Beerlytics: An Interactive Beer Recommender System Based on Community Reviews

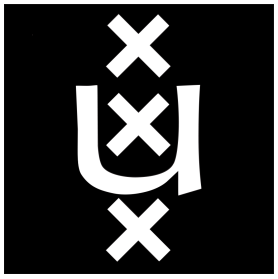
SUBMITTED IN PARTIAL FULFILLMENT FOR THE DEGREE OF
MASTER OF SCIENCE

SASCHA VAN ESSEN
10597808

MASTER INFORMATION STUDIES
HUMAN CENTERED MULTIMEDIA

FACULTY OF SCIENCE
UNIVERSITY OF AMSTERDAM

July 22, 2018



1st Supervisor
Dr. Stevan Rudinac
University of Amsterdam

2nd Supervisor
Dr. Nanne van Noord
University of Amsterdam

Beerlytics: An Interactive Beer Recommender System Based on Community Reviews

Sascha van Essen
University of Amsterdam
Amsterdam, The Netherlands
saschavanessen@gmail.com

ABSTRACT

This thesis presents *Beerlytics*, an interactive beer recommender system based on word embeddings of free-text user reviews. Using reviews gathered from the *Ratebeer* community platform, word and document embeddings are trained with *word2vec* and *StarSpace*. These embeddings are used as input for two different *SVMs*. In the *Beerlytics* system, users are able to indicate the beers they like and dislike, which are then used to train the *SVMs*. These classifiers recommend a list of 10 beers to the user. The user can change their preferences multiple times, iteratively building their profile and exploring the recommendations. The *SVMs* adjust accordingly, providing an updated list of recommendations each time. The different embedding models and classifiers are tested with artificial actors. The results show that the classifiers trained on the embeddings outperform the popularity baseline when making a list of recommendations from the beers an artificial actor consumed. When making a list of recommendations from all the beers in the system, they do not manage to surpass the popularity baseline, due to the sparsity of the data.

KEYWORDS

Recommender system, word embeddings, interactive learning, product reviews, natural language processing

1 INTRODUCTION

"You might also like ..." is a phrase that we've become used to seeing on our screens, as different online services recommend commodities or content to their users. Not only do e-commerce websites that try to sell products make use of these recommender systems, but applications have been developed with the sole purpose of helping users explore and discover music, news articles, or other items.

Presently, the field of recommender systems is trying to improve their predictions to users with methods that go beyond the user-item matrix [19]. User information is gathered from social networks as well as user-generated information such as geotags and comments. This research aims to determine whether word embeddings learned from consumer reviews on a community platform provide a good foundation for a recommender system.

The case study for this system is the community platform *RateBeer*, where users can leave reviews of beers. It currently has over 9 million reviews on over half a million beers.[1] With the textual reviews from *RateBeer*, the recommender system *Beerlytics* is made. This interactive system recommends beers to a user based on the beers they previously liked and disliked. A representation is made for each beer by training word embeddings on the *RateBeer* reviews. This representation is used to train a classifier on a user's preference and recommend new beers.

This thesis outlines the research and development of the *Beerlytics* recommender system. First, the related work is discussed (section 2), and after that the method for designing and testing the system is described (section 4). Then, the result section recites outcome of the tests (section 5), which is explained and interpreted in the discussion section (section 6). Afterwards, this study is concluded in section 7. Finally, some future research is proposed to further advance the *Beerlytics* recommender system (section 8).

[1] http://www.*RateBeer*.com/Stats.asp

2 RELATED WORK

This section gives an overview of the current state of research in the field that is relevant for the *Beerlytics* system. First, the different approaches to recommender systems are outlined. Then, two different methods for processing natural language texts are discussed: topic modelling and word embedding. Finally, some background on interactive learning is described.

2.1 Recommender Systems

Since the first algorithms that generated recommendations in the mid-1990s, various papers have summarized the history and current state of recommender systems, including publications by Gediminas Adomavicius and Alexander Tuzhilin [2], Jonathan Herlocker et al. [7], Badrul Sarwar et al. [18], and Yue Shi et al. [19]. Fundamentally there are two approaches to filtering the relevant documents for a user: collaborative filtering, and content-based filtering. A brief summary of both methods, as well as the combination thereof, is provided below.

2.1.1 *Collaborative Filtering.*

This approach compares user evaluations to recommend new items, assuming that users who have agreed on the rating of items in the past, will likely do so as well in the future. Thus, it predicts an individual user rating either based on ratings for similar products by that same user, or based on ratings for that same product by similar users. This user-item matrix can be used to predict a new item to the user with the memory-based approach or the model-based approach. The memory-based approach typically computes who the most similar users are to a given individual user, by using a similarity metric on the rating vectors, for example cosine similarity. The model-based approach uses the user-item matrix as input to train the function parameters of a model, which can then generate recommendations for individual users [18, 19].

2.1.2 *Content-based Filtering.*

This approach is based on a comparison of the content of the items, and possibly a user profile. If these items are products

to be recommended to consumers, the content of these items initially consisted mostly of tags (for example genres or colours), or descriptions provided by a manufacturer or reseller [2]. However, more recent research has been looking at additional information, most notably for this thesis: user-contributed information such as free-form text reviews and comments [19]. More on this research and the different methods used is discussed in section 2.1.4.

