AI_Lab3

September 14, 2021

1 Lab-3

1.1 Members = P. V. Sriram, Mahesh Babu

1.2 Roll No. = 1801CS37, 1801CS36

2 Setup

[1]:
```
# Import Libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
```

[2]:
```
# Read File
file = open("./smsspamcollection/SMSSpamCollection", 'r')

lines = file.readlines()
```

[3]:
```
# Extract Data
labels = []
data = []

for line in lines:
    labels.append(line.split('\t')[0])
    data.append(line.split('\t')[1])
```

[4]:
```
# Train Test Split
X_train, X_test, Y_train, Y_test = train_test_split(data, labels, test_size=0.
↪25, random_state = 10)
```

3 Multi-Nomial Naive Bayes

[5]:
```
import string
# import nltk
# nltk.download('stopwords')
from nltk.corpus import stopwords
from sklearn.model_selection import KFold
```

```
from sklearn.metrics import accuracy_score
stop_words = set(stopwords.words('english'))
from sklearn.feature_extraction.text import CountVectorizer

# Multi-Nomial Naive Bayes Class
class MNB():
    def __init__(self):
        # Initialise vectorisers
        self.vectorizer_x = CountVectorizer()
        self.vectorizer_y = CountVectorizer()
        self.cutoff_freq = 5

    # Preprocess vocabulary
    def pre_process(self):
        self.vocab = {}
        self.features = []
        for i in range(len(self.X_train)):
            for word in self.X_train[i].split():
                word_new  = word.strip(string.punctuation).lower()
                if (len(word_new)>2)  and (word_new not in stop_words):
                    if word_new in self.vocab:
                        self.vocab[word_new]+=1
                    else:
                        self.vocab[word_new]=1

        for key in self.vocab:
            if self.vocab[key] >= self.cutoff_freq:
                self.features.append(key)

    # Vectorise each sentences
    # Using count vectorisers
    def encode(self):
        self.vectorizer_x.fit(self.features)
        self.X_cv_train = self.vectorizer_x.transform(self.X_train)

        self.vectorizer_y.fit(self.classes)
        self.Y_cv_train = self.vectorizer_y.transform(self.Y_train)

    # Train the Model
    # Calculate conditional Probabilities
    def fit(self, X_train, Y_train):
        self.X_train = X_train
        self.Y_train = Y_train

        self.classes, self.class_probs = np.unique(self.Y_train, return_counts␣
↪= True)
        self.class_probs = self.class_probs / np.sum(self.class_probs)
```

```
        self.pre_process()
        self.encode()

        self.index_matrix = np.dot(self.X_cv_train.T, self.Y_cv_train).toarray()
        self.prob_matrix = (self.index_matrix + 1) / (np.sum(self.index_matrix,␣
↪axis = 0))

    # Predict class using
    # conditional probabilites
    def predict(self, X_test):
        X_cv_test = self.vectorizer_x.transform(X_test).toarray()
        prediction = []
        for sample in X_cv_test:
            current = 0
            for i, y in enumerate(self.prob_matrix.T):
                temp = np.multiply(sample.T, y)
                prob = np.prod(temp[np.where(temp != 0)]) * self.class_probs[i]
                if (prob > current):
                    current = prob
                    class_ = self.classes[i]
            prediction.append(class_)
        return (prediction)

    # Evaluate Performance
    def evaluate(self, X_val, Y_val):
        pred = self.predict(X_val)
        return (accuracy_score(pred, Y_val))
```

[6]:
```
# Initialise Model
clf1 = MNB()

# K-Fold cross validation
kf = KFold(n_splits=5)
accuracy = 0
for train_index, test_index in kf.split(X_train):
    clf1.fit(np.array(X_train)[train_index], np.array(Y_train)[train_index])
    accuracy += clf1.evaluate(np.array(X_train)[test_index], np.
↪array(Y_train)[test_index])
accuracy /= 5
print(accuracy)
```

```
0.9801435406698564
```

