Reddit text analysis for recommender systems

Polytechnique Montréal - LOG6308 Systèmes de recommandations

November 30 2019

Fabrice Charbonneau[1]

I. INTRODUCTION

A. Context

One of the most popular social forums agglomeration, news and discussions website on internet is Reddit. Users can share links, text posts, images and post comments on about 138 000 active communities called "subreddits". Some recommender systems have been integrated on the website, aiming to recommend relevant subreddits for its users. Other open-source projects developed by users themselves have been published and are able to recommend or help other users explore new subreddits and posts, based on specified interests.

On the other hand, a lot of progress regarding natural language processing has been made in recent years. Proper usage of those recent breakthroughs on text analysis could help improve performance of existing recommender systems in regards to recommandations on Reddit from text based posts and comments. This project aims to explore machine learning algorithms and perform classification on text samples.

B. Recommender system motivations

The motivation behind this project is about helping Reddit users find similar posts and recommending them subreddits where to post a text submission based on its text content. This could theoretically improve user experience as long as the recommendations keep a discrete presence on Reddit's interface. Users could write a text submission and, if unsure where it belongs, use subreddit recommendation to post it to a relevant community.

In order to provide recommendations, a simple solution could be to compare posts between other posts and comments, and recommend similar contents.

C. Dataset description

The data used for this project comes from a public repository on github (https://github.com/linanqiu/reddit-dataset). It contains around 2.6 millions text entries (either comments or posts) from 51 different subreddits, from 2012 to 2013. The subreddits are identified by 7 broad categories: entertainment, gaming, humor, learning, lifestyle, news and television. These can be used as labels and will be the variable of interest when performing classification.

[1]F. Charbonneau, Faculty of computer engineering, Polytechnique Montreal

II. LITERATURE REVIEW

Numerous research and development have been published regarding text analysis and NLP, and recommender systems for Reddit.

One notable article published this year from Andrej Janchevski and Sonja Gievska [1] presents several methods to implement a recommender system without interfering with user's anonymity. Discussed methods in the paper include network analysis of graphs and natural language processing of users' comments. In their experimentation, each recommendation was either "good" or "bad" which translates to binary classification. The most promising results were around 92% accuracy by using a combination of network analysis and text analysis to train a random forest classifier.

III. MODELS

For this project's models, three machine learning algorithms and variations will be used. The first one will be a basic k nearest neighbors classifier (KNN), and the last two ones are going to be deep learning classifier models with variations on the architectures and input formats.

A. Description of models

KNN (k nearest neighbors) algorithm computes distances between the point to predict x and other points with known labels. It then selects the k closest points to x and its prediction is the majority class of those neighbors.

The first deep learning approach takes preprocessed text posts and comments as input. This preprocessing (discussed in the next section) consists amongst other of TF-IDF and SVD. It then uses layers of ReLU neurons of which the amount and sizes is to be determined experimentally, followed by an output layer of 7 softmax neurons allowing classification of the 7 categories.

The second deep learning approach is similar to the first one, except it takes strings closer to a raw format as input. A layer of embedding then processes the input, which spits its output to a pooling layer that reduces the dimensions before the rest of the network. The behavior of the architecture after the pooling layer is the same as the first approach, although its optimal hyper-parameters are expected to be different than the later as the inputs are not in the same format.

The two neural network methods use the same optimizer and loss functions. The optimizer is Adam optimizer and the loss function is categorical crossentropy, which is what is usually used for multi-class classification.

B. Other relevant models

It is important to mention that there are other existing models that should be more efficient than what is used in this project. An important alternative to consider is BERT, a pre-trained model by Google that is considered to be state of the art. Results in this project are not expected to be ground breaking as the approaches used are probably not as performing as other are.

IV. Hypothesis

The hypothesis regarding which algorithm would give the best performance is based on recent breakthroughs in deep learning. The expected outcome is that the deep learning algorithms would out-perform the other methods. Because the classification here is a multi-class (7 categories) classification and not a binary one, we expect our performance to be quite lower than the results from previously mentioned article with an accuracy of around 92%.

A. Preliminary results

First, a very basic and rushed implementation was used in order to get a first batch of results and a sense of the general challenges to take care of. This gave results indicating that the deep learning approach were not as good as the basic KNN implementation; however the way the data was processed was simply not viable for the tested algorithms. Therefore the preliminary results with accuracy of 17% for deep learning and 50% for KNN showed during our in-class presentation were not representative of what the actual performances are. The results presented in this paper and in the attached jupyter notebook show more accurate performance metrics.