2.1.3 Hybrid Method.

The two techniques mentioned above can also be merged in a hybrid recommender system. There are different ways to do this, for example combining the scores, or choosing the most confident recommender [2]. Especially interesting is to decide the components and distribution of the hybrid through user interaction. The system iterates and adapts to user feedback, thus resulting in a personalized recommender system to fit each individual user. These hybrids have been tested to be more accurate than any of the systems with a singular approach [2].

2.1.4 Recommendation Based on Free-form Text.

Whether it being a collaborative filtering, content-based filtering, or hybrid system, various research has found that using users' reviews can improve recommendations.

For example, Silvana Aciar et al. [1] created *Informed Recommender*, a system that generates product recommendations from online free-form text consumer opinions by mapping reviews to an ontology using text-mining techniques. Taking another approach, Niklas Jakob et al. [9] extracted item aspects from movies using different clustering techniques. For fully automatic clustering they used *Latent Dirichlet Allocation* (*LDA*). Subsequently, they extract users' opinions on these found aspects from the reviews, which serve as additional features to improve the standard collaborative filtering recommendation. Similarly, Yashar Moshfeghi et al. [16] also used *LDA* to predict the rating of a movie given a user. Asher Levi et al. [12] also use unsupervised clustering to build a vocabulary of item features and combine this with semantic analysis to understand the opinion on the different features. Besides that they also profile the intent and nationality of the user to improve the discovery of similar users for recommendation.

2.2 Natural Language Processing

The problem with free-form text is that it is understandable for humans, but not for machines. The field of *Natural Language Processing* (*NLP*) focuses on this task of converting human readable texts into a representation that is manageable for computers. One aspect of this is, is making a machine understand which words are semantically related, especially in an unsupervised way [5]. One of the most frequently used methods for this is using topic models. Another, more recent method, is using word embeddings. Both methods are described below, followed by an explanation of preprocessing small documents to improve the performance of models based on term frequency.

2.2.1 Topic Modelling.

Topic models are unsupervised algorithms that analyze documents to find the latent themes that are incorporated in the texts. One of the most straightforward topic modeling approaches is the probabilistic model *latent Dirichlet allocation* (*LDA*). This model was presented as a new method for discovering topics in documents in 2003 [4]. A previously defined number N topics is distilled based on the words of the entire corpus. Each topic is represented by a list of words (appearing in the corpus) that belong to that topic. For each document these N topics are given a probability score. This number represents the model's certainty that this document's content is about that topic [3].

2.2.2 Word Embedding.

Another way to capture syntactic meaning, is through word embeddings. Ronan Collobert and Jason Weston showed that training a deep neural network that takes into account the context provides a good representation of words in an unsupervised way [5]. The context here simply means the words before and after in a sentence, thus the semantic relation between words is based on the concept that words with similar meanings are often used together. By creating a feature space from the text input, every word is represented by their position in that space in the form of a vector. Tomas Mikolov et al. continued this approach which resulted in the *word2vec* toolkit in 2013 [14]. They then applied the same method to a sequence of words, to provide a representation for sentences, paragraphs, and documents, which resulted in the *doc2vec* toolkit [15].

Recently, a new neural-embedding model has shown promising results on solving a variety of problems, based on the same principle of syntactic meaning through proximity of words: *StarSpace* [22]. *StarSpace* embeds entities of different modalities into one vectorial embedding space, and compares them against one another. It has already been tested on labeling tasks, ranking tasks, recommendation, embedding of multi-relational graphs, and learning word, sentence or document embeddings, when the model was first presented in 2017. Specifically on learning text embeddings it was evaluated using SentEval[2]. The tests showed that *StarSpace* outperformed *word2vec* on some of the tasks, though on others *word2vec* achieved better results.

2.2.3 (Tweet) Pooling.

Extracting keywords based on frequency is difficult if the documents in the corpus are short texts, like product reviews. Mehrotra et al. discuss a method for preprocessing the texts to improve the topic coherence in microblog content like Twitter [13]. By pooling tweets together by their hashtag they create larger texts to use for the *LDA* topic modelling. Trying different kinds of topic schemes, they concluded that preprocessing the tweets by using the hashtag pooling scheme significantly improved the *LDA* topic modelling [13].

2.3 Interactive Learning

There are two types of recommendations that are difficult for recommender systems to make. The "cold start" problem refers to new users (or items) that don't have a lot of ratings yet, and therefore there is not a lot of available data to base the recommendation on [7]. Another issue is the "long tail" recommendation. This concerns items that are not very popular and therefore don't have a lot of ratings [19]. Systems that recommend

[2] https://github.com/facebookresearch/SentEval

the most popular items have generally been a difficult baseline to surpass [6, 10, 20, 21]. However, they do not handle the cold start and long tail issues particularly well. Both of these can be solved with interactive learning [25].

An interactive learning approach very successful in content-based search is relevance feedback. The user is asked to label the items returned in the initial query result as "relevant" or "irrelevant". This explicit feedback is then used to refine the results to better fit the user's preference [8]. When using this technique in recommender systems, not only is the system able to give more personalized recommendations due to more input samples, it also allows the user to explore the recommendations by iteratively changing their preference [24, 25].

When formulating recommendations as the prediction of "relevant" and "irrelevant" items, it can be regarded as a classification problem. One type of classification algorithm that has shown to be highly applicable to interactive learning, is a *Support Vector Machine* (*SVM*) [8, 11]. In a binary classification problem *SVMs* learn a function that separates the relevant input samples from the irrelevant ones in the feature space. New items are then classified by their position in the feature space relative to this separating hyperplane. In an interactive learning setting, the *SVM* can re-train on every new user input, adjusting the hyperplane on each iteration of feedback.

