Documentation

April 5, 2020

1 Dialect Discrimination

1.1 Overview

"In the Romanian Dialect Identification (RDI) shared task, participants have to train a model on tweets. Therefore, participants have to build a model for a in-genre binary classification by dialect task, in which a classification model is required to discriminate between the Moldavian (label 0) and the Romanian (label 1) dialects.

The training data is composed of 7757 samples. The validation set is composed of 2656 samples. All samples are preprocessed in order to replace named entities with a special tag: NE.

Samples File:

Each line represents a data sample where:

The first column shows the ID of the data sample. The second column is the actual data sample.

Labels File

Each line represents a label associated to a data sample where:

The first column shows the ID of the data sample. The second column is the actual label."

1.2 Imported libraries

Library used for reading data

[12]:
```
import pandas as pd
```

Library used for turning the collection of data into numerical feature vectors

[13]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

Library used for Grid searching parameters for svm

[27]:
```
from sklearn.model_selection import GridSearchCV
```

Libraries used for the chosen model and computing its accuracy

[14]:
```
from sklearn import metrics, svm, linear_model
from sklearn.ensemble import BaggingClassifier
```

Library used for plotting the confussion matrix

[15]:
```
import matplotlib.pyplot as plt
```

1.3 Data reading

- header = None: the .txt file do not contain the name of the columns
- delimiter = '\t': the data read is separated by tab space

[16]:
```
# read training data

train_samples = pd.read_fwf('data/train_samples.txt', header = None, delimiter␣
↪= '\t')
train_labels = pd.read_fwf('data/train_labels.txt', header = None, delimiter =␣
↪'\t')



# read validation data

validation_samples = pd.read_fwf('data/validation_samples.txt', header = None,␣
↪delimiter = '\t')
validation_labels =  pd.read_fwf('data/validation_labels.txt', header = None,␣
↪delimiter = '\t')



# read test data

test_samples = pd.read_fwf('data/test_samples.txt', header = None, delimiter =␣
↪'\t')
```

Next, two data frames are built, for train and validation data in which each row comprises a tweet and its corresponding label. Lastly, the 'Corpus' data frame is created for final validation and training.

[17]:
```
texts = []
labels = []

# build dataframe for train data

for i  in range(len(train_samples)):
        texts.append(train_samples[1][i])
        labels.append(train_labels[1][i])

trainDF = pd.DataFrame()
trainDF['text'] = texts
trainDF['label'] = labels
trainDF['text'] = [entry.lower() for entry in trainDF['text']]
```

```
texts.clear()
labels.clear()

# build dataframe for validation data

for i  in range(len(validation_samples)):
        texts.append(validation_samples[1][i])
        labels.append(validation_labels[1][i])

validationDF = pd.DataFrame()
validationDF['text'] = texts
validationDF['label'] = labels
validationDF['text'] = [entry.lower() for entry in validationDF['text']]


## concatenate train and validation data for final training

frames = [trainDF, validationDF]
Corpus = pd.concat(frames)

print(trainDF.head())
```

```
                                                text  label
0  ;%fe mr#& crmx temjc@m %'wb: }hham@@m ykm=aa e…      1
1  safw k#xk}t fh@ae m&xd >h& @# l@rd}a @hc lit e…      1
2  zghy% @ka qcrw h@@m he|%wa eh}w@m mkzrmaah@ @(…      1
3  !ck& g@eah =f; me @hc zk&} mk@eahh jmjaafm >cg…      1
4  zpw hjreaek egae h: (avny }e m@p: ejfmz @x<yn …      0
```

1.4 Text Represenation

For extracting features from text, tf-idf from sklearn.feature_extraction.text is used.

- analyzer = 'char': the feature is made of character n-grams.
- ngram-range = (4, 5): indicates that 4-grams and 5-grams will be considered.
- max_df = 0.25: ignore terms that appear in more than 25% of the documents.
- min_df = 2: ignore terms that appear in less than 2 documents.
- max_features = 6000: build a vocabulary that only considers the top 6000 features ordered by term frequency across the corpus.
- sublinear_tf = 1: use a logarithmic form for frequency.

