Optimising a Recommendation Model for Career Discovery

FINAL REPORT

by

B. Beker
R. Brugsma
J.F. Offerijns

in partial fulfillment of the requirements for the degree of

Bachelor of Science
in Computer Science

at Delft University of Technology.

Supervisors:	Dr. T. Abeel	
	A. Nederlof MSc	
Project committee:	Dr. T. Abeel,	TU Delft
	Dr. H. Wang,	TU Delft
	O.W. Visser MSc,	TU Delft
	A. Nederlof MSc,	Magnet.me

Optimising a Recommendation Model for Career Discovery

B. Beker Project Member BSc student, TU Delft

R. Brugsma Project Member BSc student, TU Delft

J.F. Offerijns Project Member BSc student, TU Delft

Dr T. Abeel Academic Adviser Assistant Professor, TU Delft

A. Nederlof MSc Client Adviser CTO, Magnet.me

ABSTRACT

Recommendation systems are algorithms that aim to predict what items are preferred by a user, based on a recorded history of user activity. Magnet.me is a company which recommends companies and opportunities to students. Potential algorithms for recommendation systems are *memory-based* and *model-based collaborative filtering*, *graph-based approaches*, *support vector machines*, *random forest classifiers* and *wide & deep learning*. Based on a qualitative comparison of the algorithms, *model-based collaborative filtering*, which is what Magnet.me currently uses as well, was chosen to be the best fit. This is because it scored highly on the three most important factors for Magnet.me: potential performance, compatibility with the dataset and scalability. When comparing several well-known benchmarking metrics, the most suitable metric for testing the performance of the recommender was the *F1 measure*. To benchmark the model, the connection status of user-company relations should be used as the label, but must be excluded in the calculation of the implicit ratings. Logarithmically scaling the view counts before used as a factor in the implicit ratings has proven to be of negligible effect. Five other signals are found that could be used to improve the recommendations. Hyperparameter optimization with *cross-validation* is implemented to constantly improve the recommender. The recommender has also been succesfully been deployed into the Magnet.me technology stack. The possibilities for a clustering algorithm are considered in order to solve the cold start problem, but we could not determine the numeric distances between features, which is required for training a clustering algorithm.

1 INTRODUCTION [1]

A recommender algorithm uses explicit user ratings or user activity to recommend items to users. Magnet.me is a company with web and mobile applications, which students and recent graduates can use to discover jobs and career opportunities. They are using a recommendation engine to provide personal content—consisting of companies and opportunities—to students. Magnet.me has implemented a basic recommender that is able to provide this. The main goal of this project[2] is to benchmark the recommender and automatically optimise the hyperparameters which are used by the recommender algorithm.

1.1 State of the Art in Recommendation Systems

Recommendation systems can be implemented using a variety of techniques. Traditionally, the techniques that these systems use are separated into *collaborative filtering* and *content-based filtering*. In practice, companies often have developed custom algorithms that work best for their context. This can be done by combining or extending traditional recommendation techniques. For example, Google is applying *deep neural networks* in YouTube with *Tensorflow* (Covington, Adams, & Sargin, 2016). Furthermore, Facebook uses the proximity of two nodes in the social network graph in order to make recommendations for users.

Collaborative filtering is a recommendation algorithm that predicts ratings or preferences between a user and an item (Ekstrand, Riedl, & Konstan, 2011). It can predict explicit ratings, which are ratings that a user actively gives to an item (a five star rating for a movie for example), or implicit ratings, which are values that represent the preference of a user. The algorithm calculates these preferences or ratings for a user by finding users that are similar. It assumes that if a user is similar to other users, this user will be interested in content that these other users prefer.

In contrast with *collaborative filtering* algorithms, *content-based* algorithms make use of the attributes (features) of users or items. In most approaches, users are represented by a certain taste profile. This profile is

[1] Based on our research report, located in Appendix F.

[2] More information can be found in the project plan of Appendix D.

a vector of features representing different properties of a user (e.g. study background and interests). A company could be placed at a certain location, focuses on a certain industry or has a specific amount of employees working there. If the user profile is similar to the attributes of the company, the company might be of interest to the user.

Alternating Least Squares

The *collaborative filtering* feature inside *Spark*[3], which is used by Magnet.me, relies on matrix factorization to find the best recommendations, as visualised in Figure 1. *Alternating Least Squares* (ALS) is an implementation of a *collaborative filtering* algorithm (Takács, Tikk, 2012).

This matrix can be factorized in P and Q, of which the values are initialised to either 1 or to random values. ALS has to learn these matrices, which consist of a latent vector per user/item, by alternatingly updating the values of P and Q. In multiple iterations, it changes values in one of the two matrices and calculates the other matrix. After enough iterations, the values will converge, and the entries in the multiplication matrix $P \times Q$ should be predictions of how much a user matches with a specific company/opportunity.

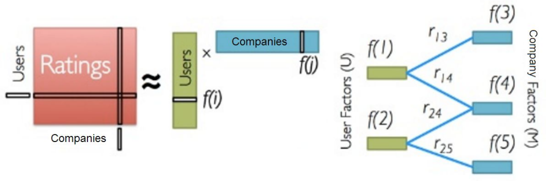


Figure 1: Example of matrix factorization calculation (source: Databricks)

In regards to ALS in *Apache Spark*, four hyperparameters (three if only using explicit data) can be configured to improve the resulting predictions. Additionally, one parameter can be passed to specify the number of blocks used for parallelization. Typical for the ALS algorithm is that it alternatingly changes one of the two factorized matrices. The optimization of the latent vectors could also be done using other algorithms like *Bayesian networks* or *probabilistic latent semantic analysis* (Su, Khoshgoftaar, 2009), but these algorithms are not included in *Spark*'s machine learning library.

Strengths and Weaknesses of Collaborative Filtering for Magnet.me

The user *collaborative filtering* algorithm has several benefits for Magnet.me. With *collaborative filtering*, companies and opportunities are recommended by analyzing the behaviour of users. Because of this, items that are not similar to each other can be recommended to a user because a similar user have viewed these items.

In a way, Magnet.me also makes use of *content-based filtering*, because it shows the most popular companies within the industry of the student. However, it is important that not only the popular companies are recommended, because letting the users explore companies that they are not familiar with is one of the goals of Magnet.me.

Another benefit of using *collaborative filtering* is that a *Spark* implementation exists. Not only does this save us considerable time to implement the actual algorithm ourselves, it also means that the *collaborative filtering* algorithm can be distributed. Because of this, the algorithm's performance remains manageable if the amount of data increases drastically.

A disadvantage of the current system is that no personal recommendations can be given in the first hour, which is known as the cold start problem. Because of this the model has to be re-trained each hour to provide recommendations for new users.

Alternative Techniques

Below we have included a list of alternative techniques that can be used for recommending companies and opportunities to students. Note that this list is not exhaustive, but includes a subset of all machine learning algorithms that we found to be the most likely candidates for this particular project.

Memory-based Collaborative Filtering

Besides the *model-based collaborative filtering* approach that is described above, a *memory-based* approach can be used where the similarity between users is calculated by similarity functions like cosine similarity or the Pearson correlation coefficient. A weighted average of events of the most similar users is determined to provide a recommendation. Until recently, *memory-based* approaches did not scale well with larger datasets, but this has also been fixed in *Spark* by the implementation of *Dimension Independent Matrix Square using MapReduce* (DIMSUM) (Zadeh, Carlsson, 2014).

Graph Analysis

Graphs can be used in certain different ways to improve recommendation accuracy. One example of graph-based filtering is *GraphRec*, a system which converts the user-item matrix to a weighted adjacency matrix, which is used to create a graph from which the entropy of all items can be calculated (Lee, Lee, 2015).

[3] Website of Apache Spark

For example, *GraphRec* has been used to recommend music artists using explicit user data. This algorithm tends more towards a content-based approach, because the items are centralized and relations are being made between items, based on their attributes. Another *graph-based* algorithm is found where users are nodes and users are connected to item-attributes via the items.

Support Vector Machines

Support Vector Machines (SVM) is a model that classifies its input into a binary output. It does so by defining an hyperplane between vectors (which are vectors with features of a user or company/opportunity). Vectors on one side of the hyperplane are items or users that are similar to the input vector of the SVM. In the Magnet.me case, input could be a vector with features of a user. The SVM will determine what class (a collection of opportunities and organizations) the user belongs, which is basically equivalent of determining what opportunities or organizations could be recommended to that user. This way, SVM can be used as a recommendation system (Gershman et al., 2011).

Unfortunately, research has shown that SVMs perform poorly on recommendation problems. This is because there is only a small percentage of users interested in a particular item/company, leading to extremely unbalanced class distributions (Zhang, Iyengar, 2002).

Random Forests

A *random forest classifier* is a model that uses several decision trees to classify its input (Ajesh, Nair, & Ps, 2016). These decision trees base their decision on training data that has been served. In the Magnet.me case, input of the random forest model could be a vector representing the user profile (in a model based implementation) (Zhang, Min, & He, 2014). Every decision tree determines what recommendation (company or opportunity) comes out of their tree. An average of all output of decision trees will be calculated to determine what recommendation suits that user best.

A disadvantage of *random forest classifiers* is that streaming data input is a problem for its model. When new training data is available, a *random forest* needs to be recreated and trained (Saffari et al., 2008). In Magnet.me data, new example data arrives all the time (new companies and new opportunities), which means that without recreating the forest constantly (which takes a lot of computation time), the system would give questionable recommendations.

Wide and Deep Learning

Neural networks are fairly new in the context of recommendation systems. *Deep neural networks* have more hidden layers in the network (mostly more than ten) in comparison with “normal” neural networks. In a recommendation case, input for the neural network would be several features of a user profile and the output would be a classification of for example companies.

Recently, *wide and deep neural network learning* has been introduced in the Google Play Store (Cheng et al., 2016). Wide and deep learning means that a network has two components, one wide (which is in this paper is a *generalized linear model*) and one deep (a *deep neural networ*k) component. It is stated that the recommendations of the Play Store improved significantly.

1.2 State of the Art in Benchmarking Metrics for Recommendation Models

There are a variety of metrics which are used in benchmarking recommendation systems. Choosing what metric should be used to benchmark a recommender, depends on the context in which the recommender operates.

For recommenders that base their recommendations on explicit data, the *root mean squared error* (RMSE) is often used (Cremonesi et al., 2015). This metric calculates how much the explicit ratings that are predicted differ from the actual ratings. When a recommender uses implicit data however, the RMSE might not be the proper metric. *Precision* and *recall*[4] are metrics that also give insight in the performance of the recommender. These values, together with the *F1* score (Lipton, Zachary C Elkan, Naryanaswamy, 2014), can be used on implicit data. These metrics use *true positives*, *false positives* and *false negatives* to determine the performance.

The *normalised discounted cumulative gain* (NDCG) is a metric that evaluates the ranking of the predicted items (Wang, He, Chen, 2013). While this metric is not often used in recommender problems, it is used in benchmarking web search engines to determine whether the most important search results are ranked on top.

[4] These metrics usually are presented together with the threshold they operate, for example *precision*@4 or *recall*@4, meaning *precision* for the first four recommendations.

1.3 Magnet.me Implementation

Magnet.me currently uses the *collaborative filtering* (CF) feature in the Machine Learning library of the *Apache Spark* framework to provide personal recommendations, because this system was thought to be the best for the Magnet.me context. One of the characteristics of Magnet.me is that lots of opportunities are added to the database constantly (about ---[5] opportunities on average per day over the last year[6]). At moment of writing, they have --- users, --- companies and --- opportunities (of which --- are active) in their database[7].

Typical for their users is the short time they are browsing through content. On average, as shown in Figure 2, only a small percentage will return after a few months, because he or she usually stops searching after finding a suitable job.

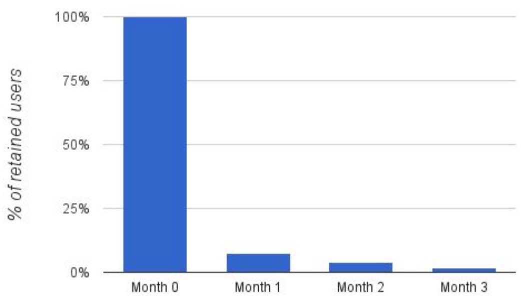


Figure 2: Retention rate of users of Magnet.me on a monthly basis on the web application (source: Google Analytics)

Apart from using *collaborative filtering* for giving personal recommendations, similar and popular recommendations are given. Each hour, the user-based *collaborative filtering* algorithm is re-trained, which takes 16 minutes on average[8], and personal recommendations will be given. To make the best use of the product, users need to receive recommendations as soon as possible, while the system has little time to learn user preferences.