3 PROBLEM STATEMENT

The goal of the *Beerlytics* system is to recommend beers a user might be interested in. Assuming the user will enjoy beers with similar qualities to the ones liked before, a user's preference can be "learned" by looking at the beers they previously liked and disliked. Thus, it is necessary that the user inputs some of their likes and dislikes into the system. This can be done in multiple steps, so the user can explore the suggestions and the system can provide more personalized recommendations. Inputting the likes and dislikes should not have to be an elaborate process, therefore the user has to be able to convey their opinion through just a few clicks, instead of providing ratings or even reviews to express their preference. Based on this input the system needs to be able to make a list of recommendations. Additionally, the system should learn every time the user inputs new beers, thus interactively providing new recommendations. In order to give a recommendation based on the qualities of the beers, the system requires some kind of representation of the beers and their features.

Focusing on users' opinion of the beer, the representation of these qualities, or features, should be based on the free-form text of the reviews added by the community on the *RateBeer* platform, instead of on manufacturers' descriptions. As can be seen in the related works described above, many recommender systems extract these features from free-from text by using *LDA*. This research takes a more recent approach of using word embeddings, specifically the *StarSpace* model. The embeddings created by *StarSpace* are compared to the more established *word2vec* embeddings. The method used for creating the embeddings and the rest of the system's backend is described in the next section.

4 METHOD

This section outlines the approach taken to the development of the *Beerlytics* recommender system.

4.1 Data Collection

The data is gathered by scraping the *RateBeer* website. Most importantly for the recommendations, this includes all the reviews and ratings given by users. Besides that, there is more information on the beers, a collection of locations (cafes, supermarkets, etcetera), and which beers can be bought there. This information is stored in a private *MySQL*[3] database. An entity-relationship diagram to further clarify the data can be found in Figure 1.

4.2 Word and Document Embedding

The documents are embedded using *word2vec* and *StarSpace*, to be compare a new method of embedding to an established method. The approach of generating these embeddings is detailed below. However, first the data is preprocessed according to the following procedure.

4.2.1 *Preprocessing.*

The reviews on *RateBeer* are written in different languages. To make sure this does not influence the performance of the word embedding model, only the reviews written in English are included as input. Thus to begin with, the language of the reviews is detected with *Google Translate*[4], by using the *Googletrans API*[5] and added to the reviews table in the database. Then the English reviews are pooled together per beer, resulting in one document for each beer. These documents are tokenized, stemmed, and stop words are removed with the *Natural Language Toolkit*[6]. This results in 2.716.159 reviews over 91.405 beers which are used as input for the word embeddings. The beers are saved in two batches (one of 50.000 and one of 41.405) to bisect the file size of the input and reduce the memory space needed.

4.2.2 *Word2vec.*

After preprocessing, the documents representing a beer are used to train the *word2vec* model. This is done in *Python* with the *Gensim* library[7], specifically the *doc2vec* tool[8]. The vector size is set to 100 dimensions, and the size of the context window is set to 5 words. Each document is labelled with the beer ID from the database. The embedding model is saved in a text file, storing only the document embeddings linked to the beer IDs, and excluding the word embeddings.

4.2.3 *StarSpace.*

With the documents preprocessed as described above, a word embedding model is trained by the *StarSpace* neural model, using trainmode 5[9]. Subsequently, an embedding is rendered by the trained model for each document with the embed_doc utility

[3] https://www.mysql.com
[4] https://translate.google.nl
[5] https://github.com/ssut/py-googletrans
[6] https://www.nltk.org/
[7] https://radimrehurek.com/gensim
[8] https://radimrehurek.com/gensim/models/doc2vec.html
[9] https://github.com/facebookresearch/StarSpace#training-mode

