Final report Optimization of Business Processes (DSS)

Yannick Hogebrug (2625424), Dominique Hopman (2624150),
Jesse Schouten (2621562), Daan van der Thiel (2624780),
Berend Mortier (2625053)

1 February 2019

Introduction

Finding the right job for the right person is a problem that becomes more complex with the more data we gather. For this we built a recommender system with performance limitations and a nice interface. Our objective is: "Build a job recommender system which, given input data from a job seeker and a large database of vacancies, recommends vacancies to the job seeker in decreasing order of fit"

During this project a decision support system (DSS) was built that provides job recommendations to potential job seekers. This is embedded in a Graphical User Interface (GUI) in which a user is able to create an account, log in to the system and apply to a recommended job. When logged in to the account a list of 100 jobs is shown on which the user can decide to apply if he/she is interested. The recommendations are constructed from an algorithm that gains information from personal details that the user gives when submitting an account, and user behavior. An user manual is added to the report to guide users through the system, as well as documentation to provide all necessary information to rebuild the system potentially. Furthermore, a scientific explanation of the algorithms and all verification which was done, is reported.

1 Data Analysis

For the data analyses we first looked at how many applications a certain vacancy on average gets. Figure 1 shows the frequency of the number of applications on a certain job. Here it can be seen that most of the job vacancies only get one application.

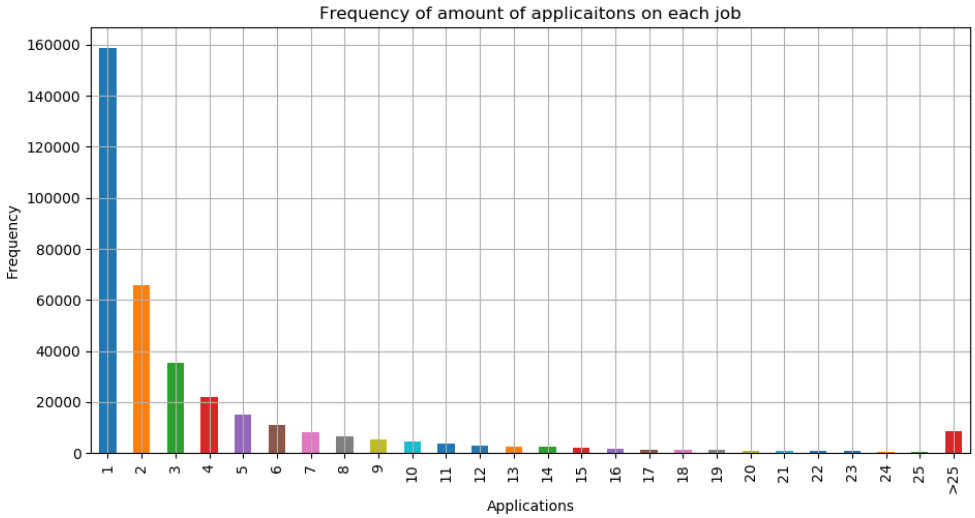

Figure 1: frequency of applications on job vacancies

Also, we looked at the people from a certain state and how many of them applied to a job in the same state as they live. For this we created a table, which can be found in the appendix. From this it can be seen that a lot of people apply to vacancies in there own state. This is why we filter the data on the state of the person on which we try to predict. We will come back to this later in this report, when we talk about the algorithms that were used.

In the user history the job history is also given. For this the user gives the name(s) of his/her previous job(s). We looked at how many times a title occurs. This is shown in Figure 2. It shows that around 570000 of the job titles only occur once. This is an absolute majority, so we concluded that comparing the job titles without any text mining would not seem to be useful.

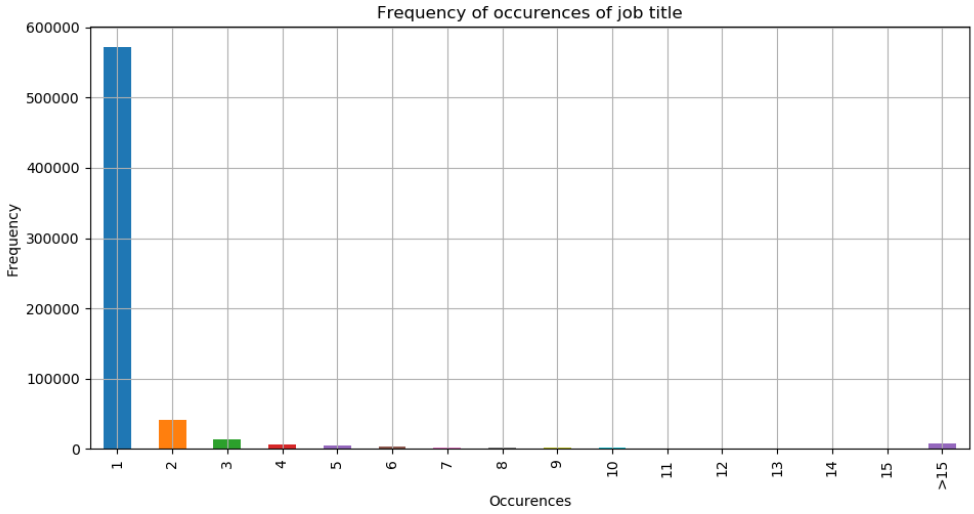

Figure 2: frequency of applications on job vacancies

We were also interested in the Major a person has. We found that there are a lot of unique majors. In Figure 3 it can be seen that 37000 of the majors only occur once. For example the major "Accounting Finance" and the major "Accounting / Finance" occur once but these could be put together under the major Accounting. As with the job title we seem to first need text analyses to match this variable. We experimented with the text mining, but this was computationally expensive. So, we decided quite soon that we will not be able to do this due to the computation time and the lack of time for this project.

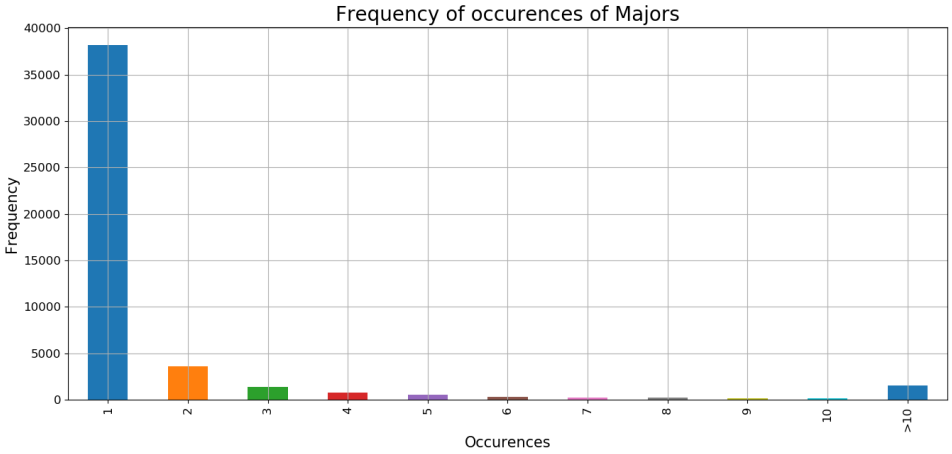

Figure 3: frequency of majors

In Figure 4 the degree type of all the users is given. Here it can be seen that only a small percentage is PHD or Vocational and the majority has No, High School or a Bachelor education.