[7]:
```
# Perfrom prediction on Test Set
ans = clf1.predict(X_test)
```

[8]:
```
# Accuracy
clf1.evaluate(X_test, Y_test)
```

[8]:
```
0.9763271162123386
```

[9]:
```
# Index matrix (For compating with Multi Variate)
clf1.index_matrix
```

[9]:
```
array([[ 0, 17],
       [ 0,  8],
       [ 0,  8],
       …,
       [ 6,  0],
       [ 2,  3],
       [26,  0]])
```

4 Multi-Variate Naive Bayes

[10]:
```
# Multi-Nomial Naive Bayes Class
class MVB():
    def __init__(self):
        # Initialise vectorisers
        self.vectorizer_x = CountVectorizer(binary = True)
        self.vectorizer_y = CountVectorizer(binary = True)
        self.cutoff_freq = 5

    # Preprocess vocabulary
    def pre_process(self):
        self.vocab = {}
        self.features = []
        for i in range(len(self.X_train)):
            for word in self.X_train[i].split():
                word_new  = word.strip(string.punctuation).lower()
                if (len(word_new)>2)  and (word_new not in stop_words):
                    if word_new in self.vocab:
                        self.vocab[word_new]+=1
                    else:
                        self.vocab[word_new]=1

        for key in self.vocab:
            if self.vocab[key] >= self.cutoff_freq:
                self.features.append(key)

    # Vectorise each sentences
    # Using count vectorisers
    def encode(self):
```

```
        self.vectorizer_x.fit(self.features)
        self.X_cv_train = self.vectorizer_x.transform(self.X_train)

        self.vectorizer_y.fit(self.classes)
        self.Y_cv_train = self.vectorizer_y.transform(self.Y_train)

    # Train the Model
    # Calculate conditional Probabilities
    def fit(self, X_train, Y_train):
        self.X_train = X_train
        self.Y_train = Y_train

        self.classes, self.class_probs = np.unique(self.Y_train, return_counts␣
↪= True)
        self.class_probs = self.class_probs / np.sum(self.class_probs)

        self.pre_process()
        self.encode()

        self.index_matrix = np.dot(self.X_cv_train.T, self.Y_cv_train).toarray()
        self.prob_matrix = (self.index_matrix + 1) / (np.sum(self.index_matrix,␣
↪axis = 0))

    # Predict class using
    # conditional probabilites
    def predict(self, X_test):
        X_cv_test = self.vectorizer_x.transform(X_test).toarray()
        prediction = []
        for sample in X_cv_test:
            current = 0
            for i, y in enumerate(self.prob_matrix.T):
                temp = np.multiply(sample.T, y)
                prob = np.prod(temp[np.where(temp != 0)]) * self.class_probs[i]
                if (prob > current):
                    current = prob
                    class_ = self.classes[i]
            prediction.append(class_)
        return (prediction)

    # Evaluate Performance
    def evaluate(self, X_val, Y_val):
        pred = self.predict(X_val)
        return (accuracy_score(pred, Y_val))
```

[11]:
```
# Initialise Model
clf2 = MVB()
```

```
# K-Fold cross validation
kf = KFold(n_splits=5)
accuracy = 0
for train_index, test_index in kf.split(X_train):
    clf2.fit(np.array(X_train)[train_index], np.array(Y_train)[train_index])
    accuracy += clf2.evaluate(np.array(X_train)[test_index], np.
 ↪array(Y_train)[test_index])
accuracy /= 5
print(accuracy)
```

```
0.9801435406698564
```

[12]:
```
# Perfrom prediction on Test Set
ans = clf2.predict(X_test)
```

[13]:
```
# Accuracy
clf2.evaluate(X_test, Y_test)
```

[13]:
```
0.9770444763271162
```

[14]:
```
# Index matrix (For compating with Multi Variate)
clf2.index_matrix
```

[14]:
```
array([[ 0, 15],
       [ 0,  8],
       [ 0,  8],
       …,
       [ 6,  0],
       [ 1,  3],
       [26,  0]])
```

[ ]: