Comp135-ML Project2 Report

1. Pre-requred

● version and package

Python (Version >=3.3, Version 3.7 used in this project)
Numpy package
String package // for string strip
Re package // for string strip
matplotlib package// for plot the error curve with hyper-parameters
To install matplotlib, type in the terminal:
```
$pip3 install  matplotlib
```
For more infomation, please check the website below:
https://matplotlib.org/users/installing.html
sklearn package // for model selection, cross-validation and bag-of-words vectorizer
To install sklearn, type in the terminal:
```
$pip3 install  scikit-learn
```
For more infomation, please check the website below:
http://scikit-learn.org/stable/install.html
keras package // for word-embedding
To install keras, type in the terminal
```
$pip3 install  keras.
```

● how to compile this project

```
$cd /directory/code.py
$python3 code.py
```
Tips:
1. Please include the train, test data files in the same directory as this program.
2. During the program running, it will generate some curve pictures. Turn them down, the program will continue and pictures will be saved in the same directory automatically.
3. If you want to change the train dataset, test dataset, pre-train method, model, train-test-ratio, please edit the code for Myclassifier parameters setting which is in the **main** function.

2. Experiment

● pre-train method

In this project, I use two pre-train methods. The one is bag-of-words, another is word-embedding. Let me briefly explain my approaches to use these two methods.

bag-of-words
function in code :
def read_data_bag_of_words_function(self, datafile_name):

First, I clean the whole dataset. If the dataset is train dataset, I take out the label and save it in Y, then exclude the punctuations, numeric in the sentences. If the dataset is test dataset, exclude the punctuations, numeric in the sentences directly.
Second, I use the CountVectorizer to convert all the vocabularies into a row, and the row contains all different vocabularies occurring in the dataset, which as the features, but exclude the useless words like “ the”, “a” …. And for each sentence, it has a corresponding number in the row, which represents the word frequency occurring in sentences.
Finally, I use Tfidftransformer to scale the data in (0,1) distribution and save in array X. If datafile is train data, return [X, Y], if datafile is test data, return X.

Note: if the model is Naive Bayes and adaboost the X should be binary, which I have done in the code.

word-embedding
function in code :
def read_data_embedding_function(self, datafile_name):

First, I clean the whole dataset. If the dataset is train dataset, I take out the label and save it in Y, then exclude the punctuations, numeric in the sentences. If the dataset is test dataset, exclude the punctuations, numeric in the sentences directly.
Second, I use tokenizer, get the number of words in corpus. And store the keys of each words in encode which is corresponding the sentences order. Then I use pad to fulfill the array, each sentence' key row have the same length.
Third, I make a dictionary from the glove.6B.50d.txt, find the value for each key. The value contains 50 features. Then, for each row, I sum up the vectors of words, save it as X.
Finally, If the dataset is train data, word-embedding, return [X,Y]. If the dataset is test data, return X.

● Learning Algorithm

I choose four learning algorithm in my project, including SVM, Adaboost, Logistic Regression and Naive Bayes.

SVM
parameters [{‘kernel’: [‘linear'], 'gamma': [0.7]}]
hyper-parameters [{‘C': [1, 10, 100, 1000]}]
soft margin : C affects the trade-off between complexity and proportion of nonseparable samples and must be selected by the user.”[1] or more specifically the trade-off between errors on training data set and margin maximization. A smaller C will allow more errors and margin errors and usually produce a larger margin. When C goes to Infinity, svm becomes a hard-margin. Note that the value of C is chosen by users based on training experiments.
here I select [1, 10, 100, 1000], from larger margin to small.

Adaboost
parameters [{“base_estimator__criterion": [“entropy”], "base_estimator__splitter": [“best”]}]
hyper-parameters ["n_estimators": [20, 30, 50, 100]}
I use the Decisiontree for base estimators, the parameters are following:
DecisionTreeClassifier(random_state=11, max_features="auto", max_depth=None)
adaboost hyper-parameter is n_estimators, which represents the number of base estimators, if n_estimator is small, it will become a normal classifier, and if the n_estimator is large, the model will be complicated and overfit. Here I select [20, 30, 50 ,100], from simple model to complicate

Logistic Regression
parameters [{‘penalty’:[’l2’]}]
hyper-parameters [{‘C': [1, 10, 100, 1000]}]
Logistic regression hyper-parameter : The trade-off parameter of logistic regression that determines the strength of the regularization is called C, and higher values of C correspond to less regularization (where we can specify the regularization function).C is actually the Inverse of regularization strength(lambda). Here I select [1, 10, 100, 1000] from more regularization to less.

