place-rating pairs, initial correlation score was calculated to use as the basis of the later recommendations. The figure below shows the correlation matrix of test data.



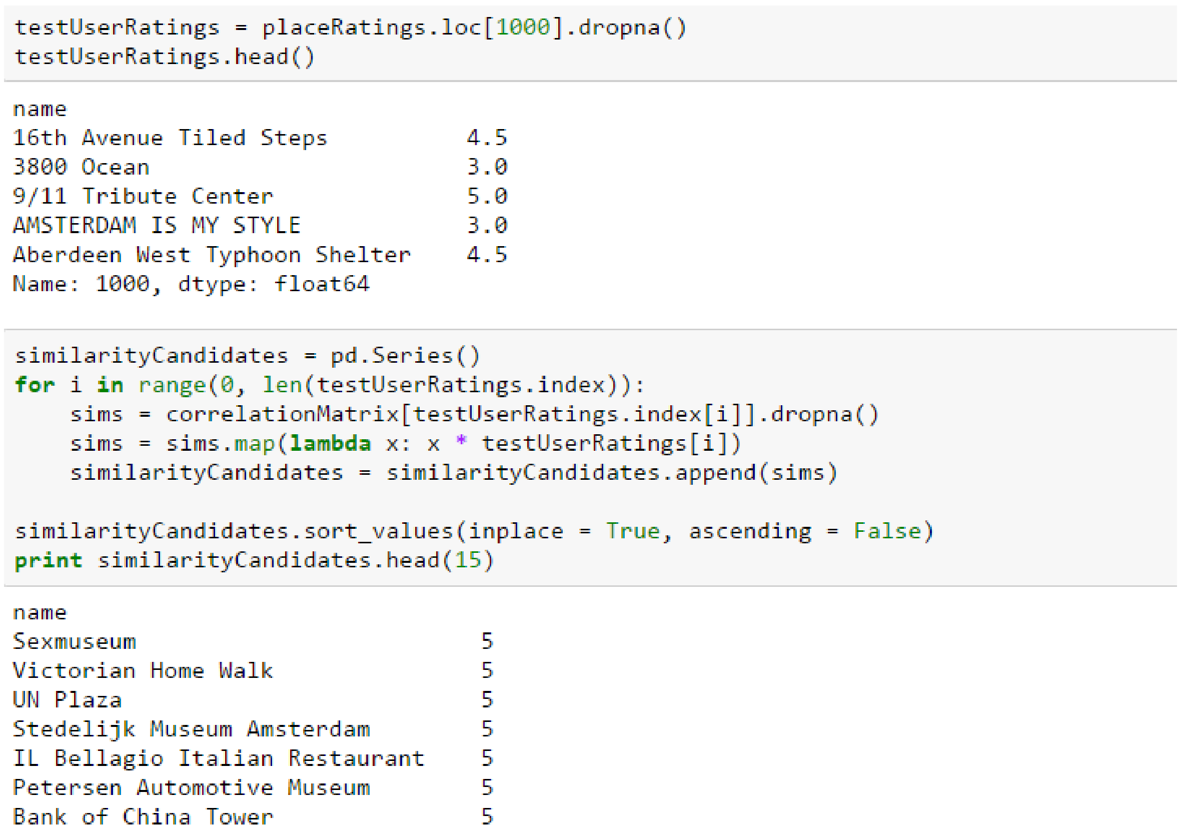
```
correlationMatrix = placeRatings.corr(method='pearson', min_periods=100)
correlationMatrix.head()
```