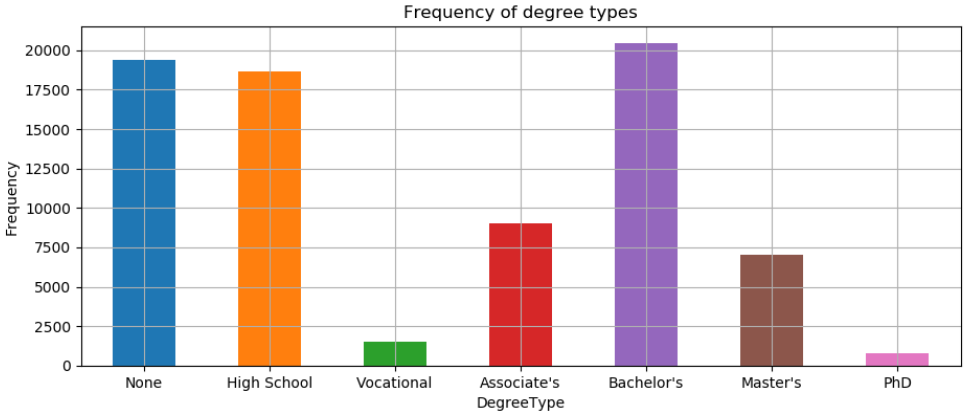

Figure 4: frequency of degrees

2 Data preprocessing

It was known some attributes in the datasets were not useful, as the data was grabbed from a kaggle competition where the objectives were different. Therefore, 'WindowID' and 'Split' are the first columns to be dropped from all the datasets. Note that the DSS does assume these columns to be present in the inputfiles when loading the data. Furthermore, only the first three digits of 'Zipcode' are used, which means the remaining part of the string is dropped.

Several small transformations where performed on the data. To sum up: in order to deal with date(time)-type variables, all features of this types, for instance 'EndDate', 'StartDate' and 'ApplicationDate', are formally transformed from a string format to a python date format. From this point, the graduation year is added as a feature to the user dataset. Also, string values like 'None', 'NA' and np.nan values are transformed to NoneType variables to make life easier later on in the modeling.

As the application history of the users is of particular interest to predict future applications, a list of all application histories was added to the user data. This method was chosen over the usage of sparse matrices, as these would get very large due to the large amount of different Job ID's. For reasons that are mentioned in other parts of the report, the job history of users was not included in the model as well, although it is noted that this is expected to be a very important predictor in the recommendation of interesting vacancies to users.

An important detail of the job recommendation system is that it should only recommend jobs that are still open for applications. Based on the date at which a user is using the DSS, the jobs data is filtered so that it only contains jobs that have not yet ended, but do have started. So, users won't get recommendations of jobs that have already ended at the time they click the recommendation button in the DSS.

All the preprocessing mentioned above is done by starting the DSS. From the project description, it is stated that someone must be able to replace the dataset(s) with another one with the same columns and names. Doing the preprocessing directly when starting the DSS, makes this as simple as possible. You only have to replace the file in your working directory and run the DSS. This is explained in further detail in the chapter 'DSS'.

3 DSS

DSS stands for Decision Support System. In this project the DSS consists of a job recommendation system. Users can register an account and receive a top 100 with personalized job recommendations. When logged in to the DSS, users can also view personal information, job history, application history and apply to vacancies for example. It is also possible to create a new account or delete an account.

3.1 Preparation

To setup the DSS the first time, the following things have to be done:

1. Install Pycharm, version 2018.3.4.
2. Open the Zip-file called 'DSS' and create a new map in a certain directory <directory> and unpack the zip-file in here.
3. Start Pycharm.
4. Create a new project and select the map 'DSS' in the directory you just determined. Click OK and you will some code appearing on the screen with multiple code tabs at the top.
5. Go to File (left top corner) → Settings → Project: DSS → Project Interpreter → click on the settings button (right top corner) and click on Add.
6. Select Virtualenv Environment and select Existing environment. Click on the ... beside Interpreter and select the map 'DSS' in the directory where you unpacked the Zip-file and go to <directory> \DSS\venv\Scripts\python.exe.
7. Click OK and you will see a lot of packages/libraries like pandas and flask.
8. Click on Apply and thereafter OK.

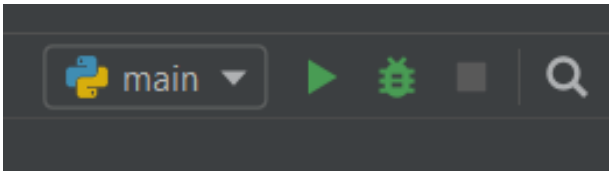

Figure 5: Configuration

9. Click in the right top corner on **main** beside the green play button (see Figure 5 and click on Edit Configurations.
10. After Python interpreter, select the interpreter you just created (with DSS in the name) and click on Apply and then OK.

11. It is possible that you have to wait a few moments now. When the play button becomes green again, you can click on it and run the script. The script runs in approximately 1,5/2 minutes on a 16GB RAM computer. When it is finished you can click on http://127.0.0.1:5000/ and the DSS wil start. The warnings which are shown, can be ignored.

12. You are now able to use the DSS. In the next subsection, all the features of the DSS are discussed.

3.2 User Manual

The DSS has multiple pages which will be explained below. When you start the DSS, the files are loaded from your directory. Be aware that you always place the data files in the same map as the maps 'venv', 'main' and the file 'run.py'. When you unpacked the Zip-file in an earlier stage, the data files are already included. If desirable, you can replace these files with other data files, but they must have exactly the same name and structure.

Features

The DSS has the following features:

- Load new dataset
- Register
- Login
- Create new account
- Delete account
- Recommendations list
- Apply to a job
- Export CSV

In the next subsection these features are discussed in a logical order.

Manual

When the DSS is started, the following home page is shown:

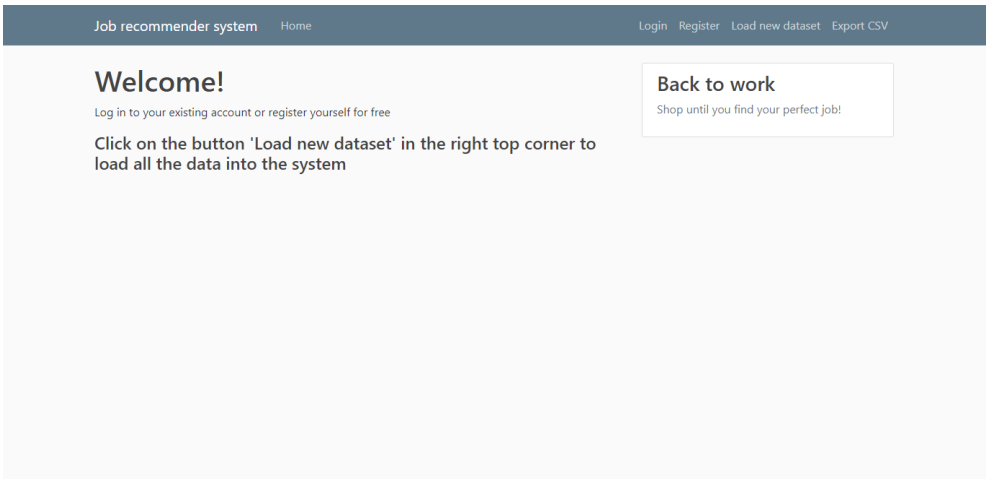

Figure 6: Initial welcome screen

You see a warning that the data must be loaded into the system first via 'Load new Dataset'. If you click on this, you go to the screen below.

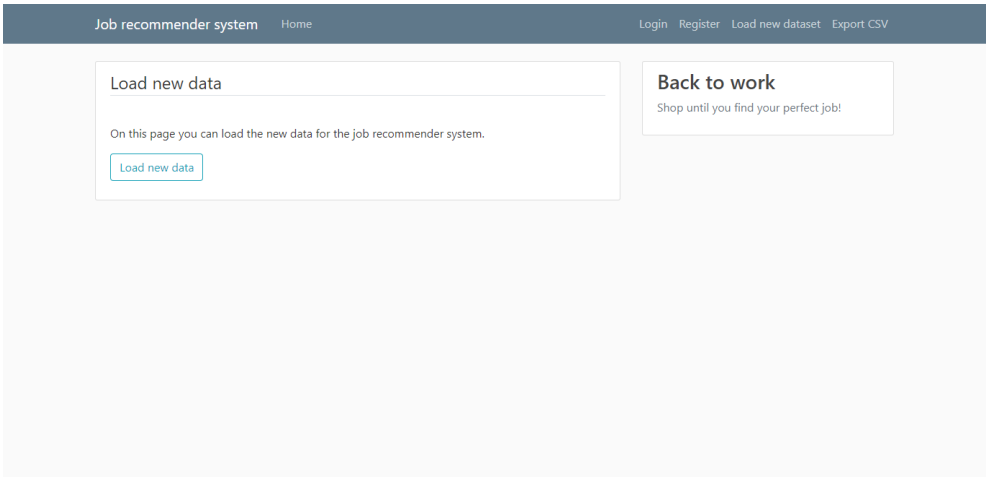

Figure 7: Load database

If you press the button 'Load new data', the data will be loaded into the system. You only have to do this the first time you visit the DSS or if you want to start all over again with the initial database, based on your data files. But don't forget to load the new dataset, because otherwise you are not able to log in. However, you are able to create a new account and log in with it, but you will not see any recommendations, because there is no application history data available.

After you have loaded the new data, you go back to the home page an you will see the screen shown in Figure 8.

The user sees some information about the data in the system and can go to the register page by pressing the button or log in to an already existing account by clicking on the blue-colored word Login below the register button..

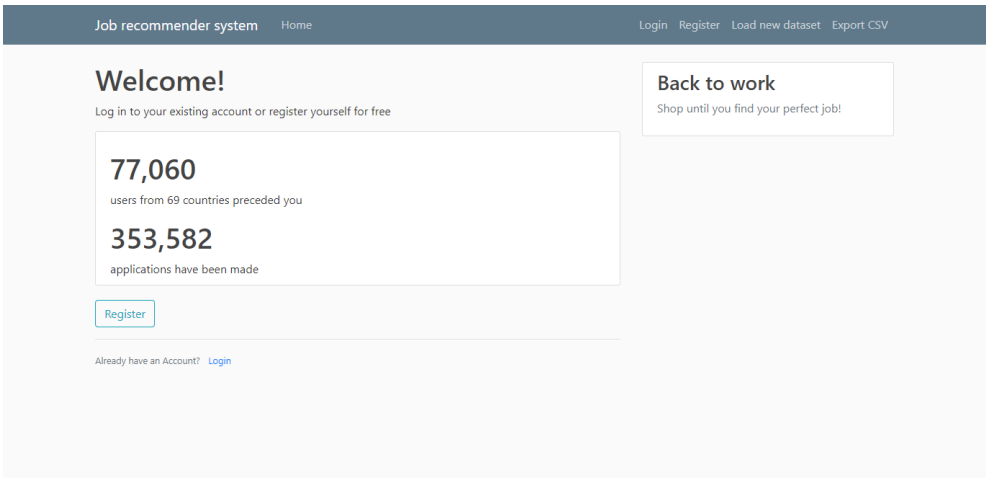

Figure 8: Welcome screen with loaded database

When a new user comes into the system, he/she can make an account by clicking on the earlier mentioned register button or on Register in the top blue bar. Then, he/she will see the following page:

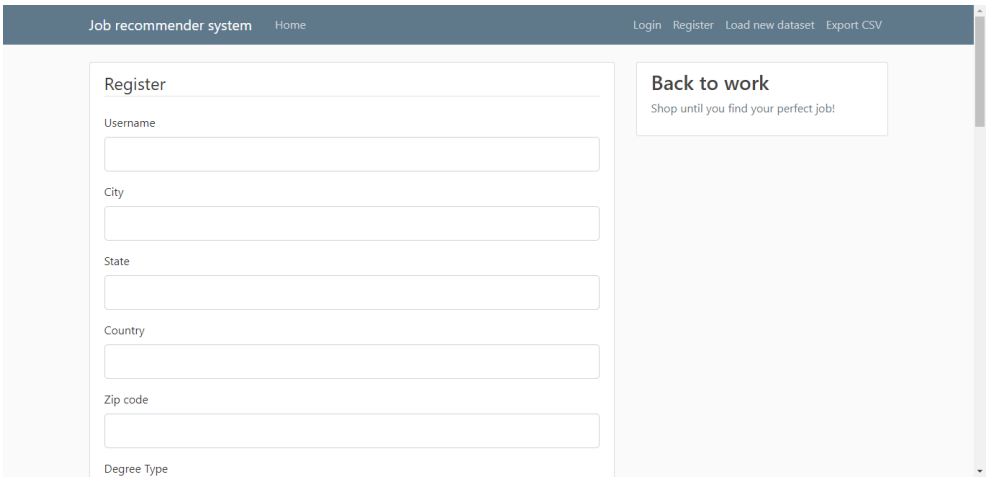

Figure 9: Register screen

All fields are optional, except for the Username and password fields. Be also aware that you type two times the same password, otherwise you get an error that they don't correspond with each other. The more data you fill in about yourself, the better recommendations we can give to you. After filling in your personal information you can sign up. You will be redirected to the login page with a green message that your account has been created succesfully and you are now able to log in. This screen is shown in Figure 10.