Naive Bayes
parameters {default}
hyper-parameters [{'alpha': [1, 5, 10, 20]}]

Naive Bayes hyper-parameter: Laplace smoothing , which is a technique for smoothing categorical data. A small-sample correction, or pseudo-count, will be incorporated in every probability estimate. Consequently, no probability will be zero. this is a way of regularizing Naive Bayes, and when the pseudo-count is zero, it is called Laplace smoothing.

● Model Selection

model selection function code :

def fit_choose_hyperparamter(self):

call the function in main function :

classifier = MyClassifier(['trainreviews.txt', 'testreviewsunlabeled.txt', 'bag-of-words', 'SVM', 50, 0.75])

classifier.fit_choose_hyperparamter()

we can change the Myclassifier setting to make different experiments

# pre-train method:['bag-of-words','word-embedding']

# model:['SVM', 'adaboost', 'Logistic Regression', 'Naive Bayes']

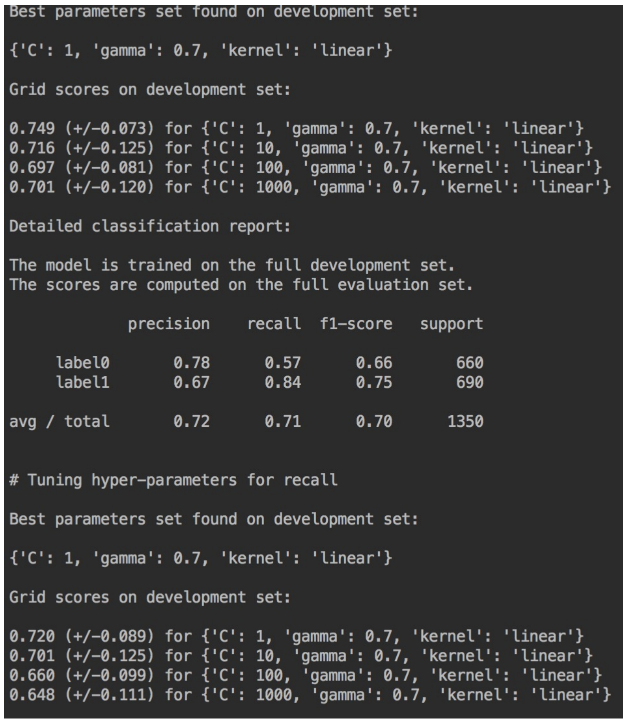
use cross-validation in GridSearchcv, train-validation ratio is 5. We can get mean, std, average accuracy for train, validation. And we also report the precision, f1-score, recall.

SVM

parameters [{‘kernel’: [‘linear'], 'gamma': [0.7]}]

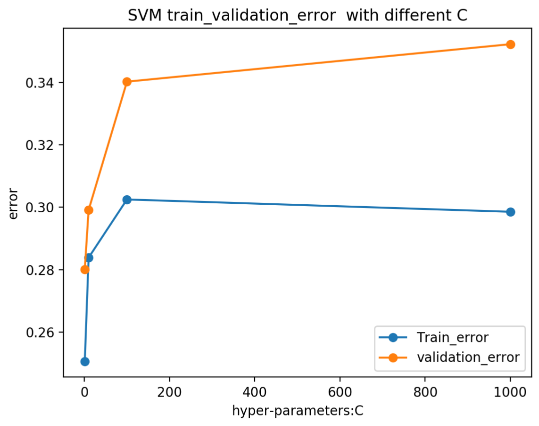
hyper-parameter : [{‘C': [1, 10, 100, 1000]}]