name	16th Avenue Tiled Steps	3800 Ocean	9/11 Memorial	9/11 Tribute Center	AMSTERDAM IS MY STYLE	ARTIS	Aberdeen West Typhoon Shelter	Access Hollywood Tours	Agora Mediterranean Kitchen	Al's Pizzeria and Restaurant	...	ZSL London Zoo
name												
16th Avenue Tiled Steps	1.000000	-0.109698	-0.195689	-0.034158	-0.038832	-0.025903	0.033973	0.125786	0.067767	0.044554	...	-0.152819
3800 Ocean	-0.109698	1.000000	-0.139629	0.033102	0.007036	0.104269	0.101060	0.021766	0.182471	0.012772	...	0.045751
9/11 Memorial	-0.195689	-0.139629	1.000000	0.053836	0.011137	-0.108573	-0.000432	0.010916	-0.032567	0.041472	...	-0.001607
9/11 Tribute Center	-0.034158	0.033102	0.053836	1.000000	-0.030121	0.041847	-0.023054	0.037408	0.050673	-0.003383	...	0.103362
AMSTERDAM IS MY STYLE	-0.038832	0.007036	0.011137	-0.030121	1.000000	0.101265	0.207887	0.086408	0.097878	0.000680	...	0.026597

Figure 9 Correlation matrix for Item-Based Collaborative Filtering

The correlation method takes parameters, like the used correlation method, here being Pearson, and min-periods which dropped rating-place -pairs with less than 100 occurrences from the dataset. This would remove potential skews in the correlation scores, occurred when places with very few ratings, would be falsely highly correlated.

One of the user was chosen to represent the imaginary test user, and all his ratings were extracted from entire dataset. Then the ratings were boosted, with lambda-function, to make more variance in the correlation scores. The figures below represent this process.



```
testUserRatings = placeRatings.loc[1000].dropna()
testUserRatings.head()
```

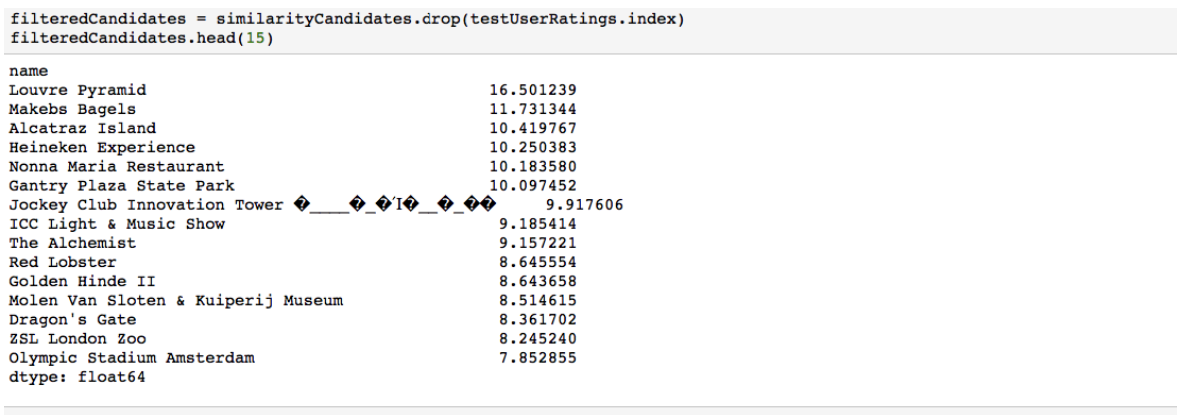
```
name
16th Avenue Tiled Steps         4.5
3800 Ocean                      3.0
9/11 Tribute Center             5.0
AMSTERDAM IS MY STYLE           3.0
Aberdeen West Typhoon Shelter   4.5
Name: 1000, dtype: float64
```

```
similarityCandidates = pd.Series()
for i in range(0, len(testUserRatings.index)):
    sims = correlationMatrix[testUserRatings.index[i]].dropna()
    sims = sims.map(lambda x: x * testUserRatings[i])
    similarityCandidates = similarityCandidates.append(sims)

similarityCandidates.sort_values(inplace = True, ascending = False)
print similarityCandidates.head(15)
```

```
name
Sexmuseum                          5
Victorian Home Walk                5
UN Plaza                           5
Stedelijk Museum Amsterdam         5
IL Bellagio Italian Restaurant     5
Petersen Automotive Museum         5
Bank of China Tower                5
```

Figure 10 Extracting the test user, with id 1000, from the entire dataset and boosting the rating scores