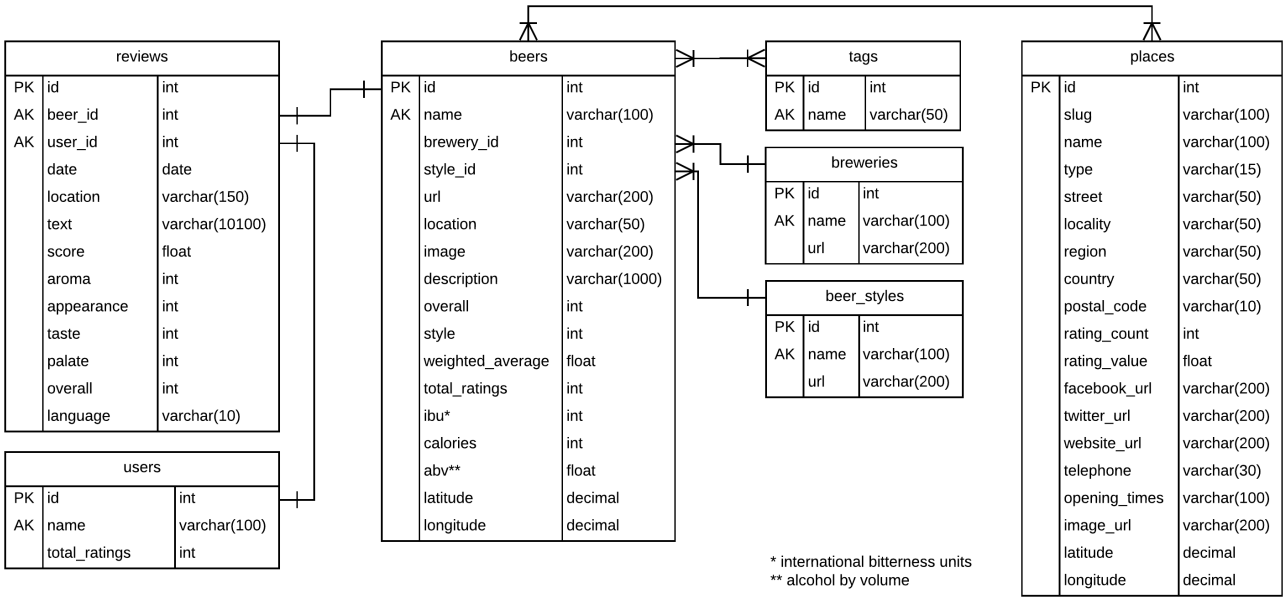


Figure 1: Entity Relationship Diagram

function[10]. This output is piped to a file, which is parsed using a *Python* script to the same format as the *word2vec* model, and again saved in a text file.

Finally, for both *word2vec* and *StarSpace* the outcome of these steps is a file that contains a 100 dimension feature vector for each beer, that is the representation of that beer in the trained model's feature space. This file can be loaded as a word embedding model into a Python script using *Gensim* to use as input for the various prediction models described below.

4.3 Predicting

With the document embeddings as input, a model needs to be created that predicts which beers a user will like. This model should recommend a list of 10 beers, which the user will most likely enjoy, as previously described. Though the system makes a list of 10 recommendations, in the evaluation the models also make a list of 20 recommendations. Therefore the amount of beers recommended is in the description of the models defined by N.

4.3.1 *Models.*

Since the system needs to be able to make personalized recommendations for a user, the classifying model is trained on the specific user's previous likes and dislikes. Two types of classifiers are tested, using the *scikit-learn*[11] tool: a linear support vector classifier and a stochastic gradient descent classifier with a logistic loss function. After training on the user input and making predictions for the remaining beers, the ranking is made by selecting the N beers furthest from the hyperplane.

[10]https://github.com/facebookresearch/StarSpace#print-sentence–document-embedding

[11]http://scikit-learn.org

The performance of these models is compared to baselines of similarity of the feature vectors, as well as popularity. The popularity baseline recommends the N most frequently reviewed beers from the database. The similarity baseline is made by computing the cosine similarity of an input sample to the rest of the items the user rated and ranking the items most to least similar. The prediction from each input sample is combined using late fusion by means of a voting system similar to Michael Pazzani's research [17]. The only difference here is that negative input is taken into account as well, to match the input of the other models. This way each model has the same information to base the recommendation on, and therefore the results are better juxtaposed. Thus the process is as follows. The item most similar to a positive input sample get N, the next most similar item gets N-1 points, and so on until the N^{th} most similar item gets 1 point. The same goes for the negative sample inputs, only then using negative points. The item most similar to a negative sample gets -N points, the next -(N-1), etcetera, until the N^{th} most similar item gets -1 points. This is done for every input sample, that being the user's 3 highest rated and 3 lowest rated beers. The final ranking is subsequently made by summing the points for every beer and recommending the N beers with the most points.

4.3.2 *Evaluation.*

The above mentioned embeddings and models are evaluated using artificial actors, as described by Jan ZahÃąlka et al. [23]. These actors are users that rated beers on the *RateBeer* platform. The performance is measured for each model by calculating the precision for each actor and averaging the result over all actors. For the evaluation, an item in the recommendation list is deemed correct (in other words, "relevant" or "true positive") if the artificial

actor rated the item with a score higher than 3 (out of 5) on the *RateBeer* platform. Due to the sparsity of the ratings, the models initially compose a ranking only of the beers an actor rated. Then, the models make a prediction ranking for all the beers with an embedding representation (those that have at least one English review). Precision is evaluated at 10, which is the intended length of the recommendation list for the system's application, and at 20, to see if the precision increases with a longer list. Each model has their own method for selection the list of predictions, which is described above.

Two different methods of dividing the train data and test data are chosen for the classifiers. One being a common practice in evaluating classifiers and the other because it is closer to the goal and application of the system. The first procedure selects 10% of an actor's ratings as test data and trains on the remaining 90%. Ratings higher than 3 are considered positive input (class 1) and lower ratings are considered negative (class 0). Considering the ranking (on the test data) needs to contain 10 items for precision at 10, actors are users selected from the database who have rated over 100 items. The same applies to precision at 20, selecting *RateBeer* users with over 200 beers rated. Actors are also discarded if the train set contains only positive or negative samples, to avoid random sampling to be able to compute the hyperplane.

The second procedure selects an actor's 3 highest rated items as positive input (class 1), and the 3 lowest items as negative input (class 0). This is naturally a much smaller dataset to train on, but this input is very similar to the final purpose of the system. The results of the tests are described in section 5.

4.4 Interaction Design

To facilitate the interactive process outlined in the problem statement, a prototype is made by creating a *JavaScript* application suitable for modern browsers. The model with the highest evaluation stemming from the above mentioned approach is incorporated in the application's backend.