The idea of user-based *collaborative filtering* algorithm is that the behaviour of past users is used to make recommendations for new users (Ekstrand, Riedl, & Konstan, 2011). In the context of Magnet.me, this is done by collecting events that users produce. Currently, the recommender gives recommendations based on data consisting of three different types of user actions:

- User views a company recommendation (positive);
- User views an opportunity recommendation (positive);
- User is connected with a company (strong positive).

To obtain data about the relation between user and content, explicit or implicit data can be gathered. Explicit data are ratings that a user actively gives to a specific item, for example giving a five star rating to a movie. Implicit data are ratings that are inferred based on user actions, for example views or clicks. The recommendations of the current algorithm are based solely on implicit user data.

It is important to note that this means an extra step in the computation of the predictions is required. The implicit data of a user has to be converted to a confidence matrix where all entries contain confidence values. (Hu, Koren, & Volinsky, 2008) The value 1 is assigned to the two click actions. A value of 10 is assigned to the connect action, because this is considered a more significant relation between a user and a company than a click action. Several million events have been collected in the past. When a user produces several events, the *collaborative filtering* compares him or her to users that are similar and gives recommendations accordingly.

1.4 Research Questions

The goal of this project is to improve and tweak the recommender that is currently in use by benchmarking the recommender and automatically optimise the hyperparameters of the recommender. In order to achieve an optimal solution for Magnet.me, we will answer the following questions:

(1) Which algorithm can be implemented, in order to provide the best recommendations of companies and opportunities to students?

(2) How can we improve the recommendations that are given?

(3) How can we set up a system that automatically optimizes the model hyperparameters at a regular

[5] --- confidential

[6] We calculated this by counting the number of opportunities added daily to the Magnet.me database and then taking the average.

[7] We have determined these numbers by querying the current Magnet.me database.

[8] We determined this by running a database query to determine the average time of each *Spark* job over the last year, after which we summed the averages of each of the 4 types of jobs.

basis that can be integrated into the Magnet.me technology stack?

(4) Can we tackle the cold start problem using clustering, in order to give recommendations to new users?

Answers to these research questions are given in several sections and are summarized in the conclusion.

2 MATERIALS & METHODS [9]

2.1 Personal Recommendations

Scoring Function

In order to train a recommendation model, a function is needed to calculate the implicit scores for each user-company relation. These scores need to represent the relations between users and companies. In the current implementation, the score for a user-company relation is equal to the amount of clicks of the user on the page of that company, plus ten points if the user is connected to that company. The weight of ten that is assigned to a user-company connection is chosen, has not proven to be the optimal value.

We have started with the same scoring function, but with a variable connection-status weight. Later on, when the evaluation metrics are fully implemented, the connection-status weight is adjusted to a value that gives better recommender performance.

Instead of simply taking the view count as part of the scoring function, the log of the view count can be used. Using the log of the view count is expected to give a more realistic score, because this has the effect that the difference between one or two views will give a bigger change in the score than the difference between 30 and 31 views.

Using the scoring function as mentioned, the values for each user-company relation can vary from zero to very high. Due to the lack of an upper limit, it is hard to compare the recommender performance with other models and with the same model with different settings. The scores will be scaled, so that all values lay between 0 and 1. This way, modifications to the model can easily be compared and evaluated. Also, some metrics expect scaled values in order to evaluate the performance.

[9] An evaluation of our software development practices can be found in Appendix B.

Model settings

The model performance will be evaluated with different values for the connection-status weight and using the log of the view count turned on and off. These model settings will be referred to as model parameters in the rest of the report.

2.2 Benchmarking

To eventually decide what the best configuration is for the recommender, specifically choosing what model parameters and hyperparameters are ideal, various benchmarks have been implemented.

Implicit Ratings

Before getting into the actual benchmarking of the recommender, it should be emphasised that the Magnet.me recommender makes use of implicit data over explicit. The ALS implementation of *Spark* has an option in which you can train implicit data. What this means is that the predictions that are provided by the trained ALS model are not explicit ratings given to items, but is a confidence (preference) value given to a user and an item. This value is a value between -1 and 1, with a value near larger than zero meaning a preference towards that item, and less than zero not a preference towards that item. Due to this, benchmarking and comparing the recommender models has been a lot more cumbersome than benchmarking explicit ratings, because a preference is not easy to compare with implicit ratings calculated by the scoring function.

The solution to this problem was not to compare the ratings that are predicted to the implicit ratings that are calculated by the scoring functions, but to compare the ratings with labels that items have. The correctness of a recommendation means that if the model predicts that a user has a preference towards a item (confidence value larger than 0) and it appears that the user has a positive label, the prediction is true positive.

To determine a label for a company recommendation, the connection status is used. When a user is connected to a company, the label is positive. For an opportunity recommendation the label is used whether a user has saved the opportunity or has applied to that opportunity.

Metrics

Some ranking metrics that are commonly used in recommender systems are *precision*, *recall*, F1, RMSE and NDCG. An implementation of these metrics is available in Spark and in an external library[10]. To get a better understanding of the metric functions, we decided to implement the ranking metrics ourselves.

When implementing the RMSE, it was noticed that this metric might not be applicable to our situation because of the use of implicit data and our binary labels. Moreover, RMSE is calculated to every rating, while these ratings might not be that important because only a specific amount of recommendations are shown to the user.

What we actually want to benchmark is that the first top K recommendations ordered by highest rating are correct. This means that a certain threshold must be provided on how many recommendations we want to test. In the applications of Magnet.me, at most six recommendations are shown initially, which is why we chose to use this as a threshold.

Monitoring

Magnet.me is already tracking clicks on recommendations and are currently implementing a system which can track the *click-through rate* (CTR) of recommendations. In the future, we can use this to see what the improvement of our model is in terms of the live CTR in the Magnet.me apps.

2.3 Hyperparameter Optimisation

At the moment, the parameters that are used with the *collaborative filtering* implementation have been predefined and have not been updated in the last year. However, the optimal parameters to achieve the best results can very well change over time, due to new incoming data that changes the underlying structure of the dataset.

Methods exist that can automatically determine the optimal set of parameters to use when training the model. The main approach is to run a parameter sweep. Quite simply, you define minimum and maximum values for each parameter and it will train the model with all possible combinations of parameters. To determine the best set of parameters, a loss function is defined which calculates the error rate of the model. A basic implementation works like this:

(1) Create separate sets of training and test datasets, which is often a 70-30 or 80-20 split;
(2) For each set of parameters:
    a. Train the recommendation model using the training data;
    b. Evaluate the performance using the test data;
(3) Choose the best set of parameters with the lowest error rate.

10 [Github page of spark-ranking-metrics]

Cross Validation

Running hyperparameter optimization on the same training and test dataset can be prone to overfitting, as the lowest error rate can easily be a model that is very specific to this training dataset. Therefore, *cross-validation* is often introduced in order to avoid overfitting the training data. For example, in *k-fold cross validation*, the entire dataset is split into k subsets, an example of which is shown in Figure 3. One subset is then used as the test dataset, while the other k-1 datasets are used as training data. The process then becomes:

1. Create k subsets of the entire dataset;
2. For each set of parameters:
    a. Repeat k times:
        i. Define one of the subsets as the test data;
        ii. Define the other k-1 subsets as the training data;
        iii. Train the recommendation model using the training data;
        iv. Evaluate the performance using the test data;
    b. Take the average of the error rates of each subset;
3. Choose the best set of parameters with the lowest average error rate.

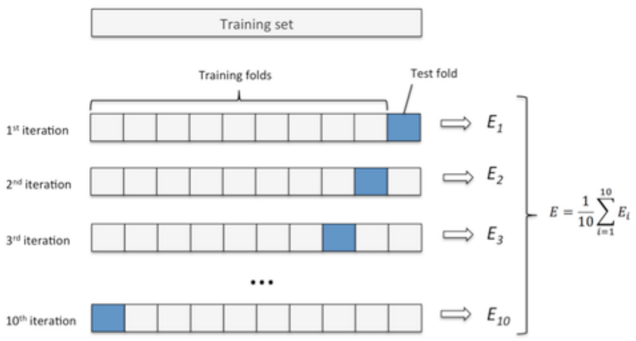


Figure 3: Example of a k-fold cross validation with k=10

ALS Parameter Description

Spark's ALS train implementation has four parameters: *lambda*, *iterations*, *rank* and *alpha*.

- *Iterations* specific the maximum number of times that ALS calculates the error rates of the latent matrices and updates them accordingly. If this parameter is set too small, a chance exist that the

ALS-algorithm is not yet done converging the values in the factor matrices. If it is set too big, the values will coverge in the first iterations. A too big value will never make the performance worse, but the last iterations are unnecessary because the predictions will not be improved substantially.

- The *rank* parameter specifies the number of latent factors of the ALS matrix factorization.
- *Lambda* is a factor used in the regularization formula of ALS.
- *Alpha* is a hyperparameter that is only used when the recommender uses implicit data. If the value is higher than it will give higher value to observed events in comparison with unobserved events.

Determining Hyperparameter Ranges

Now that a metric function is chosen in order to evaluate the recommender performance, we need to determine the range of hyperparameters for which we test the model performance. The range of values for which the hyperparameter optimisation job must be performed could be different for companies and opportunities, because the optimal hyperparameters are probably not the same for recommending companies than for recommending opportunities.

The optimal hyperparameters can vary widely for different recommenders, due to differences in amount of data, data sparsity, and the distribution of the implicit scores. No default starting values are available, except *Spark* mentioning that a value of 20 or less for the iterations parameter should be high enough in most cases.

We started with running the hyperparameter optimization with a very big range of values for each hyperparameters, and looking at which values have almost no impact on the metric score. The ranges can be narrowed by eliminating the values that result in a low NDCG score. The remaining ranges can now be tested in more detail.

2.4 Infrastructure & Deployment

Training a recommendation model with ALS is a task that requires significant resources. To accomplish this, you need to think where to store the data where *Spark* can read quickly. Also because of the fact that *Spark* will run faster if it has access to more memory, processing power and so on, finding an appropriate amount of server capacity is mandatory. First a description is giving how we have set up our development environment and after that a description is giving how we deploy our recommender implementation in the Magnet.me environment.

Development Infrastructure

Google Cloud & Kubernetes

It will be a very difficult task to handle large amounts of data locally during development. Magnet.me uses the cloud servers on the *TransIP*[11] network to handle their recommender jobs. Although this is a reliable service, we have used the *Google Cloud Platform* to handle the jobs. The reason for this is that Google provided a trial period in which using the platform is free of charge for the first two months. Moreover, we have more experience with this cloud platform.

This project relies heavily on *Spark* and *Elasticsearch (ES)*[12]. To manage these applications thoroughly and to distribute resources accordingly between these applications, an open-source system called *Kubernetes*[13] is used. The purpose of this system is to easily manage *Docker*[14] containers. *Kubernetes* requires *YAML*[15] files consisting of references to *Docker* images in repositories and container configuration. Also within these files references to persistent volumes need to be provided. Persistent volumes are basically containers that are linked to *Google Persistent Disks*. This way, data can be persisted on these disks.

Spark master and workers

Because of the fact that *Spark* makes a significant use of resources and because a single *Spark* job (a hyperparameter sweep for example) can last a long time when having relatively few resources, it would be wise to run *Spark* on a resourceful server. We have set up one Spark-master controller and several workers to make the tasks as efficient as possible. While setting up these controllers is easy to do using *Kubernetes*, sending a *Spark* job to the cluster seemed to be quite cumbersome. The lack of documentation and the instability of *Spark 2.0* (relatively new version) was the cause of this.

[11] TransIP

[12] Elasticsearch

[13] Kubernetes

[14] Docker

[15] YAML

Sending jobs to the Spark cluster

To send a job to a *Spark* cluster, one must assemble a *jar* [16] file that will be sent to the cluster. *Spark-submit* is an application provided with the *Spark* framework that has the purpose to send tasks to a cluster. Provided with an IP-address, a reference to a jar file and some configuration options, it will initiate a job on the Spark master controller that is running on the provided IP-address. The master controller will then again allocate resources to its workers with which the job will be done. The main problem is that the *Spark-submit* application will not handle the transfer of the *jar* file to the servers. What this practically means is that the *jar* file needs to be available on every worker. To accompany this, we have written a shell script that assembles the *jar* and deploys this *jar* on every worker.

Elasticsearch

Magnet.me uses *Elasticsearch* to persist their data. To deploy our recommender implementation in the Magnet.me stack, we had to set up and integrate with Elasticsearch. Again, setting up an *Elasticsearch* cluster is easily done using *Kubernetes*. This cluster is only accessible through an internal IP-address given by *Kubernetes*. *Spark* provides a library which handles reading and writing to *Elasticsearch*.

