Name: Sangeeth Kumar V
Register No: 16BIS0072

Artificial Intelligence with Python

Lab Task – 02 (L39 & L40)

Prof Hemprasad Yashwant Patil

Question:

The Census Income Data Set (Income_data.txt) presents a challenge of predicting whether income exceeds $50K/yr based on census data. The data is stored in Income_data.txt file. There are two distinct classes namely '<=50K' and '>50K'. Apply necessary pre-processing steps on this data and state them clearly. Note that there should not be more than 70% data in train. The model has to be built which will accurately classify the test data. Use various classification techniques and indicate the results in tabular form. Perform a benchmarking analysis of the results.

Solution:

Sample dataset:

```
39, State-gov, 77516, Bachelors, 13, Never-married, Adm-
clerical, Not-in-family, White, Male, 2174, 0, 40, United-
States, <=50K

50, Self-emp-not-inc, 83311, Bachelors, 13, Married-civ-spouse,
Exec-managerial, Husband, White, Male, 0, 0, 13, United-States,
<=50K
```

SVM and LinearSVM classifier

Import required modules

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
import warnings
warnings.filterwarnings("ignore")
import itertools
import timeit
```

Preprocess the data

```python
# Input file containing data
input_file = 'income_data.txt'
```

```python
# Read the data
X = []
y = []
count_class1 = 0
count_class2 = 0
max_datapoints = 25000
# dictionaries to benchmark
scores = {}
time = {}

#Pre-processing
with open(input_file, 'r') as f:
    for line in f.readlines():
        if count_class1 >= max_datapoints and count_class2 >= max_datapoints:
            break
        if '?' in line:
            continue

        data = line[:-1].split(', ')

        if data[-1] == '<=50K' and count_class1 < max_datapoints:
            X.append(data)
            count_class1 += 1

        if data[-1] == '>50K' and count_class2 < max_datapoints:
            X.append(data)
            count_class2 += 1
# Convert to numpy array
X = np.array(X)

# Convert string data to numerical data
label_encoder = []
X_encoded = np.empty(X.shape)
for i,item in enumerate(X[0]):
    if item.isdigit():
        X_encoded[:, i] = X[:, i]
    else:
        label_encoder.append(preprocessing.LabelEncoder())
        X_encoded[:, i] = label_encoder[-1].fit_transform(X[:, i])

X = X_encoded[:, :-1].astype(int)
y = X_encoded[:, -1].astype(int)
```

Linear SVM

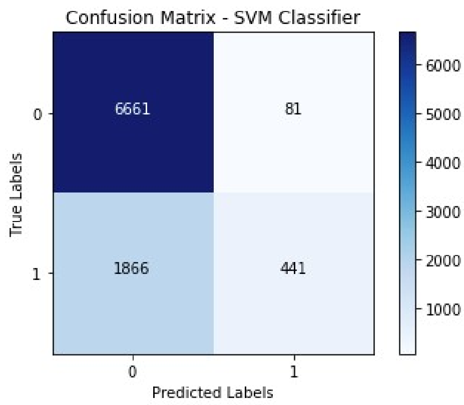
```python
# Linear SVM
from sklearn.svm import LinearSVC
from sklearn.multiclass import OneVsOneClassifier
```

Train and compute accuracy

```
# Create SVM classifier
classifier = OneVsOneClassifier(LinearSVC(random_state=0))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=5)

# Note down start time
start = timeit.default_timer()
# Train the classifier
classifier.fit(X, y)
# Cross validation
classifier.fit(X_train, y_train)
y_test_pred = classifier.predict(X_test)
# Note down stop time
stop = timeit.default_timer()

# Compute the F1 score of the SVM classifier
f1 = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=3)
print("SVM classifier results")
print("F1 score: " + str(round(100*f1.mean(), 2)) + "%")
# Accuracy
a = accuracy_score(y_test_pred,y_test)
print("Accuracy: " + str(a*100) + "%")
```

Print confusion matrix

