Description_NB_SVM

December 4, 2019

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import re
        from nltk.corpus import stopwords
        from nltk.stem import WordNetLemmatizer
        import nltk.tag
        from sklearn.model_selection import cross_val_score
        from sklearn.model_selection import KFold
        from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn import model_selection, naive_bayes, svm
        from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, recall_score, pre
        import time
```

1 I/ Text Processing:

- Split sentence into words
- Convert words into original form
- Remove stop_words, comma, number, ...
- Only keep noun, adjective, past-tense verb

```
In [2]: def preprocess_data(data):
            stop_words = set(stopwords.words('english'))
            lemmatizer = WordNetLemmatizer()
            document = []
            for i in range(0,len(data)):
                #print(data[i])
                s = re.split('[\t\s,.\'\"\d]',data.iloc[i])
                line = []
                temp = []
                for term in s:
                    term = term.lower()
                    if ((term not in stop_words) and (term)):
                        temp.append(lemmatizer.lemmatize(term))
                word_tag = nltk.pos_tag(temp)
                count = 0
                for term in temp:
```

```
                if (word_tag[count][1] in ['JJ','NN','VBD']):
                    line.append(str(term))
                count += 1
            data.iloc[i] = str(line)
            document.append(line)
        return data, document
```

2 II/ Model Selection:

2.0.1 1) Naive Bayes - Text Classification by Multinomial Model:

```
In [3]: # prior = probability of class
        def find_prior(y):
            PriorC = {0: 0,1: 0}
            PriorC[0] = sum(y == 0) / len(y)
            PriorC[1] = sum(y == 1) / len(y)
            return PriorC
```

```
In [4]: # Create Tct: count number of occurences of each term in each class
        def find_vocabulary(document,quality):
            vocabulary = {}
            pos = 0
            for line in document:
                for item in line:
                    if (item not in vocabulary.keys()):
                        vocabulary[item] = {0: 0, 1: 0}
                        vocabulary[item][quality.iloc[pos]] += 1
                    else:
                        vocabulary[item][quality.iloc[pos]] += 1
                pos += 1
            return vocabulary
```

```
In [5]: def TrainMultinomialNB(document,quality):
            prior = find_prior(quality)
            vocabulary = find_vocabulary(document,quality)
            sum_c0 = sum(vocabulary[term][0] for term in vocabulary.keys())
            sum_c1 = sum(vocabulary[term][1] for term in vocabulary.keys())
            condprob = vocabulary
            for term in vocabulary.keys():
                condprob[term][0] = (vocabulary[term][0]+1)/(sum_c0 + len(vocabulary))
                condprob[term][1] = (vocabulary[term][1]+1)/(sum_c1 + len(vocabulary))
            #print(condprob)
            return vocabulary,prior,condprob
```

```
In [6]: def ApplyMultinomialNB(document,vocabulary,prior,condprob):
            score = prior
            argmax = []
            for line in document:
```

```
            #print(line)
            if (prior[0] > 0): score[0] = np.log(prior[0])
            else: score[0] = 0
            if (prior[1] > 0): score[1] = np.log(prior[1])
            else: score[1] = 0
            for term in line:
                if term in vocabulary.keys():
                    score[0] += np.log(condprob[term][0])
                    score[1] += np.log(condprob[term][1])
            if (score[0] > score[1]): argmax.append(0)
            else: argmax.append(1)
        return argmax
```

```
In [7]: def NB_multinomial(X_train,y_train,X_test):
            vocabulary, prior, condprob = TrainMultinomialNB(X_train,y_train)
            predictions_NB = ApplyMultinomialNB(X_test,vocabulary,prior,condprob)
            return(predictions_NB)
```

2.0.2 2) Support Vector Machine

```
In [8]: def SVM(Train_X_Tfidf,Train_Y,Test_X_Tfidf):
            SVM = svm.SVC(C=1.0, kernel='linear', degree=3, gamma='auto')
            SVM.fit(Train_X_Tfidf,Train_Y)
            predictions_SVM = SVM.predict(Test_X_Tfidf)
            return predictions_SVM
```

2.0.3 3) Split train/test:

```
In [9]: #winedata = pd.read_csv('sample.csv', usecols = ['description','quality'])
        winedata = pd.read_csv('winemag-data-130k-v2.csv', usecols = ['description','quality']
        Train_X, Test_X, Train_Y, Test_Y = model_selection.train_test_split(winedata['descripti
        SVM_train, NB_train = preprocess_data(Train_X)
        SVM_test, NB_test = preprocess_data(Test_X)

        Tfidf_vect = TfidfVectorizer(max_features=5000)
        Tfidf_vect.fit(winedata['description'])
        Train_X_Tfidf = Tfidf_vect.transform(SVM_train)
        Test_X_Tfidf = Tfidf_vect.transform(SVM_test)

        predictions_NB = NB_multinomial(NB_train,Train_Y,NB_test)
        predictions_SVM = SVM(Train_X_Tfidf,Train_Y,Test_X_Tfidf)
```

3 III/ Model Evaluation:

3.0.1 1) Confusion matrix:

```
In [11]: NB_confusion = confusion_matrix(predictions_NB, Test_Y)
         TP_NB = NB_confusion[1, 1]
```

```
        TN_NB = NB_confusion[0, 0]
        FP_NB = NB_confusion[0, 1]
        FN_NB = NB_confusion[1, 0]

        svm_confusion = confusion_matrix(predictions_SVM, Test_Y)
        TP_SVM = svm_confusion[1, 1]
        TN_SVM = svm_confusion[0, 0]
        FP_SVM = svm_confusion[0, 1]
        FN_SVM = svm_confusion[1, 0]
        print('Naive Bayes Confusion Matrix: ')
        print(NB_confusion)
        print('SVM Confusion Matrix: ')
        print(svm_confusion)
```

```
Naive Bayes Confusion Matrix:
[[11857  1972]
 [ 3488  7605]]
SVM Confusion Matrix:
[[13418  2586]
 [ 1927  6991]]
```

3.0.2 2) Accuracy Score:

- Percentage of correct predictions

```
In [12]: print("Naive Bayes Accuracy Score -> ",accuracy_score(predictions_NB, Test_Y)*100)
         print("SVM Accuracy Score -> ",accuracy_score(predictions_SVM, Test_Y)*100)
```

```
Naive Bayes Accuracy Score ->  78.09164593531818
SVM Accuracy Score ->  81.89150148463206
```

3.0.3 3) True Positive Rate / Sensitivity / Recall:

- When the actual value is positive, how often is the prediction correct?
- How “sensitive” is the classifier to detecting positive instances?

```
In [13]: # True Positive Rate = True Positives / (True Positives + False Negatives)
         TPR_NB = TP_NB / float(TP_NB + FN_NB)
         TPR_SVM = TP_SVM / float(TP_SVM + FN_SVM)
         print("Naive Bayes Recall Score -> ",TPR_NB*100)
         print("SVM Recall Score -> ",TPR_SVM*100)
```

```
Naive Bayes Recall Score ->  68.55674749842244
SVM Recall Score ->  78.3920161471182
```

3.0.4 4) Specificity:

- When the actual value is negative, how often is the prediction correct?
- How “specific” (or “selective”) is the classifier in predicting positive instances?

```
In [14]: SPEC_NB = TN_NB / float(TN_NB + FP_NB)
         SPEC_SVM = TN_SVM / float(TN_SVM + FP_SVM)
         print("Naive Bayes Specificity Score -> ",SPEC_NB*100)
         print("SVM Recall Specificity -> ",SPEC_SVM*100)
```

```
Naive Bayes Specificity Score ->  85.74011136018513
SVM Recall Specificity ->  83.84153961509622
```

3.0.5 5) Precision:

- When a positive value is predicted, how often is the prediction correct?
- How “precise” is the classifier when predicting positive instances?

```
In [15]: print("Naive Bayes Precision Score -> ",precision_score(predictions_NB, Test_Y)*100)
         print("SVM Precision Score -> ",precision_score(predictions_SVM, Test_Y)*100)
```

```
Naive Bayes Precision Score ->  79.40900073091782
SVM Precision Score ->  72.99780724652814
```

3.0.6 6) ROC Curves and AUC:

- ROC Curves: Plot of the False Positive Rate (x-axis) versus the True Positive Rate (y-axis)
- AUC has an important statistical property:
  - The AUC of a classifier is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance
  - The bigger AUC the better.

```
In [16]: bayes_auc = roc_auc_score(Test_Y,predictions_NB)
         svm_auc = roc_auc_score(Test_Y,predictions_SVM)

         # AUC Score
         print("Naive Bayes AUC Score -> ",bayes_auc*100)
         print("Naive Bayes AUC Score -> ",svm_auc*100)

         # ROC curves
         FPR_NB, TPR_NB, _ = roc_curve(Test_Y,predictions_SVM)
         FPR_SVM, TPR_SVM, _ = roc_curve(Test_Y,predictions_NB)

         plt.plot(FPR_NB, TPR_NB, linestyle='--', label='Naive Bayes')
         plt.plot(FPR_SVM, TPR_SVM, marker='.', label='SVM')
         plt.xlabel('False Positive Rate')
```

```
    plt.ylabel('True Positive Rate')
    plt.legend()
    plt.show()
```

```
Naive Bayes AUC Score ->  78.33923480664497
Naive Bayes AUC Score ->  80.21998540886199
```