Production Infrastructure

To use the recommender in production, the recommender needs to be integrated within the Magnet.me build process. Magnet.me uses *Maven*[17] to build their Java application. Fortunately, there is a Maven plugin which also builds application that are written in Scala. Magnet.me uses *Jenkins*[18] as their build management tool. By setting up a hook on our repository to *Jenkins*, it will automatically picks up changes and build the application. It will also produce a .jar file which then can be sent to the Magnet.me *Spark* cluster where a specific *Spark* job can be run.

2.5 Data Analysis

The relevant data of Magnet.me that we stored on our *Elasticsearch* cluster, is divided in multiple indices[19]. One index with crucial data for the recommender is the index of all click events of the users. A selection of the information stored in a click event is:

1. User information;
2. Time of event;
3. Information about the visited company/opportunity.

Another index that contains essential data is the *student-organization* index, with all information about the companies and opportunities. From this index information about the connection status between a user and a company can be derived.

The indices also contain data which is not needed for the recommender but is needed for other goals of Magnet.me, e.g. contact-info or employees. The full schemas of the user event data can be found in the appendix.

Comparing models

We would like to get more insight in how changes to the model parameters affect the recommender performance. The scoring of implicit data for example can change the outcomes of the recommender. Three jobs are implemented for model comparison.

Compare recommender models job trains and evaluates the recommender on multiple thresholds for precision, recall and F1. This is done for the parameters of the Magnet.me implementation of the recommender and other parameters to compare results. *Compare model parameters jobs* can be used to train and evaluate recommender for a given set of model parameters. The results than can be visualized in graphs, that can be created using the *Data analysis job*.

3 RESULTS

3.1 Alternating Least Squares is the best recommendation algorithm for this project.

To decide what algorithm is used for implementing the recommendation system, certain factors are described. Using these factors, we have determined a rating for the recommendation system approaches and with that the best option for Magnet.me was chosen.

- *Potential Performance:* Clearly, the most important factor in deciding the best algorithm is its accuracy at providing recommendations of companies and opportunities to users. It should also aim to not just

[16] Explanation of JAR files

[17] Apache Maven

[18] Jenkins

[19] An overview of the exact indices and their sizes can be found in Appendix C.

provide the most popular companies as recommendations each time, but diversify its recommendations.

- *Compatibility with Dataset:* This factor determines if the algorithm is applicable on the Magnet.me dataset. Some algorithms work better with different types of data, so to determine how compatible a technique is with the data is important.

- *Scalability:* As Magnet.me is a growing company with new users and items each day, the algorithm needs to scale well. A very important factor deciding if an algorithm is scalable is if it can be used on a distributed system. *Spark* is a system which is mostly used to handle algorithms this way.

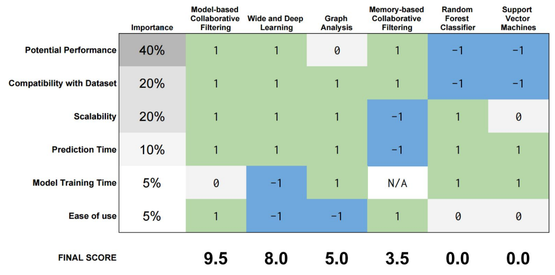
- *Prediction Time:* This factor decides how fast the calculation speed is in recommending an item to a user.

- *Model Training Time:* This factor decides what the speed is in training the model, if applicable.

- *Ease of Use:* This factor determines how easy it would be to implement and deploy the algorithm.

<ins>Algorithm Comparison</ins>

We have used the decision factors to evaluate the various algorithms that were listed in the previous section. By rating each algorithm on a scale of negative-neutral-positive, we have created a simple overview of the advantages and disadvantages of each algorithm, as can be seen in Figure 4 below. Furthermore, we have added a percentage denoting the importance of each factor for this project. Obviously, both the importance percentages and the algorithm ratings are guesses and not exact numbers. The main goal is to provide some insight into our reasoning and enable a relative ranking of the algorithms.



	Importance	Model-based Collaborative Filtering	Wide and Deep Learning	Graph Analysis	Memory-based Collaborative Filtering	Random Forest Classifier	Support Vector Machines
Potential Performance	40%	1	1	0	1	-1	-1
Compatibility with Dataset	20%	1	1	1	1	-1	-1
Scalability	20%	1	1	1	-1	1	0
Prediction Time	10%	1	1	1	-1	1	1
Model Training Time	5%	0	-1	1	N/A	1	1
Ease of use	5%	1	-1	-1	1	0	0
FINAL SCORE		**9.5**	**8.0**	**5.0**	**3.5**	**0.0**	**0.0**

Figure 4: Algorithm comparison using weighted importance factors (determined by us)

REFLECTION

These results are somewhat subjective. To give an objective evaluation of these recommenders, the factors should ideally be quantified and thoroughly tested with suitable metrics.

<ins>Comparison Results</ins>

When we take the importance of the decision factors in consideration, we can conclude that the current implementation, *model-based collaborative filtering* (CF), scores best overall. This algorithm scores well for scalability because *Spark* makes use of *Alternating Least Squares* (ALS) for the model training, which can be run distributed. *Model-based collaborative filtering* is well-known for its diverse recommendations, which is not true for *random forest classifiers* and *support vector machines*. As for the *graph-based* approaches, no accurate guessing could be made regarding to the performance.

The disadvantage of the cold start problem does still exist with the *collaborative filtering* algorithm. In order to tackle this, we tried a clustering algorithm to give predictions for the first hour, when little information of a new user is available. The results of this clustering algorithm are discussed in section 3.5.

The other approach that seems promising is *wide and deep learning*. The main disadvantage for *wide and deep learning* is that it is a relatively new and complex method. However, it has proven to be accurate at providing recommendations in the Google Play Store.

3.2 F1@6 is the optimal metric for this use case.

When benchmarking our recommender with the RMSE, inconsistent values appeared. In Figure 5, the RMSE is shown comparing four recommender models based on how much training data they have received.

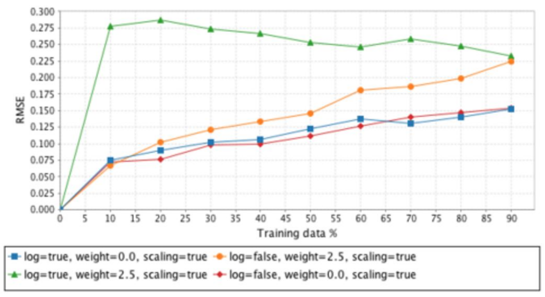


Figure 5: RMSE scores of recommender models per % of trainings data

The results imply that training the model with a higher amount of training data, would result in worse recommendations. The reason for this inconsistent behaviour is that when benchmarking the recommender, the benchmarking compares the labels of companies or opportunities with the preference of a user. Because of this, the error (which is the difference of the preference and the label in this case) would be an incorrect representation of performance of the recommender.

Normalised Discounted Cumulative Gain (NDCG) is another metric that we have implemented. This takes into account the order in which the recommendations appear. Based on discussions with the client, it appeared that the first six recommendations should only be used to test the performance of the recommender, as these are the ones that are shown prominently in the application, and the first of these six is just as important as the last of these six.

After having decided that the first six recommendations are the most important recommendations, *precision*, *recall* and the F1 score have been implemented. At first, to determine the *precision* and *recall*, an external library [20] has been used. In Figure 6, the *precision*, *recall* and F1 scores are given with their given threshold. While the *precision* curve is behaving as expected, the *recall* curve is not behaving properly. To get better insight in these metrics, a custom implementation has been made.

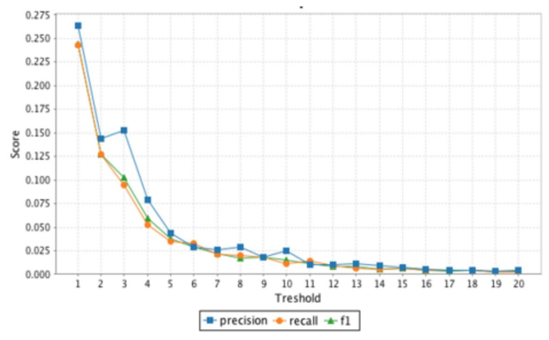


Figure 6: Faulty benchmark scores over certain thresholds

While implementing these metrics, we noticed that determining true positives, false positives, false negatives (which are parameters for these metrics) was hard to do. The reason is that there is no real rule that decides whether a company or opportunity is a recommendation (true positive) or not a recommendation (false positive). In the standard *Spark* functions the rule is arbitrary. In these functions it is checked if the rating, which can be any value and is not within a certain range, is higher than 0.5 to determine if an item is a recommendation[21]. Why this value makes the cut-off, is not documented.

To check whether the first six recommendations are correct, it is tested whether the user is connected to the company (connection status is used as a label). This way, true positives, false positives and true negatives are easily defined. And using these values as parameters for *recall*@6 and *precision*@6, a metric for benchmarking the recommender has been found. The F1@6 score has the *recall*@6 and *precision*@6 as parameters, which makes it a single representing value for measuring the performance of the recommender.

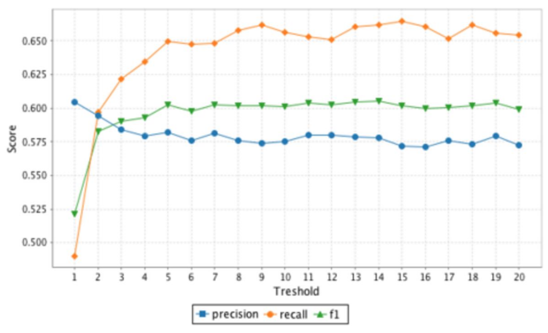


Figure 7: F1-measure per threshold

In Figure 7, the values of *precision*, *recall* and F1 at various threshold are given. Note that because of the relatively small dataset, the *precision* curve converges at ~0.575. More specifically, the amount of test data per user is low so the amount of predictions that the model can give based on the test data is low. When there is more data, the *precision* should converge at a lower value.

3.3 Connection status should not be used as a feature, but as the label of the recommender during benchmarking.

In the current implementation of the Magnet.me recommender, the connection status of a user and a company is taken into account when calculating the implicit score (an extra value of 10 is added to the score if a user is connected to a company). During benchmarking of the recommender, it uses the connection status to test if a recommendation is correct or not. Because of this, incorporating the connection status in the scoring function will mean that the recommender needs to know the connection status in order to predict the connection status (a self-fulfilling prophecy).

[20] Github page of spark-ranking-metrics

[21] Code reference in Spark

When benchmarking a recommender, labels are needed to test the correctness of recommendations. In the current Magnet.me implementation there was no benchmarking implemented and because of that no clear labels were defined for good recommendations. In this project we have thought of the connection status as a label, but other signals as labels can also be considered. In the opportunity recommender case, other labels need to be thought of. Whether a user has applied to an opportunity or a user has saved an opportunity can be used as a label.

3.4 View counts should not be scaled logarithmically to discount large values and other signals should be considered.

In some cases, user-company relations have very high view counts, numbering in the hundreds. Our hypothesis was that this could skew the results, as a single view will be discounted in comparison to the large view counts. Therefore, we implemented a scaling function to take the logarithm (with base 10) of the view count as the score.

However, when comparing the recommender model with this scaling function enabled and without, we can see that it makes almost no difference in Figure 8. While the training dataset grows, the F1 measure grows nearly at the same rate in both models.

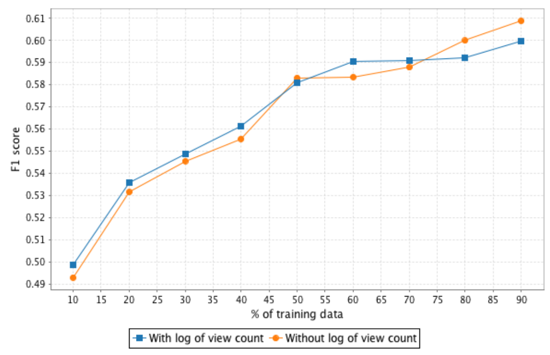


Figure 8: F1-measure at a growing % of training data

REFLECTION

In hindsight, we should have run this analysis with *cross-validation* and showed in Figure 8 not only the mean value, but also the standard deviation.

Furthermore, the current model is based only on this view count as a feature. Seeing as a high view count is not a good predictor of connection status, more signals should be taken into account. Several signals that could be implemented in the future:

(1) When a student connects with a company, but later decides to disconnect, this is interpreted to be the same as never connecting in the first place. However, we would argue that this is a fairly strong signal that the user is not interested anymore in this particular company.

(2) Recruiters can visit the profile of a user, which can be a strong indication that the company of the recruiter could be a good fit for this user.

(3) Recruiters can invite a user to apply for a opportunity as part of a recruitment campaign. This is a strong signal that the company has deemed this user to be a good fit. How a user responds (dismiss invitation; apply for the opportunity etc.) to this invitation can be a good signal as well.

(4) Users can send and receive messages to and from recruiters, which is a strong signal that the user is interested in the company of the recruiter.

