Generating event recommendations with neural networks

Anonymous Henri Immonen Anonymous
- henri.immonen@aalto.fi -

February 2, 2018

Recommendation systems are a difficult task that can be solved using machine learning techniques, such as deep neural networks. In this paper the problem was addressed with Multi-Layer Perceptron Classifier (MLP) based on event and user context as well as with Restricted Boltzmann Machine as a collaborative design problem. MLP classifier was tuned in various ways: its capacity, activation function, solver functions as well as regurazing methods were varied in order to improve the results. Final testing accuracy for the whole dataset was 72.4 %. 80% of the data was used for training and 20% was used for validation.

1 Introduction

In our project we have addressed a recommendation system machine learning problem, more specifically the problem of recommending events for individual users, using the data from a Kaggle competition as described more in detail in chapter 4.1. Recommender systems are effective way of tackling the problem of finding the most suitable items (products or services) to particular users (individuals or businesses) in the ever-increasing complexity and volume of online information [1]. Depending on the wanted output, the recommender system tasks are usually either rating prediction, ranking prediction (top-n recommendation) or classification [2]. Recommendation models are commonly divided into three central categories: collaborative, content-based and hybrid recommendation approaches. The collaborative model uses previous valuations from other users of items to recommend an item to the user. Unlike collaborative filtering, the context-based recommender systems estimates the utility of an item for the user is based on the utilities of the same user to other items "similar" to the item being estimated. Both of these methods have their limitations, and therefore hybrid approaches combining both collaborative and content-based methods are also used in recommender systems. [3].

In our project we have experimented with both collaborative filtering and context-based methods. We have approached the collaborative filtering using Restricted Boltzmann Machine (RBM), which is described more in detail in chapter 3.1. In this approach the information about other users interest of events together with the tested user's c earlier interests were used to recommend interesting events for the the user c, thereby providing us ranking prediction to a user c. Context-Based method was used in terms of classification with the help of Multi-layer Perceptron (MLP) network. Details of method used are described in chapter 3.1.1.

2 Related work

Recommendation systems are difficult to solve using traditional means and therefore excellent challenge for deep learning techniques. One example of collaborative deep learning for recommendation system is described in paper [4]. Another example with context based music recommendation with MLP network can be seen from paper [5].

3 Method

3.1 Restricted Boltzmann Machine

Restricted Boltzmann Machine is a generative stochastic artificial neural network, that is used to learn the probability distribution of its input [6]. In our project we have used a standard type of RBM that has a layer of visible units connected to a layer of hidden units forming a bipartite graph. The connection between a hidden unit h_j and visible unit v_i consists of a weights $w_{i,j}$ between the units, the bias b_i of the visible unit and the bias c_j of the hidden unit.

As Markov Chain Monte Carlo can take a long time to converge, we have used a function called Contrastive Divergence (CD), which is minimized by taking only running few steps of Markov Chain, where the learning is still shown being able to work well [7]. The CD contains of a positive gradient part and a negative gradient part. In this project we have used logistic sigmoid as activation function for negative gradient part of the CD and rectified linear unit (ReLU) activation function for the positive gradient part of the CD function.

3.1.1 Multi-Layer Perceptron Classifier

Multi-layer perceptron network [8] was built to classify events according to users interests. We experimented with various different the internal components of the network to achieve better results. Rectified linear unit

(ReLU) and Logistic activation function was tested for hidden layer [8]. Equation X shows the ReLU activation function and equation Y shows the logistic function. We also used two different solvers for weight optimization of the net: stochastic gradient descent and Adam [8].

$$f(x) = x^+ = max(0, x) \tag{1}$$

Where x is the input to a perceptron

$$f(x) = \frac{L}{1 + exp{-k(x - x_0)}} \tag{2}$$

Where x_0 is the x-value of sigmoid's midpoint, L is curve's maximum and k is the steepness of the curve.