It was also not at all what the hypothesis was predicting and did not really make sense.

V. Methodology

In order to get relevant results from the models, a number of steps will be taken to ensure the whole process is reliable.

A. Data selection & filtering

The data was selected following these steps:

- remove posts and comments that has been deleted or removed
- remove empty posts and comments
- remove duplicated posts

Because the whole dataset contains about 2.6 millions of strings, the manipulation of such a high amount of data put heavy loads on machines used. Not a lot of resources was available for this project, which is why only a subset of the dataset was used. Choosing that dataset was another decision to be made.

First the dataset was sub sampled by randomly choosing 3000 entries from each subreddit, leaving a sub-dataset of 150 000 entries. Then, a further specific subset was selected based on the amount of words present in a single entry. The entries were kept only if it contained at least 10 words, leaving a filtered dataset of 28 745 entries.

A lot of information was present on the dataset, including number of upvotes, downvotes, gold given, username, etc. The only information kept was the category (class), the subreddit and of course the text itself.

Data was then filtered using the following rules:

- tokenize data by isolating each word
- stem each token
- remove punctuation
- remove digits
- remove links

Finally, labels of categories were processed with one-hot encoding.

B. Data transformation

Two main transformation approach will be used.

The first one, mentioned earlier, is performing TF-IDF and SVD on the strings. The amount of latent factors with SVD was determined empirically but without a rigorous method, as it would have taken too much time with available resources for the project. It is important to note that because the matrices were sparse, classic SVD was not used, and a variation called truncatedSVD provided by SKLearn was used instead. It will still be referred to as SVD for simplification.

The second transformation process was exclusively used for the deep learning implementation. It consists of constructing a dictionary based on all the unique words present in the data, then replacing tokens by their integer value in the dictionary. The value 0 is reserved in the dictionary for a padding token. Each entry, or sequence of tokens, have then been padded with 0 so that every sequence has an equal length. This length was lower than some sequences, which implies longer sequences have been truncated. This was one of the reasons our preliminary results were not accurate: the value of sequence length was set to the maximum sequence length in the data, which was 10 000. This made the model unable to extract any sort of meaning from the data.

C. Training and test sets

Each of the models will be trained on the same training set. The whole data is randomized then separated 30% for tests and 70% for training. The KNN model will be trained on the whole training set, while the deep learning models will be trained on 70% of the training set, the rest 30% will be their validation set. Each experiment always share the same sets.

D. Hyper-parameters wrapper approach

In general, the way the hyper-parameters were optimized were by trial and error, using a wrapper approach to a certain extent. For example, the determined value of K in KNN was found by iterating through different values and choosing the optimal number by measuring the accuracy of each model. One disadvantage of this approach is that it is very time and resource consuming, which is why the actual method used was more the one of a manual trial and error, mostly

for the deep learning algorithms, which are even more time demanding than KNN.

For the deep learning algorithms, the loss and accuracy metrics were plotted along with the number of epochs. The minimum loss value and maximum accuracy helped choosing the right amount of epochs to train the models. The batch size is also a useful parameter that was tweaked in order to get more accurate results faster by choosing how many entries are averaged before performing back propagation.

The models' accuracy are going to be compared to the proportion of the majority class, which in this case is the "gaming" category representing about 23% of the data.

VI. Implementation

The entirety of the project was developed using Python3 on a jupyter notebook. The libraries used for data preprocessing and basic classifiers such as KNN were Pandas, nltk, numpy and SKLearn. The deep learning and graphs were done using Keras/Tensorflow and Matplotlib.

Some operations have been parallelized on 6 cores, 12 threads on CPU Ryzen 5 1600.

A. KNN

By testing each value from 1 to 50 neighbors, the best one seemed to be only 1 neighbor. It is the only parameter that was ajusted for this method.

```
from sklearn.neighbors import KNeighborsClassifier
knn_classifier = KNeighborsClassifier(n_neighbors=1)
knn_classifier.fit(X_train, y_train)
```

B. Neural network with SVD input

Architecture of this NN is relatively simple: two hidden layers of 32 units with ReLU activation function before the final output layer.

```
model = keras.Sequential()
model.add(keras.layers.Flatten(input_shape=(LATENT_FACTORS,)))
model.add(keras.layers.Dense(32, activation=tf.nn.relu))
model.add(keras.layers.Dense(32, activation=tf.nn.relu))
model.add(keras.layers.Dense(7, activation=tf.nn.sigmoid))

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['acc'])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=128,
                    validation_data=(x_valid, y_valid))
```