[18]:
```
# characters level tf-idf

tfidf_vect_ngram_chars = TfidfVectorizer(analyzer = 'char', ngram_range = (4,␣
↪5), max_df = 0.25, min_df = 2, max_features = 6000, sublinear_tf = 1)

tfidf_vect_ngram_chars.fit(Corpus['text'])
```

```
xtrain_tfidf_ngram_chars = tfidf_vect_ngram_chars.transform(trainDF['text'])

xvalid_tfidf_ngram_chars = tfidf_vect_ngram_chars.
 ↪transform(validationDF['text'])

xcorpus_tfidf_ngram_chars = tfidf_vect_ngram_chars.transform(Corpus['text'])

xtest_tfidf_ngram_chars = tfidf_vect_ngram_chars.transform(test_samples[1])
```

1.5 Model Training

train_model function prints the accuracy, confussion matrix and F1-score of the classifier, trained on feature_vector_train with the corresponding train_label, validated on feature_vector_valid with its correspoding valid_label. 'test_or_valid' parameter indicates wether the predictions should be written in a .csv file(validate on test_data) or print the model's accuracy(validate on validation_data)

[19]:
```
# Utility function for training the model and computing its accuracy

def train_model(classifier, feature_vector_train, label, valid_label,␣
 ↪feature_vector_valid, test_or_valid):

    # fit the training dataset on the classifier

    classifier.fit(feature_vector_train, label)


    # Plot non-normalized confusion matrix
    titles_options = [("Confusion matrix, without normalization", None),
                      ("Normalized confusion matrix", 'true')]

    for title, normalize in titles_options:
        disp = metrics.plot_confusion_matrix(classifier, feature_vector_valid,␣
 ↪valid_label,
                                 display_labels = [0, 1],
                                 cmap = plt.cm.Blues,
                                 normalize = normalize)
        disp.ax_.set_title(title)

        print(title)
        print(disp.confusion_matrix)

    plt.show()

    # predict the labels on validation dataset
```

```
    predictions = classifier.predict(feature_vector_valid)

    # write predictions in .csv file

    if test_or_valid == 'test':
        prediction = pd.DataFrame(predictions, test_samples[0]).
↪to_csv('prediction.csv', index_label = 'id', header = ['label'])

    if test_or_valid == 'validation':
        # compute F1-score

        print("F1-score: ", metrics.f1_score(valid_label, predictions,␣
↪average='macro'))

        # compute accuracy

        print("accuracy: ", metrics.accuracy_score(predictions, valid_label))
```

1.5.1 SVM

SVM parameters:

- C = 10: penalty parameter of the error term. After grid search, 10 is the value that rendered the highest accuracy out of [1, 10, 100, 1000]
- gamma = 1.0: After grid search, out of [1e-4, 1.0] 1.0 rendered the highest accuracy

1.5.2 BaggingClassifier

BaggingClassifier parameters:

- base_estimator= svm.SVC(): SVC will be fitted on random subsets of the dataset
- n_estimators = 30: number of base classifiers
- random_state = 0: seed used by the random number generator
- bootstrap_features = True: features will be drawn with replacement

Perform GridSearch

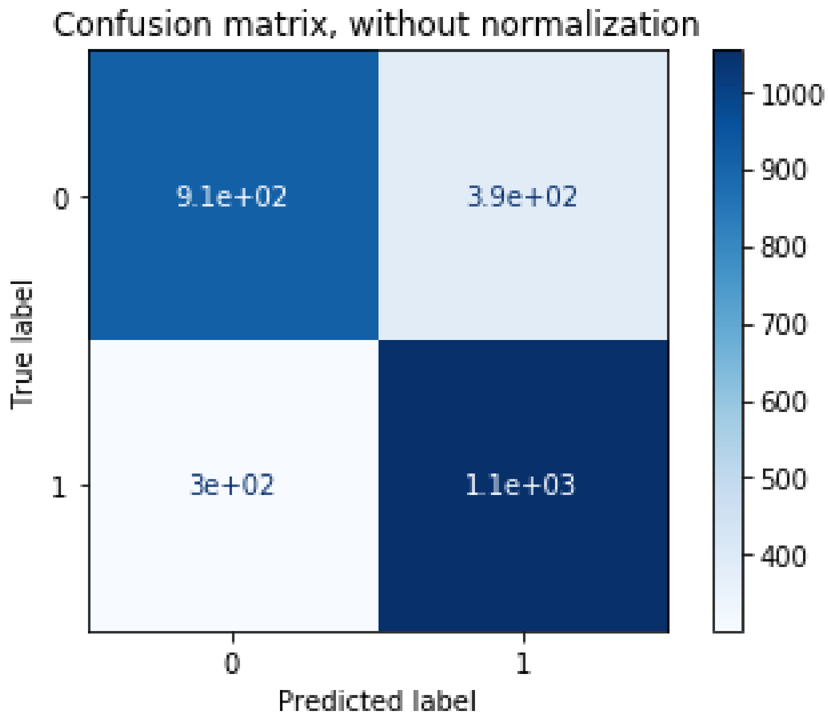
[28]:
```
# svm
clf_ = svm.SVC(kernel='rbf')
hyperparameters = {
                        'C' : [1, 10, 100, 1000],
                        'gamma' : [ 1e-4, 1.0]
                  }
clf = GridSearchCV(clf_, hyperparameters, cv = 3)

clf.fit(xtrain_tfidf_ngram_chars, trainDF['label'])

print("best parameters for svm: ", clf.best_params_)
```