The screenshot from the code above, shows how all the place ratings for test-user with id 1000, are extracted from all the place ratings. Then for every similar place in correlation matrix, the correlation score is scaled accordingly to how well the test user had rated the place. This will add more weight to places that have higher correlation/similarity score. These scores are then grouped together to calculate sum of correlation for places, that correlate with more than one place. The figure below shows the final list of candidates for potential locations, that could be recommended for user 1000. All the places that user have already visited are dropped from the result.



```
filteredCandidates = similarityCandidates.drop(testUserRatings.index)
filteredCandidates.head(15)
```

```
name
Louvre Pyramid                                 16.501239
Makebs Bagels                                  11.731344
Alcatraz Island                                10.419767
Heineken Experience                            10.250383
Nonna Maria Restaurant                         10.183580
Gantry Plaza State Park                        10.097452
Jockey Club Innovation Tower �____�_�'I�__�_��       9.917606
ICC Light & Music Show                          9.185414
The Alchemist                                   9.157221
Red Lobster                                     8.645554
Golden Hinde II                                 8.643658
Molen Van Sloten & Kuiperij Museum              8.514615
Dragon's Gate                                   8.361702
ZSL London Zoo                                  8.245240
Olympic Stadium Amsterdam                       7.852855
dtype: float64
```

Figure 11 Showing potential locations for visiting

The tested item-based collaborative filtering approach gave the most meaningful results, but the problem with it is the need of large amount of data to make it precise enough. Some of the other potential problems, with this approach, are the possibility of users that want to game the system and distort the results. For some e-commerce sites, that might use item-based collaborative filtering for their product recommendations, it is logical to count in only those items, that were bought by the user. For place recommendations, it would be harder to ensure that user visited the place, without making it mandatory to check in at the place to validate the visit.

For the item-based collaborative system, it would be beneficial to host the place and rating data, as part of the system. All the known places should be found from the system, even though there were no ratings given by the users yet. This would remove the issue, that the current solution had, that user could request recommendations in the locations that are not known by the system. For missing ratings, some fake ratings could be used, like taking the mean of all the ratings for places with same category.

Other feature that was missing from the item-collaborative system test case, was the user's ability to get recommendations for only currently nearby places. The longitude and latitude values were forgotten, when the place data was programmatically fetched from Places API. The proximity calculations would also be out of the scope of this research so they weren't the focus point.

If these limitations of item-based collaborative system, in the tourism context would be overcome, this approach is very flexible in various ways. For example, the place type categories could be used, e.g. compute ratings for places that are missing them and used as a base for correlations. At the same time, these categories would not be treated in that sense, that similarities for places with different categories could not be found. These facts would help for places, that have too ambiguous category. This was the reality that occurred in all the researched sources of place and point of interest related data.

7 Results and discussion

Finding feasible solution for the given context turned out to be extremely difficult. Choices for source of data were hard to find. Some of them were poorly documented or unreliable in terms of continuous accessibility.

OpenStreetMap had the option to download their entire database of location data, but they had very few data-attributes for places. Only the category attribute would have qualified as a feature for the machine learning model. Google Places API, on the other hand, had more choices for feature candidates, but there were lot of inconsistencies in the presence of these values, especially in smaller cities. With Google Places the system would also be reliable on third-party service, which is a downside in longer run.

Overall, it was hard to come up with the feature candidates, since the most of the location related data would not be meaningful in recommendation service. Longitude, latitude information was available in every data source. But when the goal is to present the user potentially interesting places, the past location-coordinates will not matter. Opening hours could be one feature, but people are generally more flexible to variable opening hours, during their holidays. Price information wasn't directly and consistently available in any of the services. The only potential features, that were left, were calculating the distances from starting point to visited place, known rating scores and place types.

Categorical data found out to be difficult one to deal with Scientific Python libraries, and after preprocessing those values to numerical representatives, it would be complex operation to find those numerical mappings from remote service, like using a smartphone to request classification for *n*-places, with given feature parameters. Maintaining all the train data locally, would take too much storage space from the smartphone and computing complex machine learning algorithms, would dry the smartphone batteries in no time. For these reasons, a remote machine-learning server, would be almost mandatory, for good user-experience.