using bag-of-words pre-train, the performance as following:



```
Best parameters set found on development set:

{'C': 1, 'gamma': 0.7, 'kernel': 'linear'}

Grid scores on development set:

0.749 (+/-0.073) for {'C': 1, 'gamma': 0.7, 'kernel': 'linear'}
0.716 (+/-0.125) for {'C': 10, 'gamma': 0.7, 'kernel': 'linear'}
0.697 (+/-0.081) for {'C': 100, 'gamma': 0.7, 'kernel': 'linear'}
0.701 (+/-0.120) for {'C': 1000, 'gamma': 0.7, 'kernel': 'linear'}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

             precision    recall  f1-score   support

     label0       0.78      0.57      0.66       660
     label1       0.67      0.84      0.75       690

avg / total       0.72      0.71      0.70      1350


# Tuning hyper-parameters for recall

Best parameters set found on development set:

{'C': 1, 'gamma': 0.7, 'kernel': 'linear'}

Grid scores on development set:

0.720 (+/-0.089) for {'C': 1, 'gamma': 0.7, 'kernel': 'linear'}
0.701 (+/-0.125) for {'C': 10, 'gamma': 0.7, 'kernel': 'linear'}
0.660 (+/-0.099) for {'C': 100, 'gamma': 0.7, 'kernel': 'linear'}
0.648 (+/-0.111) for {'C': 1000, 'gamma': 0.7, 'kernel': 'linear'}
```

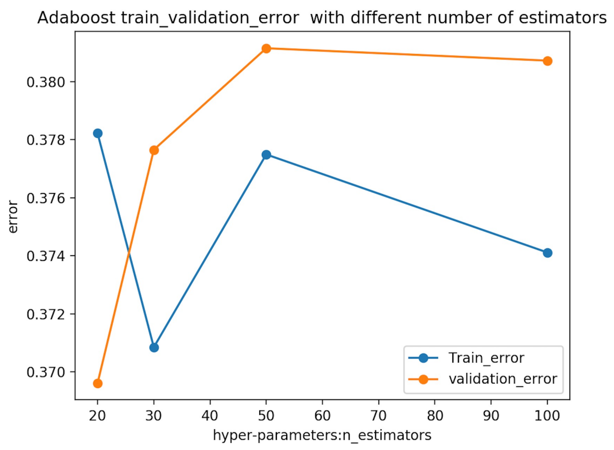
picture1. SVM performance (bag)



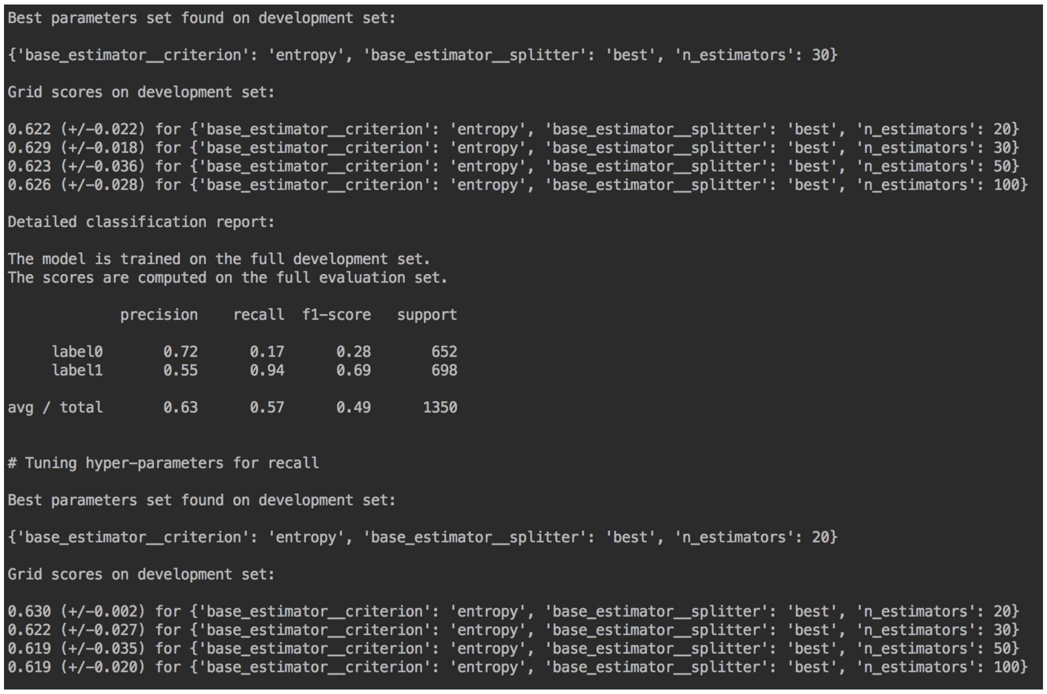
picture2. SVM error-hyper:C curve (bag)

Note: the word-embedding method is a time consuming method for SVM, so I didn’t use it for SVM.