4.4.1 *Frontend.*

The recommendation is split into two parts, which are presented on two different pages. The first part allows the user to select their preference. The page shows a grid of beers which each has a "like" or "dislike" button. By clicking on these buttons the user can select a few beers they like and a few beers they do not like. By default the 16 most reviewed beers are shown in the grid, since it is most likely the user has tasted those. Each beer is represented by the name of the beer, the style, and the image of the bottle, can, logo, or other (if available). At the top of the page there is a search bar, where the user can query beers by name from the private *MySQL* database. The grid is updated on key input in the search bar to show the query results. When the user is done inputting their preference, they can click on the "Recommend" button, which redirects the user to the second page. A screenshot of this preference page can be found in Figure 2.

The second part displays the personalized list of 10 beers the the system recommends for the user based in their input in part one. For each of these beers the user can again convey their preference through the "like" and "dislike" buttons. They can also go back to the previous page to search for other beers they would like to

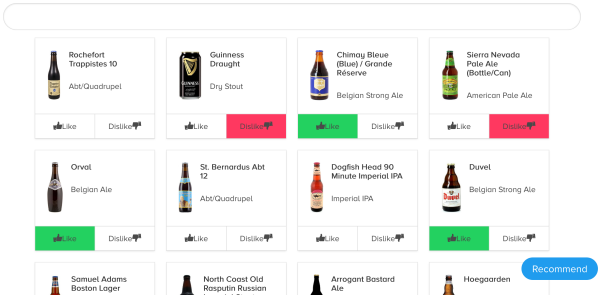


Figure 2: Select Preference Page

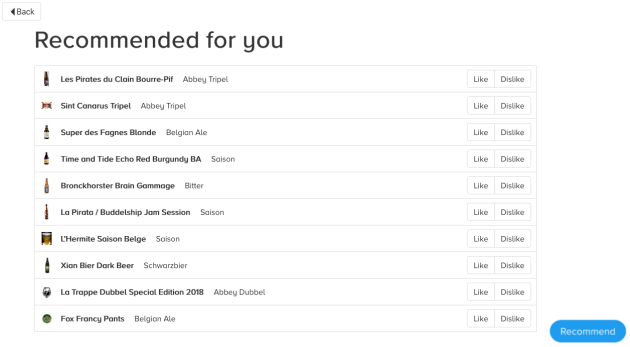


Figure 3: Show Recommendation Page

input for the recommendation, or delete some beers that currently part of the selection. Every time the user clicks the "Recommend" button on either of the pages, they get a recommendation based on the current selection of positive and negative input. This way the user can explore the system and interactively build their preference profile by doing multiple iterations of selecting beers and as a result get more personal recommendations with each iteration. A screenshot of this recommendation page can be found in Figure 3.

4.4.2 *Backend.*

The functionality demonstrated by the frontend is dependent on two API's which provide the information exchange with the backend. The first API communicates with the private database that holds the information scraped from *RateBeer*. The second API provides the recommendation using one of the models as described in section 4.3.1.

The database API is implemented in *JavaScript* using the *Node.js*[12] runtime and *ZEIT's Micro* asynchronous HTTP microservices[13]. It provides two endpoints: one for the search functionality, and one for extracting data about a beer. The search endpoint returns the name, style and image of the beers that match the user's search query. If the query is empty it returns the information for the most frequently rated beers. A call is made to this endpoint when the page is first loaded, and on key input in the search bar. The second endpoint returns the

[12] https://nodejs.org
[13] https://github.com/zeit/micro

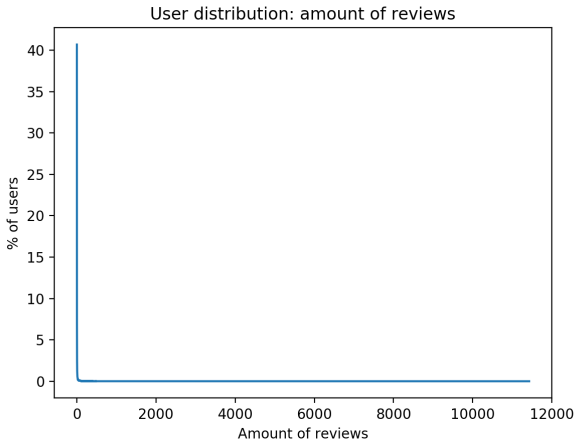


Figure 4: User Distribution Percentage

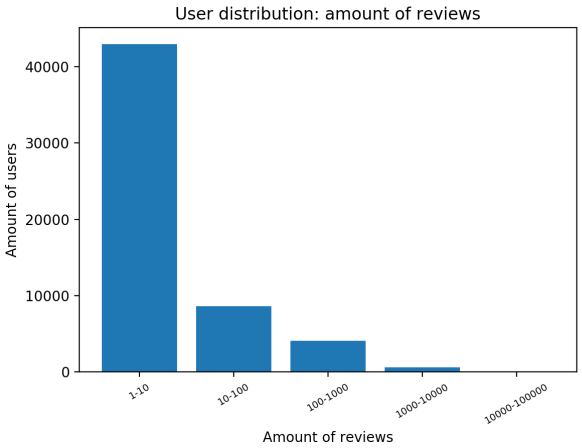


Figure 5: User Distribution Absolute

Table 1: Evaluation Results (user's consumption)

Classifier	Embedding	Samples	Precision	
			@10	@20
Linear SVC	word2vec	6	0,901	0,906
SGD	word2vec	6	0,896	0,902
Linear SVC	StarSpace	6	0,888	0,889
SGD	StarSpace	6	0,887	0,888
Linear SVC	word2vec	90%	0,871	0,821
SGD	word2vec	90%	0,867	0,817
Similarity	word2vec	6	0,865	0,872
Linear SVC	StarSpace	90%	0,860	0,808
SGD	StarSpace	90%	0,855	0,805
Most popular	-	6	0,846	0,826
Similarity	StarSpace	6	0,843	0,851

Table 2: Evaluation Results (all beers)

Classifier	Embedding	Samples	Precision	
			@10	@20
Most popular	-	6	0,316	0,429
Linear SVC	StarSpace	6	0,083	0,000
SGD	StarSpace	6	0,079	0,010
Similarity	StarSpace	6	0,040	0,044
Similarity	word2vec	6	0,038	0,046
Linear SVC	word2vec	6	0,012	0,016
SGD	word2vec	6	0,012	0,015
SGD	StarSpace	90%	0,008	0,000
Linear SVC	StarSpace	90%	0,008	0,001
SGD	word2vec	90%	0,001	0,000
Linear SVC	word2vec	90%	0,001	0,000

information for a certain beer given it's ID. For each beer in the list of recommendations a call is made to this endpoint to collect the name, style and image of the beer.

The recommendation API is implemented in *Python* using the *Falcon* web API framework[14]. Given the IDs from the liked and disliked beers as separate parameters, the API gets the embedded vector representations from these IDs and inputs them as positive and negative samples into the model. The model then provides the list of 10 recommendations. The endpoint finally returns the beer IDs corresponding to these recommendations. A call is made to this endpoint every time the user clicks the "Recommend" button.

5 RESULTS

This section lays out the outcome of the above mentioned evaluation tests. The interpretations of these results are discussed in section 6. Though first, some insight is given into the users used for these test below.

[14]https://falconframework.org

5.1 User-review distribution

The *RateBeer* platform holds 67.029 users that have added their opinion via a rating and reviewing a beer. Figure 4 shows the percentage of users that has rated a certain amount of beers. Figure 5 shows the absolute value of users that have rated certain amount of beers in a logarithmic range.

Ultimately there were 4633 users with over 100 reviews that had both positive and negative ratings for classification, which were used for the precision at 10 tests. There were 3078 with over 200 reviews, which were used for the precision at 20 tests.

5.2 Performance

The results of the evaluation are split into two tables. Table 1 shows the performance of the models with the two types of embeddings on ranking only the beers the users consumed. Table 2 shows the results for the models when comprising a ranking of all the beers in the system. Both tables have been sorted on best performance on precision at 10. All the numbers have been rounded off to 3 decimals.

6 DISCUSSION

As can be seen in Table 1 and Table 2 all the classifiers have a higher precision than the popularity baseline when testing on beers already consumed. However, the popularity baseline still performs much better than any model when testing on all the beers. This is unsurprising, since the chance that an artificial actor has rated the beers in the recommended list (whether positive or negative) is much higher in the popularity recommendations, and beers that are not rated are deemed irrelevant by default.

Notably, the same models perform better when trained the *word2vec* embeddings compared to the *StarSpace* embeddings on the test filtered by consumption, while the opposite is true for the test on all beers. The two classifiers perform almost equally well, though *Linear SVC* has a higher precision on any test except for the one on all beers, with *StarSpace* embeddings and 90% samples. However, the difference on that test is so small that it is only noticeable in the 4^{th} decimal. As was to be expected, all models perform worse when composing a ranking of the entire set of beers, instead of just the beers the user provided a rating for. An explanation for this, as well as other deliberations on the evaluation of the system, is outlined below. Though first the user-review distribution and it's impact on the tests is analyzed.

6.1 Selecting Actors

Examining the user distribution, it is clear that most *RateBeer* users could not be used as artificial actors in the evaluation. Since the ranked recommendation consists of 10 beers, the users that rated 10 beers or less were unsuitable. The users that ranked between 16 and 100 beers could technically have been used in the tests with 6 samples. However, the decision was made to keep the group of actors consistent to better compare results, thus discarding these users as well. It is clear though, that the artificial actors selected for the tests are not the "typical users" of *RateBeer*, since those only review a few beers. The non-normal distribution of the amount of reviews per user was additionally the reason for not picking random users as artificial actors for the test. The chance of picking a user with an insufficient amount of reviews is simply too high.

When selecting the artificial actors the only criteria was that they could partake in the same test for all of the models, meaning they rated enough beers on *RateBeer* to be able to calculate precision@10 or precision@20, allowing to take 90% of the ratings as samples while still having enough left to rank. It could therefore very well be possible that for some actors the system had to rate 10 out of 10 (or 20 out of 20) beers, which all had a positive rating. This has of course a beneficial influence on the average precision. On the other hand, for another actor the system could have had to make a ranking out of beers with only negative labels, which decreases the precision. The effect of these cases are estimated to cancel each other out, but it would have been better to more thoroughly research the actors to see if this is in fact the case.

6.2 Evaluation

Due to the sparsity of the user ratings, the results of test that composed a ranking of all the beers, are most likely lower than reality would be. The beers a user has not consumed cannot be correctly classified as negative, since their opinion on those items is simply unknown. In other words, it cannot be claimed that all the false positives in the ranking are actually true negatives. More research would need to be done to more accurately test the evaluation on all beers, which will be discussed in section 8.

This lack of knowledge due to the sparsity of the data is also one of the reasons it was decided to not include a recall test in the evaluation, even though this is a very popular metric used together with precision in information retrieval and recommender systems [7]. It is unknown which beers are true positives, if the user has not rated them. The other reason is that recall is not relevant if the system does not aim to recommend all of the true positives, even if that information would be available. The goal is to recommend a few beers to consume, not every single beer the user will like. That list would be too long for the user to process. On that account, precision was also calculated no higher than at 20. Looking at the results, when the list of returned items gets longer the precision increases in some cases, but in others it decreases. This inconsistency as well as the unexceptional increase is not judged to weigh up to the costs of showing the user a longer list of recommendations, therefore the initial design of the system remains unchanged.

It is noteworthy that especially in the unfiltered test, the method of training the classifiers on 90% has a lower precision than training on the 3 best and 3 worst rated. It seems to be that the quality of the samples is more important for the prediction than the amount of samples. After all, it is probable that a user's preference can be better learned based on a few items they loved and hated, rather than on a lot of items they felt moderately positive or negative about. Since the model is a binary classifier and only knows whether an item was rated positively or negatively, and not the extent of the (dis)like, it cannot give weights to the samples. Regression would then be a better fit for this task. However, the users of the *Beerlytics* system ultimately do not specify the degree of their (dis)like either, therefore the weight of the rating was not used with the artificial actors, and regression was not considered.

This scale of preference provides another point of discussion in the evaluation tests. Currently the boundary between like and dislike is set at 3 (out of 5). However, not every user rates using the same scale. For some users a 3 might be a beer they've mildly enjoyed, while others could more generous with their points and would not want to drink a beer they rated with 3 points again. For the tests using the most positive and negative samples this is less of a problem when choosing the samples, since it does not look at the ratings at that time, meaning the samples are positive and negative relatively to the rest of the rated items. However, for all tests it is a complication when evaluating the precision with artificial actors. This issue could be solved by repeating the tests using the adjusted weighted sum, or preference-based filtering when evaluating the ranking [2].

Overall most issues in this research arise from the discrepancies between the tests and the system's application. The data available was applied to simulate the system's usage, but in order to do this some compromises had to be made. Preferably the performance of the models would have been evaluated exactly in the same way the system operates.

7 CONCLUSION

The results show that all classifiers outperform the popularity baseline when testing on the beers a user consumed. The classifier with the highest precision is Linear SVC, using the *word2vec* embeddings. When testing on the entire set of beers, none of the models manages to surpass the popularity baseline, which can be explained by the sparsity of the data. However, after the popularity ranking, the most precise model is again the *Linear SVC classifier*, this time with *StarSpace* embeddings. Since the models were not expected to perform well on such sparse data (as explained in section 6), and they did perform well on recommending only beers an actor consumed, it can be concluded that word embeddings learned from consumer reviews on a beer community platform provide a good foundation for the *Beerlytics* recommender system. This conclusion can be expanded to suppose systems based on word embeddings will also perform well on other applications, such as recommending different products, or retrieving documents based on a query.

Since the models based on the *word2vec* embeddings perform better when testing the consumed beers, and the models based on the *StarSpace* embeddings perform better when testing all beers, it cannot be said conclusively that one embedding is better than the other. More research would have to be done to reach a conclusion on the type of embedding that is best for the *Beerlytics* system.

8 FUTURE WORK

As can be drawn from the discussion, the next step for the *Beerlytics* system would be to evaluate the recommendations with a test that can confidently identify a user's true negatives, and all of the true positives. One proposed way to do this, would be to do an actual user study instead of a simulation with artificial actors. These users can then use the system as intended, by inputting a few beers they liked and disliked, optionally in multiple iterations, and then consuming the beers recommended by the system. At the same time, this user study can be combined with testing the system's frontend. The design needs to be assessed to make sure that the user interaction is clear and intuitive. Doing multiple iterations of this study can improve the frontend's design in many ways.

Additionally, it would be interesting to look not only at the quality of the recommendations, but also at the variance. The interactive component of the system is supposed to support long tail recommendations, therefore the beers that are recommended should be analyzed based on popularity. Does the system recommend any beers the user is unfamiliar with? And in succession to that, does the user prefer serendipitous recommendations? Specifically, does the user prefer the *StarSpace* or the *word2vec* recommendations?

Furthermore, the classifiers used in this thesis are simple and established, but not necessarily the best for this use case. A more extensive classifier, for instance one that learns online from the input of all users, could researched to try to increase the performance of the recommendations. Once the system is deployed this model could train alongside the present classifier, which would be replaced once the new model outperforms the old one. Another approach is to combine collaborative filtering with the current content filtering recommender. Again, the variance is of interest here, to see if the type of recommendations change when using a different model.

REFERENCES

[1] Silvana Aciar, Debbie Zhang, Simeon Simoff, and John Debenham. 2007. Informed Recommender: Basing Recommendations on Consumer Product Reviews. *IEEE Intelligent Systems* 22, 3 (May 2007), 39–47. https://doi.org/10.1109/MIS.2007.55

[2] Gediminas Adomavicius and Alexander Tuzhilin. 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge & Data Engineering* 6 (2005), 734–749.

