Scientific report of assignment 2

Authors:
K. Hu (2586224) , J.D. Janssen (2586789) and J.F. Uleman (2509032)
October 10, 2018

1 Business understanding

With the coming of the Internet, booking a hotel can easily be done using online websites such as booking.com or expedia.com. Due to the wide variety of websites it is important for a website to match users to the right hotels, so that the user won't switch to another online travel agency. Hence the importance of ranking hotels in any given search to specific user preferences. Ranking hotels by such preferences may be done by using many different properties. For example, the country the user is from or the average star-rating of hotels that a user has booked in the past.
A system that predicts the rating or preference of a certain item is called a recommender system. There are two types of recommender systems: collaborative filtering systems and content-based filtering systems. A collaborative filtering system only utilizes the historical data to predict a specific individual, while a content-based filtering system uses the characteristics of an item (or sets of items) to predict a similar item (or items) [1]. For example, Facebook, MySpace, LinkedIn, and other social networks use collaborative filtering to recommend new friends, groups, and other social connections. On the other hand, the music website Pandora recommends new music according to the type of music a user has played. This is an example of content-based filtering. Hybrid techniques combine both of these methods. In this report, we will use a model-based approach to predict which hotel a user is most likely to book. This prediction is based on the historical data from Expedia, but also on certain characteristics of the hotel and the user.
The data used for this assignment originated from the 2013 Kaggle competition, where the best hotel ranks for personal searches on the Expedia website had to be predicted. The same had to be done in this assignment, with a few alterations to the data set. When opening expedia.com and searching for a hotel in a certain destination, a hotel can be filtered on a lot of different options. These options are for example: star-rating, price per night, area, type of accommodation and meals included. Furthermore, Expedia knows from the users their IP addresses in which country and city the user is searching and with the help of cookies Expedia can build up a history database for every individual user. This results in an enormous database. In this research, the enormous amount of data had to be dealt with in efficient ways and used to predict the hotel most likely to be booked given a certain search.
The winner of the Kaggle competition showed that the most important features of the Expedia database for prediction were: the position of the hotel on the search result page, the price and the location desirability of the hotel [2]. In our research, the hotel position on the Expedia search

results page is only available in the training set. The prediction of the test set gives as a result the rank of hotels that are most likely to be booked as a result of other features. In the approach of the winning team, a distinction was made between different types of features. Besides the original features, also some composite features were made. For example: price order, which is the order of the price within the same search id. Some of the categorical features were converted to numerical data. Because some types of predictive modeling techniques work better when working with continuous predictors, while other types of models are more designed for categorical variables, converting the data to a numerical value can make the data more appropriate for certain models. The model used by the winning team was an Ensemble of Gradient Boosting Machines (GBM). Two types of models for the ensemble were used: with and without features converted to numerical data. Gradient boosting is a technique that produces a prediction model in the form of an ensemble of weak decision trees. The model is built in stages, and in each stage a weak learner to compensate the shortcomings of existing weak learners is added [3].

2 Data Exploration

2.1 Data understanding

This assignment requires the analysis of two sets of data: "training set VU DM 2014.csv" and "test set VU DM 2014.csv", both containing information about a search query for a hotel, the hotel properties that resulted, and for the training set specifically: whether the user clicked on the hotel and booked it. The main task is to use the training set to train an algorithm to predict which hotel a given search_id is most likely to book in the test set, and outputs the ranking of the best hotels, i.e. prop_id, for each srch_id.

First, **basic characteristics** the training and test sets were explored. The training test sets consist out of 4958347 and 4959183 instances, 199795 and 199549 srch_id and 54 and 50 attributes, respectively. The data sets have 129113 and 129438 prop_id values. It can be noticed that the test data overall contains more unique prop_id values than the training data. Perhaps the test data is more recent, this would explain an increased amount of hotels in the Expedia assortment. There are altogether 221879 searched hotels in the training data are clicked, of which 138390 are booked.

Second, **the missing data percentage** was explored for each attribute in training set. Figure 1 shows that of the 54 attributes, 28 have missing data in more than half of the instances. This indicated that stringent feature selection and/or imputing the missing data would be necessary.

Third, the **correlation** between prop_id and other attributes was explored. Figure 2 indicates that non of the features correlate well with prop_id, therefore linear models like Logistic Regression will not perform well on this data. As a result, some non-linear models like LambdaMart, RandomForest, Gradient Boosting, and SVM, were used in the process of training the data.