Adaboost
parameters   [{“base_estimator__criterion": [“entropy”], "base_estimator__splitter": [“best”]}]
hyper-parameters   ["n_estimators": [20, 30, 50, 100]}
using bag-of-words pre-train, the performance as following:



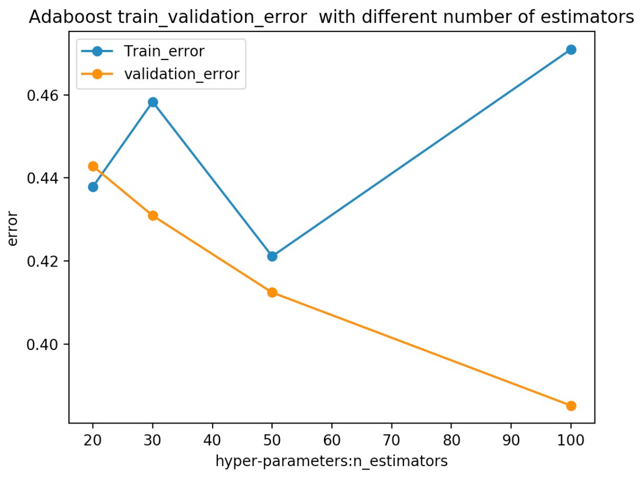
picure3. adaboost error-hyper:n_estimators curve (bag)



