Information Retrieval - Assignment 1

Silvan Robert Adrianzlp432

June 12, 2019

Contents

Evaluate Existing Word Embeddings Models

Pretrained word embedding

I decided on using the `GoogleNews-vectors-negative300.bin` word embeddings. That word embedding got trained on Google news articles. It includes 3 million words over all put into vectors of size 300.

5 nearest neighbors

For the nearest neighbors search I used the `most_similar` function from the `gensim` package. But that one technically uses only the cosine similarity for finding similar words, so the euclidean distance I implemented myself by using the `scipy` package.

Nearest neighbors cosine similarity

Word	**Score**
Jackson	0.532635
Prince	0.530633
Tupou_V.	0.529283
KIng	0.522750
e_mail_robert.king_@	0.517362

Table 1: Similar words to King by cosine similarity

Word	**Score**
EURASIAN_NATURAL_RESOURCES_CORP.	0.673970
Londons	0.653613
Islamabad_Slyvia_Hui	0.637556
Wandsworth	0.613382
Canary_Wharf	0.611928

Table 2: Similar words to London by cosine similarity

Word	**Score**
Bad	0.617220
good	0.558616
Decent	0.516819
Better	0.503792
LAKE_WYLIE_Largemouth_Bass	0.500470

Table 3: Similar words to Good by cosine similarity

Word	Score
Apple_AAPL	0.745699
Apple_Nasdaq_AAPL	0.730041
Apple_NASDAQ_AAPL	0.717509
Apple_Computer	0.714597
iPhone	0.692427

Table 4: Similar words to Apple by cosine similarity

Nearest neighbors euclidean distance

Word	Score
Tupou_V.	1.9726125001907349
e_mail_robert.king_@	2.014012098312378
Singer_songwriter_Carole	2.0307531356811523
Geoffrey_Rush_Exit	2.0331478118896484
KIng	2.0411529541015625

Table 5: Similar words to King by euclidean distance

Word	Score
EURASIAN_NATURAL_RESOURCES_CORP.	2.0974204540252686
Sarah_Hills_FoodBizDaily.com	2.166471242904663
o2_arena	2.191667079925537
Cricklewood_north	2.198361396789551
Canary_Warf	2.202068567276001

Table 6: Similar words to London by euclidean distance

Word	Score
good	2.4779608249664307
LAKE_WYLIE_Largemouth_Bass	2.5142698287963867
Reprint_Practices	2.567570209503174
Harm_Than	2.5735108852386475
LAKE_HARTWELL_Largemouth_Bass	2.586975336074829

Table 7: Similar words to Good by euclidean distance

Word	**Score**
Apple_Nasdaq_AAPL	2.404542922973633
AAPL_PriceWatch_Alert	2.4422295093536377
Apple_AAPL	2.4825079441070557
RIM_NSDQ_RIMM	2.5051934719085693
NASDAQ_AAPL_iPhone	2.5330886840820312

Table 8: Similar words to Apple by euclidean distance

Conclusion

By now comparing the tables I can see that we mostly get similar words either with euclidean distance or cosine similarity, but cosine similarity seems to produce better results.

SVD plot

In this case I only took the first 500 words from the model and outputted it with SVD as the following plot. I created the plot by using numpy and the function `np.linalg.svd`. The Words get clustered according to their distance to each other, see in the plot for example numbers (2, 6 etc.) are relatively near to each other.

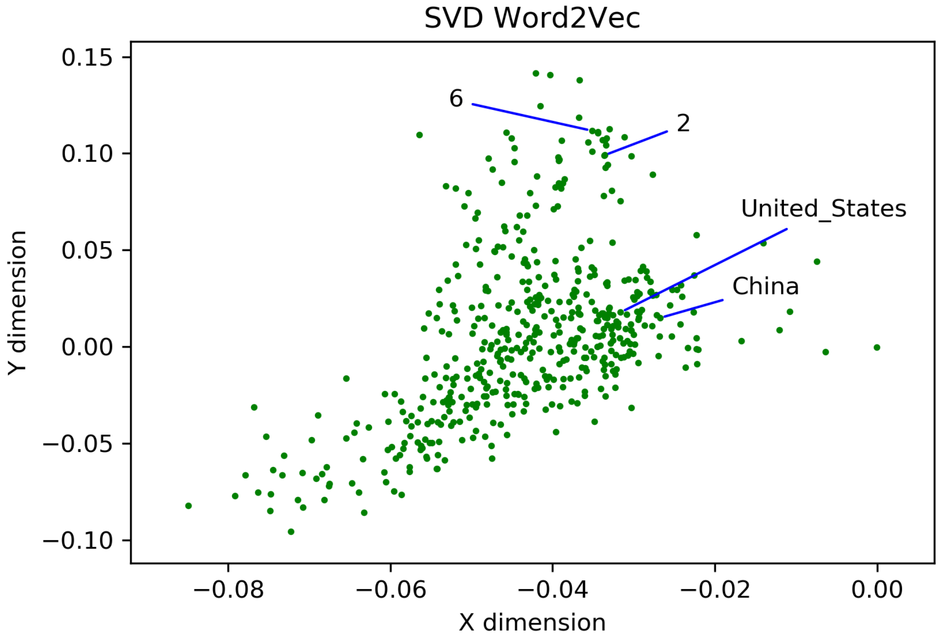

Figure 1: 500 words SVD plot

Main Task

Also for that task I used the `gensim` package again with the `most_similar` function. For getting the fourth word I only had to look at the distance between the first two words to be able to predict the fourth word. Example for predicting Iraq:

```
Athens Greece Baghdad X (Iraq)
X = Baghdad + Greece - Athens
```

Accuracy scores

I calculated the accuracy score in the end by `sklearns` accuracy_score function. The Predictions I set according to the highest similar word if it's equal to the one I expect otherwise not the right prediction.

Syntax	**Semantics**	**Overall**	**Average**
0.7400	0.7308	0.7358	0.7355

Table 9: Results

Word Embeddings for Text Classification

Also here using the embedding from the first task to do the text classifications.

I'm only taking the vectors of the embedding which are showing up in the texts, those vectors I use then to train a `AdaBoostClassifier`, this way I train the Classifier only on texts which have shown up and not on all 3'000'000 words. I create vectors for each document by taking the mean over all the words from the document to have a single vector for each document. But this representation is not that great since sentences with similar words will have the same vector even though they are totally different sentences. Word mover distance (WMD) or other representations will perform much better of course but for a baseline it seems sufficient.

Preprocessing

I also do some preprocessing on the data, for example remove punctuation which is not needed information, as well as stopwords (short words like to, and etc.) which also don't have a high value for representing a document. So I rather want to only get the important words which represent a document the best.

Model

As a model I used `AdaBoostClassifier`. I mostly decided for that model since it uses boosting (splitting up features over several estimators to hopefully get a better prediction in the end). So I ended up on `500` estimators, so the decision for predictions will be run on up to 500 estimators.

Accuracy

Overall the model seems to perform rather well with an accuracy of `81.1%` on the test data provided.

Contextually Propagated Term Weights for Document Representation

Data

For this task I used the reuters data set which consists of training and test data which include many articles and a label to it (for example tagged as trade etc.)

TF-IDF

For TF-IDF I just used the `sklearn` implementation to transform the training and the test data into TF-IDF. Then I used a `KNeighborsClassifier` as a model to train on TF-IDF and then make prediction with that model.

CPTW

For CPTW I implemented the version without IDF, where I first needed to create my unique words vocabulary. Which I built upon the worde2vec model (Google News) by only including the words which exist in the training and test collections. Further I created a $n \times n$ big similarity matrix to get the cosine similarities from word to word, on which I then can filter according to my t parameter.

Experiments

TF-IDF

For TF-IDF I only can tune one parameter and that's the k neighbors of kNN. So I used as $k = 1...19$ and according to the best result I chose my best k. This best k I choose according to cross-validation by 5 folds (training 4 folds, 1 fold validation) until I went through all 5 folds each. So I get a list with resulting scores from which I take the best score with the best k and then run the whole thing again on the whole training data (80% of whole data) and test data (20% of whole data). From which then I have a final result with my best found parameter k.

CPTW

Also for CPTW I go through the same process of getting the best parameters for k (also for kNN) and t (which is the threshold parameter of how many similar words should be chosen). So same story as before going through 5 folds, take the best parameters and then run it on the training and test data (which I haven't fully done since calculating CPTW took too long for all documents). As parameters I used for $k = 1..2$ and $t = 0.8, 0.9$, which is not a very big selection but the runtime is really bad so that I wasn't able to add many more tuning options.

Results

Now from running either TF-IDF with kNN and CPTW we get the following f1 scores. But on CPTW I only used a very small subset of data, so those numbers would be much interesting if I was ableto run it on the whole dataset.

Dataset	TD-IDF macro	TD-IDF micro	CPTW macro	CPTW micro
Reuters	0.8472	0.9244	result missing	result missing

Discussion

I would have gladly wanted to run the CPTW on the whole data set but sadly it took too long so I had to run it on a very small subset of the whole data, which of course also took a toll on the results and are not too useful. But the code is there to run all the experiments expect for that it took very long. Also as an addition would be to remove stopwords from the vocabulary which might help to achieve even a better performance.