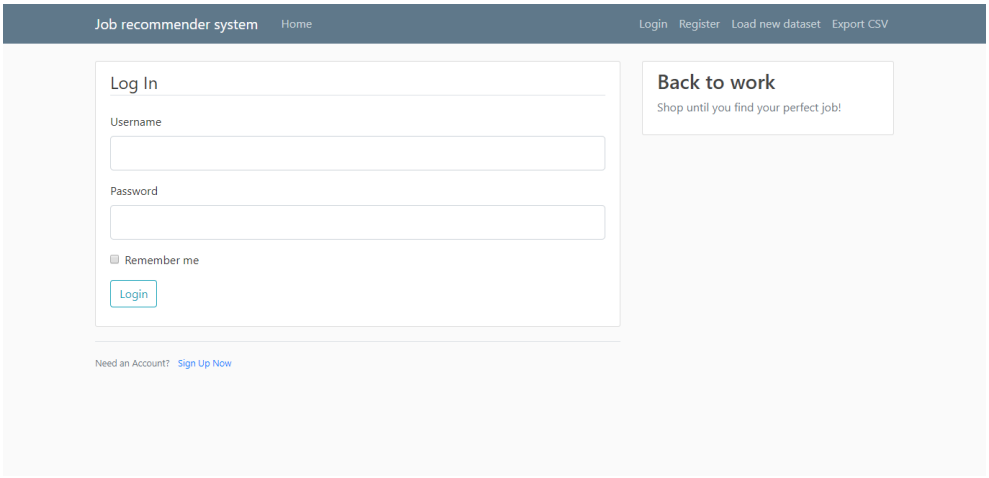

Figure 10: Login screen

If you fill in the right combination of your username and password, you will be logged in into the system and see the screen shown in Figure 11

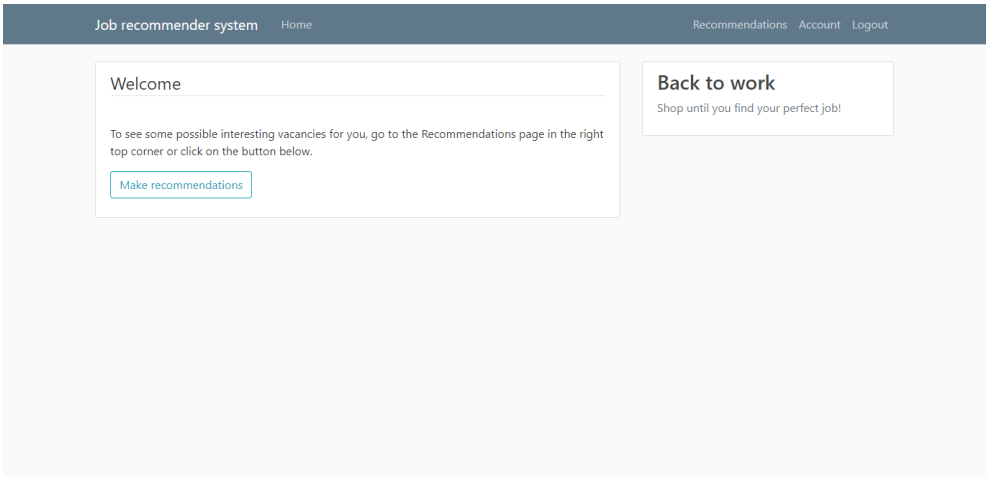

Figure 11: Welcome screen with user logged in

You get a green flash message that you are logged in and you can see that the elements in the top blue bar are changed. It is now among others possible to go to a recommendations page. This can also be done by clicking on the button 'Make recommendations'. When you click on either the button or in the top blue bar, your specific recommendations are calculated based on your personal information and probably your historical behaviour in the system. You will get a top 100 recommendations, like shown in Figure 12.

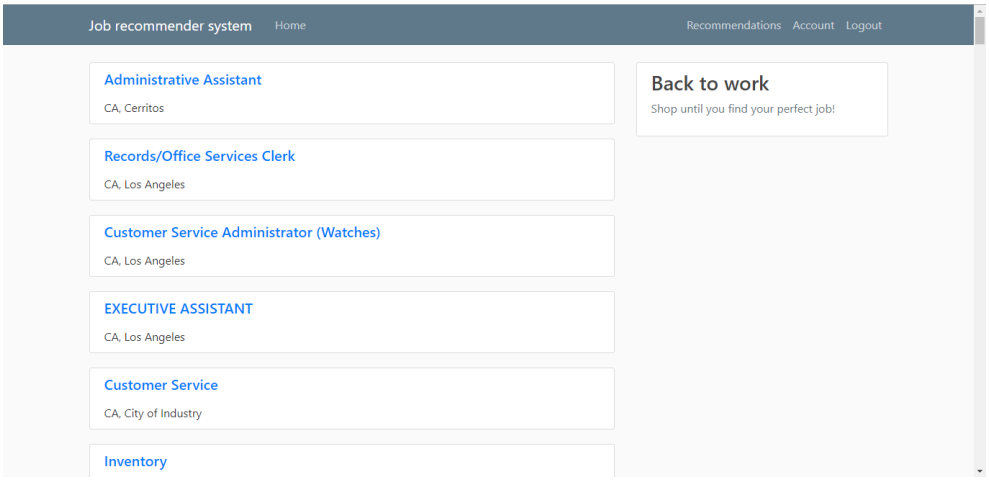

Figure 12: Recommendations

You will see only the title of the vacancy and some information about the location. The user can click on a vacancy in the recommendation list and view the more detailed information. This looks like shown in Figure 13.

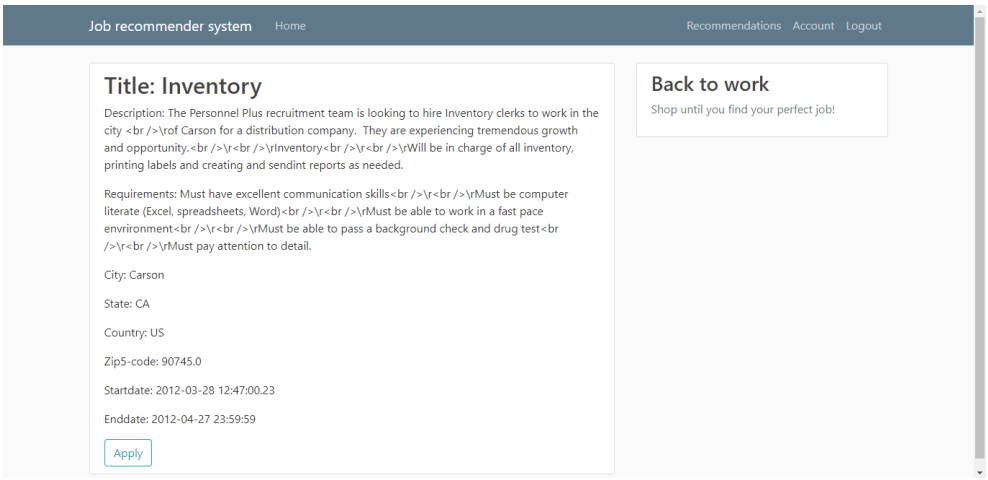

Figure 13: Vacancy

So, the user sees the more detailed information now. Based on this information, he/she can consider if he/she is interested in the vacancy. If not, the user can go back to the recommendation page by going one page back. If the user is interested, he/she can apply on the vacancy on the bottom of the vacancy. If he/she applies, a message will appear that the application has been successful. If the user go back to the recommendations now, the vacancy where the user just applied on, has been removed in the recommendations. Also, a new top100 of recommendations has been constructed, keeping in account the new application.