When tuning the network we varied its capacity by changing number of hidden layers and number of perceptrons inside the hidden layers. One way to achieve good results is by increasing capacity of the perceptron network until training error is as small as possible and then regularizing the network to improve the generalization error. For this we used L2 regularization and early stopping [8].

4 Data

4.1 Data set

Data we used for training and testing is from Kaggle competition [9]. We had an abundance of possible data features so we carefully selected the best fitting set for the classification task by hand. In some non-obvious cases we trained the neural network with different feature sets and then trimmed more useless features from the feature set. Sample size was hundreds of millions but we settled for roughly 100,000 because of the data loading and processing as well as training time. 80

Chosen data set consisted of events and user data. Selected user data features were:

- Age
- Location
- Gender

Age was calculated using the user's birth year and gender was either male or female. Users with unknown or unrecognized location where discarded.

Event data we used for the final training consisted of:

- Start time
- Country
- City
- Description

For the classification task we used boolean values (one or zero) describing if the user was in the same country or city with the event. Quality of this location feature could have been greatly improved by using Geocoding API, that, for example, google provided for a fee. Feature that would tell distance between user's location and event's location would have been much more meaningful. Description consisted of 100 features, where each feature represented how many times the nth most common word stems occurred in the event description.

4.2 Preprocessing

Data set was too big to process on its original format so we selected a smaller set. This set consisted of features used for training and testing that had no features missing. We created Python script to generate former data sets when needed and could be given custom data set size.

We scaled the data set before training using primarily standard score: ($z = \frac{x-\mu}{\sigma}$)

This helped the training immensely as the neural network used for classification is susceptible to get stuck in local optima if the data is not normalized in some way. This also helps with overfitting problem as weights in neural network are closer zero.

We also tried using PCA for dimension reduction with different dataset sizes. We noticed that a low number of principal components resulted the best results when the dataset size was lower, but with a larger dataset, PCA performed worse 1.

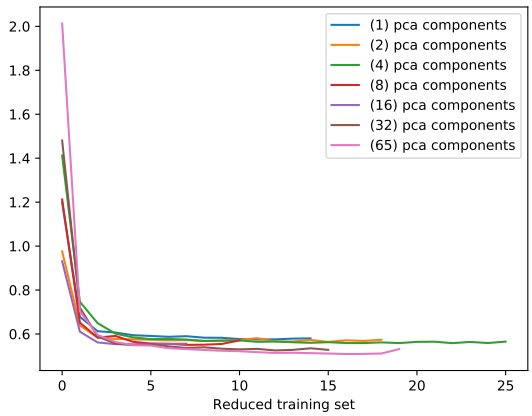

Figure 1: Number of principal components used in PCA

Interestingly, when we trained the classifier with a smaller dataset the PCA transformed data gained better results than simply normalized data, as can be seen in 1p. We suspected that this was because PCA prevents overfitting by removing noise, but with larger data set PCA reduced the score by 0.015.

Data size	PCA	Testing error
(1151, 106)	used	0.7315
1151, 106)	not used	0.6883
(5317, 106)	used	0.7011
(5317, 106)	not used	0.7161

Table 1: PCA transformed data compared to data that is only normalized. Two principal components used with PCA. Smaller data set benefits PCA transformation but the larger set does not.

5 Experiments

MLP network was experimented with extensively using different activation function for hidden layer, different weight optimization solvers, different number of layers and different amount of perceptrons in hidden layers. Scikit-learn library allowed this wide variety of experimentation in limited time due to high level of abstraction. Details of the experiments are given below

In addition to MLP network we also experimented collaborative filtering with Restricted Boltzmann Machine. We used contrastive divergence with sigmoidal and ReLU activation functions for learning in the RBM. More details of the hyperparameters are described below

5.1 Experiments with Scikit-Learn MLP Classification

5.1.1 Comparing Activation Function for Hidden Layers

ReLU activation function and Logistic activation function were compared for performance and accuracy. The test were run with following parameters:

- Data size: 100,000 entries, 20% for validation
- Size of hidden layers: 100, 100
- L2 regularization with $\alpha = 0.0001$
- Adam solver

Results of experiment are summarized in Table X. Generally speaking we can conclude that using ReLU improved training speed in this case by 360% which suggest that it is much more efficient activation function compared to Logistic activation function. Although Logistic activation had better testing accuracy it can be concluded that ReLU is a better choice for activation functions since it was significantly faster and gave a better training accuracy. A worse testing accuracy of ReLU was most likely caused by overfitting (since it had better training accuracy) and could be compensated with regularization.