C. Neural network with embedding

The first layer of embedding seems to give better results when the amount of latent dimensions is set to 32. Having tested multiple hidden layers pattern, only one layer of about 16 units seemed to achieve better results. Global average pooling is achieved right after the embedding to simplify embedding's output before ReLU layers.

```
model = keras.Sequential()
model.add(keras.layers.Embedding(vocab_size + 1, 32))
model.add(keras.layers.GlobalAveragePooling1D())
model.add(keras.layers.Dense(16, activation=tf.nn.relu))
model.add(keras.layers.Dense(7, activation=tf.nn.sigmoid))

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['acc'])

model.fit(partial_x_train,
          partial_y_train,
          epochs=20,
          batch_size=64,
          validation_data=(x_valid, y_valid))
```

VII. Results

A. Accuracy of models

Results were obtained by measuring accuracy for all models and loss for the deep learning models.

Table I shows the accuracy for each models.

model	KNN	DL SVD	DL Embedding
accuracy (%)	30.23	0.622	64.24

TABLE I
ACCURACY ACHIEVED

The deep learning models can achieve much higher accuracy than the KNN model. The two neural network gave about the same results, although the embedding seems slightly better than the SVD input.

B. Time Consumption

Time consumption varied a lot between the three models. Tables II, III and IV show wall time and user time on Ryzen 5 1600 CPU for TF-IDF and SVD as well as training and predicting time for the 3 models.

operation	TF-IDF	SVD	pre-embedding
wall time	34.1 s	7min 42s	635 ms
user time	33.9 s	26min 46s	572 ms

TABLE II
TIME CONSUMPTION FOR PROCESSING OPERATIONS

model	KNN	DL SVD	DL Embedding
wall time (s)	885 ms	18.4 s	1min 36s
user time (s)	876 ms	25 s	5min 19s

TABLE III
TIME CONSUMPTION FOR TRAINING

model	KNN	DL SVD	DL Embedding
wall time (s)	4min 50s	343 ms	406 ms
user time (s)	4min 49s	395 ms	520 ms

TABLE IV
TIME CONSUMPTION FOR PREDICTING

C. Loss and Accuracy for epochs

Figures 1 and 2 show the loss and accuracy of the model with SVD input at each epoch. This indicated the optimal number of epoch seems to be near 7 epochs. The same process is done for the embedding model.