Furthermore, there are a few other features in the DSS. One of them is the Account page, which is shown in Figure 14.

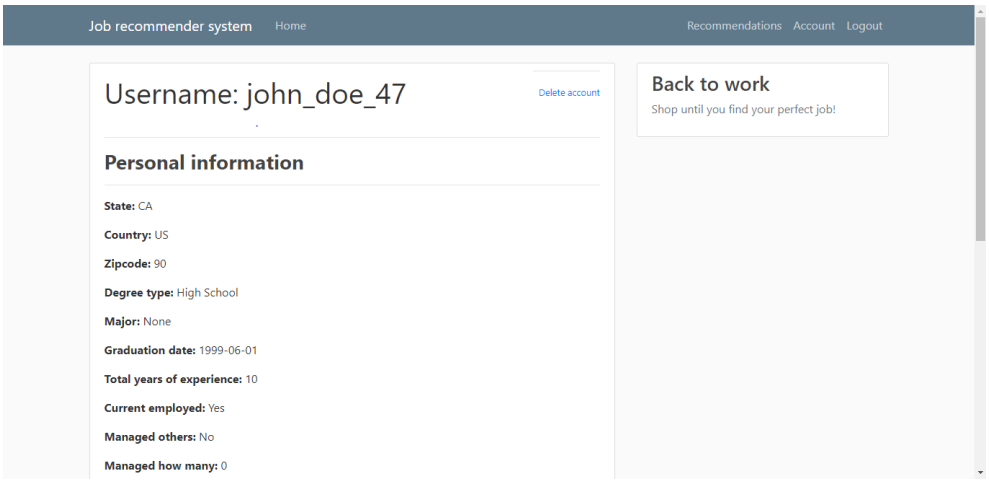

Figure 14: Account screen

An user can see all his/her personal information, his/her job history and the applications he/she already applied on. At the top, you are able to delete your account and all your corresponding data. By clicking on 'Delete account', you will be redirected to a new page, where you can deleted your account permanently (See Figure 15. Thereafter, you will be automatically redirected to the home page and you see a flash message that your account has been deleted.

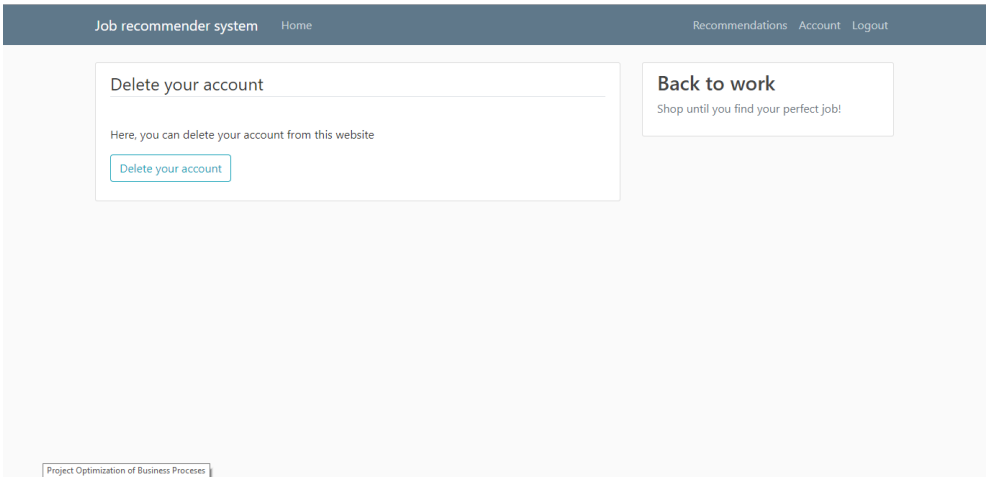

Figure 15: Delete account

You are also always able to log out of your account by pressing 'Logout' in the top bar besides 'Account'.

The last feature of the app worth mentioning is the possibility to export a top100 for all users into a csv-file. This can be done via 'Export CSV' in the top bar and is only visible if there is no user logged in. The resulting screen is shown in Figure 16

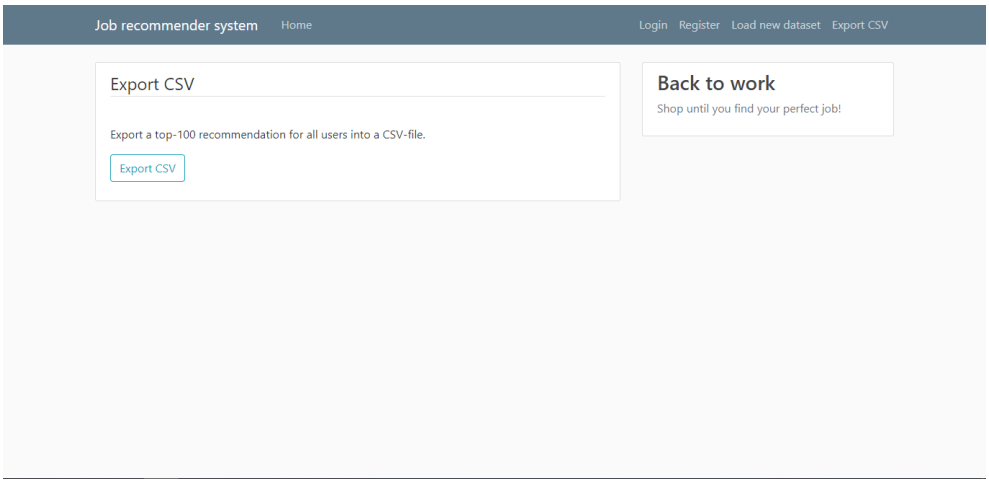

Figure 16: Export CSV

By pressing on the button 'Export CSV', the top100 for every user will be calculated and stored in a csv-file. Keep in mind that this can take some time, because the system has to construct a different top100 many times. If the export is finished, a flash message will appear that the export is successfull. The csv-file can now be found in your earlier adjusted directory.

You can choose by yourself from which users you want the top100 in the mentioned csv-file. In the file 'testusers.csv', add in the first row the id's of the users from which you want a top100. See the example in the included csv-file in which format you have to specify the user id's.

3.3 Documentation

The DSS is programmed in Python. For the Graphical User Interface (GUI), the program Pycharm is used. As earlier mentioned in the subsection 'Preparation', the user has to conduct some steps to use it. These are elaborately explained down there.

For an optimal look of the DSS, an internet connection is required.

3.3.1 Packages/libraries used

To build the DSS several packages were used which will be explained below. These packages are automatically available by following the steps explained in the subsection 'Preparation'. A short description of every package/library can be found below.