(5) Lastly, the applications currently only track click (or tap, on mobile) events. Another type of data that could be tracked is user focus: how long and how actively are users looking at a particular company profile or card in a list? Another event that could be checked if a certain recommendations enters the viewport of a certain user.

It could also be worth exploring whether a time bias should be included in the recommendation system. If user A applied for a company three years ago, while user B applied for a company last month, are these signals of the same strength? Interests of students can change over time, which makes older recommendations irrelevant. Most likely, there should be some level of regularization to account for the time that has passed since the signal occurred.

3.5 Clustering cannot be used in this context for solving the cold start problem.

Our initial idea for solving the cold start problem was to cluster new users with similar users. By recommending to the new user items that are recommend to similar users, it could be away to solve the cold start problem.

First, we have implemented a *k-means clustering* algorithm. However, we quickly replaced this algorithm with an approximate *nearest neighbours approach*, because the main disadvantage of *k-means clustering* is that there is no well-founded way of determining the proper *k* (Hamerly, Elkan, 2004).

By inspecting the results, the algorithm appeared not to work properly, as users were not being clustered with similar users. The reason for this is that the features that users have need to be of numeric. The standard way by converting string value to numeric values is that the most common string value will get a value of 1 and so on.

For example, when using the study background as a feature, the most common study backgrounds will be clustered together, while this is most definitely not a good indicator of discipline similarity. Because of this undesired behaviour, another way had to be found. This resulted in another problem as it is not clear how to give numeric values to certain features. It is hard to give a value to the difference between two study disciplines, for example the difference between law and medicine.

4 CONCLUSION

4.1 Recommender algorithm

↳ RESEARCH QUESTION 1

To determine the best recommendation system, we have analysed and compared the most commonly used algorithms and we can conclude that the current approach, *model-based collaborative filtering*, is the best recommender for the Magnet.me context. This has been decided by evaluating each approach and comparing them using qualitative metrics. *Model-based collaborative filtering* scored well for the three most important factors for Magnet.me: potential performance, compatibility with dataset and scalability.

4.2 Model improvements

↳ RESEARCH QUESTION 2

To decide how we can improve the recommendations given by the recommender, we have implemented a way to benchmark the implemented recommenders. To do this, we have decided that the F1 score is a metric that represents the performance of a recommender in the Magnet.me case. The configuration (model parameters and hyperparameters) that scores the highest F1 score, should be used to give the optimal recommendations.

The F1 score varies dependent on the threshold for which is evaluated. This means that if the presentation of the recommendations on the Magnet.me application changes, the threshold must be changed as well when benchmarking the recommender. This should be taken into account when Magnet.me decides to show a different amount of recommendations.

4.3 Automatic Hyperparameter Optimisation

↳ RESEARCH QUESTION 3

To figure out how to set up an automatic process of hyperparameter optimisation, we first determined the correct parameter ranges for which the model performance must be evaluated and then set up a combination of *k-fold cross validation* and benchmarking to fully automate the steps.

We applied *cross-validation* in order to prevent the model being overfit to the training data. When this is omitted, benchmarking could give a good evaluation result while in reality the model could give suboptimal recommendations when new data comes available.

The parameter tuning job, has succesfully been deployed into the Magnet.me technology stack. This job will be automated by Magnet.me using *Jenkins*, so that the parameters stay optimized every month.

4.4 Student Clustering

↳ RESEARCH QUESTION 4

To attempt to solve the cold start problem, we implemented both *k-means clustering* and *nearest neighbour clustering*. However, in both cases we came across the same problem: in order to create a set of numeric features from the user profiles, we need to convert study backgrounds into numbers. The algorithms assume that similar numbers imply similar study backgrounds, thus we needed to determine how similar study backgrounds are to each other, which was not possible.

4.5 Future Improvements

Wide and deep learning seemed promising, but it has the disadvantage that it is a relatively new and complex method. However, our hypothesis is that it could work quite well. When the amount of users has grown substantially in the next few years, we suggest Magnet.me to consider a *deep and wide learning* approach.

For *graph-based* recommendations, it was hard to give a prediction of the performance as this is a relatively new approach. However, this approach scored well for most other factors. When more research is done for this approach, and an effective distributed implementation comes available, it could also be a good solution for Magnet.me.

To improve the recommender even further, time should be spent on how the implicit ratings are calculated. While our implementation only handles view events to calculate the score, more signals could be used to improve the recommendations. Moreover, decisions on how to label items (companies and opportunities) need to be made to make a properly functioning benchmark. What good recommendations actually are is hard to determine in the current system. Without that information it will be hard to evaluate the recommender.

5 REFLECTION [22]

We encountered several problems during the process. This started when linking *Elasticsearch* to our *Spark* cluster. We had to set up our development infrastructure ourselves, which we decided to do using the *Google Cloud* services. While it was relatively easy to setup *Spark* and *Elasticsearch*, a considerable amount of time was spent trying to let them interact with each other. Also actually sending and running Spark jobs required a lot of tweaking of server configurations. While we had hoped to have finished this in week two, we actually completed it in week six.

Because of the fact that *Google Cloud* services is not a free service, we have made use of a trial account. The problem with this account became apparent when we were blocked from the services because we have reached the limit in the use of resources. Because of this we had to create a new trial account and we had to set up everything again.

We also spent more time than we planned to implement the benchmarking metric. We planned to spend about a week to set this up, but it ended up taking us about three weeks. At first we thought that making use of implementations of *Spark* and an open-source alternative would be sufficient. But because of faulty behaviour of these methods, we ended up implementing these metrics ourselves.

[22] A detailed timeline of our work can be found in Appendix A.

Debugging the metrics was very time consuming, mainly because of the fact that it was very hard to determine what values were expected. Deciding whether the faulty behaviour arises from a faulty implementation, faulty structured data, or *Spark* bugs was hard to determine. Moreover, benchmarking the recommender takes a long time, so analysing improved implementations took a long time.

6 ETHICAL ANALYSIS

To make a recommendation system work, sufficient data is needed for training the system. Moreover, this project makes an extensive use of implicit data, which tracks user behaviour in the form of clicks and views. Based on this behaviour, recommendations are given. This data should be handled carefully as a user profile can be made from this data. This potential sensitive data can be leaked or misused, meaning that it could be sold to companies. Magnet.me has strict security requirements to tackle this problem.

Recommendations are based on how much certain items are “consumed”. This means that companies in the Magnet.me case that are not consumed much, like new companies, will not be recommended to users. Magnet.me solves this problem by showing new companies to users as well. Because of this, new companies will be consume as well, even though they might not be recommended at first.

When implementing a clustering algorithm for clustering users, it needs to be noted that specific demographics can be formed. It might be possible that male users will be clustered together, or female users. Some opportunities that are recommended to users in these clusters might be discriminating in nature.

REFERENCES

Ajesh, A., Nair, J., & Ps, J. (2016). A Random Forest Approach for Rating–based Recommender System. *Intl. Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 1293–1297.

Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., … Inc, G. (n.d.). Wide & Deep Learning for Recommender Systems.

Covington, P., Adams, J., & Sargin, E. (n.d.). Deep Neural Networks for YouTube Recommendations. http://doi.org/10.1145/2959100.2959190

Cremonesi, P., Milano, P., Cremonesi, P., Milano, P., Koren, Y., & Turrin, R. (2015). Performance of recommender algorithms on top-N recommendation tasks Performance of Recommender Algorithms on Top-N Recommendation Tasks, (September). http://doi.org/10.1145/1864708.1864721

Ekstrand, M. D., Riedl, J. T., & Konstan, J. A. (2011). Collaborative Filtering Recommender Systems. *Human–Computer Interaction*, *4*(2), 81–173. http://doi.org/10.1561/1100000009

Gershman, A., Wolfe, T., Fink, E., & Carbonell, J. (2011). News Personalization using Support Vector Machines, 28–31. Retrieved from http://repository.cmu.edu/cgi/viewcontent.cgi?article=1051&context=lti

Hamerly, G., & Elkan, C. (2004). Learning the k in kmeans. *Advances in Neural Information Processing …*, *17*, 1–8. http://doi.org/10.1.1.9.3574

Hu, Y., Volinsky, C., & Koren, Y. (2008). Collaborative filtering for implicit feedback datasets. *Proceedings - IEEE International Conference on Data Mining, ICDM*, 263–272. http://doi.org/10.1109/ICDM.2008.22

Lee, K., & Lee, K. (2015). Escaping your comfort zone: A graph-based recommender system for finding novel recommendations among relevant items. *Expert Systems with Applications*. http://doi.org/10.1016/j.eswa.2014.07.024

Lipton, Z. C., Elkan, C., & Naryanaswamy, B. (n.d.). Thresholding Classifiers to Maximize F1 Score. Retrieved from https://arxiv.org/pdf/1402.1892.pdf

Saffari, A., Leistner, C., Santner, J., Godec, M., & Bischof, H. (n.d.). On-line Random Forests.

Steck, H. (n.d.). Evaluation of Recommendations: Rating-Prediction and Ranking. http://doi.org/10.1145/2507157.2507160

Su, X., & Taghi M.Khoshgoftaar. (2009). A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, *2009*(Section 3), 1–19. http://doi.org/10.1561/1100000009

Takács, G., & Tikk, D. (n.d.). Alternating Least Squares for Personalized Ranking.

Wang, Y., He, D., & Chen, W. (2013). A Theoretical Analysis of NDCG Ranking Measures, 1–30. Retrieved from http://www.jmlr.org/proceedings/papers/v30/Wang13.pdf

Zadeh, R. B., & Carlsson, G. (2014). Dimension Independent Matrix Square using MapReduce (DIMSUM).

Zhang, H. R., Min, F., & He, X. (2014). Aggregated recommendation through random forests. *Scientific World Journal*, *2014*. http://doi.org/10.1155/2014/649596

Zhang, T., & Iyengar, V. S. (2002). Recommender Systems Using Linear Classifiers. *Journal of Machine Learning Research*, *2*, 313–334. http://doi.org/10.1162/153244302760200641

APPENDIX A: TIMELINE

Week	Description	% of time spent
Week 1	Write project plan	25%
	Draft research report	25%
	Complete Coursera course: Introduction to Recommender Systems (University of Minnesota)	50%
Week 2	Finish research report	75%
	Set up development infrastructure	25%
Week 3	Start to implement collaborative filtering algorithm	50%
	Set up Spark in development infrastructure	50%
Week 4	Mirror Elasticsearch data in development infrastructure	25%
	Implement popular organizations job	25%
	Fix collaborative filtering implementation	50%
Week 5	Implement analyses of data	25%
	Add hyperparameter optimization and cross validation	25%
	Integrate Elasticsearch with Spark jobs	50%
Week 6	Fix running Spark job on development infrastructure	25%
	Start to implement k-means clustering	25%
	Develop custom ranking metrics calculation	50%
Week 7	Set up deployment to production infrastructure	25%
	Draft final report	25%
	Implement k-means streaming and approximate nearest neighbours	25%
	Set up opportunity recommendations	25%
Week 8	Continue to write final report	25%
	Improve ranking metrics calculation	25%
	Implement various other Spark jobs	25%
	Refactor codebase according to SIG feedback	25%
Week 9	Finish final report	50%
	Improve ranking metrics calculation	25%
	Draft presentation	25%
Week 10	Finish presentation	75%
	Deliver code to Magnet.me	25%

APPENDIX B: SOFTWARE EVALUATION

The *Scala* source code of our project was sent to the *Software Improvement Group* (SIG) on December 22th and reviewed by them soon after. They assigned a score of 4.0 out of 5.5 to the code. A summary of the feedback they gave us:

1. *Components*: a clear component structure in the file system is missing;
2. *Component independence*: too large top-level components;
3. *Unit interfacing*: above-average number of parameters for some units;
4. *Testing*: lack of unit tests.

We have processed the feedback by adding unit tests for the ranking metric functions. We also refactored most *data access object* (DAO) components, because that was the most unorganized part of the file system. Job components regarding data analysis and model comparison were moved to a folder called *analysis*.

The original Magnet.me recommender jobs were written in *Java* and Magnet.me has not used the *Scala* language in any part of the company. This is why our *Scala* code must be well documented, so that even developers that have no experience with *Scala* understand the workings. We tried to accomplish this by making the names of components and parameters as obvious as possible, and using the same naming conventions as Magnet.me does. Also, additional comments are added at places where *Scala* specific patterns are used, in order to clarify the functioning.

APPENDIX C: LIST OF EVENTS

By running aggregation functions on the Elasticsearch database, we managed to retrieve all possible event types and list the total counts of each event type.

--- For confidentiality reasons, the table has been removed from this public document ---

APPENDIX D: PROJECT PLAN

Magnet.me is a young company that lets students and graduates discover all their relevant career opportunities. To accomplish this, a recommendation engine is used to discover and provide these opportunities. By analyzing the behaviour of Magnet.me users, the engine recommends companies or opportunities to the users.