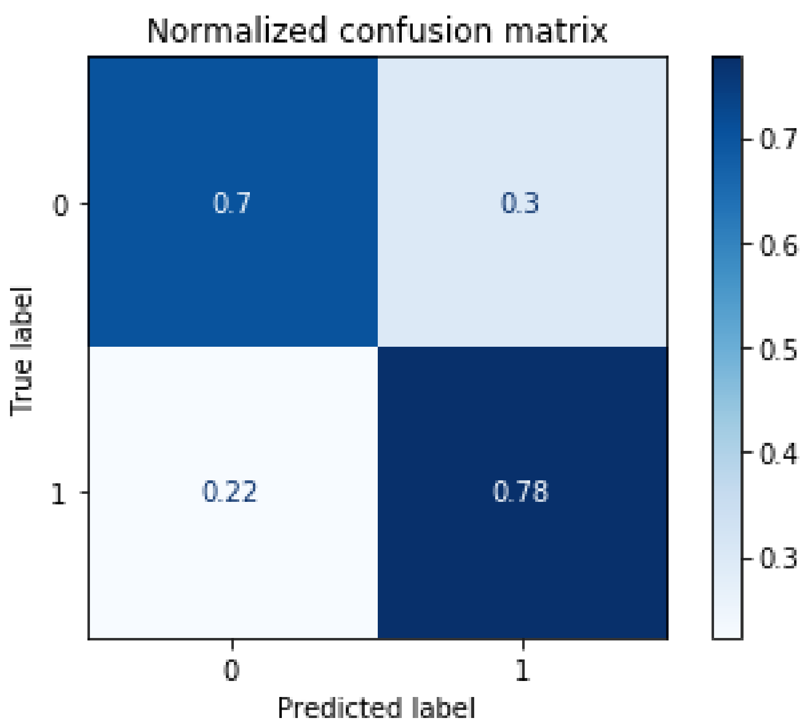
```
# baggingClasiifier
clf_ = BaggingClassifier(random_state = 0, bootstrap_features = True)
hyperparameters = {
                        'n_estimators' : [ 10, 20, 30]
                  }
clf = GridSearchCV(clf_, hyperparameters, cv = 3)

clf.fit(xtrain_tfidf_ngram_chars, trainDF['label'])

print("best parameters for BaggingClassifiers: ", clf.best_params_)
```

```
best parameters for svm:  {'C': 10, 'gamma': 1.0}
best parameters for BaggingClassifiers:  {'n_estimators': 30}
```

[16]:
```
train_model(BaggingClassifier(base_estimator= svm.SVC(C = 10, gamma = 1.0),␣
↪n_estimators = 30, random_state= 0, bootstrap_features = True),␣
↪xtrain_tfidf_ngram_chars, trainDF['label'], validationDF['label'],␣
↪xvalid_tfidf_ngram_chars, 'validation')

# number of tweets in each class
print(validationDF['label'].value_counts())
```

```
Confusion matrix, without normalization
[[ 914  387]
 [ 302 1053]]
Normalized confusion matrix
[[0.70253651 0.29746349]
 [0.22287823 0.77712177]]
```


Confusion matrix, without normalization
9.1e+02
3.9e+02
3e+02
1.1e+03
True label
Predicted label
0
1
0
1
1000
900
800
700
600
500
400

