Lesson 7- Sklearn and Randomforest

May 9, 2019

1 Ensemble methods

The goal of ensemble methods is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability / robustness over a single estimator.

Two families of ensemble methods are usually distinguished:

- Averaging methods: Here the driving principle is to build several estimators independently and then to average their predictions. On average, the combined estimator is usually better than any of the single base estimator because its variance is reduced. Example: Random forest
- Boosting methods: Here base estimators are built sequentially and one tries to reduce the bias of the combined estimator. The motivation is to combine several weak models to produce a powerful ensemble. Example: Adaboost

1.0.1 Random Forest classifier

In random forests, each tree in the ensemble is built from a sample drawn with replacement from the training set. In addition, when splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features. Instead, the split that is picked is the best split among a random subset of the features. As a result of this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree) but, due to averaging, its variance also decreases, usually more than compensating for the increase in bias, hence yielding an overall better model.

1.0.2 Sklearn

High level module built on NumPy, SciPy, and matplotlib. Scikit-learn is a Python module integrating a wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised problems. This package focuses on bringing machine learning to non-specialists using a general-purpose high-level language. Emphasis is put on ease of use, performance, documentation, and API consistency.

Step 1: Import data *Create dataframes from the csv data*

```
In [ ]: import pandas as pd
        import numpy as np
```

```
TRAIN_CSV = "C:\\Users\kmpoo\Dropbox\HEC\Teaching\Python for PhD May 2019\python4phd\S
dataframe = pd.read_csv(TRAIN_CSV, sep=",",error_bad_lines=False,header= 0, low_memory=
print(dataframe)
```

Step 2: Manipulate the dataframe and add new features *Create a new feature "length of the sentence"*

```
In [ ]: dataframe = dataframe.assign(nWords = lambda x : x['review_text'].str.split().str.len()
        dataframe['bi_senti'] = [ "positive" if x >= 4 else "negative" for x in dataframe['sen
        print(dataframe)
        print(dataframe['bi_senti'].value_counts())
```

Step 3: Split into train and test samples

```
In [ ]: from sklearn.utils import shuffle #To shuffle the dataframe
        from sklearn.model_selection import train_test_split

        dataframe = shuffle(dataframe)
        df_train, df_test = train_test_split(dataframe, test_size=0.2)
        print("size of trainig data ", len(df_train))
```

Step 4: Tokenization and veectorize the sentence Text data requires special preparation before you can start using it for predictive modeling.

The text must be parsed to remove words, called tokenization. Then the words need to be encoded as integers or floating point values for use as input to a machine learning algorithm, called feature extraction (or vectorization). ##### Convert sentences to TF-IDF vectors tf-idf (term frequency–inverse document frequency), is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.

```
In [ ]: from sklearn.feature_extraction.text import TfidfVectorizer


        def tfidf_vectorizer(corpus):
          """This function converts the input sentence into a sparse matrix of tfidf vectors""
          tokenizer = TfidfVectorizer( strip_accents = "unicode", analyzer="word", stop_words='
          tokenizer.fit(corpus)
          return tokenizer


        vectorizer = tfidf_vectorizer(dataframe['review_text'])
        train_x = vectorizer.transform(df_train['review_text'])
        test_x = vectorizer.transform(df_test['review_text'])

        print(train_x)
```

Step 5:Generate the classifier

```
In [ ]: from sklearn.ensemble import RandomForestClassifier

        rfc = RandomForestClassifier(n_estimators=10)
        classifier = rfc.fit(train_x, df_train['bi_senti'])
        acc = classifier.score(test_x,df_test['bi_senti'])
        print("accuracy of rfc is = ", acc)
        rfc.predict_proba(test_x)[0:10]
```

Step 6: Use the model to predict the sentiment of your sentence

```
In [ ]: s = pd.Series("his movie was absolutely horrible. A boring, random, nonsensical mess fi
        x = vectorizer.transform(s)
        print(rfc.predict_proba(x))
```