A user creates an account with some information about his education and his interests. After that the user can browse through Magnet.me (web and mobile application) to find companies or opportunities that he or she might be interested in. A user can connect with companies, that way these users will be added to the network of a company. Recruiters from companies know what users are in their network. That way, recruiters know what students/graduates are interested in their company and contact them if needed. Users can also apply for opportunities through Magnet.me (events, jobs, internships etc.).

Problem description

Magnet.me currently uses the user collaborative filtering feature of the Apache Spark framework to provide recommendations for its users. Using this feature, similar, popular and personal recommendations are provided. The parameters in the current system were determined years ago when magnet.me had less users and companies attached to it. The system needs to be optimized by tweaking its parameters. Currently, it is not known or possible to read the performance of the recommendation engine. A program needs to be written that automatically reads and tries to improve the performance of the recommendation system to accomplish this.

Research questions

In order to meet the academic requirements and achieve an optimal solution for Magnet.me, we will answer the following question and corresponding sub questions during the project:

In what way can the Magnet.me recommendation system be improved?

(1) Which recommender systems can be applied to the user data of Magnet.me, in order to realise the best matches between people and companies/opportunities?
(2) How can we set up a system that automatically trains a new model, based on the entire dataset available at that time, at a regular basis (monthly), integrated into the Magnet.me technology stack?
(3) How can we deploy our model, in such a way that the Magnet.me API can easily retrieve recommendations for specific users, as determined by the recommendation model that we train monthly and run in Spark?
(4) Optional: can we use other machine learning techniques, such as supervised machine learning or deep neural networks, to generate recommendations based on not only user actions, but also the contents of a user profile?

To get more insight in recommendation systems, we will also participate into a Coursera course "Introduction to Recommender Systems: Non-Personalized and Content-Based". This course was advised by the client. This course provides a basic theoretical understanding of how certain recommendation systems work.

Data & algorithms

Currently Magnet.me makes recommendations based on data of three different types of user-actions:

- User clicks on a company recommendation (positive data)
- User clicks on a opportunity recommendation (vacancy) (positive data)
- User ignores a recommendation (company/opportunity) (negative data)

Several million user actions have been collected in the past few years. Several times per day, the Spark server runs the algorithm to update the recommendations for each user, using Spark's collaborative filtering feature. We will need this data in order to make a program that evaluates the performance of the Magnet.me recommender. Our final software product has the task to repeatedly tweak the parameters of the recommender model run in Spark, with the goal to get the best performance out of the recommender by keeping the parameter settings up-to-date.

Evaluation and benchmark testing
The metric for evaluation of the model is accuracy. We will measure what the recommender predicted versus what actually got clicked. This test can easily be run offline, so an improvement can be measured and verified. A side product of this research can be to figure out weaknesses in the way Magnet.me currently logs signals. Are they missing any signals or are they using the wrong signals?

Deliverables
First a program that reads the performance of the current configuration of the recommender needs to be written. Second, a way to improve the recommender needs to be found. Lastly, suggestions for improving the recommendations even further by using other techniques, signals or configurations need to be given or implemented.

Deployment
To train and run the recommendation models, we will set up an Apache Spark cluster on Google Container Engine. This can also be used for setting up a mirror database of ElasticSearch, so we can train our model on the entire dataset without impacting the production environment of Magnet.me. SSH keys are used to gain access and run commands on these Docker instances and all recommended security settings are being used.

We are using the $300,- initial credit that is provided by Google Cloud to all new customers in order to run these cloud instances.

Constraints
1. We need to take security measures to protect the data, this includes setting up hard drive encryption on our computers and enabling two-factor authentication on Github;
2. The solution has to be integrated into the current Magnet.me toolchain (consisting of Maven, Jenkins and ElasticSearch).

Timeline
Below is an initial outline of our plans for this project. Note that this is preliminary and will most likely change dependent upon our progress and further research in the first few weeks.

Nov 14 - Nov 20	Finish project plan, draft research report, coursera course
Nov 21 - Nov 27	Finish research report, setup technology infrastructure
Nov 28 - Dec 4	Work on sub question (2)

Dec 5 - Dec 11	Work on sub questions (2) and (3)
Dec 12 - Dec 18	Work on sub question (3)
Dec 19 - Dec 25	Work on sub question (4), start working on final report, First SIG submission
Dec 26 - Jan 1	Christmas
Jan 2 - Jan 8	Christmas
Jan 9 - Jan 15	Work on subquestion (4), draft final report, implement SIG feedback
Jan 16 - Jan 22	Work on subquestion (4) and remaining, Finish final report
Jan 23 - Jan 29	Deadline final report (24 January), prepare final presentation, Final SIG submission
Jan 30 - Jan 31	Final presentation (31 January)

Group members

Borek Beker	Rick Brugsma	Jeroen Offerijns
bekerborek@gmail.com	rbrugsma@gmail.com	jeroen@offerijns.nl
+316 832 171 44	+316 203 925 73	+316 814 875 87
#4118650	#4163788	#4221524

Client advisor and TU coach

Thomas Abeel	Alex Nederlof
Assistant Professor, TU Delft	CTO, Magnet.me
T.Abeel@tudelft.nl	alex.nederlof@magnet.me
+31 15 27 85114	+316 246 931 80

APPENDIX E: INFO SHEET

Optimising a Recommendation Model for Career Discovery

Client
Magnet.me
Goudsesingel 200
3011 KD Rotterdam

Final presentation
31 January, 2017 at 9.00
EEMCS faculty
TU Delft

Description

Recommendation systems are algorithms that aim to predict what items are preferred by a user, based on a recorded history of user activity. Magnet.me is a company which recommends companies and opportunities to students. Potential algorithms for recommendation systems are *memory-based* and *model-based collaborative filtering*, *graph-based approaches*, *support vector machines*, *random forest classifiers* and *wide & deep learning*.

Based on a qualitative comparison of the algorithms, *model-based collaborative filtering*, which is what Magnet.me currently uses as well, was chosen to be the best fit. This is because it scored highly on the three most important factors for Magnet.me: potential performance, compatibility with the dataset and scalability. When comparing several well-known benchmarking metrics, the most suitable metric for testing the performance of the recommender was the *F1 measure*. To benchmark the model, the connection status of user-company relations should be used as the label, but must be excluded in the calculation of the implicit ratings.

Logarithmically scaling the view counts before used as a factor in the implicit ratings has proven to be of negligible effect. Five other signals are found that could be used to improve the recommendations. Hyperparameter optimization with *cross validation* is implemented and has succesfully been deployed into the Magnet.me technology stack. The possibilities for a clustering algorithm are considered in order to solve the cold start problem, but we could not determine the numeric distances between features, which is required for training an accurate clustering algorithm.

An unexpected challenge of this project was setting up the development infrastructure. It consisted on setting up an Elasticsearch cluster and a Spark cluster that can interact with each other on Google Cloud services. Another challenge was in benchmarking the recommender with proper metrics.

Team

B. Beker	**R. Brugsma**	**J.F. Offerijns**	**Dr T. Abeel**	**A. Nederlof MSc**
Project Member	Project Member	Project Member	Academic Adviser	Client Adviser
bekerborek@gmail.com	rbrugsma@gmail.com	jeroen@offerijns.nl	Assistant Professor, TU Delft	CTO, Magnet.me

The final report for this project can be found at: http://repository.tudelft.nl.

APPENDIX F: RESEARCH REPORT

Optimizing a Recommendation Model for Career Discovery

B. Beker
Project Member
#4118650

R. Brugsma
Project Member
#4163788

J.F. Offerijns
Project Member
#4221524

Dr T. Abeel
Academic Adviser
Assistant Professor, TU Delft

A. Nederlof MSc
Client Adviser
CTO, Magnet.me

1 INTRODUCTION

Recommendation systems are algorithms that aim to predict the rating a user would give to an item, which can be used to provide recommended content to users. Magnet.me is a company with a website and mobile apps, which students and recent graduates can use to discover jobs and career opportunities. They are using a recommendation engine to provide personal recommendations of companies and opportunities to students. A basic recommendation system has already been implemented, but there is definitely room for improvement.

The goal of this project is to improve and tweak the recommender that is currently in use. In order to achieve an optimal solution for Magnet.me, we will answer the following question: how can the recommendation system of Magnet.me be improved? We will do so by working on the following sub-questions:

(1) Which recommender systems can be applied to the user data of Magnet.me, in order to realise the best matches between people and companies/opportunities?

(2) How can we set up a system that automatically optimizes the model training parameters, based on the entire dataset available at that time, at a regular basis, integrated into the Magnet.me technology stack?

(3) How can we deploy our model, in such a way that the Magnet.me API can easily retrieve recommendations for specific users?

(4) Optional: can we use other machine learning techniques, such as deep neural networks, to generate better recommendations based on not only user actions, but also the contents of a user profile?

To answer these questions, the following sections are presented in this research report: in section 2, jobs-to-be-done, user stages and signals will be discussed to give insight in how the product is used. In section 3, a selection of applicable recommendation algorithms are described. Section 4 gives a comparison of the selected algorithms and what algorithms will be implemented. In section 5, hyperparameter tuning will be discussed; and in section 6, a description will be given of how we will benchmark the implemented recommender system. Finally, we will discuss our conclusions.

2 CONTEXT

To decide what technique will be used to provide recommendations for the users[23] of Magnet.me, a description of the context where it operates needs to be provided.

2.1 Jobs-to-be-Done Analysis

In this section, the *jobs-to-be-done framework*[24] is applied to give insight into the needs of the users of Magnet.me. Another way to do this is by defining personas and user stories. But we found out when describing these, that they tend to give questionable insights, because they do not acknowledge causality. The jobs-to-be-done framework describes causality and motivation in a way that would be more suitable for this case.

The main job for Magnet.me is to provide opportunities and companies to its users that they are interested in. Job stories are defined to describe what the goals and jobs are of typical users:

I. When users browse the application anonymously[25] and get directed to an opportunity or company they are interested in, they want to know what similar companies or opportunities there are, so they are aware of what opportunities or companies they like.

[23] Users are in this case students or graduates. Users can also be recruiters or employees of Magnet.me, but when referring to users in this report, specifically students or graduates are meant.

[24] Jobs to Be Done (Harvard Business Review, 2016)

[25] Anonymous use of the application is only applicable on web, not on mobile.

II. When anonymous users browse opportunities, they want to see opportunities or companies that are interesting to them, so they can find more relevant opportunities.

III. When users have just signed up and are looking for a job, they want to see opportunities or companies that are interesting to them, so they can apply for relevant opportunities and connect to interesting companies.

IV. When users that already have an account are looking for a job, they want to see relevant opportunities and companies they might be interested in and they want to see what is going on with companies that they are connected to, so they can apply for relevant opportunities.

V. When users make an account on Magnet.me, they want to see more relevant opportunities and companies they might be interested in, so they can find relevant jobs faster.

VI. When users are watching multiple opportunities in a short time, they want to see relevant opportunities and companies they might be interested in, so they have a better understanding of the diversity of jobs available.

Within the job stories, four types of users are defined. The following table summarises the properties of these users.

	User A	**User B**	**User C**	**User D**
Account age	0 days	1 hour	0 days	2 weeks
Signed up	no	no	yes	yes
Status	Recently graduated	Recently graduated	Bachelor student	Master student
Matching job-to-be-done	I	II	III	IV

2.2 User Stages

Users go through several stages. In every stage, recommendations could potentially be given or signals can be derived for the recommender. Below, a description will be given of the most important stages the customer goes through. In appendix B, you can find a list of screenshots to accompany these stages.

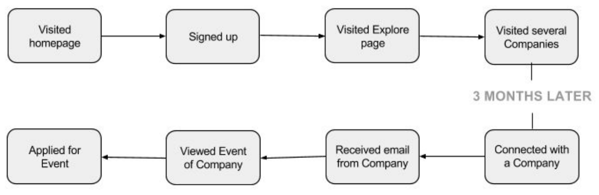


Figure 1: Common user flow

<u>a) Visited homepage without account</u>
To make the best use of Magnet.me, a user needs to create an account with details about his or her study background. Currently, no recommendations can be given on the homepage to new users because not much is known about the user.

<u>b) Visited Explore page</u>
When a user has created an account, he or she will be redirected to the *Explore page*. On web, this page consists of several lists of recommended opportunities and organizations. On mobile, the equivalent view would be several company recommendations through which the user can swipe. The user can connect to these companies or ignore them.

<u>c) Visit company</u>
Every company has their own page. On this page, a description is given along with some meta information. A user can connect with this company through a connect button to join the company's network if they have an account. Also, recommendations are given for companies that are either similar to the currently shown company, popular in general or just interesting for the current user. This is called the *browse next experience* (BNE).

