escudo_fc.jpg

Project 2 - Analysing Tweeter data

escudo_ct.jpg

Ali Nehrani

[a] *Ozyegin Universidad*

Resumen

The goal of this project was to predict sentiment for the given Twitter post using Python Data sciece packages. I implemented to data processing approaches; Kfold train-test split method and sklearn `train-test-split` method. I used 5 different types of data obtained from main data and different classification methods. Method score and accuracy is reported.

Contents

1. Used Python Libraries

In my code I used the following modules:

```python
import os
import nltk
import numpy as np
import re as regex
import pandas as pd
from collections import Counter
import matplotlib.pyplot as plt
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import f1_score,
    precision_score, recall_score, accuracy_score
from sklearn.model_selection import train_test_split
    , cross_val_score, GridSearchCV,
    RandomizedSearchCV
from sklearn import metrics, svm, model_selection,
    feature_extraction, linear_model, tree, ensemble
    , neural_network
```

2. Evaluating Function

I developed the following code for evaluating classification methods over the data. This is based on the similarity if applying different classification methods in `sklearn`.
I this function **dev** I used "Kfold" from sklearn `model-selection` module to split data into training and test and k is set to 10. In the following I calculated accuracy score with different measures including: "metrics.accuracy-score", "precision-score", "recall-score", "accuracy-score" and "f1-score" in kfold case. At the end I returned the average of the score called kfold score.
In this **dev** the `train-test-split` method from "model-selection" is also applied. I computed method score "clss.score".

```python
def cl_ts_evaluation(clss, k, data, target, method=
    None, test_ratio=0.25):
    metric_ac_score = []
    precision = []
    recall = []
    accuracy = []
    f1 = []
    if method == 'kfold':
        kfold = model_selection.KFold(n_splits=k)
        for ind_train, ind_test in kfold.split(data):
            clss.fit(data[ind_train], target[
    ind_train])
            y_pred = clss.predict(data[ind_test]) #
            metric_ac_score.append(metrics.
    accuracy_score(y_pred, target[ind_test]))
            precision.append(precision_score(y_pred,
     target[ind_test], average=None, pos_label=None)
    )
            recall.append(recall_score(y_pred,
    target[ind_test], average=None, pos_label=None))
            accuracy.append(accuracy_score(y_pred,
    target[ind_test]))
            f1.append(f1_score(y_pred, target[
    ind_test], average=None, pos_label=None))
            return [np.mean(metric_ac_score), np.
    mean(precision), np.mean(recall), np.mean(
    accuracy), np.mean(f1)]

    else:
        for I in range(k):
            x_train, x_test, y_train, y_test =
    model_selection.train_test_split(data,target,
    test_size=test_ratio)
            clss.fit(x_train,y_train)
            metric_ac_score.append(clss.score(x_test
    ,y_test))

    return np.mean(metric_ac_score)
```

Correo electrónico: (Ali Nehrani)

3. Preprocessing

3.1. Loading Data

Loading data is simple: first we provide the address of the data file in the following lines:

```
fileAddress=os.path.join(os.getcwd(), 'twitter_sentiment_corpus.csv')
print('Start to loading first data from this file: \n', fileAddress)
```

Then the `.csv` is red:

```
tweetsData = pd.read_csv(fileAddress)
```

Finally I used stemmer

```
stemmer=nltk.PorterStemmer()
df["TweetText"] = list(map(lambda str: stemmer.stem(str.lower()), df["TweetText"]))
```

3.2. Data prepration

In this section I first removed the improper data i.e. the data in which the emotion is not equal to "positive", "negetive" or "neutral":

```
tweetsData = tweetsData.drop(tweetsData[tweetsData['Sentiment'] == 'irrelevant'].index)
```

The I tried to remove unavailable data, numbers and spaces by the following commands respectively:

```
df = df[df["TweetText"] != "Not Available"]
r_num = regex.compile(r"\s?[0-9]+\.?[0-9]*")
df.loc[:, "TweetText"].replace(r_num, "", inplace=True)
```