```
# Confusion matrix
conf_mat = confusion_matrix(y_test,y_test_pred)
plt.imshow(conf_mat, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix - SVM Classifier')
plt.colorbar()
ticks = np.arange(2)
plt.xticks(ticks,ticks)
plt.yticks(ticks,ticks)

thresh = conf_mat.max()/2.
for i,j in itertools.product(range(conf_mat.shape[0]),range(conf_mat.shape[1])):
    plt.text(j,i,format(conf_mat[i,j],'d'),
            horizontalalignment = "center",
            color="white" if conf_mat[i,j]>thresh else"black")
plt.ylabel('True Labels')
plt.xlabel('Predicted Labels')
plt.show()
```



Accuracy and classification report

```
# Accuracy and Classification report
print(classification_report(y_test_pred,y_test))
print('Time taken for SVM',stop-start)
scores['SVM'] = a*100
time['SVM'] = stop-start
```

```
              precision    recall  f1-score   support

           0       0.99      0.78      0.87      8527
           1       0.19      0.84      0.31       522

    accuracy                           0.78      9049
   macro avg       0.59      0.81      0.59      9049
weighted avg       0.94      0.78      0.84      9049

Time taken for SVM 7.138061488000062
```

GaussiaNB (naive Bayes classifier)

```
# GaussianNB classifier
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
```

Train model

```
# Note start time
start = timeit.default_timer()
# Train classifier
classifier.fit(X,y)
# predict the values for training data
y_pred = classifier.predict(X)
# Note stop time
stop = timeit.default_timer()
```

Compute accuracy and time taken

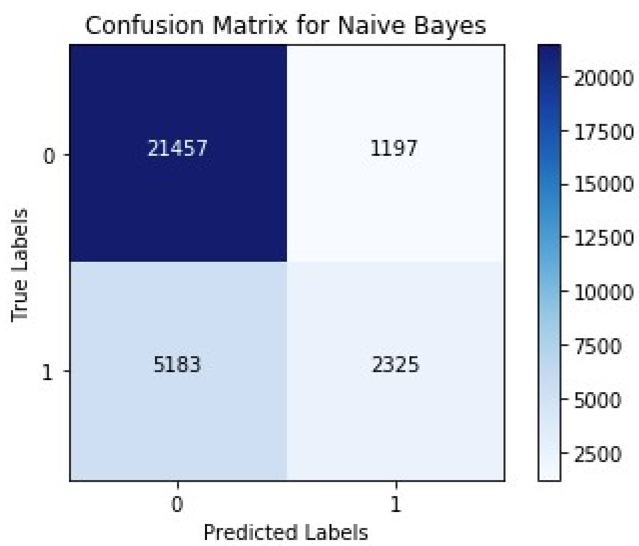
```
# Computing accuracy and time taken
accuracy = 100*(y == y_pred).sum()/X.shape[0]
print("Gaussian Native Bayes classfier results")
print("Accuracy: " + str(round(accuracy,2)), "%")
```

```
print("Time",stop-start)
time['Naive Bayes'] = stop-start
```

```
Gaussian Native Bayes classfier results
Accuracy: 78.85 %
Time 0.034026517000029344
```

Print confusion matrix

```
conf_mat = confusion_matrix(y,y_pred)
plt.imshow(conf_mat, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix for Naive Bayes')
plt.colorbar()
ticks = np.arange(2)
plt.xticks(ticks,ticks)
plt.yticks(ticks,ticks)

thresh = conf_mat.max()/2
for i,j in itertools.product(range(conf_mat.shape[0]),range(conf_mat.shape[1])):
   plt.text(j,i, format(conf_mat[i,j],'d'),
          horizontalalignment = 'center',
          color = 'white' if conf_mat[i,j]>thresh else 'black')
plt.ylabel('True Labels')
plt.xlabel('Predicted Labels')
plt.show()
```



Accuracy of naive bayes classification

