IMDB_sent_an_baseline_models

August 14, 2018

1 Sentiment analysis on IMDB dataset: simple models

In this notebook, I will try to implement some basic models not relying on neural networks to perform sentiment analysis on the IMDB data set. This will give a baseline to measure the performance of my more advanced models. This script has been stronlgy inspired by the "nlp.ipynb" notebook from the fastai dl1 course (17.06.18 version).

1.1 Libraries

```
In [1]: from glob import glob
        import os
        import numpy as np
```

1.2 Loading the data

The data is located on my personal machine but is also available online.

```
In [2]: PATH = "/home/aritz/Documents/CS_Programming_Machine_Learning/Machine_learning_and_AI/On
        #PATH = "/home/aritz/Documents/CS_Programming_Machine_Learning/Machine_learning_and_AI/O
        TRAIN = PATH+'train/'
        TEST = PATH+'test/'
        TRAIN_ALL = PATH+'train/all/'
        TEST_ALL = PATH+'test/all/'
```

The next function was taken from the text.py file from fastai (17.06.18).

```
In [3]: def texts_labels_from_folders(path, folders):
            texts,labels = [],[]
            for idx,label in enumerate(folders):
                for fname in glob(os.path.join(path, label, '*.*')):
                    texts.append(open(fname, 'r').read())
                    labels.append(idx)
            return texts, np.array(labels).astype(np.int64)
```

The r in X_trn_r stands for raw, as the text is still not preprocessed.

```
In [4]: names = ['neg','pos']
        X_trn_r,y_trn = texts_labels_from_folders(TRAIN,names)
        X_val_r,y_val = texts_labels_from_folders(TEST,names)
```

1.3 Creating a BOW representation

1.3.1 BOW with only frequency

Here we build a BOW representation of the data. In this case every column displays how many times a word appears in the given text. And we are using 3-grams.

This is the size of the vocabulary. I took it from the fastai notebook. I have no idea how it is chosen.

```
In [5]: VOCAB_SIZE = 200000
```

The next variable represents the minimum frequency of a word among the different documents. Given the fact that we have 25K files, I guess than 0.1% should be reasonable.

```
In [6]: MIN_FREQ = 0.001
```

The next variable represents if the column for a token only represent the presence (1) or absence (0) of a word (i.e. BINARY = True) or if it counts the number of times a word appears in the given text (i.e. BINARY = FALSE). According to this paper (Section 2.1): https://www.aclweb.org/anthology/P12-2018 binarizing gives better results for Naive Bayes estimator.

```
In [7]: BINARY = False
```

```
In [8]: from sklearn.feature_extraction.text import CountVectorizer
```

This will be the tool to transform each text into a long vector which entries indicate how many times the corresponding words from the vocabulary (vocabulary which is also built during the fitting process) has been found in the given text.

```
In [9]: veczr_freq = CountVectorizer(ngram_range=(1,3), min_df=MIN_FREQ, max_features=VOCAB_SIZE
```

In the following cell, the training set is transformed into vectors and the vocabulary is built.

```
In [10]: X_trn_freq = veczr_freq.fit_transform(X_trn_r)
```

In the following cell, the validation set is transformed into vectors.

```
In [11]: X_val_freq = veczr_freq.transform(X_val_r)
```

Note that

```
In [12]: X_val_freq.shape
```

```
Out[12]: (25000, 45467)
```

This gives us the vocabulary.

```
In [13]: voc_freq = veczr_freq.get_feature_names()
```

1.3.2 BOW with binary entries

Here we build a BOW representation of the data, where every column indicates the presence or absence of a word/token/n-gram.

```
In [14]: veczr_bin = CountVectorizer(ngram_range=(1,3), min_df=MIN_FREQ, max_features=VOCAB_SIZE
```

```
In [15]: X_trn_bin = veczr_bin.fit_transform(X_trn_r)
```

```
In [16]: X_val_bin = veczr_bin.transform(X_val_r)
```

```
In [17]: voc_bin = veczr_bin.get_feature_names()
```

1.3.3 BOW with tf-idf coefficient

Here we build a BOW representation of the data, where every column indicates the tf-idf score of the word/token/n-gram for the given document.

```
In [18]: from sklearn.feature_extraction.text import TfidfTransformer
```

```
In [19]: tfidf_transformer = TfidfTransformer()
```

```
In [20]: X_trn_tfidf = tfidf_transformer.fit_transform(X_trn_freq)
```

```
In [21]: X_val_tfidf = tfidf_transformer.transform(X_val_freq)
```