- **Flask**
  Version 1.0.2
  A simple framework for building complex web applications. This package is used to build up the DSS application.

- **Flask WTF**
  Version 0.14.2
  Simple integration of Flask and WTForms. This package is used to integrate the packages Flask and WTForms.

- **WTForms**
  Version 2.2.1
  A flexible forms validation and rendering library for Python web development. This package is used to set all the forms on the different (HTML) pages.

- **Flask Login**
  Version 0.4.1
  User session management for Flask. This package is used to make it possible to login and register an user account. Also we can check if the username and password of an user are correct.

- **Pandas**
  Version 0.23.4
  Powerful data structures for data analysis, time series and statistics. This package is e.g. used to construct a dataframe and write it to a csv-file.

- **Numpy**
  Version 1.16.0
  NumPy is the fundamental package for array computing with Python. This package is used in the algorithm to improve the computing performance.

- **SQLAlchemy**
  Version 1.2.16
  Database Abstraction Library This package is used to make it possible to use a database.

3.3.2 Database

To use different functions, such as login authentication, there is a local database set up with the help of the package SQLAlchemy. The algorithm itself uses a dataframe which is also initialized based on the input data files. We found out

that this is much faster than getting the data from the database and run the algorithm on this. New users or applications will be stored in both the database as the dataframe in exactly the same way.

3.3.3 Runtime

During the project, we found out that the runtime was one of the biggest issues. However, this is an important factor in a recommender system, because you don't want to let the user wait for a long time, when he/she is in the system. The run time of the activities which are not completed (almost) immediately, are considered below. Because the DSS will be runned on a 12GB RAM computer and we have 8GB and 16GB RAM computers available, we will give the runtimes on these computers below.

The current available datasets are used. When new data is loaded, the runtime can change, dependent on the size of the data. The runtime of the DSS with a windows computer with 8 gig RAM:

- Time it takes to load new data (only needs to be done once): 7 minutes
- Time it takes for the DSS to start up: 104 seconds
- Time it takes to produce recommendations: 110 seconds

The runtime of the DSS for a windows computer with 16 gig RAM:

- Time it takes to load new data (only needs to be done once): 3 minutes and 17 seconds
- Time it takes for the DSS to start up: 103 seconds
- Time it takes to produce recommendations: 2 minutes and 25 seconds

3.3.4 Restrictions

The DSS has also some restrictions:

- In the DSS it is not possible for a user to replace his/her (personal) input data.
- When a new account is created, the user is not automatically logged in. This can be either seen as a advantage or a disadvantage.
- The description and requirements text for a vacancy are sometimes a litte messy. As stated in the project description, the main focus is not on decode the texts in this fields. However, the html tags are removed from the text.

4 Algorithms

There exist a number of different algorithms that are frequently used in recommendation systems. All have advantages and disadvantages compared to other models. As computational efficiency was quite important, a case-based approach was used during this project as there is a lot of control over the magnitude of the algorithm: it is very easy to start small and extend from there on. Also, item-based collaborative filtering is highlighted as it might tackle some of the disadvantages of this approach, although it was not used in the DSS due to computational reasons.

4.1 User cased-based recommendation

By user cased-based recommendation a similarity score is calculated between users. The users are represented by a set of features and feature values. Based on these features a similarity score can be computed between users and a top-k peer group can be selected. The ranking of the jobs is based on the similarity score and the jobs these peers applied to that the user itself did not apply to yet. The score of a job is the sum of the similarity scores of all the users in the peer group that applied on this job. Finally the top 100 of the job scores give the ranking.

The downside of this approach is when nobody has ever applied to a certain job, that job will never be recommended. This problem can be solved with item-based collaborative filtering.

4.2 Filtering

The similarity scores are computed over all the users. However, a user might get high similarity scores with people on the other side of the country and will get job recommendations on the other side of the country, which is not very likely that a user will apply on. Therefore the algorithm will use the scheme shown in Figure 17.

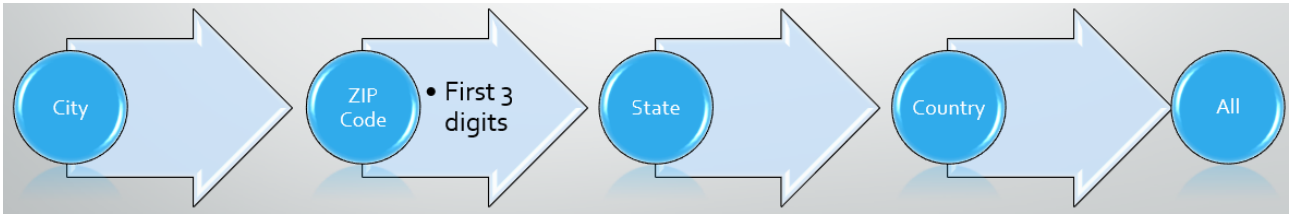

Figure 17: Algorithm scheme

Basically, the algorithm tries to construct a top 100, starting by comparing the user to a small subset in the dataset: other users that live in the same city. If

the algorithm is not able to construct a top 100, it moves to the zipcode (for which only the first 3 digits are taken). The subset will be expanded according to Figure 17 until a top 100 is made for the particular user.

The jobs in the top 100 should meet the following criteria:

1. The user should not have already applied to the job.
2. Considering the time at which a user clicks the recommendations button, this time must be between the start- and enddate of the vacancy.

It is noted that a subset does not have to be evaluated more then once. Every time the top k items can be stored when moving to a new subset. For instance: if it is not possible to construct a top 100 comparing a user to other users in the same city, we can remember all the vacancies that are relevant from those users and move to people sharing the same zipcode that don't live in the same city. This is a nice way of shortening the process time of the algorithm.

4.3 NaN values

When a new user adds his or her information everything is optional, which results in a lot of undefined values, also described as NaN values. There were two options considered how to deal with these NaN values. The first option was not take the feature into account if one or both of the users you calculate the similarity for, has a nan value. The other option is do take the NaN values in account, thus when both users have a NaN value for a feature, this will be treated as similar. The difference between the two options for the score is minimal. Therefore the DDS uses option two, because this seemed to be the faster one.

4.4 Similarity

A user can fill in several characteristics, also known as features, in the DSS, such as the state, job history and total years of experience. To recommended fitting jobs to a user, all those features could be of potential help. Based on computational efficiency and the data analyses, multiple features were selected and grouped in sets based on data type and expected importance: location, degree, categories, numerical categories and application history. In these feature sets, a number of 12 features are used to calculate similarity scores between users. This is constructed as following:

$$f_1 = [City, ZipCode, State, Country]$$
$$f_2 = [Degreetype]$$
$$f_3 = [Major, CurrentlyEmloyed, Managedothers]$$
$$f_4 = [WorkHistoryCount, TotalYearsExp, GraduationYear]$$
$$f_5 = [ApplicationHistory]$$

The next step is to calculate similarity scores based on these feature sets. The approach in this report is inspired from Guo et al. [2], although the authors of these papers proposed the formula for seperate features instead of feature sets. Though, the approach is pretty much the same, but just repeated multiple times for all features in a set. The similarity score between two users is computed by the function:

$$sim(U_a, U_b) = (\frac{\sum_{i=1}^{m} w_i \times sim(f_i^a, f_i^b)}{\sum_{i=1}^{m} w_i})^{\eta},$$

where w_i is the weight for the similarity for feature f_i and η is some constant value larger or equal to one. η could function as a parameter that gives an advantage to a user with a relatively large similarity score, as powers affect large values more than small values. Whether this is useful will follow from the tests later on in this chapter. This formula is exactly the same in guo et al., except for this η. [2]

When a categorical feature consists of one value the similarity is 0 or 1, 0 when the users have difference values for that feature and 1 when they have the same value. Therefore, the similarity score in a feature set with categorical features is equal to the number of equal categories. For the numerical feature set the following function will be used to determine the similarity[2]:

$$sim(f_i^a, f_i^b) = \sum_{k=1}^{m} (1 - \frac{|v_k^a - v_k^b|}{max(v_k^a, v_k^b)}),$$

where v_k^a and v_k^b are the numerical values and m is the number of features in this set.
Furthermore, the users all have a feature 'application history', which is a list of the jobs they applied to. To determine the similarity between two of these lists, the Jaccard index will be used. This formula basically describes the intercept of a set as a fraction of the union of the set. It is defined as following[2]:

$$sim(f_i^a, f_i^b) = \frac{|v_i^a \cap v_i^b|}{|v_i^a \cup v_i^b|}$$

4.5 Weights

The weights were determined by a grid search. For each feature set f_i that was denoted earlier, we fitted parameters, shown the following matrix:

$$W = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ \eta \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \\ \theta \\ \eta \end{bmatrix}$$

In the grid search each time one weight will be increased. If this gives a positive result the weight will be increased even more. This will give a clear vision of the influence of the weights on the features.

The formula of similarity also has an element η. This is a certain power to the similarity score. This way high similarity scores between two users get rewarded with an extra amount. For example without η user A has a similarity score of 0.8 with user B and 0.3 with three other users. User B did job j_1 and the other three did job j_2, the score for job j_1 is then 0.8 and for job j_2 it will be 0.9. When $\eta = 2$ each individual score is returned with a power of 2, so the score for job j_1 is $0.8^2 = 0.64$ and for job j_2 $0.3^2 \times 3 = 0.27$. When the power is too high the individual scores gets to much influence. When the power is too low, the popularity of job gets more influence. A balance has to be found between these similarities.

4.6 Job score

When the similarity scores are calculated a job ranking can be made for user a. The job score is based on the similarity scores and whether other users did or did not apply on that job. The formula for computing the job score for job j is:

$$jobScore(j) = \sum_{i=1}^{k} sim(U_a, U_i) \times r_{i,j},$$

where k is the number of other users the similarity is computed over and $r_{i,j}$ is 1 if user i did apply on job j and 0 if not.

4.7 Item-based collaborative filtering

The downside of the user cased-based recommendation approach is when nobody has ever applied to a certain job, it will never be recommended. This problem can be solved with item-based collaborative filtering, as described in Aggarwal [1]. Item-based collaborative filtering uses the interest of a user in a product to find similar products , which can then be recommended. Though, Aggarwal proposed this method in the context of movie recommendations, it could potentially be used in job recommendations by finding jobs similar to jobs that a user applied to, and recommending those jobs to the user.

Item based collaborative filtering was tried in this project. To start off: only

applications in the same state were taken into consideration, as the data analysis showed users applied into the same state most of the time.

The similarity of the jobs was determined based on the description. Before the similarity between jobs can be calculated some steps have to be taken. As the description was given in html code, this was first transformed to normal text. After the text was transformed to lowercase and punctuation was removed, stop words, most common words and the most rare words were removed, because these words will add little information for calculating the similarity. Now, each description contains only words that does not make a sentence anymore.

A next step could be stemming. Stemming is rewriting a word to the stem of that word. Unfortunately this is a very time consuming step and the gain is very low, therefore stemming is skipped. From the remaining words a Bag of Words can be created. A Bag of Words is a matrix of the most common words, where the rows are the jobs and the columns are those most common words. The matrix is filled with 0's and 1's. If a job has that word in the job's description, the value is 1 and otherwise the value is 0. With an term-frequency times inverse document-frequency (TF-IDF) transformation words that occur more frequently will get a lower value than words that occur in a small fraction.

The TF-IDF is calculated by:

$$tf - idf(f_i, J_a) = tf(f_i, J_a) \times idf(f_i) = tf(f_i, J_a) \times log(N/N_i)$$

where $tf(f_i, J_a)$ is the number of times a word occurs in a job description, N is the total number of jobs and N_i the number of jobs that contains that word. With this the similarity can be calculated between jobs. The similarity is calculated with the cosine similarity. The cosine similarity is calculated between two vectors[3]. This formula is:

$$sim(J_a, J_b) = \frac{\sum_{i=1}^{m} j_i^a \times j_i^b}{\sqrt{\sum_{i=1}^{m} (j_i^a)^2} \times \sqrt{\sum_{i=1}^{m} (j_i^b)^2}}.$$

The ranking is given by iterating over the jobs a user applied to and sum the scores for the similar jobs. In the end item based collaborative filtering turned out to be very time consuming computational wise, therefore it is not used in the final model. It could be used in a future approach when there is possibly more time to find out how the computational time can be improved.

5 Verification and Validation

For verifying the model the average reciprocal hit rate (ARHR) was used. The ARHR for user u is calculated by the following function:

$$ARHR(u) = \sum_{j \in \{I_u \cap V_u\}} \frac{r_{u,j}}{v_{u,j}},$$

where $r_{u,j}$ is 1 if user u applied to job j and zero if user u did not apply, $v_{u,j}$ is the rank of job j in the recommendation list. The average ARHR is calculated by:

$$ARHR = \frac{\sum_{u=1}^{m} ARHR(u)}{m}$$

5.1 Training, validation and test set

The model will be tested on a test set of a time period of two weeks, therefore the validation set will also be the last two weeks. In these two weeks there are 59686 unique users that apply in that period. The different models will be validated on a sample of a thousand users.

5.2 Baseline

Before comparing the models first a baseline is set. The baseline is usually based on a simple model. In this case a simple popularity recommender was used. This model will recommend the most popular jobs from the city the user lives in. When there are not enough jobs in a city for making a top 100, then the first three digits of the zip code will be taken in account. If that is not enough the top 100 will be filled up with most popular jobs of that state, and otherwise country.

Results

In the grid search different weights were tried. If a feature weight had positive influence, the weight for that feature was raised. Results are shown in Table 1. It shows that every model defeats the baseline and that some of the weights do improve the score. The biggest influence on the ARHR score is the application history. The model should not only be based on application history. If it is, then new users that never did an application will never get a similarity score with other users. Against expectation giving degree type a higher weight has a negative influence on the ARHR score. Besides application history, the set major, currently employed and managed others and η have a positive effect on the ARHR score. Combining the weights that give a higher score gives an even higher score, which will be the final model.