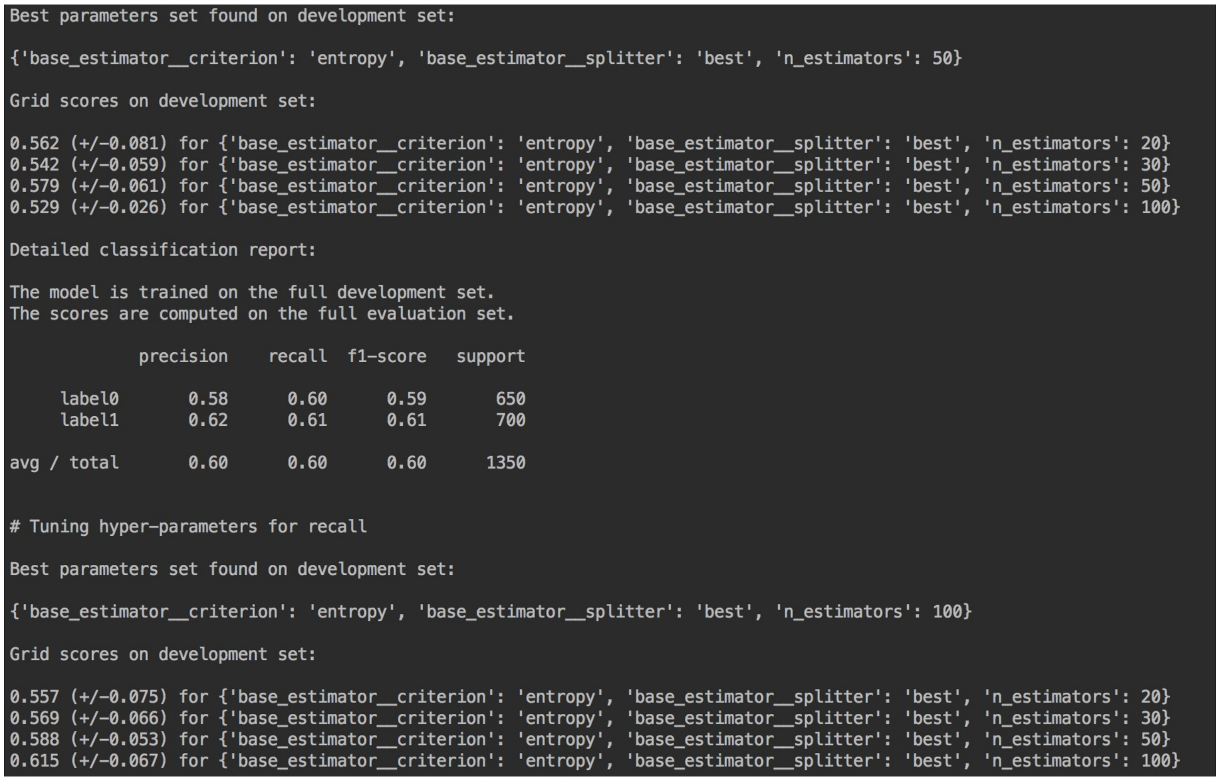
```
Best parameters set found on development set:

{'base_estimator__criterion': 'entropy', 'base_estimator__splitter': 'best', 'n_estimators': 30}

Grid scores on development set:

0.622 (+/-0.022) for {'base_estimator__criterion': 'entropy', 'base_estimator__splitter': 'best', 'n_estimators': 20}
0.629 (+/-0.018) for {'base_estimator__criterion': 'entropy', 'base_estimator__splitter': 'best', 'n_estimators': 30}
0.623 (+/-0.036) for {'base_estimator__criterion': 'entropy', 'base_estimator__splitter': 'best', 'n_estimators': 50}
0.626 (+/-0.028) for {'base_estimator__criterion': 'entropy', 'base_estimator__splitter': 'best', 'n_estimators': 100}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

             precision    recall  f1-score   support

     label0       0.72      0.17      0.28       652
     label1       0.55      0.94      0.69       698

avg / total       0.63      0.57      0.49      1350


# Tuning hyper-parameters for recall

Best parameters set found on development set:

{'base_estimator__criterion': 'entropy', 'base_estimator__splitter': 'best', 'n_estimators': 20}

Grid scores on development set:

0.630 (+/-0.002) for {'base_estimator__criterion': 'entropy', 'base_estimator__splitter': 'best', 'n_estimators': 20}
0.622 (+/-0.027) for {'base_estimator__criterion': 'entropy', 'base_estimator__splitter': 'best', 'n_estimators': 30}
0.619 (+/-0.035) for {'base_estimator__criterion': 'entropy', 'base_estimator__splitter': 'best', 'n_estimators': 50}
0.619 (+/-0.020) for {'base_estimator__criterion': 'entropy', 'base_estimator__splitter': 'best', 'n_estimators': 100}
```

picture4. adaboost performance(bag)

using word-embedding method, the performance as following:



picture5. adaboost error-hyper:n_estimators curve (embedding)