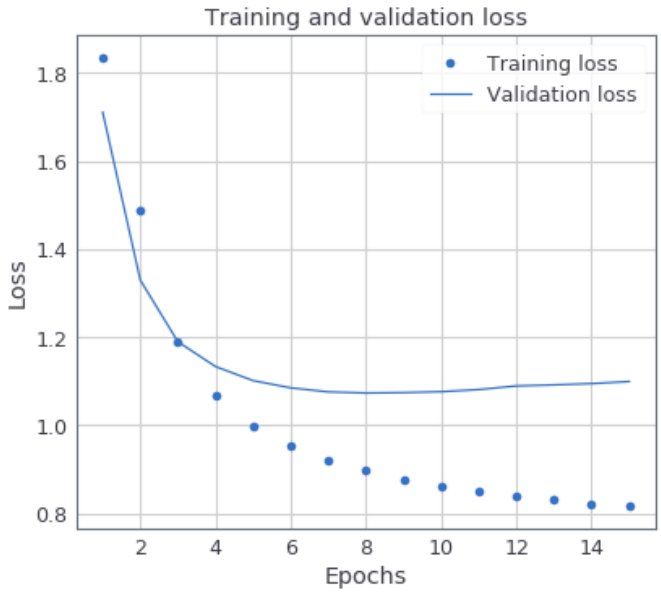

Fig. 1. Loss training and validation for SVD input DL model

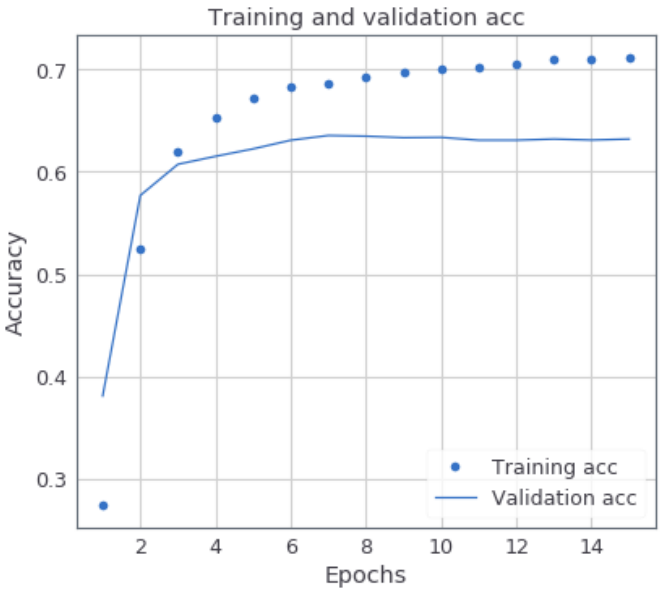

Fig. 2. Accuracy training and validation for SVD input DL model

Figures 3 and 4 show the loss and accuracy of the embedding model and suggest the model can still iterate over additional epochs without losing too much performance.

D. Latent Factors

Finally, another experiment made was to determine the optimal number of latent factors from SVD for the two models that use it. Figures 5 and 6 show the accuracy of models with the number of latent factors used for training data.

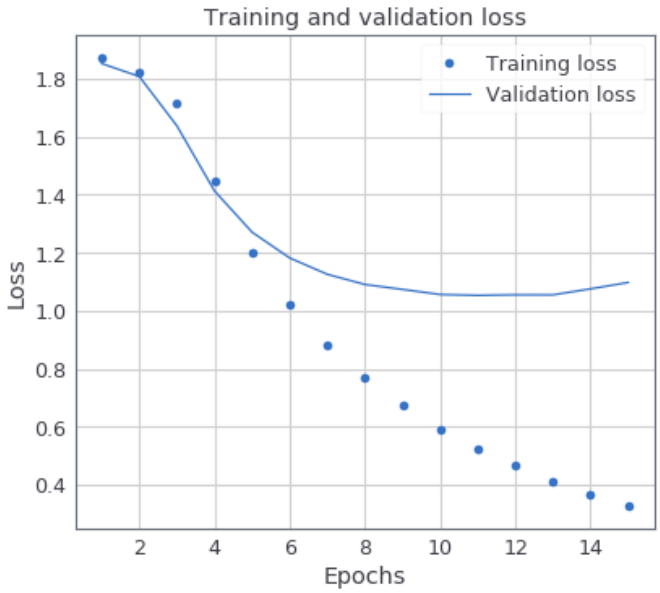

Fig. 3. Loss training and validation for embedding DL model

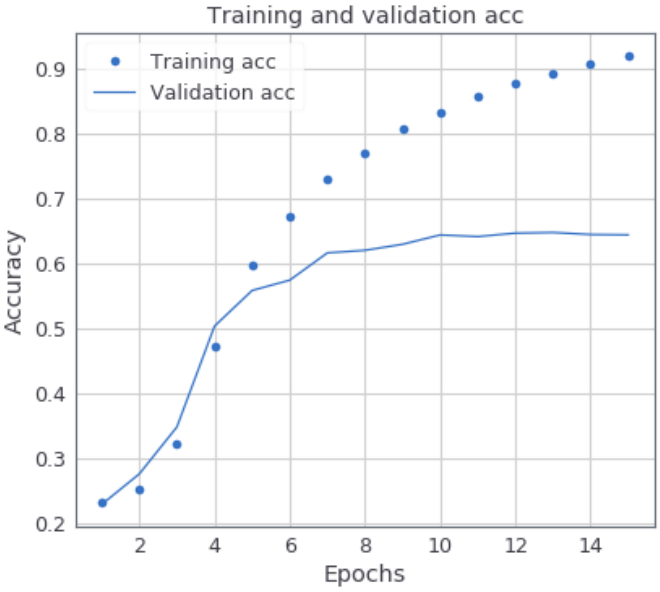

Fig. 4. Accuracy training and validation for embedding DL model

VIII. Analysis

A. Performance

It was reassuring to observe that the deep learning models were indeed more accurate than a knn algorithm. This was an expected result, but it wasn't expected that the two models would give such close performance. This was probably due to the data itself, with an accuracy of about 64% probably the best results that can be achieved using these methods. The embedding seemed to consistently give better results than NN with SVD input. Classification accuracy of 64% is subjectively pretty high considering the data consisted of random text posts and comments from Reddit and the fact