Parameter values in iteration	
(α,β,γ,δ,θ,η)	ARHR
Baseline	0.0069
(1,1,1,1,1,1)	0.0237
(10,1,1,1,1,1)	0.0233
(1,10,1,1,1,1)	0.0236
(1,1,10,1,1,1)	0.0252
(1,1,20,1,1,1)	0.0249
(1,1,1,10,1,1)	0.0202
(1,1,1,1,10,1)	0.0279
(1,1,1,1,1,1.5)	0.0245
(1,1,1,1,100,1)	0.0351
(1,1,1,1,200,1)	0.0392
(1,1,1,1,400,1)	0.0423
(1,1,1,1,1000,1)	0.0433
(1,1,1,1,1,2)	0.0250
(1,1,10,1,1000,2)	0.0445

Table 1: Grid search results

Following from these results, the final parameters in the similarity measure, as described in section 4.4, were set as following in the DSS:

$$W = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ \eta \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 10 \\ 1 \\ 100 \\ 2 \end{bmatrix}$$

6 Conclusion

The objective of this project was to build a job recommender system which recommends vacancies to the job seeker in decreasing order of fit. The result is a DDS which is easy to use and can give a personalized top 100 of vacancies in a short time. The algorithm that makes this top 100 is based on user case-based recommendation. Although this approach has the potential to do better, the baseline, which was a popularity recommender, was beaten by a factor of 10 approximately. Therefore, it can be concluded the recommendation system is at least not useless, and might be able to help people find fitting jobs.

7 Discussion

Because of lack of time the weights of the similarity measure are not fully explored. Now only one parameter at a time was changed, maybe changing multiple parameters at a time would give even better results. Also, not that much was spent on experimenting with the grouping of the features into the proposed feature sets. It might be worth it to try and experiment with these, to see how other sets perform. Though, it could be expected that increasing the number of sets will result in larger computational times: in python it is more efficient to calculate scores for grouped features rather then single features for example. Also, it would add more weights to the similarity measure, which makes the model more complex.

The disadvantage of the model is that jobs where nobody applied to, will never be recommended. A solution for this problem could be item-based collaborative filtering. For the model in this project, it was not suitable because of computational constraints. For further research a faster algorithm could be found that finds similar jobs a users applied on, or the algorithms could be ran on a faster machine.

Based on the context, it would be expected that the job history of a user captures a lot of information about their interests in other vacancies. Though, the current model does not take the job history into account at all. As explained in the data analysis, there were too many different job titles in the dataset. It might be worth it to find solutions that make it more relevant to add these job titles to the model, for instance using clustering and/or text mining.

What can also be done for further research, is trying other similarity functions, for example the cosine similarity function. This might also have influence on the job ranking.

References

[1] Charu C Aggarwal et al. *Recommender systems*. Springer, 2016.

[2] Xingsheng Guo, Houssem Jerbi, and Michael P O'Mahony. An analysis framework for content-based job recommendation.

[3] Shubman Jain. Ultimate guide to deal with text data (using python) – for data scientists engineers.

Appendix

Data Analysis

	Applications				Jobs		
	% In state	Amount	% of total	Cum % of total	amount of Jobs	% Total Jobs	Cum % Total Jobs
TX	94.89	175848	10.97	10.97	25669	9.00	9.00
CA	94.12	129509	8.08	19.05	28087	9.85	18.86
FL	93.85	191000	11.91	30.96	18525	6.50	25.35
AZ	92.77	55006	3.43	34.39	7135	2.50	27.86
IL	91.07	119951	7.48	41.88	15296	5.37	33.22
MN	90.94	11976	0.75	42.62	5699	2.00	35.22
CO	89.13	15873	0.99	43.61	5558	1.95	37.17
MI	88.11	47299	2.95	46.56	7693	2.70	39.87
NC	86.77	59054	3.68	50.25	9361	3.28	43.15
GA	86.20	78365	4.89	55.14	8802	3.09	46.24
OH	85.56	54385	3.39	58.53	11756	4.12	50.36
NV	85.12	10951	0.68	59.21	1806	0.63	51.00
TN	84.68	28915	1.80	61.01	6654	2.33	53.33
WA	84.27	9959	0.62	61.64	5929	2.08	55.41
CT	83.97	23430	1.46	63.10	4294	1.51	56.92
NY	83.19	94445	5.89	68.99	14587	5.12	62.03
IA	82.24	7437	0.46	69.45	3155	1.11	63.14
HI	81.40	731	0.05	69.50	616	0.22	63.36
UT	81.29	3143	0.20	69.69	1920	0.67	64.03
IN	81.13	56765	3.54	73.24	6824	2.39	66.42
PA	79.32	62771	3.92	77.15	12560	4.41	70.83
LA	79.12	11717	0.73	77.88	3136	1.10	71.93
OR	78.93	3242	0.20	78.08	2652	0.93	72.86
WI	78.36	12774	0.80	78.88	6106	2.14	75.00
OK	77.51	3918	0.24	79.13	2380	0.83	75.84
AK	77.05	366	0.02	79.15	508	0.18	76.01
MA	76.52	13457	0.84	79.99	7113	2.50	78.51
NE	75.96	1506	0.09	80.08	1229	0.43	78.94
KS	75.06	20780	1.30	81.38	3522	1.24	80.18
KY	74.72	24478	1.53	82.90	4234	1.49	81.66
ID	73.82	1631	0.10	83.01	818	0.29	81.95
AL	72.94	12576	0.78	83.79	2981	1.05	82.99
SC	71.60	28181	1.76	85.55	4006	1.41	84.40
VT	70.76	472	0.03	85.58	513	0.18	84.58
VA	70.61	31371	1.96	87.53	8371	2.94	87.51
MD	69.06	51899	3.24	90.77	7599	2.67	90.18
NM	67.12	2360	0.15	90.92	1377	0.48	90.66
MS	65.07	9204	0.57	91.49	1584	0.56	91.22
MO	62.49	34320	2.14	93.63	5139	1.80	93.02
DE	62.46	10850	0.68	94.31	1411	0.49	93.52
SD	61.17	824	0.05	94.36	633	0.22	93.74
NJ	60.67	65668	4.10	98.46	9215	3.23	96.97
AR	57.25	3366	0.21	98.67	1944	0.68	97.65
NH	42.67	1788	0.11	98.78	900	0.32	97.97
RI	42.60	2087	0.13	98.91	683	0.24	98.21
ME	35.54	363	0.02	98.93	555	0.19	98.40
WV	32.06	1444	0.09	99.02	1121	0.39	98.80
DC	29.50	5833	0.36	99.39	1855	0.65	99.45
ND	22.47	178	0.01	99.40	620	0.22	99.66
MT	21.43	98	0.01	99.40	367	0.13	99.79
WY	11.79	195	0.01	99.42	255	0.09	99.88
PR	3.16	1077	0.07	99.48	72	0.03	99.91