5.1.2 Comparing solvers for weight optimization

Before training the classifier with a larger dataset, we made stochastic gradient descent compete against Adam optimizer. For the Adam we used default values as proposed by Diederik Kingma, and Jimmy Ba [10]: $(\alpha = 0.001)$, $(\beta_1 = 0.9)$, $(\beta_2 = 0.999)$ and $(\epsilon = 10^{-8})$

The classifier was trained with different solvers and hidden layer sizes. For this, we used a reduced training set with a size of 100,000 entries. Stochastic gradient descent performed worse than Adam with both hidden layer sizes as can be seen in 2.

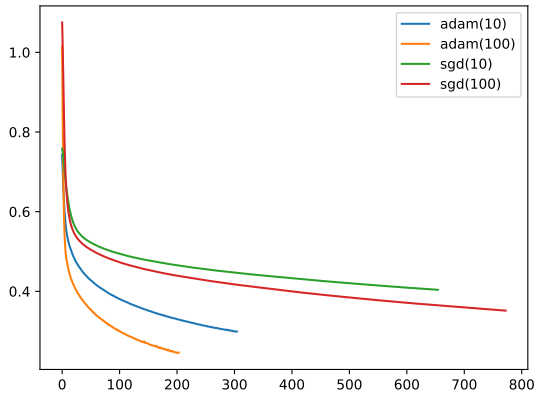

Figure 2: For training the classifier stochastic gradient descent (sgd in the figure) and Adam optimizer were trained with hidden layer sizes of 10 and 100. Adam had consistently faster descent rate.

5.1.3 Comparing Amount of Hidden Layers and Number of Perceptron in Hidden Layers Used

Various combinations of numbers of hidden layers as well as numbers of perceptrons in hidden layers were tried and compared. The test were run with following parameters:

- Data size: 100,000 entries, 20% for validation
- Activation function for hidden layers: ReLU
- L2 regularization with $\alpha = 0.0001$
- Adam solver

Table 3 summarises the results. Experiments were begun with a single layer and by increasing number of perceptrons in that layer until results did not improve. After that we pinpointed the exact number of neurons needed in a single layer for best performance. The result was a hidden layer with 83 perceptrons. This gave training accuracy of 85.7 %. Testing accuracy was not accounted for because overfitting was to be corrected by regularizing the classifier.

Also deeper networks were tested. Making the network deeper did not seem to improve the results significantly, so did further tests mostly with single layer network. The surprising training speeds were considered e.g. how could a network with three hidden layers each 100 perceptrons train faster than a single layered network with 10 perceptrons? This could be explained by fact that network stopped training after the score did not improve and perhaps a network with higher capacity reached this point earlier then one with lower capacity.

5.1.4 Comparing regularization methods

First we tested L2 regularization with different α values. The tests were run with following parameters:

- Data size: 100,000 entries, 20% for validation

- Hidden layer size: single layer with 100 perceptrons
- Activation function for hidden layers: ReLU
- L2 regularization
- Adam solver

Table 4 summarizes the results. Beside improving testing accuracy, regularization also improved the training speed significantly. α value of around 0.1 gave the best results, testing accuracy of 73.2%. We also tested early stopping by removing L2 regularization and increasing the capacity to a single hidden layer with 1000 perceptrons. This test trained the network in 2.2 seconds and had training accuracy of 75.9% and testing accuracy of 73.0 %. This is slightly worse result then L2 regularization with α value of 0.1.