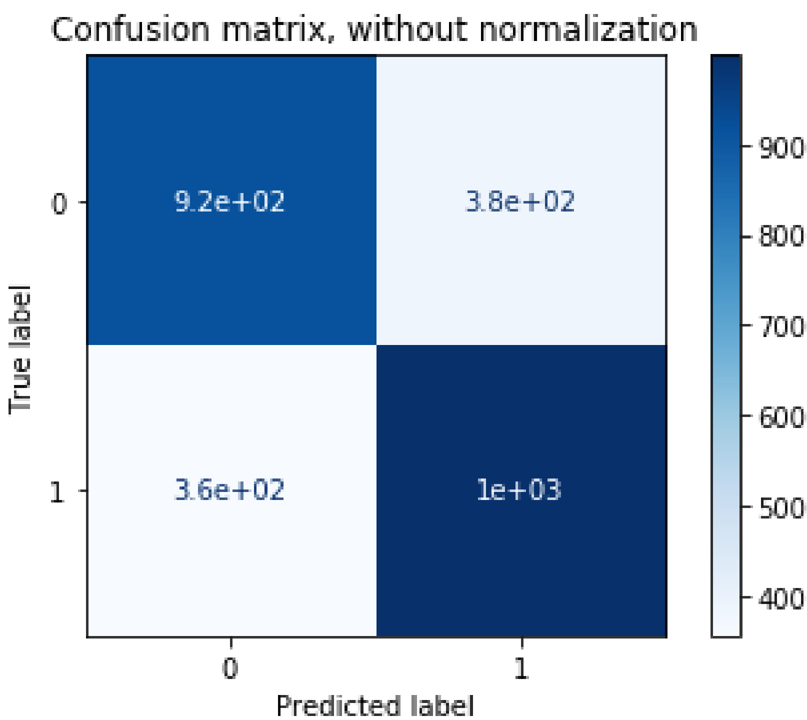

Normalized confusion matrix
0.7
0.3
0.22
0.78
True label
Predicted label
0
1
0
1
0.7
0.6
0.5
0.4
0.3

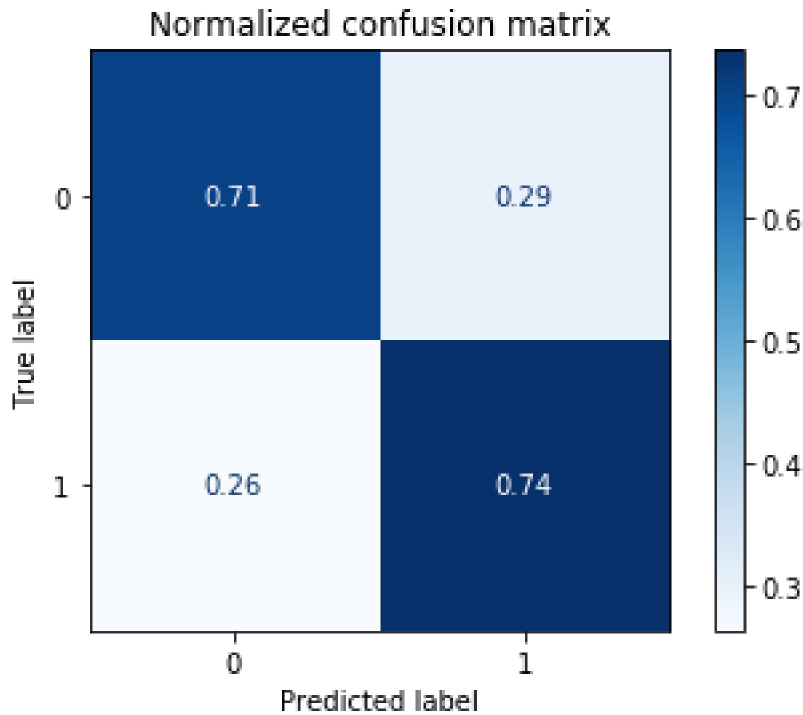
```
F1-score:  0.7398748972105958
accuracy:  0.7405873493975904
1    1355
0    1301
Name: label, dtype: int64
```

1.5.3 Logistic Regression

[21]:
```
train_model(linear_model.LogisticRegression(C = 10), xtrain_tfidf_ngram_chars,␣
↪trainDF['label'], validationDF['label'], xvalid_tfidf_ngram_chars,␣
↪'validation')
```

```
Confusion matrix, without normalization
[[918 383]
 [356 999]]
Normalized confusion matrix
[[0.70561107 0.29438893]
 [0.26273063 0.73726937]]
```






```
F1-score:  0.7215030275512484
accuracy:  0.7217620481927711
```