CS410 Technology Review
Text Classification using StarSpace

Maximilian Trumpf, November 2018

Introduction

StarSpace was introduced by Facebook AI Research in November 2017 as a general-purpose neural embedding model. It can solve a wide variety of problems and is also stated to be outperforming more “traditional” approaches (such as TF-IDF) as well as other pioneering approaches (such as fastText) in many scenarios. However, so far only a limited set of usage-examples can be found besides the official StarSpace github page. This post is meant to serve as a practical additional reading to the StarSpace paper (see **Sources**) and therefore to help readers to understand how StarSpace can be used in practice.

As a practical problem this article will deal with the text classification of a collection of ~10.000 police reports from the German Berlin Police. The full dataset including a more detailed explanation of the data and the results of the text classification can be found on the following GitHub page which is also authored by the author of this post: https://github.com/MaxTru/Berlin-Police-Reports-NLP. For the classification task, the author labeled 750 police reports into the categories of:

1. Traffic Offense
2. Violent Crime
3. Criminal Damage or Fire

500 of the 750 labeled reports were used as training dataset, while 250 are used as test dataset. It needs to be highlighted and noted by the reader that this is a very small training and test dataset which clearly impacts the result.

Background

StarSpace uses embeddings on word, sentence or document level. In short, using embeddings these entities are represented as vectors. For example, the training dataset used in context of this post, produced the following embedding / 10-dimensional vector for the word “radfahrer” (engl. Cyclist):

```
radfahrer   -0.000656946   0.0274024  0.0678048  0.00220479 0.0137081
-0.0244335 0.041197   -0.00979346    -0.0189197 -0.0442938
```

The embedding of words with similar or the same meanings shall have close embeddings / vectors, whereas words with very different meanings shall have distant embeddings / vectors. Also, the difference of vectors also shall carry meaning. For example the embedding / vector for “Queen” minus the vector for “Woman” plus the vector for “Man” shall be close to the vector of “King”.

These embeddings or vectors can be trained using several approaches. A famous approach is the usage of Word2Vec. In Word2Vec a neural network with one hidden layer is used. The input layer consist of n neurons, where n equals the size of the vocabulary, same applies to the output layer. The hidden layer consist of i neurons, where i represents the dimensionality of the embeddings to be trained (in our example above i=10). Now, the core idea of embeddings is that the context of entities (e.g., words) is closely related to their meaning. This idea can forced into the hidden layer (and therefore into the vector) using the following approach:

- In the input layer, the neuron of the word for which the embedding is to be trained is set to 1, all other neurons are set to 0.
- In the output layer, the neurons of the words around the respective words (the words in context) are set to 1, all other neurons are set to 0.
- Now, since two words with similar or same meanings will have similar words in the output layer set to 1, the neural network must set the hidden layer of these similar words to similar values and therefore to close vectors.

Using this approach the hidden layer must carry the words' meaning since two words with a similar meaning in the input layer must produce a similar output in the output layer. This is based on the core idea that meaning is closely related to the context of a word. The hidden layer is therefore the result and later used as embedding.

As stated previously, StarSpace allows the embedding of words, sentences and documents. In the training process of a Text Classification problem, StarSpace will take into account the similarity of *a* and *b*, where *a* is a document and *b* is the correct label describing that document. Likewise it takes into account the similarity of *a* and *b-*, where *a* is again a document and *b-* is a randomly sampled incorrect label to that document. The vector of a document is represented as the sum of the respective words in the document (bag of unigram or n-gram words representation possible). The vector of a label is likewise represented as an embedding.

Practical Usage

After Git cloning StarSpace to your local machine, a fairly straight forward shell script is sufficient to prepare the data, train a model, test the model and apply it to the entire dataset. The full script is available here: https://github.com/MaxTru/Berlin-Police-Reports-NLP/blob/master/starspace/train_and_apply_model.sh, the key steps will be described in this post.