<u>d) Visit opportunity</u>
Every opportunity—which can be a job, internship or an event—has a specific page with details about this opportunity. A user can save or apply for an opportunity if he or she finds it interesting and if the user has an account. Just like the company page, the opportunity page also has a BNE in which opportunities are recommended.

<u>e) Visit news feed</u>
When a user is connected to several companies, it is possible for them to stay up-to-date with these companies. The news feed shows recent updates posted by companies (text, photos or posted opportunities). A user can like these news posts as well.

<u>f) Company actions</u>
On several stages, it is possible for the user to connect to a company. By connecting, the user will enter the network of that company to receive updates and letting that company know that the user is interested in their company. This can lead to direct messages from recruiters to the user.

For a user, it is also possible to actively ignore a company recommendation. On the web application, a user can decide it is not interested in a company recommendation. On mobile, this works different than on web. A user needs to decide if he or she wants to connect to a company before seeing new recommendations. This means that on mobile replying to a company's recommendation happens more often than on web.

g) Opportunity actions

On the opportunity page it is possible for users to save an opportunity. By saving, they can easily retrieve the opportunities that they are interested in. Users can also apply for opportunities by sending a message to a specific recruiter of a company.

h) Invited to apply actions

Companies are able to send out invites for opportunities to users that are within their network. A user receives an invitation via email and they can accept it by applying for an opportunity. A user can also decide that he or she is not interested in the opportunity. Lastly, the user can be reminded of their invitation or they can ask a question about the opportunity.

i) Clicked opportunity link in email

Occasionally an email is sent to users with personal recommendations. The user will be redirected to a company or opportunity page if he/she clicks a link in the email.

Recommendations can be given within three different media: in the web-application, in the mobile app and via email. The web medium includes both the full-blown web application and the responsive mobile version of this application. The mobile medium includes the native iOS and Android apps. All mentioned stages can occur both in the web application and the mobile app, except visiting the homepage (this can only occur in the web application).

2.3 Signal and Event Tracking

In Appendix B, we have listed the events that are currently being tracked by Magnet.me. This includes viewing companies and opportunities, viewing news posts and other general user actions such as logging in and out. It also records clicks on recommended companies or opportunities, which we can use to benchmark our recommendation model.

2.4 Recommendation Types

At the moment, recommendations are given in three different locations: on the Explore page; in the *Browse Next Experience* (BNE), which is a list of similar companies or opportunities on their profiles; and in emails that are sent to users.

We can define six different types of recommendations that are currently provided:

(1) Personal company recommendations;
(2) Personal opportunity recommendations;
(3) Popular companies within the interests of a user;
(4) Popular opportunities within the interests of a user;
(5) Most similar companies;
(6) Most similar opportunities.

2.5 Signal Suggestions

Several signals are currently missing that might help in improving the recommendation engine:

(1) Ignoring a company on web and ignoring a company on mobile is seen as the same signal for the recommender. Seeing as the equivalent method of ignoring a company on the web is to click a small X icon in the top right of a company, whereas the reject action on mobile is a necessary step for proceeding to the next company, there is definitely a significant difference in the weight of these signals.

(2) When a student connects with a company, but later decides to disconnect, this is interpreted to be the same as never connecting in the first place. However, we would argue that this is a fairly strong signal that the user is not interested anymore in this particular company.

(3) Recruiters can visit the profile of a user, which can be a strong indication that the company of the recruiter could be a good fit for this user.

(4) Recruiters can invite a user to apply for a opportunity as part of a recruitment campaign. This is a strong signal that the company has deemed this user to be a good fit. How a user responds (dismiss invitation; apply for the opportunity etc.) to this invitation can be a good signal as well.

(5) Users can send and receive messages to and from recruiters, which is a strong signal that the user is interested in the company of the recruiter.

(6) Lastly, the apps currently only track click (or tap, on mobile) events and other actions. Another type of data that could be tracked is user focus: how long and how actively are users looking at a particular company profile or card in a list? Another event that could be checked if a certain recommendations enters the viewport of a certain user.

We also plan to explore whether a time bias should be included in the recommendation system. If user A applied for a company three years ago, while user B applied for a company last month, are these signals of the same strength? As shown in the customer journeys (section 2.2), interests of students can change over time, which makes older recommendations irrelevant. Most likely, there should be some level of regularization to account for the time that has passed since the signal occurred.

3 LIST OF ALGORITHMS

Recommendation systems can be implemented using a variety of techniques. Traditionally, the techniques that these systems use are separated into collaborative filtering and content-based filtering. In practice, companies often have developed custom algorithms that work best for their context. This can be done by combining or extending traditional recommendation techniques. For example, Google is applying deep neural network algorithms in YouTube with *Tensorflow* (Covington, Adams, & Sargin, 2016). Furthermore, Facebook uses the proximity of two nodes in the social network graph in order to make recommendations for users.

Content-based algorithms make use of the attributes (features) of users or items. In most approaches users are represented by a certain taste profile. A user builds up a profile with features based, in the case of Magnet.me, on connections with companies. A company could be placed at a certain location, focuses on a certain industry or has a specific amount of employees working there. If the user profile is similar to the attributes of the company, the company might be of interest to the user.

In order to research the various recommender systems that currently exist, we completed the online course *Introduction to Recommender Systems: Non-Personalized and Content-Based*[26], which is provided by the University of Minnesota. It gave us a good overview of the existing collaborative filtering and content-based filtering methods and the way these models can work.

3.1 Current Implementation

Magnet.me currently uses the collaborative filtering feature in the Machine Learning library of the *Apache Spark*[27] framework to provide personal recommendations, because this system was thought to be the best for the Magnet.me context. One of the characteristics of Magnet.me is that lots of opportunities are added to the database constantly (about --- opportunities on average per day over the last year[28]). At moment of writing, they have --- users, --- companies and --- opportunities in their database[29].

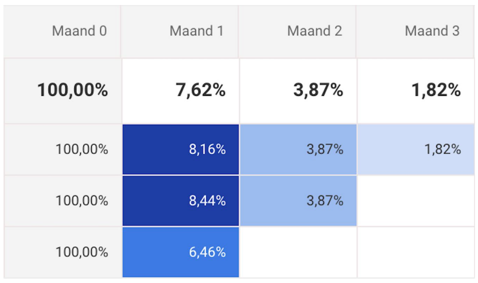
[26] Introduction to Recommender Systems on Coursera

[27] Website of Apache Spark

[28] We calculated this by counting the number of opportunities added daily to the ElasticSearch database and then taking the average.

[29] We have determined these numbers by querying the current Magnet.me database.

Typical for their users is the short time they are browsing through content. On average, as shown in figure 3, only a small percentage will return after a few months, because he or she usually stops searching after finding a suitable job.



Maand 0	Maand 1	Maand 2	Maand 3
100,00%	**7,62%**	**3,87%**	**1,82%**
100,00%	8,16%	3,87%	1,82%
100,00%	8,44%	3,87%	
100,00%	6,46%		

Figure 2: Retention rate of users of Magnet.me on a monthly basis on the web application (source: Google Analytics)

Apart from using collaborative filtering for giving personal recommendations, similar and popular recommendations are given. The CF model is trained once an hour, which means that until this moment, no personal recommendations can be given. This is why only popular and similar recommendations are given in this time. After an hour, the user-based collaborative filtering algorithm is re-trained, which takes 16 minutes on average[30], and personal recommendations will be given. To make the best use of the product, users need to receive recommendations as soon as possible, while the system has little time to learn user preferences.

The idea of user based collaborative filtering algorithm is that the behaviour of past users is used to make recommendations for new users (Ekstrand, Riedl, & Konstan, 2011). In the context of Magnet.me, this is done by collecting events that users produce. Currently, the recommender gives recommendations based on data consisting of three different types of user actions:

- User views a company recommendation (positive);
- User views an opportunity recommendation (positive);
- User connects with a company (strong positive).

To obtain data about the relation between user and content, explicit or implicit data can be gathered. The difference is that with explicit data, the user is asked to give a score for specific content. With implicit data, no score is explicitly asked and this must be inferred from user actions like clicks. The recommendations of the current algorithm are based solely on implicit user data. It is important to note that this means an extra step in the computation of the predictions is required. The implicit data of a user has to be converted to a confidence

[30] We determined this by running a database query in ElasticSearch to determine the average time of each Spark job over the last year, after which we summed the averages of each of the 4 types of jobs.

matrix where all entries contain confidence values. (Hu, Koren, & Volinsky, 2008) The value 1 is assigned to the two click actions. A value of 10 is assigned to the connect action, because this is considered a more significant relation between a user and a company than a click action. Several million events have been collected in the past. When a user produces several events, the collaborative filtering compares him or her to users that are similar and gives recommendations accordingly.

Alternating Least Squares

The collaborative filtering feature inside Spark, which is used by Magnet.me, relies on matrix factorization to find the best recommendations. *Alternating Least Squares* (ALS) is an optimization algorithm that is used by Spark to solve this matrix factorization (Takács, Tikk, 2012). Before the algorithm starts, the data in the user-content matrix is sparse. After running the algorithm, this matrix contains all predictions for how much a user matches with a specific company.

This matrix can be factorized in P and Q, of which the values are unknown in the beginning, and can be initially set to 1 or small random values. ALS has to learn these latent vectors by alternatingly updating the values of P and Q. In multiple iterations, it changes values in one of the two matrices and calculates the other matrix. After enough iterations, the values will converge, and the entries in the multiplication matrix $P \times Q$ should be good predictions of how much a user matches with a specific company/opportunity.

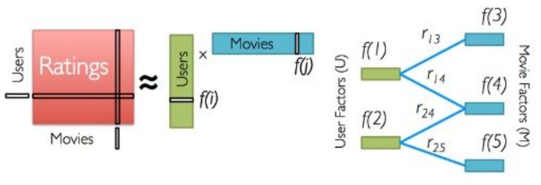


Figure 3: Example of matrix factorization calculation (source: Databricks)

In regards to ALS in Spark, five parameters can be configured to improve the resulting predictions. Additionally, one parameter can be passed to specify the number of blocks used for parallelization. Typical for the ALS algorithm is that it alternatingly changes one of the two factorized matrices. The optimization of the latent vectors could also be done using other algorithms like Bayesian networks or probabilistic latent semantic analysis (Su, Khoshgoftaar, 2009), but these algorithms are not included in Spark's machine learning library.

Strengths and Weaknesses

The user collaborative filtering (CF) algorithm has a number of benefits for Magnet.me. First, the content that is recommended does not have to be similar to what the user searched for in the past. If other users, that are similar to a user, first searched for company X and later searched for another unrelated company, the significantly different company can be recommended to a new user that searched for company X, This advantage applies to memory-based CF-algorithms as well.

Another benefit is that the recommended content actually has matched with previous users. This can also be a problem because no events exist for recently added companies or opportunities, which could result in new items not being recommended at all. This problem is partly solved by also presenting content that is new to users.

Previously no distributed user-CF algorithm existed, but Spark makes the algorithm distributed and thus more scalable. A disadvantage of the current system is that no personal recommendations can be given in the first hour, because the CF-algorithm only updates all predictions once an hour in the current implementation at Magnet.me.

3.2 Alternative Techniques

Below we have included a list of potential alternative techniques that can be used for recommending companies and opportunities to students. Note that this list is not exhaustive, but includes a subset of all machine learning algorithms that we found to be the most likely candidates for this particular project.

Memory-based Collaborative Filtering

Besides the model-based collaborative-filtering approach that is described above, a *memory-based* approach can be used where the similarity between users is calculated by similarity functions like cosine similarity or the Pearson correlation coefficient. A weighted average of events of the most similar users is determined to provide a recommendation. Until recently, memory-based approaches did not scale well with larger datasets, but this has also been fixed in Spark by the implementation of Dimension Independent Matrix Square using MapReduce (DIMSUM) (Zadeh, Carlsson, 2014).

Graph Analysis

Graphs can be used in certain different ways to improve recommendation accuracy. One example of graph-based filtering is *GraphRec*, a system which converts the user-item matrix to a weighted adjacency matrix, which is used to create a graph from which the entropy of all items can be calculated (Lee, Lee, 2015).

For example, GraphRec has been used to recommend music artists using explicit user data. This algorithm tends more towards a content-based approach, because the items are centralized and relations are being made between items, based on their attributes. Another graph-based algorithm is found where users are nodes and users are connected to item-attributes via the items, as shown in Figure 4 (Yu et al., 2014).