[3] David M. Blei. 2011. Introduction to probabilistic topic models. In *In Communications of the ACM*. https://doi.org/10.1145/2133806.2133826

[4] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet Allocation. *J. Mach. Learn. Res.* 3 (March 2003), 993–1022. http://dl.acm.org/citation.cfm?id=944919.944937

[5] Ronan Collobert and Jason Weston. 2008. A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In *Proceedings of the 25th International Conference on Machine Learning (ICML '08)*. ACM, New York, NY, USA, 160–167. https://doi.org/10.1145/1390156.1390177

[6] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. 2010. Performance of Recommender Algorithms on Top-n Recommendation Tasks. In *Proceedings of the Fourth ACM Conference on Recommender Systems (RecSys '10)*. ACM, New York, NY, USA, 39–46. https://doi.org/10.1145/1864708.1864721

[7] Jonathan Herlocker, Joseph Konstan, Loren Terveen, and John Riedl. 2004. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)* 22, 1 (2004), 5–53.

[8] T. S. Huang, C. K. Dagli, S. Rajaram, E. Y. Chang, M. I. Mandel, G. E. Poliner, and D. P. W. Ellis. [n. d.]. *Proc. IEEE* ([n. d.]).

[9] Niklas Jakob, Stefan Hagen Weber, Mark Christoph Müller, and Iryna Gurevych. 2009. Beyond the Stars: Exploiting Free-text User Reviews to Improve the Accuracy of Movie Recommendations. In *Proceedings of the 1st International CIKM Workshop on Topic-sentiment Analysis for Mass Opinion (TSA '09)*. ACM, New York, NY, USA, 57–64. https://doi.org/10.1145/1651461.1651473

[10] Dietmar Jannach, Lukas Lerche, Fatih Gedikli, and Geoffray Bonnin. 2013. What Recommenders Recommend – An Analysis of Accuracy, Popularity, and Sales Diversity Effects. In *User Modeling, Adaptation, and Personalization*, Sandra Carberry, Stephan Weibelzahl, Alessandro Micarelli, and Giovanni Semeraro (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 25–37.

[11] Adriana Kovashka, Devi Parikh, and Kristen Grauman. 2015. WhittleSearch: Interactive Image Search with Relative Attribute Feedback. *International Journal of Computer Vision* 115, 2 (01 Nov 2015), 185–210. https://doi.org/10.1007/s11263-015-0814-0

[12] Asher Levi, Osnat Mokryn, Christophe Diot, and Nina Taft. 2012. Finding a Needle in a Haystack of Reviews: Cold Start Context-based Hotel Recommender System. In *Proceedings of the Sixth ACM Conference on Recommender Systems (RecSys '12)*. ACM, New York, NY, USA, 115–122. https://doi.org/10.1145/2365952.2365977

[13] Rishabh Mehrotra, Scott Sanner, Wray Buntine, and Lexing Xie. 2013. Improving LDA Topic Models for Microblogs via Tweet Pooling and Automatic Labeling. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*. ACM, 889–892. https://doi.org/10.1145/2484028.2484166

[14] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *CoRR* abs/1301.3781 (2013). arXiv:1301.3781 http://arxiv.org/abs/1301.3781

[15] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 3111–3119. http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

[16] Yashar Moshfeghi, Benjamin Piwowarski, and Joemon M. Jose. 2011. Handling Data Sparsity in Collaborative Filtering Using Emotion and Semantic Based Features. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '11)*. ACM, New York, NY, USA, 625–634. https://doi.org/10.1145/2009916.2010001

[17] Michael J Pazzani. 1999. A framework for collaborative, content-based and demographic filtering. *Artificial intelligence review* 13, 5-6 (1999), 393–408.

[18] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*. ACM, 285–295.

[19] Yue Shi, Martha Larson, and Alan Hanjalic. 2014. Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Computing Surveys (CSUR)* 47, 1 (2014), 3.

[20] Yue Shi, Pavel Serdyukov, Alan Hanjalic, and Martha Larson. 2013. Nontrivial Landmark Recommendation Using Geotagged Photos. *ACM Trans. Intell. Syst.*

Technol. 4, 3, Article 47 (July 2013), 27 pages. https://doi.org/10.1145/2483669.2483680

[21] Harald Steck. 2011. Item Popularity and Recommendation Accuracy. In *Proceedings of the Fifth ACM Conference on Recommender Systems (RecSys '11)*. ACM, New York, NY, USA, 125–132. https://doi.org/10.1145/2043932.2043957

[22] Ledell Wu, Adam Fisch, Sumit Chopra, Keith Adams, Antoine Bordes, and Jason Weston. 2017. StarSpace: Embed All The Things! *CoRR* abs/1709.03856 (2017). arXiv:1709.03856 http://arxiv.org/abs/1709.03856

[23] Jan Zahálka, Stevan Rudinac, and Marcel Worring. 2015. Analytic Quality: Evaluation of Performance and Insight in Multimedia Collection Analysis. In *Proceedings of the 23rd ACM International Conference on Multimedia (MM '15)*. ACM, New York, NY, USA, 231–240. https://doi.org/10.1145/2733373.2806279

[24] J. ZahÃąlka, S. Rudinac, B. Ãð. JÃşnsson, D. C. Koelma, and M. Worring. 2018. Blackthorn: Large-Scale Interactive Multimodal Learning. *IEEE Transactions on Multimedia* 20, 3 (March 2018), 687–698. https://doi.org/10.1109/TMM.2017.2755986

[25] J. ZahÃąlka, S. Rudinac, and M. Worring. 2015. Interactive Multimodal Learning for Venue Recommendation. *IEEE Transactions on Multimedia* 17, 12 (Dec 2015), 2235–2244. https://doi.org/10.1109/TMM.2015.2480007