```
# Accuracy and Classification of naive bayes
tn, fp, fn, tp = conf_mat.ravel()
print("True Negatives: ",tn)
print("True Positives: ",tp)
print("False Negatives: ",fn)
print("False Positives: ",fp)
Accuracy = (tn+tp)*100/(tp+tn+fp+fn)
print("Accuracy %.2f"%(Accuracy))
scores['Naive Bayes'] = Accuracy
```

```
True Negatives:  21457
True Positives:  2325
False Negatives:  5183
False Positives:  1197
Accuracy 78.85
```

Classification report

```
# Classification report
print(classification_report(y,y_pred))
```

	precision	recall	f1-score	support
0	0.81	0.95	0.87	22654
1	0.66	0.31	0.42	7508
accuracy			0.79	30162
macro avg	0.73	0.63	0.65	30162
weighted avg	0.77	0.79	0.76	30162

Logistic Regression

Import modules

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()

x_train,x_test,y_train,y_test = train_test_split(X,y,test_size = 0.3)
```

Train model

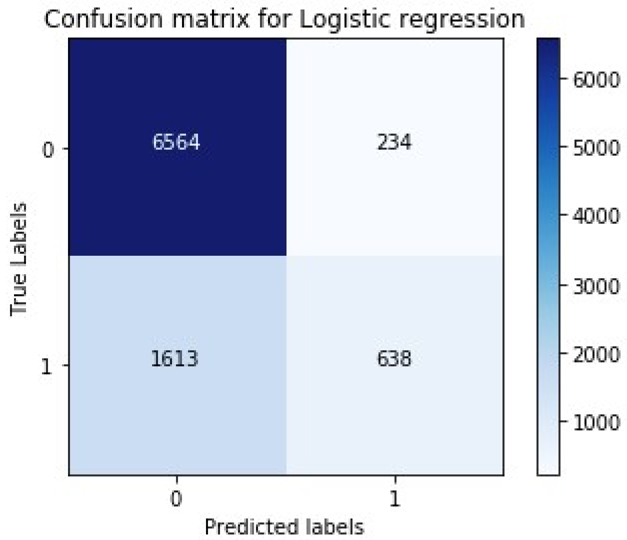
```
# Note start time
start = timeit.default_timer()
# Train classifier
classifier.fit(x_train,y_train)
# Predict the value of training data
predicted = classifier.predict(x_test)
# Note stop time
stop = timeit.default_timer()
print(predicted)
```

```
[0 0 0 ... 0 0 0]
```

Print confusion matrix

```
conf_mat = confusion_matrix(y_test,predicted)
plt.imshow(conf_mat, interpolation = 'nearest', cmap=plt.cm.Blues)
plt.title("Confusion matrix for Logistic regression")
plt.colorbar()
ticks = np.arange(2)
plt.xticks(ticks,ticks)
plt.yticks(ticks,ticks)
```

```
thresh = conf_mat.max()/2
for i,j in itertools.product(range(conf_mat.shape[0]),range(conf_mat.shape[1])):
   plt.text(j,i,format(conf_mat[i,j],'d'),
         horizontalalignment='center',
         color = 'white' if conf_mat[i,j]>thresh else 'black')
plt.ylabel('True Labels')
plt.xlabel('Predicted labels')
plt.show()
```



Accuracy and time taken

```
accuracy = accuracy_score(predicted,y_test)
print("Accuracy: " + str(accuracy*100) + "%")
scores['Logistic Regression'] = accuracy*100
print("Time taken", stop-start)
time['Logistic Regression'] = stop-start
```

```
Accuracy: 79.58890485136479%
Time taken 0.22942288800004462
```

Classification report

```
print(classification_report(y_test,predicted))
```

```
              precision    recall  f1-score   support

           0       0.80      0.97      0.88      6798
           1       0.73      0.28      0.41      2251

    accuracy                           0.80      9049
   macro avg       0.77      0.62      0.64      9049
weighted avg       0.79      0.80      0.76      9049
```