1.3.4 All data sets

```
In [22]: X_trn_all = [X_trn_freq, X_trn_bin, X_trn_tfidf]
```

```
In [23]: X_val_all = [X_val_freq, X_val_bin, X_val_tfidf]
```

```
In [24]: data_names = ['BOW frequency', 'BOW binary', 'BOW tf-idf']
```

1.4 Classification model

1.4.1 Naive Bayes

```
In [25]: from sklearn.naive_bayes import MultinomialNB
```

```
In [26]: multi_nb = MultinomialNB()
```

1.4.2 Logistic Regression

```
In [27]: from sklearn.linear_model import LogisticRegression
```

```
In [28]: logreg = LogisticRegression()
```

1.4.3 All models

```
In [29]: models = [multi_nb, logreg]
```

```
In [30]: models_names = ['Multinomial Naive Bayes', 'Logistic Regression']
```

```
In [31]: logreg.fit(X=X_trn_freq, y=y_trn)
```

```
Out[31]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)
```

```
In [32]: y_val_pred = logreg.predict(X=X_val_freq)
```

```
In [33]: y_val_prob = logreg.predict_proba(X=X_val_freq)[:, 1]
```

```
In [34]: y_val_prob.shape
```

```
Out[34]: (25000,)
```

```
In [35]: y_val_pred
```

```
Out[35]: array([0, 0, 0, ..., 1, 1, 1])
```

1.5 Model evaluation

Now let's look at the performances of our different models:

```
In [36]: from sklearn.metrics import accuracy_score
         from sklearn.metrics import f1_score
         from sklearn.metrics import recall_score
         from sklearn.metrics import average_precision_score
         from sklearn.metrics import roc_auc_score
```

```
In [39]: def print_evaluation_scores(y_true, y_pred, y_prob):
             print('accuracy = ', accuracy_score(y_true=y_true, y_pred=y_pred))
             print('F1 score binary = ', f1_score(y_true=y_true, y_pred=y_pred, average='binary'
             print('Recall score = ', recall_score(y_true=y_true, y_pred=y_pred, average='macro'
             print('Average precision score = ', average_precision_score(y_true=y_true, y_score=
             print('Air under the ROC = ', roc_auc_score(y_true=y_true, y_score=y_prob))
```

Now we compare the different models.

```
In [40]: for i in range(len(data_names)):
             print(data_names[i])
             print('')
             X_trn = X_trn_all[i]
             X_val = X_val_all[i]
             for j in range(len(models_names)):
                 print(models_names[j])
```

```
            clf = models[j]
            clf.fit(X=X_trn, y=y_trn)
            y_pred = clf.predict(X=X_val)
            y_prob = clf.predict_proba(X=X_val)[:, 1]
            print_evaluation_scores(y_val, y_pred, y_prob)
            print('**********************')
        print('######################')
```

```
BOW frequency

Multinomial Naive Bayes
accuracy =  0.8608
F1 score binary =  0.85989210081327
Recall score =  0.8608
Average precision score =  0.9079606563446728
Air under the ROC =  0.9265069152
**********************
Logistic Regression
accuracy =  0.88716
F1 score binary =  0.8874346594309883
Recall score =  0.88716
Average precision score =  0.9500510501470008
Air under the ROC =  0.9531011584
**********************
######################
BOW binary

Multinomial Naive Bayes
accuracy =  0.87164
F1 score binary =  0.8716656668666266
Recall score =  0.87164
Average precision score =  0.927364166914979
Air under the ROC =  0.9388091423999999
**********************
Logistic Regression
accuracy =  0.89084
F1 score binary =  0.8912185594132419
Recall score =  0.8908400000000001
Average precision score =  0.9520401370998973
Air under the ROC =  0.9550804736
**********************
######################
BOW tf-idf

Multinomial Naive Bayes
accuracy =  0.87396
F1 score binary =  0.8741060369970833
Recall score =  0.8739600000000001
```

```
Average precision score =  0.9433772099774644
Air under the ROC =  0.946014144
**********************
Logistic Regression
accuracy =  0.89568
F1 score binary =  0.8962196577795464
Recall score =  0.89568
Average precision score =  0.9589803412241802
Air under the ROC =  0.9605377216000001
**********************
######################
```