Finally, the **distribution bias** of some of the attributes were explored. The 5 most common prop_ids were related to booking and clicking. The two left graphs in figure 3 indicates that of all the prop_ids that have been booked/clicked, some (the most left ones on the graph) show an extremely high frequency. It can be observed that the most popular hotels have around 250 bookings while most of the other hotels only have 1 booking. The right graph explores the hotel

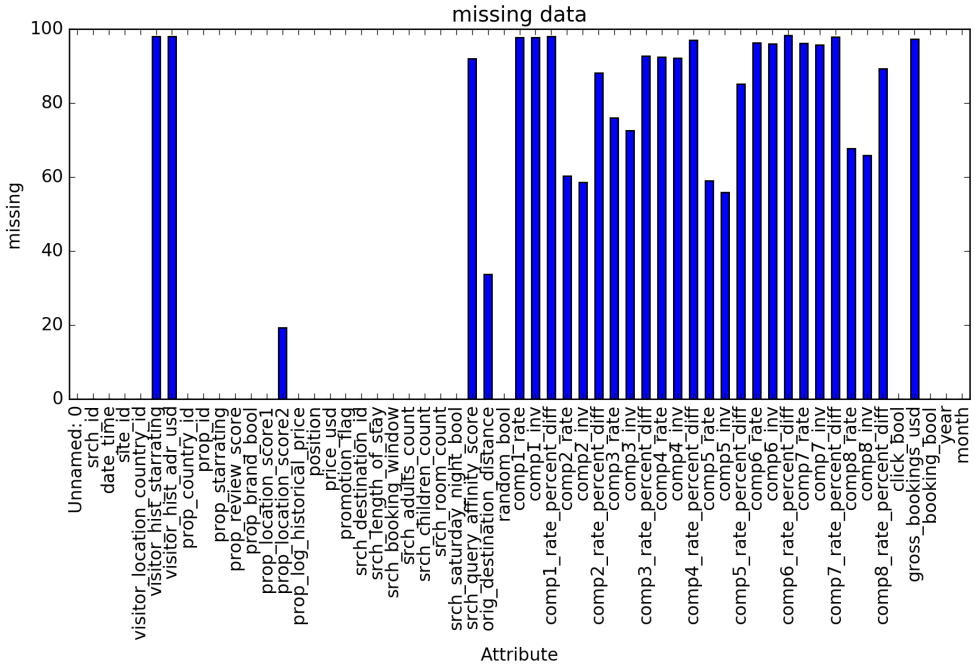

Figure 1: Exploring the missing data. On the x-axis are all the different attributes and on the y-axis is the percentage of missing data.

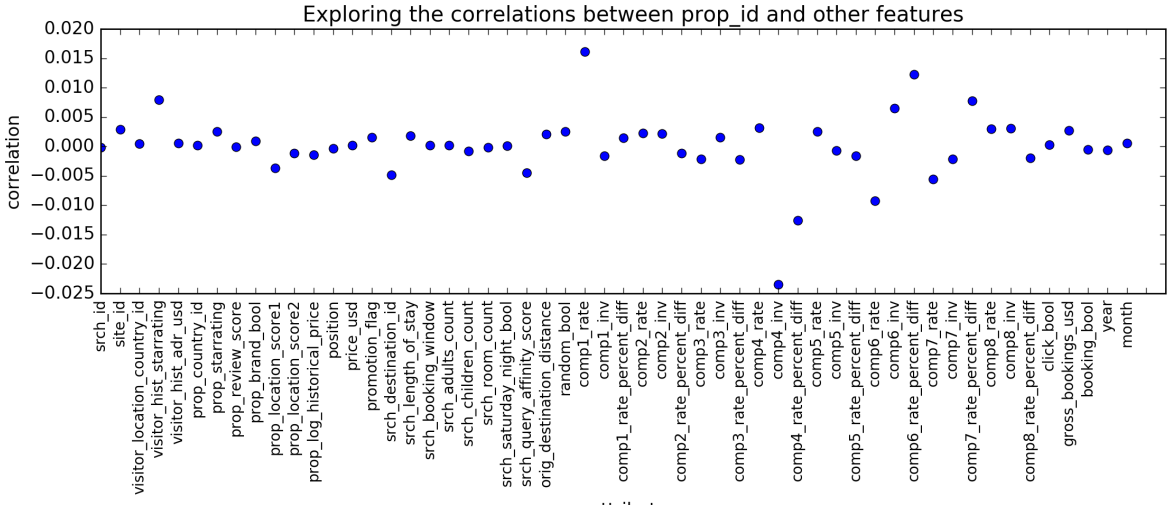

Figure 2: Exploring the correlations between prop_id and other attributes. On the x-axis are all the different attributes and on the y-axis is the correlation with each attribute to prop_id.