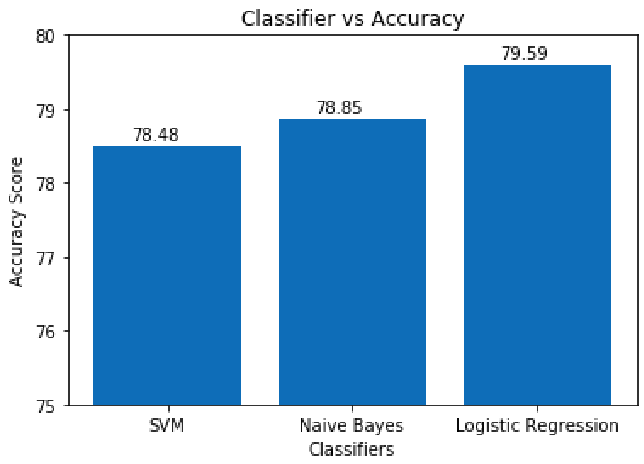
BENCHMARKING RESULTS OBTAINED FROM DIFFERENT CLASSIFIERS

Comparing accuracy of algorithms

```
# BENCHMARKING THE RESULTS OBTAINED IN DIFFERENT CLASSIFIERS

plot_y = np.array(list(scores.keys()))
plot_x = np.array(list(scores.values()))
```

```
print(*plot_x,sep = " ")
print(*plot_y,sep = " ")

index = np.arange(len(plot_y))
plt.bar(index,plot_x)
plt.title("Classifier vs Accuracy")
plt.ylabel('Accuracy Score')
plt.xlabel('Classifiers')
plt.xticks(index,plot_y,fontsize = 10)
plt.ylim(75,80)
for i in range(len(index)):
   plt.text(x = i-0.2, y = plot_x[i]+0.1, s = plot_x[i].round(2),size = 10)
plt.show()
```

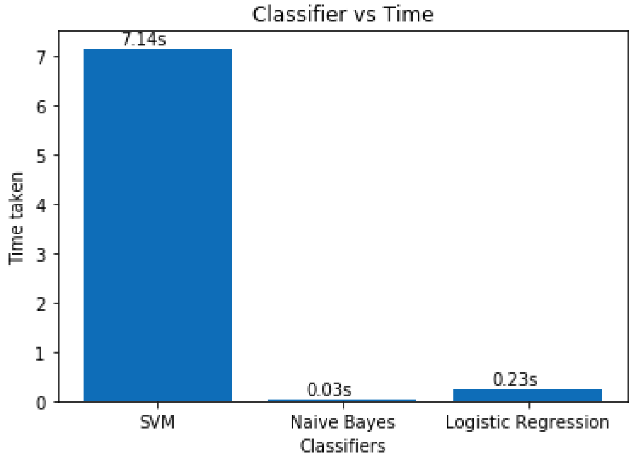
```
78.48381036578628 78.8475565280817 79.58890485136479
SVM Naive Bayes Logistic Regression
```



Comparing the time taken for each algorithms

```
time_x = np.array(list(time.values()))
time_y = np.array(list(time.keys()))
print(*time_x,sep=" ")
print(*time_y,sep=" ")

index = np.arange(len(time_y))
plt.bar(index,time_x)
plt.title("Classifier vs Time")
plt.ylabel('Time taken')
plt.xlabel('Classifiers')
plt.xticks(index,time_y,fontsize = 10)
for i in range(len(index)):
   plt.text(x = i-0.2, y = time_x[i]+0.1, s = str(time_x[i].round(2))+'s',size = 10)
plt.show()
```

```
7.138061488000062 0.034026517000029344 0.22942288800004462
SVM Naive Bayes Logistic Regression
```



Conclusion

Thus from the observation tabulated while benchmarking accuracy and time of the algorithms taken into account we can notice that **Logistic regression** has more accuracy when we train with 70% of the dataset. And considering the time consumed **Naive Bayes** works 9 time more efficiently compared to Logistic regression. Hence, a trade-off should be made for accuracy and time when using Logistic regression and Naive Bayes for classification. Overall Logistic regression performs better than all other.