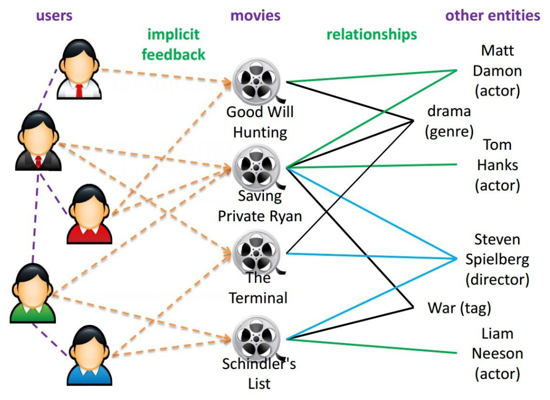


Figure 4: Knowledge Graph used for movie recommendation (source: Yu et al., 2014)

Support Vector Machines
SVM is a model that classifies its input into a binary output. It does so by defining an hyperplane between vectors (which are vectors with features of a user or organization/opportunity). Vectors on one side of the hyperplane are items or users that are similar to the input vector of the SVM. In the Magnet.me case, input could be a vector with features of a user. The SVM will determine what class (a collection of opportunities and organizations) the user belongs, which is basically equivalent of determining what opportunities or organizations could be recommended to that user. This way, SVM can be used as a recommendation system (Gershman et al., 2011).

Unfortunately, research has shown that SVMs perform poorly on recommendation problems. This is because there is only a small percentage of users interested in a particular item/company, leading to extremely unbalanced class distributions (Zhang, Iyengar, 2002).

Random Forests
A *random forest classifier* is a model that uses several decision trees to classify its input (Ajesh, Nair, & Ps, 2016). These decision trees base their decision on training data that has been served. In the Magnet.me case, input of the random forest model could be a vector representing the user profile (in a model based implementation) (Zhang, Min, & He, 2014). Every decision tree determines what recommendation (company or opportunity) comes out of their tree. An average of all output of decision trees will be calculated to determine what recommendation suits that user best.

A disadvantage of random forests is that streaming data input is a problem for its model. When new training data is available, a random forest needs to be recreated and trained (Saffari et al., 2008). In Magnet.me data, new example data arrives all the time (new companies and new opportunities), which means that without recreating the forest constantly (which takes a lot of computation time), the system would give questionable recommendations.

Wide and Deep Learning
Neural networks are fairly new in the context of recommendation systems. *Deep neural networks* have more hidden layers in the network (mostly more than ten) in comparison with "normal" neural networks. In a recommendation case, input for the neural network would be several features of a user profile and the output would be a classification of for example companies.

Recently, *wide and deep neural network learning* has been introduced in the Google Play Store (Cheng et al., 2016). Wide and deep learning means that a network has two components, one wide (which is in this paper is a generalized linear model) and one deep (a deep neural network) component. It is stated that the recommendations of the Play Store improved significantly.

3.3 Addressing the Cold-Start Problem

One of the main problems of recommenders is that little information of new users is available. This problem is called the cold start problem. This is especially a problem when recommendations are mainly based on previous user behaviour. As the first recommendations are currently only based on what companies are popular for the study background of the student, it is preferable to use all available user data, like interests, age and educational level. This is currently not possible.

To tackle this problem, we would be able to create an online algorithm of the model-based collaborative filtering algorithm. Unfortunately, there are no production-ready implementations available. However, the machine learning library in Apache Spark does contain a streaming implementation of the k-means clustering algorithm. While this cannot be used directly for providing recommendations, we can apply it indirectly. If we can cluster users together based on their profile content, we can provide recommendations to new users immediately after completing their profile, by looking at the recommendations that were provided to existing users that are also in their cluster.

4 CHOOSING AN ALGORITHM

4.1 Decision Factors

To decide what algorithm will be used for implementing the recommendation system, certain factors are described. These factors will determine a rating for the recommendation system and with that the best option for Magnet.me will be presented.

- Potential Performance*:* Clearly, the most important factor in deciding the best algorithm is its accuracy at providing recommendations of companies and opportunities to users. It should also aim to not just provide the most popular companies as recommendations each time, but diversify its recommendations.

- Compatibility with Dataset*:* This factor determines if the algorithm is applicable on the Magnet.me dataset. Some algorithms work better with different types of data, so to determine how compatible a technique is with the data is important.

- Scalability*:* As Magnet.me is a growing company with new users and items each day, the algorithm needs to scale well. A very important factor deciding if an algorithm is scalable is if it can be used on a distributed system. Spark is a system which is mostly used to handle algorithms this way. An algorithm that is scalable is in this case the same as saying an algorithm has a Spark implementation.

- Prediction Time*:* This factor decides how fast the calculation speed is in recommending an item to a user.

- Model Training Time*:* This factor decides what the speed is in training the model, if applicable.

- Ease of Use*:* This factor determines how easy it would be to implement and deploy the algorithm.

4.2 Algorithm Comparison

We have used the decision factors to evaluate the various algorithms that were listed in the previous section. By rating each algorithm on a scale of negative-neutral-positive, we can create a simple overview of the advantages and disadvantages of each algorithm, as can be seen in the figure below. Furthermore, we have added a percentage denoting the importance of each factor for this project. Obviously, both the importance percentages and the algorithm ratings are guesses and not exact numbers. The main goal is to provide some insight into our reasoning and enable a relative ranking of the algorithms.

4.3 Proposed Implementation

When we take the importance of the decision factors in consideration, it can be concluded that the current implementation, model-based CF, scores best overall. This algorithm scores well for scalability because Spark makes use of ALS for the model training, which can be run distributed. Model-based CF is well-known for its diverse recommendations, which is not true for algorithms like RFC and SVM. As for the graph-based approaches, no accurate guessing could be made regarding to the performance.

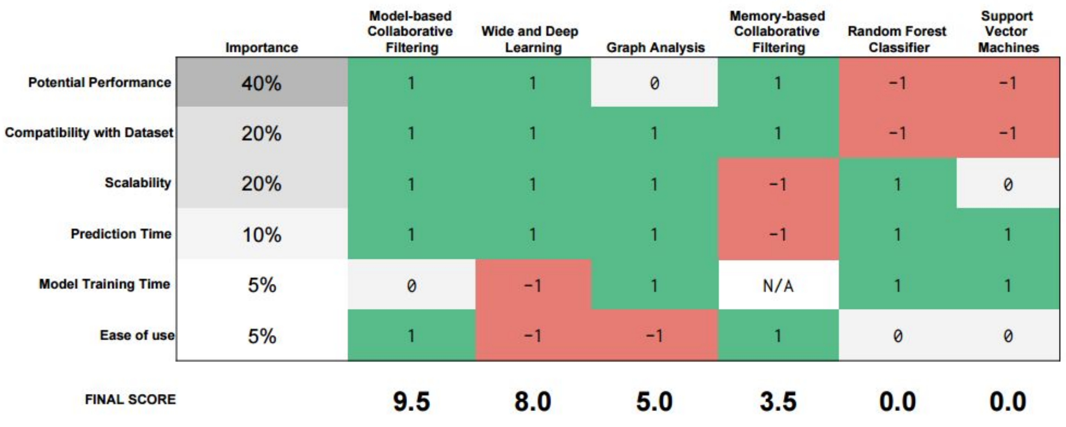
In order to tackle the cold-start problem, another algorithm can be used to give predictions for the first hour, when little information of a new user is available. We propose to integrate a streaming k-means algorithm for providing recommendations to new users. By clustering new users together with existing similar users, we can take the recommendations for existing, more established users with more data and provide these recommendations to the new users as well.

The other approach that scored well is wide and deep learning. The main disadvantage for wide and deep learning is that it is a relatively new and complex method. However, it has proven to be accurate at providing recommendations in the Google Play Store. Therefore, our thesis is that it could work quite well in this context, but we plan to explore it as an optional next step after fully implementing the model-based CF and testing the k-means clustering algorithm.

5 MODEL SELECTION

5.1 Hyperparameter Optimization

At the moment, the parameters that are used with the collaborative filtering implementation have been predefined and have not been updated in the last year. However, the optimal parameters to achieve the best results can very well



	Importance	Model-based Collaborative Filtering	Wide and Deep Learning	Graph Analysis	Memory-based Collaborative Filtering	Random Forest Classifier	Support Vector Machines
Potential Performance	40%	1	1	0	1	-1	-1
Compatibility with Dataset	20%	1	1	1	1	-1	-1
Scalability	20%	1	1	1	-1	1	0
Prediction Time	10%	1	1	1	-1	1	1
Model Training Time	5%	0	-1	1	N/A	1	1
Ease of use	5%	1	-1	-1	1	0	0
FINAL SCORE		9.5	8.0	5.0	3.5	0.0	0.0

change over time, due to new incoming data that changes the underlying structure of the dataset.

Methods exist that can automatically determine the optimal set of parameters to use when training the model. The main approach is to run a parameter sweep. Quite simply, you define minimum and maximum values for each parameter and it will train the model with all possible combinations of parameters. To determine the best set of parameters, a loss function is defined which calculates the error rate of the model. A basic implementation works like this:

(1) Create separate sets of training and test datasets, which is often a 70-30 or 80-20 split;
(2) For each set of parameters:
    a. Train the recommendation model using the training data;
    b. Evaluate the performance using the test data;
(3) Choose the best set of parameters with the lowest error rate.

In the Spark implementation of collaborative filtering, we can run a parameter sweep on the following parameters:

- rank: the number of latent factors in the model;
- iterations: the number of iterations of ALS to run;
- lambda: the regularization parameter in ALS;
- alpha: governs the baseline confidence in preference observations.

Furthermore, we can explore whether it would be worthwhile to tune the weights of the various signals. At the moment, each company profile view weighs as 1 point while being connected to a company weighs as 10 points. We can attempt to include these in the parameter sweep as well and see if this leads to better results.

5.2 Cross Validation

However, this approach can be prone to overfitting, as the lowest error rate can easily be a model that is very specific to this training dataset. Therefore, cross-validation is often introduced in order to avoid overfitting the training data. For example, in k-fold cross validation, the entire dataset is split into k subsets. One subset is then used as the test dataset, while the other k-1 datasets are used as training data. The process then becomes:

(1) Create k subsets of the entire dataset;
(2) For each set of parameters:
    a. Repeat k times:
        i. Define one of the subsets as the test data;
        ii. Define the other k-1 subsets as the training data;
        iii. Train the recommendation model using the training data;
        iv. Evaluate the performance using the test data;
    b. Take the average of the error rates of each subset;
(3) Choose the best set of parameters with the lowest average error rate.

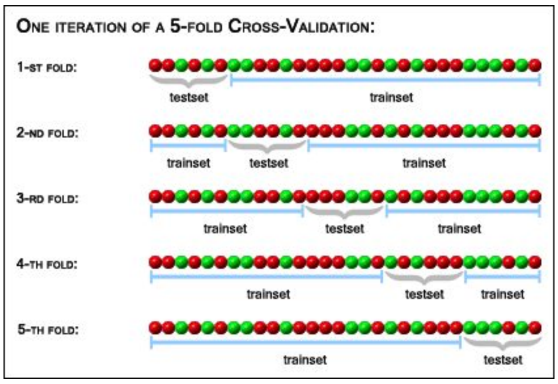


Figure 5: Example of a k-fold cross validation with k=5 (source: Graz University of Technology)

6 BENCHMARKING & MONITORING

Initially, we will develop a recommendation system that looks only at the various user actions and events that occur. These include viewing a company page, viewing an opportunity and more. We can then determine which companies or opportunities are the best matches for a particular user, based on these signals. These will be the recommendations.

6.1 Collaborative Filtering Model

Magnet.me has provided us with a dataset of company-user relations, which includes whether or not a user is connected to a company. We can use this to verify the accuracy of our recommendation system. Our plan is to filter the data on all user-company relations of which the status is *CONNECTED*. We will then use k-fold cross validation (as explained in section 5.2), of which we will use training data for creating our model to predict the best companies for a user. Then we can calculate the *root mean squared error* (RMSE) for the test data between the predicted best companies of a user and the companies to which the user is actually connected to.

By applying the same technique, we can benchmark the opportunity recommendations. We will filter the data on what opportunities have been applied to[31] and which opportunities have been saved. By calculating the RMSE between the predictions and actual applications or saves, we can benchmark the opportunity recommendations.

[31] In this case, applying means just clicking on the button to apply, as whether the user has fully completed the application form is not being tracked.

We will then run the same benchmarking model on the existing recommender system, that was implemented a year ago by Magnet.me. Since that model was trained using parameters chosen specifically for the data from more than a year ago (which was a significantly smaller dataset), we can most likely improve the model by choosing parameters that fit better with the current dataset.

Furthermore, we can start to set up other variations of our collaborative filtering model and benchmark this against both the existing model and our basic model. This can include a model with regularization for older events, a model which uses hyperparameter tuning to optimize the signal weights, or a model which looks at other signals which have so far been ignored.