Some more complex ensemble learning techniques, could solve the problem of overfitting and skewed classification results. That could reduce the need of initial data to produce meaningful predictions. Given the scope of this research, ensemble learning techniques and other potential improvements, there would be a need for more resources to achieve better results.

The research gave good insight in the world of machine learning and data science. This field of information technology, is very difficult to master in short time. Building the models and using different statistical methodologies to manipulate raw data, forced to evaluate the system requirements and architecture aspects from early on. The research could have been more successful one, if some of the models, that could work with smaller initial dataset, would have turned out to be more reliable. These models could have been used as basis for the proof-of-concept application, to taken into further development. Now the results support item-based collaborative filtering, which requires lot of data to become part of accurate recommendation system.

References

Adomavicius, G., & Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*.

Beylkin, G., Garcke, J., & Mohlenkamp, M. J. (2009). Multivariate Regression and Machine Learning with Sums of Separable Functions. *SIAM Journal on Scientific Computing*, *31*(December), 1840–1857. https://doi.org/10.1137/070710524

Brownlee, J. (2013). How to Evaluate Machine Learning Algorithms - Machine Learning Mastery. Retrieved December 11, 2016, from http://machinelearningmastery.com/how-to-evaluate-machine-learning-algorithms/

Brownlee, J. (2016a). Bagging and Random Forest Ensemble Algorithms for Machine Learning - Machine Learning Mastery. Retrieved December 11, 2016, from http://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-for-machine-learning/

Brownlee, J. (2016b). Evaluate the Performance of Machine Learning Algorithms in Python using Resampling - Machine Learning Mastery. Retrieved December 11, 2016, from http://machinelearningmastery.com/evaluate-performance-machine-learning-algorithms-python-using-resampling/

Chih-Wei Hsu, Chih-Chung Chang, and C.-J. L. (2008a). A Practical Guide to Support Vector Classification. *BJU International*, *101*(1), 1396–400. https://doi.org/10.1177/02632760022050997

Chih-Wei Hsu, Chih-Chung Chang, and C.-J. L. (2008b). A Practical Guide to Support Vector Classification. *BJU International*, *101*(1), 1396–400. Retrieved from http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

Chirita, P.-A., Nejdl, W., & Zamfir, C. (2005). Preventing shilling attacks in online recommender systems. *Proceedings of the Seventh ACM International Workshop on Web Information and Data Management WIDM 05*, *55*(2), 67. https://doi.org/10.1145/1097047.1097061

Davidian, M., & Louis, T. A. (2012). Why Statistics? *Science*, *336*(6077), 12–12. https://doi.org/10.1126/science.1218685

Enthought Inc. (2016). Welcome to Enthought Canopy — Canopy 1.7.4-final documentation. Retrieved December 11, 2016, from http://docs.enthought.com/canopy/

Ericson, G., Gyger, A., Whitlatch, K., Nevil, T., & Franks, L. (2016). Create a Machine Learning workspace | Microsoft Docs. Retrieved December 11, 2016, from https://docs.microsoft.com/en-us/azure/machine-learning/machine-learning-create-workspace

Expedia Media Solutions. (2013). *The Traveler' s Path to Purchase*. Retrieved from http://cdn2.hubspot.net/hub/149354/file-271132325-pdf/docs/Path_to_Purchase_Expedia_Media_Solutions_MillwardBrown.pdf?t=1458338275763

Gershman, A., Meisels, A., Lüke, K.-H., Rokach, L., Schclar, A., & Sturm, A. (2010). A Decision Tree Based Recommender System. *Iics*, 170–179. Retrieved from http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:A+Decision+Tree+Based+Recommender+System#0%5Cnhttp://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:A+Decision+Tree+Based+Recommender+System.%230

Goldberg, D., & Holland, J. (1988). Genetic Algorithms and Machine Learning. *Machine Learning*, *3*, 95–99. https://doi.org/10.1023/A:1022602019183