Also the punctuation is removed by the following command:

```
for remove in map(lambda r: regex.compile(regex.escape(r)), [",", ":", "\"", "=", "&", ";", "%", "$","@", "%", "^", "*", "(", ")", "{", "}","[", "]", "|", "/", "\\",">", "<", "-","!", "?", ".", "'","--", "---", "#"]):
df.loc[:, "TweetText"].replace(remove, "", inplace=True)
```

The distribution of the emotions is computed by counting number of each emotion in data:

```
negetive = len(df[df["Sentiment"] == "negative"])
positive = len(df[df["Sentiment"] == "positive"])
neutral = len(df[df["Sentiment"] == "neutral"])
t_size = negetive + positive + neutral
dist_negetive = negetive/t_size
dist_positive = positive/t_size
dist_neutral = neutral/t_size
```

In order to enumerate the target, in the following I changed the emotions into numbers; `positive` to 1, `negetive` to -1, and `neutral` to 0:

```
row_index = tweetsData[tweetsData['Sentiment'] == 'positive'].index
tweetsData.loc[row_index, 'Sentiment'] = 1
row_index = tweetsData[tweetsData['Sentiment'] == 'neutral'].index
tweetsData.loc[row_index, 'Sentiment'] = 0
row_index = tweetsData[tweetsData['Sentiment'] == 'negative'].index
tweetsData.loc[row_index, 'Sentiment'] = -1
```

Determining the data and target:

```
tweetData = np.array(tweetsData['TweetText'])
tweetTarget = np.array(tweetsData['Sentiment'].values, dtype="|S6")
```

constructing a stopwords by using `corpus.stopwords.words` from `nltk` package:

```
stopwords=nltk.corpus.stopwords.words("english")
```

4. Input Data Generating

In order to prepare input data for processing, which helps to increase the accuracy, I used vectorizatio method. One proper vectorization method in `sklearn` is `TfidfVectorizer` vectorized from `feature-extraction.text`. This process is done in the following lines and we get **Data-1**:

```
# TfidfVectorizer
tfid=feature_extraction.text.TfidfVectorizer(use_idf=True,sublinear_tf=True, stop_words=stopwords) #'english'
data_1=tfid.fit_transform(tweetData)
```

Data-2 is generated by using `bigram TfidfVectorizer` method; I set `ngram-range=(1, 2)` and also token pattern `r'_+_'`. Regular expression denoting what constitutes a "token", only used if `analyzer == 'word'`.
When building the vocabulary ignore terms that have a document frequency strictly lower than the threshold `min-df=1`.

```
tfid2=feature_extraction.text.TfidfVectorizer(use_idf=True, sublinear_tf=True, ngram_range=(1, 2), token_pattern=r'\b\w+\b', min_df=1, stop_words=stopwords)#'english'
data_2=tfid2.fit_transform(tweetData)
```

Data-3 is generated by using `CountVectorizer` method which Convert a collection of text documents to a matrix of token counts. This implementation produces a sparse representation of the counts using `scipy.sparse.csr-matrix`. Prameters are left as default.

```
countVectorize = feature_extraction.text.CountVectorizer(stop_words=stopwords) #'english'
data_3=countVectorize.fit_transform(tweetData)
```

Data-4 is generated by using bigram `CountVectorizer` method in which the bigram parameters are considered: Token pattern and threshold is set as Data-2 case:

```
#  bigram vectorizer
bigram_vectorizer = feature_extraction.text.CountVectorizer(ngram_range=(1, 2), min_df=1, stop_words=stopwords)#'english'
data_4 = bigram_vectorizer.fit_transform(tweetData)
```

Data-5 is generated by using bigram `HashingVectorizer` method. `HashingVectorizer` converts a collection of text documents to a matrix of token occurrences.
It turns a collection of text documents into a scipy.sparse matrix holding token occurrence counts.
This text vectorizer implementation uses the hashing trick to find the token string name to feature integer index mapping.
This strategy has several advantages:

- it is very low memory scalable to large datasets as there is no need to store a vocabulary dictionary in memory
- it is fast to pickle and un-pickle as it holds no state besides the constructor parameters
- it can be used in a streaming (partial fit) or parallel pipeline as there is no state computed during fit.