position bias, and shows that some positions are clicked or booked more frequently. This graph shows hotels ranking high in the search engine are more likely to be clicked and booked.

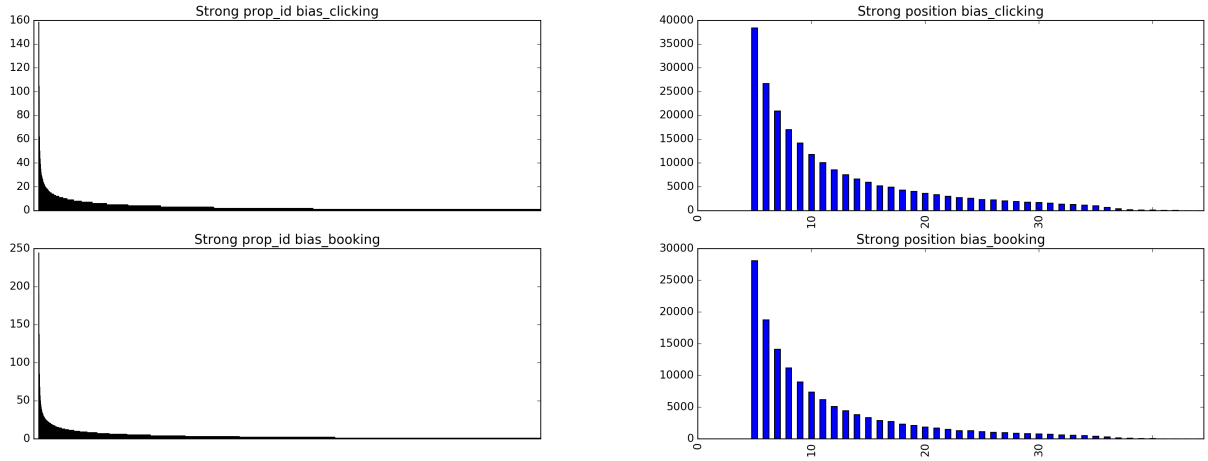

Figure 3: Left graph: Exploring the most common 5 prop_id's, on the x-axis the prop_id's of the most popular hotels and on the y-axis the number of clicks or bookings. Right graph: Exploring the hotel position bias, on the x-axis the ranking of the hotels in the search engine and on the y-axis the number of clicks or bookings.

2.2 Feature Selection

The first step in the feature selection process was to select the 'core features'. In order to compare them, features with lots of missing data (figure 1) were temporarily dropped from the data set. Next, to deal with features that had lower percentages of missing data, the missing data would either be imputed with -1 or the median, or a composite feature of two similar features would be created. For example, a composite feature, 'prop_loc_desir', which contains the mean of location score 1 and the non-NaN values of location score 2, was created. Other composite features were generated as well, such as 'person_count', which aggregated the adult and children feature into one, and a 'prop_rating' score, which is the aggregate of the star- and review scores, and three summed features over the 3 classes of competitor features, i.e. the non-NaN values in the '_rate','_rate_percentage_diff' and '_inv' features. Instances that had missing data in all of the columns in the respective classes had their value imputed with -1. Furthermore, the 'date' feature was turned into the more informative features 'month' and 'year'. And after this, feature scaling was applied to the scoring features in order to make them scale from 0-10.
All features were correlated to an aggregate feature ('clickbool') that was generated for this comparison only. The aggregate feature summed the click_bool and booking_bool features and multiplied the booking_bool by 5. Then all features were correlated to the 'clickbool' feature using Python's corr function with the 'spearman' method. And after building a model using all features, the Feature Importances' (FI) option of the Random Forest classifier (sklearn, Python) was used. Both the correlations and FI allowed for the comparison of features to determine their respective share in the prediction.
From here, a smaller group of features was selected, including some composite features, and they were put into the Random Forest (sklearn, Python) model, where the FI option was again used for

further sophistication. The results of this FI are plotted in figure 4. From the graph it becomes evident that the composite features are valuable since 'prop_loc_desir' has the highest FI score and the 'prop_score' aggregate has a greater FI score than its constituent 'prop_review_score'. Furthermore, the IDs themselves are generally more predictive than most of the other features. This was informative for the final feature selection.