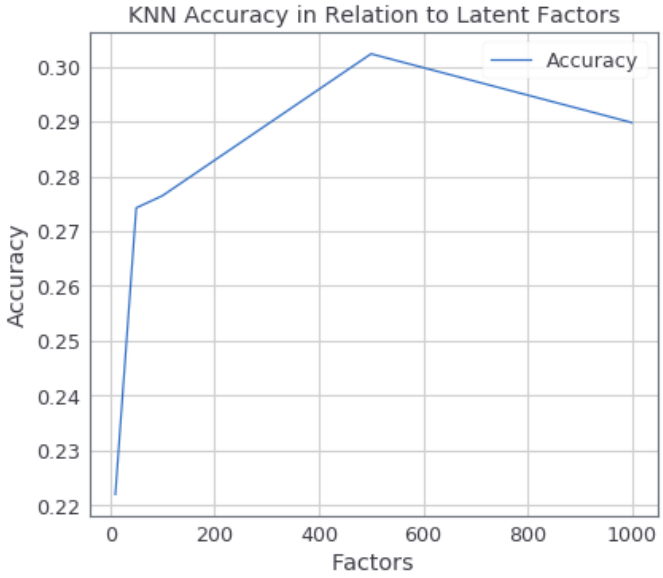

Fig. 5. Accuracy and latent factors for KNN

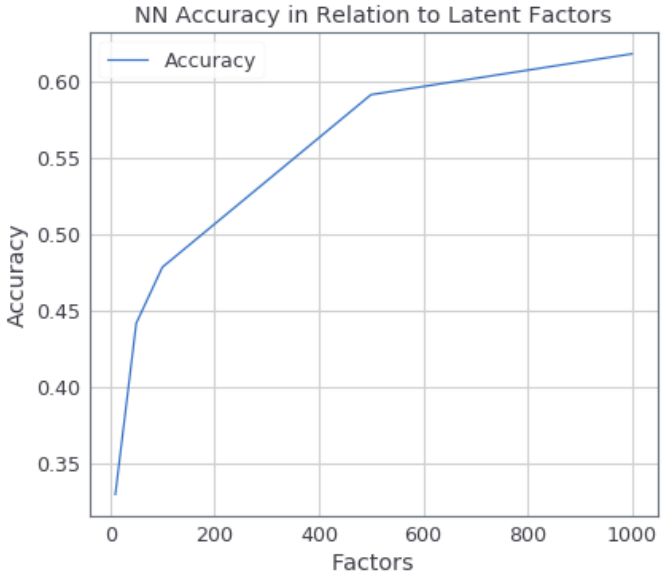

Fig. 6. Accuracy and latent factors for the neural network

that there is 7 classes.

B. Time Consumption

It is also interesting to note that the time consumption was very bad for KNN. Not only does it need time for SVD processing, it also takes a lot of time to predict the test. Deep learning SVD seems the fastest at first glance, but it needs to take into account the time spent for SVD processing. All things considered, the NN with embedding is the fastest model, and is also the most performing. This is probably due to the nature of embedding itself, which actively tries to make sense of a language like input instead of TF-IDF and SVD, that does not take into account within proximity of tokens.

C. SVD Latent Factors

For a deep learning approach, it seems like there is no virtual limit to the amount of latent factors, based on this experiment. Even after 1000 latent factors, it seems like it will continue to improve the accuracy as the model has enough complexity to handle additional information. However, it is not the case for KNN. The accuracy hits a peak of around 30% at 500 latent factors before dropping lower again at 1000 latent factors. KNN is too simple as an algorithm and does not guarantee a better performance even with more information.

D. Comparison with other results published in literature

The results are still far from the ones showed in [1]. As mentioned earlier, this is probably due to the fact that this is not a binary classification but a multiclass classification with 7 classes, which is fair to assume a lower accuracy will be obtained. Also, using state of the art methods like the BERT model, LSTMs, etc. would have help getting better results.

IX. Conclusion

As a conclusion, three machine learning algorithms were implemented and compared: KNN, neural network with same SVD input format as KNN and a second neural network approach taking raw data as input and using embedding instead of TF-IDF and SVD. The experiments showed that the neural networks performed better than the KNN approach, and that the embedding showed a slight increase in the accuracy while being faster. KNN only returned an accuracy a bit higher than the proportion of the majority class. Therefore, the best model found during this project is the neural network with embedding.

It would be interesting to do a similar work for Reddit but with images instead of text posts. As a very high percentage of Reddit's posts are images, it would make sense to implement such a model, though probably much more complex than what was accomplished in this project.

References

[1] Sonja Gievska Andrej Janchevski. A study of different models for subreddit recommendation based on user-community interaction. *Communications in Computer and Information Science*, 2019.