5.2 Experiments with Restricted Boltzmann Machine

We have used the interest of a limited number of users for all of the events as the input for the Restricted Boltzmann Machine (RBM). The input matrix I contained the users in rows and the interest of the users for each of the events in columns. For this purpose we grouped the dataset by the users. If a user u_i had clicked that he or she was interested about an event e_j, then a value of 1 was assigned to $I_{i,j}$. In case the user u_i had indicated a disinterest for an event e_j then a value of 0 was assigned instead, and in case the user had indicated neither interest nor disinterest towards the event then a value of 0.5 was assigned to the event e_j for that user. Due to bad result discussed more thoroughly in chapter 6.2, we changed the interest value $I_{i,j}$ to be 1 when the user u_i had showed interest towards the event and 0 otherwise.

Due to the large dataset, we tried to run the RBM for only a limited number of users, and we have experimented with only 100, 200 and 500 users, as this was computationally very heavy and took a lot of time. The computations could have been made more effective by limiting the amount of events, however, limiting strictly only the number of events causes conflicts, and therefore we decided to use the whole events dataset. The RBM was run through 5 epochs with batch size of 10 and learning rate $\alpha = 0.5$.

To test the correctness of the results from the RBM, we have separated a part of the events from the list of events that the user had attended to, and compared the users actual interest to those events with the recommendation from RBM. As each user from the dataset had attended to minimum of four events and most of the users had attended to clearly more events, we set the limit of test events to be at least two events for each user and at most $\lfloor 0.2 \cdot n_i \rfloor$, where n_i is the number of events user u_i had attended to.

Activation Function	**Training speed (s)**	**Training accuracy (%)**	**Testing accuracy (%)**
ReLU	1.2	86.6	70.5
Logistic	4.3	84.4	0.71.5

Table 2: Summary of results from comparison of activation functions

Hidden layers (# per layers)	**Training speed (s)**	**Training accuracy (%)**	**Testing accuracy (%)**
(1)	1.1	74.5	71.5
(10)	3.9	83.4	70.1
(100)	3.9	85.1	72.5
(1000)	4.3	85.1	72.5
(83)	3.5	85.7	72.3
(100, 100)	4.2	84.4	71.5
(100, 50)	3.2	84.6	71.5
(50, 25)	2.7	84.7	71.4
(20, 10)	4.3	85.1	71.0
(100, 100, 100)	1.2	76.2	73.5
(100, 100, 100, 100)	6.1	83.5	70.3

Table 3: Summary of results from comparison of different number of hidden layers and perceptrons inside the hidden layers.

α **value**	**Training speed (s)**	**Training accuracy (%)**	**Testing accuracy (%)**
0.0001	4.6	85.1	72.5
0.01	3.9	84.2	72.9
1	0.4	75.3	72.6
3	0.3	74.7	71.7
0.1	0.9	76.4	73.2
(100, 100, 100, 100)	6.1	83.5	70.3

Table 4: Summary of results from comparison of α values of L2 regularization

6 Results

6.1 Results from experiments with Scikit-Learn MLP Classification

From these experiments described in experiment section we can conclude that best results were achieved with following parameters:

- Data size: 100 000 entries, 20% for validation
- Activation function for hidden layers: ReLU
- L2 regularization with $\alpha = 0.0001$
- Adam solver
- 3 hidden layers with 100 perceptrons each

This test had yielded following results

Table X Results for optimal parameters with smaller dataset Training accuracy (%) 76.2 Testing accuracy (%) 73.5

When the network was trained with all of our training data (X entries) out of which 20% was used for validation following results were obtained:

Table X Results for optimal parameters with whole data Training accuracy (%) 82.4 Testing accuracy (%) 72.4

Training took 11.1 seconds of time and it gave a confusion matrix shown in the figure 3