In the end, the original features that were used in the model were:
- site_id
- srch_id
- visitor_location_country_id
- prop_country_id
- prop_id
- prop_brand_bool
- prop_log_historical_price
- price_usd
- promotion_flag
- srch_destination_id
- srch_length_of_stay
- srch_booking_window
- srch_room_count
- srch_saturday_night_bool
- random_bool
- month

These composite features were used:
- srch_person_count
- prop_loc_desir (hotel locationscore aggregate)
- prop_score (hotel score aggregate)

And these competitor feature aggregates:
- comp_price
- comp_perc_diff
- comp_avail

Two models were build using either all features (excluding the date and strictly training features) or the features mentioned above. These sets of features will from here onwards be referred to as feature set 1 and 2 respectively.

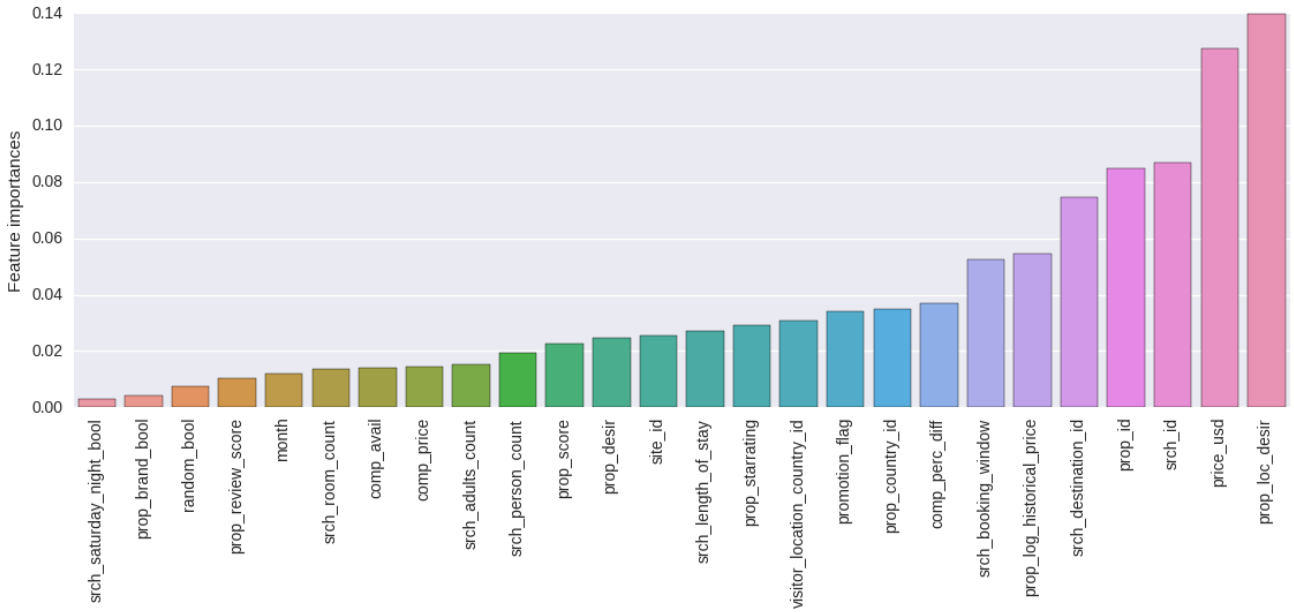

Figure 4: The feature importances of our selected set of features.