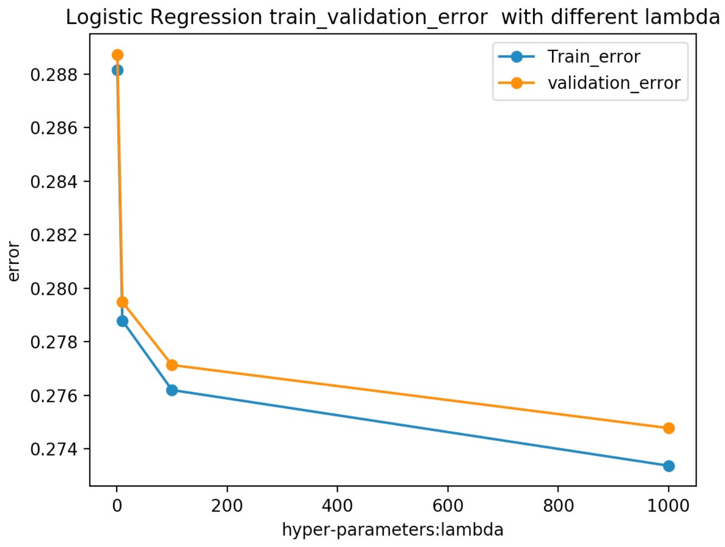


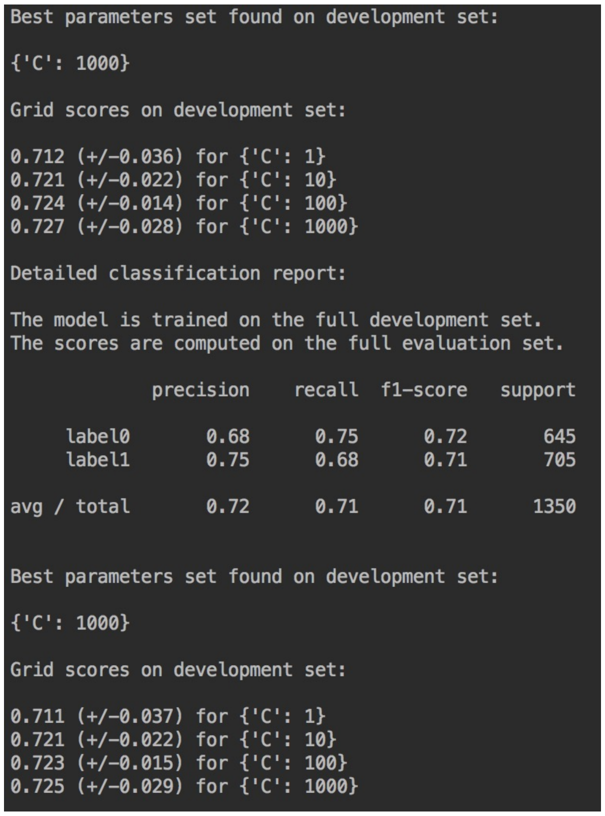
```
Best parameters set found on development set:

{'base_estimator__criterion': 'entropy', 'base_estimator__splitter': 'best', 'n_estimators': 50}

Grid scores on development set:

0.562 (+/-0.081) for {'base_estimator__criterion': 'entropy', 'base_estimator__splitter': 'best', 'n_estimators': 20}
0.542 (+/-0.059) for {'base_estimator__criterion': 'entropy', 'base_estimator__splitter': 'best', 'n_estimators': 30}
0.579 (+/-0.061) for {'base_estimator__criterion': 'entropy', 'base_estimator__splitter': 'best', 'n_estimators': 50}
0.529 (+/-0.026) for {'base_estimator__criterion': 'entropy', 'base_estimator__splitter': 'best', 'n_estimators': 100}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

             precision    recall  f1-score   support

     label0       0.58      0.60      0.59       650
     label1       0.62      0.61      0.61       700

avg / total       0.60      0.60      0.60      1350


# Tuning hyper-parameters for recall

Best parameters set found on development set:

{'base_estimator__criterion': 'entropy', 'base_estimator__splitter': 'best', 'n_estimators': 100}

Grid scores on development set:

0.557 (+/-0.075) for {'base_estimator__criterion': 'entropy', 'base_estimator__splitter': 'best', 'n_estimators': 20}
0.569 (+/-0.066) for {'base_estimator__criterion': 'entropy', 'base_estimator__splitter': 'best', 'n_estimators': 30}
0.588 (+/-0.053) for {'base_estimator__criterion': 'entropy', 'base_estimator__splitter': 'best', 'n_estimators': 50}
0.615 (+/-0.067) for {'base_estimator__criterion': 'entropy', 'base_estimator__splitter': 'best', 'n_estimators': 100}
```

picture6. adaboost performance(embedding)

Note: for adaboost algorithm using word-embedding, the accuracy is low and curve for error-hyperparameter is weird. I guess the reason is adaboost using decision tree classifier as base estimator, so the feature need be category. But I find it is difficult to make a category when using word-embedding method.

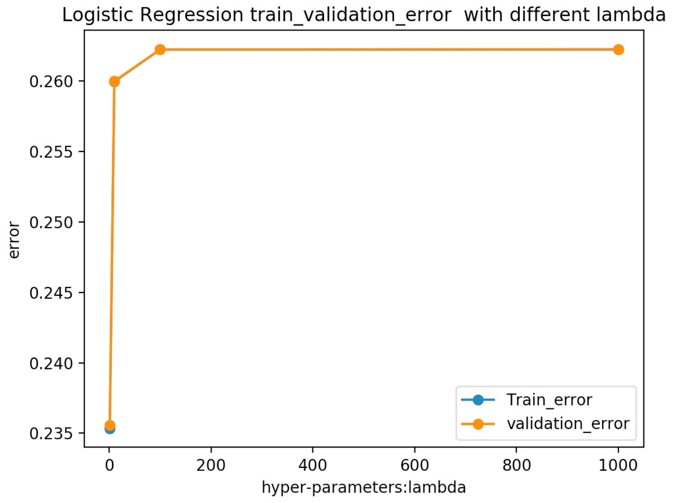
Logistic Regression
parameters [{‘penalty’:[’l2’]}]
hyper-parameters [{‘C': [1, 10, 100, 1000]}]
using bag-of-words method:

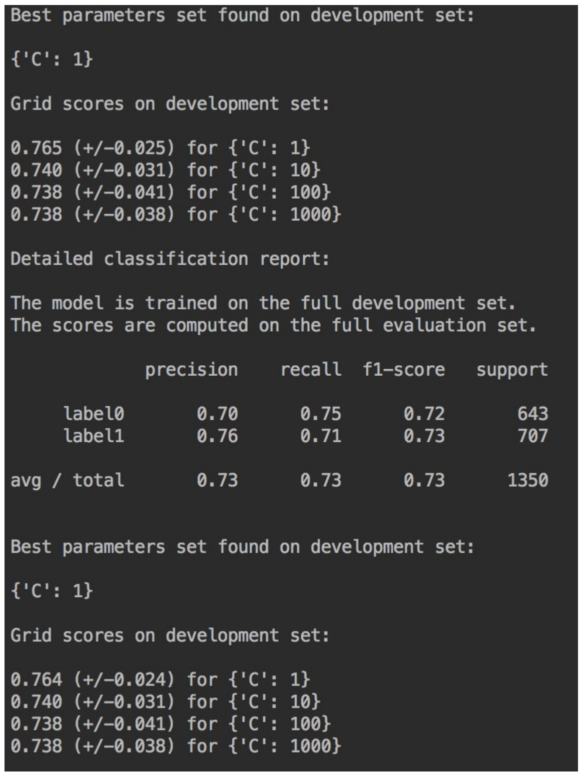




```
Best parameters set found on development set:

{'C': 1000}

Grid scores on development set:

0.712 (+/-0.036) for {'C': 1}
0.721 (+/-0.022) for {'C': 10}
0.724 (+/-0.014) for {'C': 100}
0.727 (+/-0.028) for {'C': 1000}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

             precision    recall  f1-score   support

     label0       0.68      0.75      0.72       645
     label1       0.75      0.68      0.71       705

avg / total       0.72      0.71      0.71      1350


Best parameters set found on development set:

{'C': 1000}

Grid scores on development set:

0.711 (+/-0.037) for {'C': 1}
0.721 (+/-0.022) for {'C': 10}
0.723 (+/-0.015) for {'C': 100}
0.725 (+/-0.029) for {'C': 1000}
```

picture7. Logistic Regression error-hyper:C curve (bag) picture8. Logistic performance(bag) using word-embedding method:



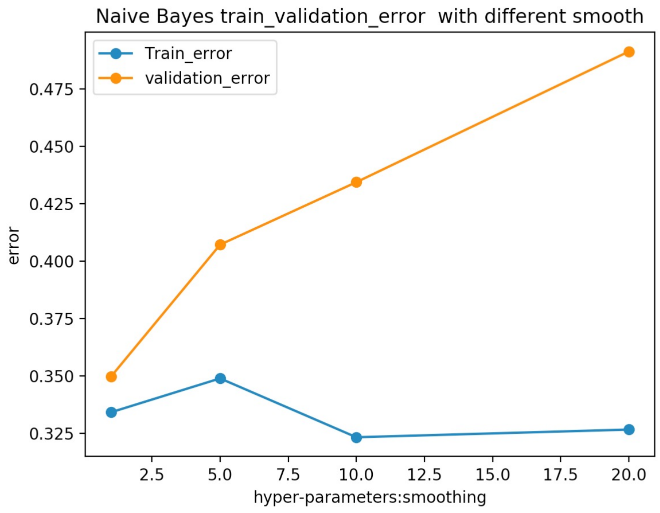


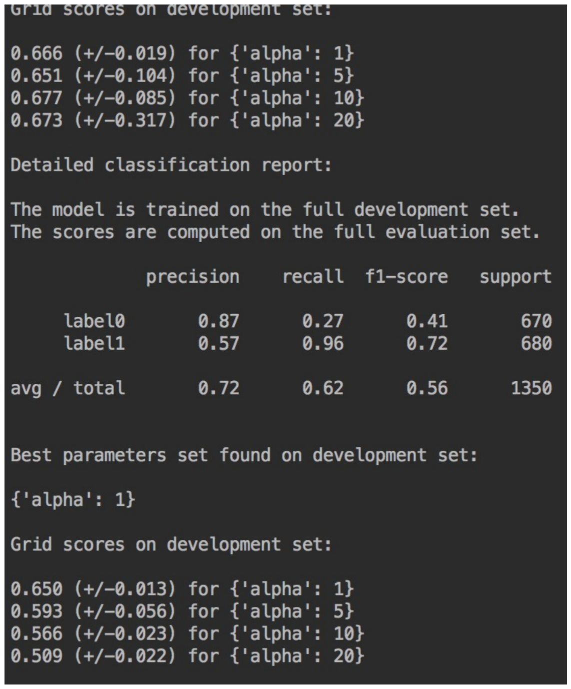
```
Best parameters set found on development set:

{'C': 1}

Grid scores on development set:

0.765 (+/-0.025) for {'C': 1}
0.740 (+/-0.031) for {'C': 10}
0.738 (+/-0.041) for {'C': 100}
0.738 (+/-0.038) for {'C': 1000}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

             precision    recall  f1-score   support

     label0       0.70      0.75      0.72       643
     label1       0.76      0.71      0.73       707

avg / total       0.73      0.73      0.73      1350


Best parameters set found on development set:

{'C': 1}

Grid scores on development set:

0.764 (+/-0.024) for {'C': 1}
0.740 (+/-0.031) for {'C': 10}
0.738 (+/-0.041) for {'C': 100}
0.738 (+/-0.038) for {'C': 1000}
```

picture9. Logistic error-hyper:C curve (embedding) picture10. Logistic performance(embedding)

Note: Logistic Regression using word-embedding has the good performance, the train and validation error are extremely close. I think the reason is Logistic Regression needs more features with a value to train the w and w0, word-embedding has more exist values features than bag-of-words. So the performance is better.

Naive Bayes:
parameters {default}
hyper-parameters [{'alpha': [1, 5, 10, 20]}]
word-of-bag performance as following:





```
Grid scores on development set:

0.666 (+/-0.019) for {'alpha': 1}
0.651 (+/-0.104) for {'alpha': 5}
0.677 (+/-0.085) for {'alpha': 10}
0.673 (+/-0.317) for {'alpha': 20}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

             precision    recall  f1-score   support

     label0       0.87      0.27      0.41       670
     label1       0.57      0.96      0.72       680

avg / total       0.72      0.62      0.56      1350


Best parameters set found on development set:

{'alpha': 1}

Grid scores on development set:

0.650 (+/-0.013) for {'alpha': 1}
0.593 (+/-0.056) for {'alpha': 5}
0.566 (+/-0.023) for {'alpha': 10}
0.509 (+/-0.022) for {'alpha': 20}
```

picture11. Naive Bayes error-hyper:smooth curve (bag-of-words)
picture12. Naive Bayes performance(bag-of-words)

Note: I don't use the word-embedding method for Naive Bayes, because the model need category features, but it is difficult to make a category using word-embedding method.

Finally, I choose word-embedding method and Logistic Regression model with C=1 to make the test. The accuracy is 0.765 +/-0.025.

3. Some interesting finding

- when we use bag-of-words to vectorize the corpus. We should convert the train and test data to the same features set. So, we would better fit_transform the train data and transform the test data.
- word-embedding method is more time-consuming than bag-of-words.
- Naive Bayes didn't perform well as expected, but the precision for label 0 and recall for label 1 are extremely high.
- logistic regression with word-embedding method has a good performance, the train-validation accuracies are both high and similar.
- the accuracy of train-validation is always changing, because we shuffle the data when use cross-validation.