Scikit Learn
Working with text data

Ishant Nayer
M10669373
Special Topics in BANA (002)-Python

Note: Code is given in a separate file

Dataset:

Dataset that is used to build and test the model is obtained from www.imdb.com - an internet movie database. A total of 2000 reviews are taken for building and validating the model. 1000 for positive and 1000 for negative. Every review is stored as a text file.

Approach:

Before building the model, the dataset is loaded using 'load_files' function. Entire file/ folder containing both positive and negative reviews separately, is loaded. The dataset is divided into 'training' and 'testing' datasets in 75:25 ratios, i.e. 75% of the original dataset as training ad remaining 25% as testing. The main objective to split the dataset as training and testing is to build the model using the training set and validate the performance of the built model using the testing set. This is implemented using train_test_split() function.

Explanation:

While classifying the reviews based on the sentiments, we convert the text content into numerical vectors to extract features, tokenize the text, identify the stop words and build a dictionary based on features.

Pipeline Classifier:

Classifiers denote the technique that will be implemented to classify the reviews based on the sentiments. Most commonly used classifiers are naïve Bayes classifier and support vector machine. Using 'Pipeline' function, linear support vector classifier is chosen and trained. It is invoked by setting 'clf' as LinearSVC with error penalty parameter C set to some fixed value.

Along with that, filtering is done to remove words with low occurrence and very high occurrence. By setting the values for 'min_df' and 'max_df', low and high occurring words are filtered and removed. Following are the values of min_df and 'max_df':
Min_df = 5- removes words that occur less than 5 reviews
Max_df = 0.95- words that are present in more than 95% of reviews

Parameter Tuning using Grid Search:

Classifiers has one or parameters which could be an error penalty factor, smoothing parameter or configurable loss in the objective function. We use a grid search technique. It involves an extensive and exhaustive search technique and automatically tweaking the parameters to find their best values.

Fitting Classifier:

Once the optimal value of the parameter is determined, the classifier is fitted with the training dataset along with the dependent variable. It is implemented using the function '.fit()' .

Model Prediction/Performance:

Once the model is trained using the training dataset with optimal parameter values, it is used to predict the polarity/ classify the reviews present in the testing dataset. It is implemented using '.predict()' function.

Performance of the model with respect to prediction of the testing dataset is determined using misclassification rate or confusion matrix.

Final Output:

```
n_samples: 2000
GridSearchCV(cv=None, error_score='raise',
       estimator=Pipeline(steps=[('vect', TfidfVectorizer(analyzer='word', binary=Fa
lse, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=0.95, max_features=None, min_df=3,
        ngram_range=(1, 1), norm='l2', preprocessor=None, smooth_idf=True,
 ...ax_iter=100,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=0))]),
       fit_params={}, iid=True, n_jobs=-1,
       param_grid={'vect__ngram_range': [(1, 1), (1, 2)]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=None, verbose=0)
0 params - {'vect__ngram_range': (1, 1)}; mean - 0.83; std - 0.01
1 params - {'vect__ngram_range': (1, 2)}; mean - 0.85; std - 0.01
             precision    recall  f1-score   support

        neg       0.87      0.88      0.87       240
        pos       0.88      0.88      0.88       260

avg / total       0.88      0.88      0.88       500

[[210  30]
 [ 31 229]]
```