6.2 Clustering Model

Using an online implementation of k-means, we plan to cluster users with similar users and then provide recommendations to a new user based on the recommendations of the existing, similar users. Our collaborative filtering model creates a user-feature matrix that we can use to find the most similar users. We can use this matrix as a benchmark for the k-means clustering model.

By looking at the X most similar users of each user in the k-means model, comparing this with the X most similar users of each user in the user-feature matrix of the collaborative filtering model and then calculating the RMSE, we can determine the accuracy of our clustering model.

6.3 Monitoring

When we can show that our model is an improvement over the existing model using sufficient benchmarking data, Magnet.me will deploy our new model to their production servers. They are already tracking clicks on recommendations and are currently implementing a system which can track the *click-through rate* (CTR) of recommendations. We can then see what the improvement of our model will be in terms of the live CTR in the Magnet.me apps.

7 CONCLUSION

In this research report, we have analyzed how recommendations are used at Magnet.me. We determined what the jobs-to-be-done are and how recommendations could help to do those jobs. Also, exploratory research has been done which recommendation systems exists and what the advantages and disadvantages are of these systems. We have managed to choose what algorithms could best be implemented to improve the recommendations. By doing so, discovery of career opportunities for Magnet.me users will be improved.

First, the model-based collaborative filtering with ALS algorithm will be implemented using Spark. The algorithm seemed to be the best solution for the Magnet.me case. Moreover, by implementing this using Spark, it will be scalable. We will attempt to implement streaming k-means after that. This could be a good solution for the cold-start problem. When this has been implemented, we will try to improve the recommendations by integrating deep and wide learning using Tensorflow.

Apart from implementing the aforementioned recommendation algorithms, automatic hyperparameter optimization will be implemented to constantly improve the recommender. To determine if the proposed system will be an improvement over the old system, measureable benchmarks will be determined. When an improvement by our system has been shown, Magnet.me will deploy our system on production. In appendix A, a project timeline is listed, giving an overview of our plans.

REFERENCES

Ajesh, A., Nair, J., & Ps, J. (2016). A Random Forest Approach for Rating–based Recommender System. *Intl. Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 1293–1297.

Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., … Shah, H. (2016). Wide & Deep Learning for Recommender Systems. *arXiv Preprint*, 1–4. https://doi.org/10.1145/2988450.2988454

Covington, P., Adams, J., & Sargin, E. (n.d.). Deep Neural Networks for YouTube Recommendations. https://doi.org/10.1145/2959100.2959190

Ekstrand, M. D., Riedl, J. T., & Konstan, J. A. (2011). Collaborative Filtering Recommender Systems. *Human–Computer Interaction*, *4*(2), 81–173. https://doi.org/10.1561/1100000009

Gershman, A., Wolfe, T., Fink, E., & Carbonell, J. (2011). News Personalization using Support Vector Machines, 28–31. Retrieved from http://repository.cmu.edu/cgi/viewcontent.cgi?article=1051&context=lti

Hu, Y., Volinsky, C., & Koren, Y. (2008). Collaborative filtering for implicit feedback datasets. *Proceedings - IEEE International Conference on Data Mining, ICDM*, 263–272. https://doi.org/10.1109/ICDM.2008.22

Lee, K., & Lee, K. (2015). Escaping your comfort zone: A graph-based recommender system for finding novel recommendations among relevant items. *Expert Systems with Applications*. https://doi.org/10.1016/j.eswa.2014.07.024

Saffari, A., Leistner, C., Santner, J., Godec, M., & Bischof, H. (n.d.). On-line Random Forests.

Su, X., & Taghi M.Khoshgoftaar. (2009). A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, *2009*(Section 3), 1–19. https://doi.org/10.1561/1100000009

Takács, G., & Tikk, D. (n.d.). Alternating Least Squares for Personalized Ranking.

Yu, X., Ren, X., Sun, Y., Gu, Q., Sturt, B., Khandelwal, U., … Han, J. (n.d.). Personalized Entity Recommendation: A Heterogeneous Information Network Approach. https://doi.org/10.1145/2556195.2556259

Zadeh, R. B., & Carlsson, G. (2014). Dimension Independent Matrix Square using MapReduce (DIMSUM).

Zhang, H. R., Min, F., & He, X. (2014). Aggregated recommendation through random forests. *Scientific World Journal*, *2014*. https://doi.org/10.1155/2014/649596

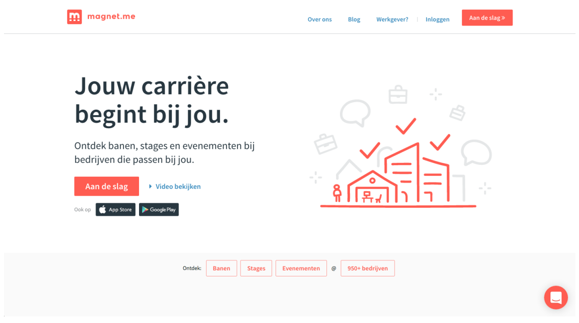
Zhang, T., & Iyengar, V. S. (2002). Recommender Systems Using Linear Classifiers. *Journal of Machine Learning Research*, *2*, 313–334. https://doi.org/10.1162/153244302760200641

APPENDIX E.A: PROJECT TIMELINE

Nov 14 - Nov 20	Finish project plan; Draft research report; Finish Coursera course.
Nov 21- Nov 27	Finish research report; Set up technology infrastructure.
Nov 28 - Dec 4	Implement the model-based collaborative filtering system in Spark.
Dec 5 - Dec 11	Improve CF system; Implement hyperparameter tuning.
Dec 12 - Dec 18	Deploy CF system with hyperparameter tuning; Implement k-means streaming.
Dec 19 - Dec 25	Improve k-means streaming; Start working on final report; Submit code to SIG.
Dec 26 - Jan 1	Break.
Jan 2 - Jan 8	Break.
Jan 9 - Jan 15	Deploy k-means streaming; Implement SIG feedback; Work on final report.
Jan 16 - Jan 22	Test wide and deep learning network; Finish draft of final report.
Jan 23 - Jan 29	Submit code to SIG; Finish final report.
Jan 30 - Jan 31	Final presentation.

APPENDIX E.C: PRODUCT SCREENSHOTS

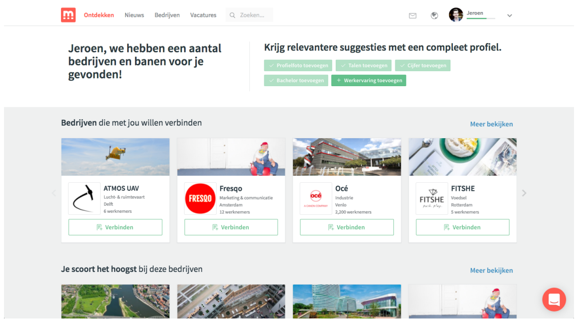
<u>Homepage</u>



Jouw carrière begint bij jou.

Ontdek banen, stages en evenementen bij bedrijven die passen bij jou.

<u>Explore page</u>



Jeroen, we hebben een aantal bedrijven en banen voor je gevonden!

Krijg relevantere suggesties met een compleet profiel.

Bedrijven die met jou willen verbinden

ATMOS UAV
Lucht- & ruimtevaart
Delft
6 werknemers

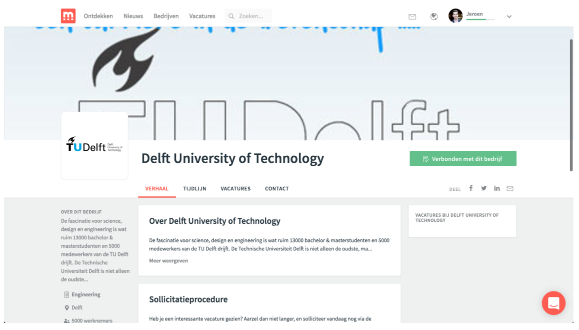
Fresqo
Marketing & communicatie
Amsterdam
12 werknemers

Océ
Industrie
Venlo
2,200 werknemers

FITSHE
Voedsel
Rotterdam
5 werknemers

Je scoort het hoogst bij deze bedrijven

<u>Company profile</u>



Delft University of Technology

OVER DIT BEDRIJF
De fascinatie voor science, design en engineering is wat ruim 13000 bachelor & masterstudenten en 5000 medewerkers van de TU Delft drijft. De Technische Universiteit Delft is niet alleen de oudste...

Engineering
Delft
5000 werknemers

Over Delft University of Technology

De fascinatie voor science, design en engineering is wat ruim 13000 bachelor & masterstudenten en 5000 medewerkers van de TU Delft drijft. De Technische Universiteit Delft is niet alleen de oudste, ma...

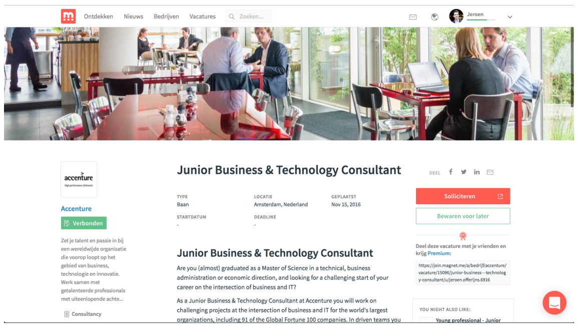
Meer weergeven

Sollicitatieprocedure

Heb je een interessante vacature gezien? Aarzel dan niet langer, en solliciteer vandaag nog via de

VACATURES BIJ DELFT UNIVERSITY OF TECHNOLOGY

<u>Opportunity</u>



Accenture

Zet je talent en passie in bij een wereldwijde organisatie die voorop loopt op het gebied van business, technologie en innovatie. Werk samen met getalenteerde professionals met uiteenlopende achte...

Consultancy

Junior Business & Technology Consultant

TYPE	LOCATIE	GEPLAATST
Baan	Amsterdam, Nederland	Nov 15, 2016
STARTDATUM	DEADLINE	
-	-	

Junior Business & Technology Consultant

Are you (almost) graduated as a Master of Science in a technical, business administration or economic direction, and looking for a challenging start of your career on the intersection of business and IT?

As a Junior Business & Technology Consultant at Accenture you will work on challenging projects at the intersection of business and IT for the world's largest organizations, including 91 of the Global Fortune 100 companies. In driven teams you

Deel deze vacature met je vrienden en krijg Premium:

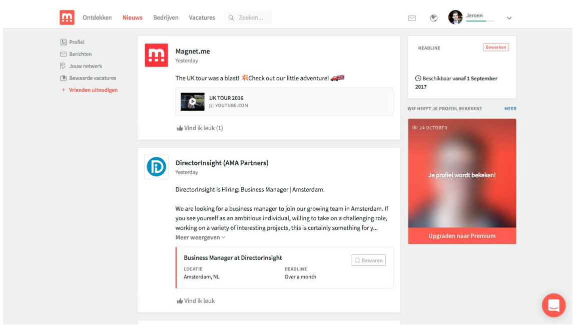
https://join.magnet.me/a/bedrijf/accenture/vacature/15096/junior-business---technology-consultant/u/jeroen.offerijns.6916

YOU MIGHT ALSO LIKE:

Young professional - Junior

<u>News feed</u>

<u>Mobile card-swiping interface (iOS)</u>



Magnet.me
Yesterday

The UK tour was a blast! 💥Check out our little adventure! 🚗🇬🇧

UK TOUR 2016
YOUTUBE.COM

Vind ik leuk (1)

DirectorInsight (AMA Partners)
Yesterday

DirectorInsight is Hiring: Business Manager | Amsterdam.

We are looking for a business manager to join our growing team in Amsterdam. If you see yourself as an ambitious individual, willing to take on a challenging role, working on a variety of interesting projects, this is certainly something for y...
Meer weergeven

Business Manager at DirectorInsight

LOCATIE	DEADLINE
Amsterdam, NL	Over a month

Bewaren

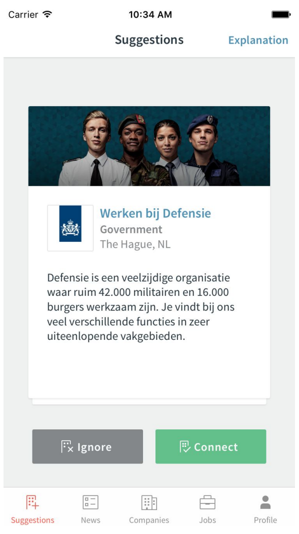
Vind ik leuk

HEADLINE
Bewerken

Beschikbaar vanaf 1 September 2017

WIE HEEFT JE PROFIEL BEKEKEN? MEER

24 OCTOBER
Je profiel wordt bekeken!
Upgraden naar Premium



Werken bij Defensie
Government
The Hague, NL

Defensie is een veelzijdige organisatie waar ruim 42.000 militairen en 16.000 burgers werkzaam zijn. Je vindt bij ons veel verschillende functies in zeer uiteenlopende vakgebieden.