3 Modeling and Evaluation

3.1 Methods

3.1.1 Machine learning techniques: LambdaMart

One popular and particularly effective learning-to-rank method is the LambdaMart method, which is based on LambdaRank and RankNet. Its effectiveness becomes evident from the fact that "an ensemble of LambdaMART rankers won Track 1 of the 2010 Yahoo! Learning to Rank Challenge."[4] Furthermore, several of the winners of the 2013 Kaggle Competition reported having used LambdaMart with more than satisfactory results.
In essence, LambdaMart is a combination of LambdaRank and Mart, the latter of which is a boosted tree model in which the output is a linear combination of regression trees, and LambdaRank is an algorithm that uses gradients of a costfunction rather than the costs themselves. LambdaMart is less concerned with the actual scores that it assigns to items and more with the ranking that it provides. [4] This makes it an ideal option for a learning-to-rank system. Furthermore, LambdaRank is able learn NDCG directly (see: Validation methods), which likely also adds to its success. For this assignment, it was attempted to implement a LambdaMart model but due to a lack of time, it was ultimately decided to abandon it for the supervised Machine Learning algorithms we were more familiar with.

3.1.2 Machine learning techniques: Random Forest

Decision trees tend to overfit to the training data quite quickly when the tree is too "deep", because then the tree is based on rules which are actually just noise of the training data. This is where a random forest algorithm becomes handy. In this algorithm, many (tens - hundreds) decision trees are build. All trees get a randomly chosen part of the training data and each split point in every tree is performed on a random subset of the potential columns to split on. In the end of the algorithm, the predictions of all trees are averaged and this gives a strong overall prediction with less overfitting [5]. The random forest algorithm can be implemented using the $RandomForestClassifier$ function from the scikit-learn python library.

3.1.3 Machine learning techniques: Gradient Boosting Machine

The idea of gradient boosting is to repeatedly perform the following procedure. First, learn a simple regression predictor (so starting with an one-layer decision tree) of the data. Then, compute the error residual, this is the error per data point in the predictions. Finally, learn a new model that tries to predict this error residual, this new function is a bit more complex than the first one-layer decision tree and will have a smaller residual error. At each boosting iteration, the algorithm applies weights $w_1, w_2, ..., w_N$ to each of the training samples, with the poorly predicted instances' weights increasing. As iterations proceed, examples that are poorly predicted at early stages receive ever-increasing influence, thus the accuracy of the model to the training data gets slightly better. The overall prediction is given by a weighted sum of the collection of predictors [6]. The gradient boosting algorithm can be implemented using the $GradientBoostingClassifier$ function from the scikit-learn python library.

3.1.4 Machine learning techniques: SVM

The goal of a Support Vector Machine (SVM) is to train a model that assigns new items into a particularly category. These categories are the result of a linear partition of the feature space. The separate categories are divided by a gap that is as wide as possible. This separator is the hyperplane that has the largest distance to the nearest training-data point of any category. The nearest training-data points are called the support vectors. When the hyperplane has a larger distance to the nearest support vectors, this gives in general a lower generalization error. Once the categories are made, new items getting assigned to a certain category based on specific characteristics. The SVM algorithm can be implemented using the SVM function from the scikit-learn python library.

3.1.5 Validation method

Normalized Discounted Cumulative Gain. A good evaluation metric of the ranking quality for the learned models is the Normalized Discounted Cumulative Gain (nDCG). It measures the performance of a recommendation system based on the predicted score of the recommended entities. It varies from 0.0 to 1.0, with 1.0 representing the ideal ranking of the entities[7].

The Discounted Cumulative Gain (DCG) can be calculated as follows:

$$DCG_k = \sum_{i=1}^{k} \frac{2^{reli_i} - 1}{log_2(i+1)} \quad (1)$$

$$reli_i = \begin{cases} 5 : \text{for booked hotels} \\ 1 : \text{for clicked hotels} \\ 0 : \text{for all the rest} \end{cases}$$

With k: the maximum number of entities that can be recommended
After this calculation, the DCG can be normalized by dividing it by the $IDCG_k$, which is the ideal DCG_k for the given set of rankings.

Holdout validation. Since there is a sufficient amount of data: with 199795 search IDs and 4958347 instances in the "training set VU DM 2014.csv" file we won't use cross-validation, which swaps roles of several folds of data in one training set, so that we could make full use of all the instances. A holdout validation method is used, where a certain amount of data is reserved for training and the rest for validation. First, 100 srch_id values were randomly selected, and the corresponding instances in "training set VU DM 2014.csv" file are selected, forming the training set. And this data set consists of 2479 instances. Second, 50 srch_id values in training set are randomly selected for training, with the remaining 50 for validation.
Specifically, each model is first trained on the training set to obtain a set of rules to classify the instances in test set to two groups: clicked and not clicked. Next, each model is trained on the training set to obtain another set of rules to classify the instances in test set to two groups: booked and not booked. And then another attribute "score" is created accordingly, with 5 representing booked is predicted, 1 representing clicked is predicted but not booked, and 0 for the rest. So for the validation set, the DCG_k is calculated based on the "score" attribute, and $IDCG_k$ is calculated after ranking according to the "click_bool" and 'booking_bool' attributes.