(All the following steps are performed in a directory where StarSpace was cloned to)

1. Building

While we need to build StarSpace, we also need to build query_predict which is later taking the trained model and is then used to predict the labels for the entire dataset.

```
# Build
echo "Building StarSpace..."
make
echo "Building StarSpace... Done"
echo "Building query_predict..."
make query_predict
echo "Building query_predict... Done"
```

2. Tokenization

Three input files are used in the sample shell script: the training, the test and the entire dataset. While the first two datasets are labeled, the third is obviously not labeled yet. In the files each line contains a document. A label is expressed as the first word of a line in the format of:

```
__label__1, <document text>
```

All three documents are tokenized including an all lowercase transformation and a separation of all words and punctuations using single white spaces:

```
# Normalize input files
echo "Normalizing input files..."
```

```
normalize_text() {
  tr '[:upper:]' '[:lower:]' | \
    sed -e "s/'/ ' /g" -e 's/"//g' -e 's/\./ \. /g' -e 's/<br \/>/ /g'
\
        -e 's/,/ , /g' -e 's/(/ ( /g' -e 's/)/ ) /g' -e 's/\!/ \! /g'
\
        -e 's/\?/ \? /g' -e 's/\;/ /g' -e 's/\:/ /g' | tr -s " "
}

cat $TRAINDOCS | normalize_text > $TRAINDOCS_NORM
cat $TESTDOCS | normalize_text > $TESTDOCS_NORM
cat $ALLDOCS | normalize_text > $ALLDOCS_NORM
echo "Normalizing input files... Done"
```

3. **Model training and model testing**

The model training and testing is very straightforward and is performed by using

```
./starspace train […]
./starspace test […]
```

The full list of parameters is not stated in this article but can be found in the linked script.

With the very small training dataset used in this post, the StarSpace models reaches a Hit@1 performance of 0.912.

4. Prediction of labels

Using the query_predict utility tool provided by the StarSpace team, the model is used to predict the labels of all documents of the dataset. Hereby the predictions are stored in a separate file:

```
# Predicitons for full dataset
echo "Making predictions for full dataset..."
./query_predict $MODEL 1 < $ALLDOCS_NORM | sed '/^$/d' | sed '/^Enter
some text/!d' | sed 's/Enter some text: 0\[[0-9].[0-9]*\]: //' >
$PREDICTIONS
echo "Making predictions for full dataset... Done"
```

Evaluation

StarSpace was easy to use to solve the described text classification problem. It needs to be considered that the dataset was relatively small (n=500 training and n=250 test documents). However, in this setup StarSpace could not achieve the praised high performance and could not outperform a more “traditional” Naïve Bayes Classifier. Detailed results:

StarSpace

```
Evaluation Metrics :
hit@1: 0.912 hit@10: 1 hit@20: 1 hit@50: 1 mean ranks : 1.12 Total
examples : 250
```

NaiveBayes

```
-------------------------------------------------------------
Class       F1 Score    Precision   Recall      Class Dist
-------------------------------------------------------------
__label__1  0.828       0.928       0.748       0.137
__label__3  0.95        0.94        0.961       0.612
__label__4  0.959       0.934       0.984       0.251
```

```
-------------------------------------------------------------
Total       0.937       0.937       0.937
-------------------------------------------------------------
750 predictions attempted, overall accuracy: 0.937
```

Clearly, the NaiveBayes classifier was able to outperform the StarSpace approach in this particular scenario (with a very small dataset and training data).

Also the tools to evaluate a model which are shipped with StarSpace were relatively limited in comparison to for instance the MeTa framework. So does StarSpace not offer K-Fold cross validation and also no detailed view on the evaluation results per class (see results above). For the optimization of more complex model, such a more detailed evaluation of StarSpace models would clearly be relevant.

Sources

1. Ledell Wu, Adam Fisch, Sumit Chopra, Keith Adams, Antoine Bordes, Jason Weston: StarSpace: Embed All The Things! (v5) | https://arxiv.org/abs/1709.03856
2. Macheads101 | https://www.youtube.com/watch?v=5PL0TmQhItY