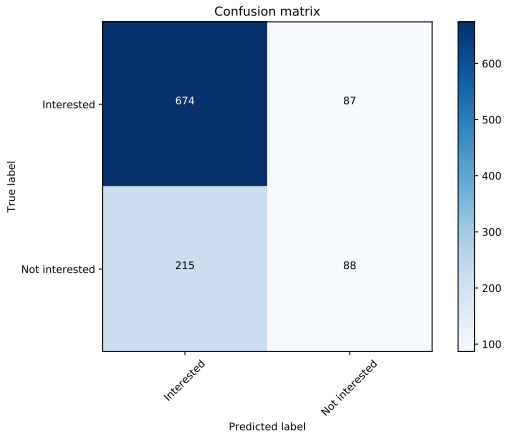

Figure 3: Confusion matrix of the results

From confusion matrix in Figure X we can see that our classifier gave a lot of false positives for events that users were actually not interested in, and was pretty good at identifying events that users were not interested in. This may be due to fact that most of users were uninterested in most of the events.

6.2 Results from experiments with Restricted Boltzmann Machine

We first used interest value of 0 to indicate users u_i disinterest to event e_j and the value of 0.5 when user was neither indicated disinterest or interest towards the event the user had attended. However, this approach didn't provide valuable results with RBM, and we supposed that this was due to not using only binary unit values as described in literature. Therefore, we end up choosing to set a value of 1 when the user was interested of the event and 0 otherwise. Even though the error function was clearly more consistently decreasing with more more training the results did not correlate either after the change.

7 Discussion

Given the nature of our data the results were okay. However the results could have been improved greatly by, for example, having a feature that would tell the distance between user's location and event's location instead of just boolean value that was true if the location was same and false otherwise.

The efficacy of the computations could have been improved by using a more wise filtering in compressing the size of the event dataset. For example using only the events that were included in the dataset used for training and testing, instead of only strictly reducing the amount of events. This would have been especially useful in the part of experimenting RBM, as it would have enabled using a larger set of users, batch size and epochs.

There was more data available then we had time to properly use. Processing the data and training with very large dataset took more time then we could spare. There are also some alternative methods that could have been used to recommend events. User data could be used for segmenting to get different user types. Then a predictor could be trained to recommend events for the user types instead of individual users, or alternatively, use both alongside each other. Events could be segmented as well as users, and events similar to ones the user has shown interest, could be recommended.

8 Conclusions

We tried to solve event recommendation task by using a classifier that predicts if the user would be interested in previously unseen events or not based on the context. We handpicked various sets of features for the predictor and trained it with a different number of hidden layers, different hidden layer activation functions, different solvers for weight optimization during training and different regularization methods. These attempts gave results with testing accuracy of about 70

References

[1] J. Lu, D. Wu, M. Mao, W. Wang, and G. Zhang, "Recommender system application developments: A survey," *Decision Support Systems*, vol. 74, pp. 12–32, 2015.

[2] S. Zhang, L. Yao, and A. Sun, "Deep Learning based Recommender System: A Survey and New Perspectives," vol. 1, no. 1, pp. 1–35, 2017.

[3] G. Adomavicius and a. Tuzhilin, "Toward the Next Generation of Recommender Systems: a Survey of the State of the Art and Possible Extensions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 734–749, 2005.

[4] H. Wang, N. Wang, and D.-Y. Yeung, "Collaborative deep learning for recommender systems," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1235–1244, ACM, 2015.

[5] A. Van den Oord, S. Dieleman, and B. Schrauwen, "Deep content-based music recommendation," in *Advances in neural information processing systems*, pp. 2643–2651, 2013.

[6] B. Wicht, A. Fischer, and J. Hennebert, "On CPU performance optimization of restricted Boltzmann machine and Convolutional RBM," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9896 LNAI, pp. 163–174, 2016.

[7] M. a. Carreira-Perpiñán and G. E. Hinton, "On Contrastive Divergence Learning," *Artificial Intelligence and Statistics*, vol. 0, p. 17, 2005.

[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[9] "kaggle - Event Recommendation Engine Challenge," 2012.

[10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.