3.2 Results

Three machine learning schemes, i.e. RandomForest,Gradient Boosting Machine, and SVM were trained with the training set using two sets of features, i.e. Feature set 1 and 2, mentioned above.

Everything was implemented in Python using the sklearn library with the parameters are set as follows:
- RandomForest: n_estimators=10, min_weight_fraction_leaf=0.01, others default;
- Gradient Boosting Machine:n_estimators=100, learning_rate=1.0,max_depth=1, random_state=0, others default;
- SVM: kernel='rbf', others default;
- SVM: kernel='sigmoid', others default;

Figure 5 shows the nDCG for each machine learning algorithm, with each of the feature sets. It can be observed that feature set 2 produces slightly more robust models than feature set 1 for the three schemes. And of all the models, the Gradient Boosting Machine outperforms the others.

This indicates that an ensemble of weaker models can produce more robust predictions for this task.

Finally, according to the outcomes of the nDCG evaluation metric, the Gradient Boosting Machine scheme was used to predict the "test set VU DM 2014.csv" file. Similarly as in the validation test, the instances were classified by whether they were clicked and booked, and accordingly, their "score" was calculated. The instances within the same srch_id were then ranked according to the probability that a certain prop_id will booked. A csv file containing only srch_id and prop_id was then generated.

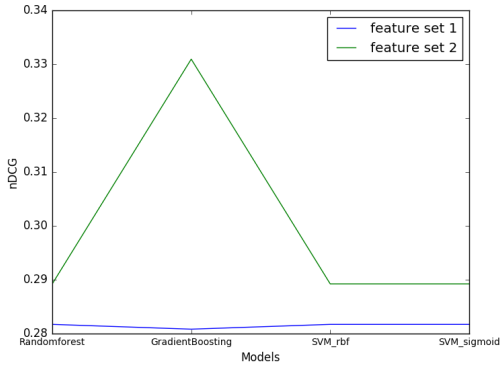

Figure 5: Comparison of training models with two set of feature sets

4 Conclusion

For the model training part, ensemble methods have been used to improve robustness over a single estimator in our work. And it actually showed robust outcomes. There are two families of ensemble methods: averaging methods, in which the driving principle is to build several estimators independently and then to average their predictions, and boosting methods, in which base estimators are built in a stage-wise fashion and tries to reduce the bias of the combined estimator by allowing optimization of a differentiable loss function.[8] For the former, Random Forests was deployed, and for the latter, Gradient Boosting Method was deployed.

For the model selection part, there are various performance measures, of which we have considered two evaluation metrics, i.e. nDCG and Mean Average Precision [9]. The latter is used on the occasions when there are m missing outbound edges from a user in a social graph, and it can predict up to n other nodes that the user is likely to follow. And the former, nDcg, is used in our task, which evaluate the gain of ranking based on each instances position in the result list and the most ideal ranking positions.

4.1 Skills and knowledge gained from this assignment

We have gained various skills and knowledge from working on this assignment, including how to work with machine learning algorithms (Gradient Boosing, Random Forest, SVM) and tricks used in data processing. The programming was mostly done in Python with the help of the scikit-library. For all of us it was the first time we used this library (for the course Data Mining techniques, we also used this library for the first assignment), so we gained some familiarity and experience with it. Furthermore, we learned how to work with very large data sets. For example, the data set was to large to open at once at one of the computers. So, some modifications had to be made before the data set could even be loaded into Python on our computers.
Another thing we learned is how the process of feature selection is not a one time thing. We used creativity and logic at first to make a first selection of features and composites. However, we had to continuously return to the process of feature selection after the creation of yet another model to keep experimenting. Correlating the features to the booking/clicking and using the 'Feature Importances' option of the Random Forest classifier proved very insightful. It can be easy to get overwhelmed with features but we now feel more confident that we could handle them. We only regret that our feature selection did not pay off more in terms of a greater increase in nDCG.

References

[1] Prem Melville and Vikas Sindhwani. Recommender systems. In *Encyclopedia of machine learning*, pages 829–838. Springer, 2011.