Google. (2016). Place Search | Google Places API Web Service | Google Developers. Retrieved December 11, 2016, from https://developers.google.com/places/web-service/search

Groleau, R. (2003). Imagining Other Dimensions. *Nova*. Retrieved from http://www.pbs.org/wgbh/nova/physics/imagining-other-dimensions.html

Ham, N., Dirin, A., & Laine, T. H. (2016). Machine learning and dynamic user interfaces in a context aware nurse application environment. *Journal of Ambient Intelligence and Humanized Computing*, 23. https://doi.org/10.1007/s12652-016-0384-1

Herrmann, P., Kundisch, D., Nicolau, V., & Zimmermann, S. (2014). Mobile Data Roaming Regulations. *Twenty Second European Conference on Information Systems*, 1–11.

IPK International. (2015). *ITB WORLD TRAVEL TRENDS REPORT 2015 / 2016*.

Retrieved from http://www.itb-berlin.de/media/itbk/itbk_dl_all/itbk_dl_all_itbkongress/itbk_dl_all_itbkongress_itbkongress365/itbk_dl_all_itbkongress_itbkongress365_itblibrary/itbk_dl_all_itbkongress_itbkongress365_itblibrary_studien/ITB_World_Travel_Trends_Report_2015_

Jolliffe, I. T. (2005). Principal component analysis. *Applied Optics*, *44*(May), 6486. https://doi.org/10.1007/SpringerReference_205537

Jupyter. (n.d.). Jupyter and the future of IPython — IPython. Retrieved December 11, 2016, from https://ipython.org/

Kuusisto, L. (2014). New York City Restaurants Multiply, Despite High-Profile Closures - WSJ. Retrieved March 20, 2016, from http://www.wsj.com/articles/new-york-city-restaurants-multiply-despite-high-profile-closures-1412816142

Lane, T. (2003). On the Origin and Destiny of Inductive Machine Learning. *Journal of Machine Learning Gossip*, *1*, 21–27. https://doi.org/10.1.1.59.8430

Michie, E. D., Spiegelhalter, D. J., & Taylor, C. C. (1994). Machine Learning , Neural and Statistical Classification. *Proceeding*. https://doi.org/10.2307/1269742

Mortazavi, S. (2015). Introducing Jupyter Notebooks in Azure ML Studio | Cortana Intelligence and Machine Learning Blog. Retrieved December 11, 2016, from https://blogs.technet.microsoft.com/machinelearning/2015/07/24/introducing-jupyter-notebooks-in-azure-ml-studio/

Netflix Prize. (2016). Retrieved December 1, 2016, from https://en.wikipedia.org/wiki/Netflix_Prize

OpenStreetMap. (2016). Planet OSM. Retrieved December 11, 2016, from http://planet.openstreetmap.org/

Peter Armitage,Geoffrey Berry, J. N. S. (2002). *Statistical Methods in Medical Research*. (Fiona Pattinson, Ed.) (4th ed.). Cornwall: Blackwell Science Ltd. Retrieved from https://books.google.fi/books?hl=fi&lr=&id=OevV49Dhn_YC&oi=fnd&pg=PR5&dq=Bayes%27+Theorem+Medical+field&ots=3JqhFs3mdn&sig=oCJYybwFVwMSa-YK7UtfBgkRjxo

Precision and recall. (2016). Retrieved December 1, 2016, from https://en.wikipedia.org/wiki/Precision_and_recall

Prokhorov, A. V. (2012). Regression analysis. Retrieved from https://www.encyclopediaofmath.org/index.php/Regression_analysis

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. *Proceedings of the 10th …*, *1*, 285–295. https://doi.org/10.1145/371920.372071

Wiktionary. (2016). *Http://en.wiktionary.org/wiki/latent*. Retrieved from https://en.wiktionary.org/wiki/latent

Your Europe. (2016). Mobile roaming costs - Your Europe. Retrieved March 20, 2016, from http://europa.eu/youreurope/citizens/travel/money-charges/mobile-roaming-costs/index_en.htm