[Ref. from sklearn website]

```
#  HashingVectorizer
hashingVectorizer = feature_extraction.text.
     HashingVectorizer(n_features=100)
data_5 = hashingVectorizer.fit_transform(tweetData)
```

The following part of the code computes classification and prodiuces the results:

```
for meth in method:
    i=0
    for model in models:
        for dat in data:
            print("Model: {}===>".format(str(type(
    model).__name__), data_ind[i], meth))
            print("---------------------------------"
    )
            print("Data: {}".format(data_ind[i]))
            print("---------------------------------"
    )
            print("Method: {}".format(meth))
            print("---------------------------------"
    )
            accuracy = cl_ts_evaluation(clss=model,
    k=10, data=dat, target=tweetTarget, method=meth)
            if meth == 'kfold':
                print("          ----- Results
    -------              ")
                print("
                     ")
                print("metric_ac_score: " + str(
    accuracy[0]))
                print("Precision: " + str(accuracy
    [1]))
                print("Recall: " + str(accuracy[2]))
                print("Accuracy:  " + str(accuracy
    [3]))
                print("F1: " + str(accuracy[4]))
                print("
    ============================================="
    )
            else:
                print('Accuracy of the method: \t\t
    {0:.4f}'.format(accuracy))
                print("
    ***********************************************************
    ")
        i+=1
```

5. Results

In this report I applied the following classification methods:

```
models = [linear_model.LogisticRegression(),
          svm.LinearSVC(),
          tree.DecisionTreeClassifier(max_depth=10),
          ensemble.RandomForestClassifier(),
          BernoulliNB()]
```

The following results are obtained in the case of using `train-test-split` method of sklearn: In the following I report three maximum scores that I obtained during computation:

```
1. Model: LinearSVC
--------------------------------
Data: data_1
--------------------------------
Accuracy of the method:     0.7669
**************************************
2. Model: LogisticRegression
--------------------------------
Data: data_4
--------------------------------
Accuracy of the method:     0.7614
**************************************
3. Model: LinearSVC
--------------------------------
Data: data_4
--------------------------------
Accuracy of the method:     0.7600
**************************************
4. Model: LinearSVC
--------------------------------
Data: data_2
--------------------------------

Accuracy of the method:     0.7595
*************************************
5. Model: LogisticRegression
--------------------------------
Data: data_3
--------------------------------
Accuracy of the method:     0.7592
**************************************
```

In the following I report the results of train test splitting by Kfold. I used different approaches here which all are reporting the accuracy of the classification. First I want to describe the scores;
`recall score`: The recall is the ratio $\frac{tp}{tp+fn}$ where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples.
`metrics.accuracy-score`: Accuracy classification score. In multilabel classification, this function computes subset accuracy: the set of labels predicted for a sample must exactly match the corresponding set of labels in y-true.
`metrics.accuracy-score`: The precision is the ratio $\frac{tp}{(tp+fp)}$ where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative.
`F1 score`: Compute the F1 score, also known as balanced F-score or F-measure The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

$$F1 = 2 * \frac{(precision * recall)}{(precision + recall)}$$

In the multi-class and multi-label case, this is the weighted average of the F1 score of each class.

```

==================================================
Model: LogisticRegression==>
---------------------------------
Data: data_4
---------------------------------
Method: kfold
---------------------------------
Precision: 0.63963963964
==================================================
Model: LinearSVC==>
---------------------------------
Data: data_2
---------------------------------
Method: kfold
---------------------------------
Precision: 0.638888888889
==================================================
Model: LinearSVC==>
---------------------------------
Data: data_4
---------------------------------
Method: kfold
---------------------------------
Precision: 0.617486338798
==================================================
Model: RandomForestClassifier==>
---------------------------------
Data: data_3
---------------------------------
Method: kfold
---------------------------------
Precision: 0.604166666667
==============================================
Model: RandomForestClassifier==>
---------------------------------
Data: data_4
---------------------------------
Method: kfold
--------------------------------
Precision: 0.607142857143
```