[2] Owen Zhang, winner kaggle expedia competition 2013. Presentation: `https://www.dropbox.com/sh/5kedakjizgrog0y/AABhKazV866m3uDGa2QYoFsoa/ICDM_2013/3_owen.pdf?dl=0`.

[3] Cheng Li. A gentle introduction to gradient boosting. Presentation: `https://www.google.nl/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&ved=0ahUKEwjjiITTsvzMAhXlDMAKHT2MCQAQFggpMAE&url=http%3A%2F%2Fwww.ccs.neu.edu%2Fhome%2Fvip%2Fteach%2FMLcourse%2F4_boosting%2Fslides%2Fgradient_boosting.pdf&usg=AFQjCNG1guO3QiX0FzhPAs-rehxZma4MXA&cad=rja`.

[4] Christopher JC Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11:23–581, 2010.

[5] Tutorial about improving your prediction on a dataset. Website: `https://www.dataquest.io/mission/75/improving-your-submission/`.

[6] Alex Ihler. Introduction to machine learning and data mining. Lecture slides: `http://sli.ics.uci.edu/Classes/2015W-273a`.

[7] Kaggle: normalized discounted cumulative gain, howpublished = `https://www.kaggle.com/wiki/normalizeddiscountedcumulativegain`, note = Accessed: 2016-05-28.

[8] Scikit-learn: ensemble method. `http://scikit-learn.org/stable/modules/ensemble.html`. Accessed: 2016-05-24.

[9] Kaggle: evaluation. `https://www.kaggle.com/wiki/MeanAveragePrecision`. Accessed: 2016-05-26.

Process report of assignment 2

As with our first assignment, our overall teamwork was very good. We had enlightening discussions, pushed one another to come up with new ideas and each of us contributed equally to the end product.
An elaborate overview of our weekly Friday (12:00-16:00) meetings and the work that each of the group-members performed can be found below.
We were all present during the lectures and usually spend some time discussing the assignment in the breaks and after the lectures. At the end of the second assignment's introductory lecture we decided to all start do some reading on learning-to-rank tasks individually.

29-04-2016 - We brainstormed on the project together between 12:00 and 16:00. We loaded the data, discussed the relevancy of certain features, and divided roles for the Exploratory Data Analysis. Jeroen is going to keep track of the hour-by-hour work schedule.

This week we all worked separately on a conceptual understanding of the assignment, feature selection and what kind of model we'd like to implement.

06-05-2016 All of us came up with ideas on which type of model we'd like to implement. And each of us came up with unique ideas on feature selection and model implementation. Jeroen wanted to try implementing LambdaMart, Kaixin wanted to implement a SVM and gradient boosting, and Jiske was considering a simple algorithm with only a few features and seeing how good of a prediction we could make using that.

This week we all worked on an implementation of our respective models.

13-05-2016 We discussed the progress of our models. Jiske has been working on a random forest model with only a few features and nearly has it implemented. Kaixin was still working on her model and has already made some nice graphs for data exploration. Jeroen was still working on his implementation of LambdaMart and has been working on feature selection and the creation of composite features.

20-05-2016 Due to the upcoming deadline, it was decided to drop the LambdaMart model and start working with the random forest model that by then had already been implemented. We experimented with our composite features and started working with the Feature Importances option.
Kaixin decided to keep working on the SVM and the Gradient Boosting Machine so that we could ultimately ensemble the models.

During the week we all kept working individually. We decided that Jeroen would focus on the feature selection process, while Kaixin would finish her model implementation and Jiske would stick to working with the random forest. We all started writing the corresponding parts of the report and kept in close contact through live chats.

27-05-2016 We all did some more writing and discussed how we would feedback, edit and hand in the report. We then all worked on the report the entire weekend. We also noticed a typo in one of the feature names in the assignment manual that we had copied and had to generate our prediction file again.

Regarding the code, we all wrote equally work-intensive parts (as mentioned above) and exchanged everything through GitHub so that the others could work with it as well.
In the final report, Jiske was responsible for the Business Understanding and Methods parts. Kaixin was responsible for the Data Exploration and Results parts. Jeroen was responsible for the Feature Selection part and process report. Furthermore, we all wrote parts of the conclusion and the Machine Learning techniques part. We all feedbacked and edited the report and finished the report on Sunday the 29th of May.