final_models.py

Co-authored by Tanya, Bhavik, Mudit, Srihit and Jaykumar as a result of our ME781 Data Mining final project.

Based on Shopping Data set from UCI's Machine Learning Repository

1. **Predict.ai Website** http://www.predictai.design/
2. **GitHub Repo** https://github.com/Tannybuoy/predictai
3. **Demo Video** https://www.youtube.com/watch?v=xFt4cl4daKM/

Importing necessary libraries for running Machine Lerning models

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import RobustScaler, StandardScaler, MinMaxSca
from sklearn.model_selection import cross_val_score, train_test_split, cr
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
import xgboost as xgb
from xgboost import XGBClassifier
```

Google Drive Mount in Google Colab

```
from google.colab import drive
drive.mount('/content/drive')
```

Go to directory containing the dataset

```
cd "/content/drive/My Drive/Colab Notebooks"
```

Reading shopping data

```
X_train = pd.read_csv('ShoppingData.csv')
df = X_train.copy()
df.head()
```

Producing dummy variables for categorical data and cleaning data

A dummy dataframe is created to clean the dataset and preprocess

1. **Visitor** - Columns based on New, Returning or Other
2. **Month** - New columns for each month
3. **Class** - Column after changing data type to int

```
dummiesdf = pd.get_dummies(df['VisitorType'])
df.drop('VisitorType', inplace = True, axis = 1)
df['New_Visitor'] = dummiesdf['New_Visitor']
df['Other'] = dummiesdf['Other']
df['Returning_Visitor'] = dummiesdf['Returning_Visitor']

dfmonth = pd.get_dummies(df['Month'])
df.drop('Month', inplace = True, axis = 1)
dfwithdummies = pd.concat([df, dfmonth], axis = 1, sort = False)
```

```
dfwithdummies['Class'] = df['Revenue'].astype(int)
dfwithdummies.drop('Revenue', axis = 1, inplace = True)
dfwithdummies['Weekend'] = df['Weekend'].astype(int)
dfwithdummies.drop('Returning_Visitor', axis = 1, inplace = True)
dfcleaned = dfwithdummies.copy()

X = dfcleaned.drop('Class', axis = 1)
Y = dfcleaned['Class'].copy()
```

Checking for Collinearity Between Features and Creating Reducing Feature Size

> The cor and heatmap help in visualising correlation between various features. Accordingly we do remove the columns/ pre-process.

```
cor = X.corr()
sns.heatmap(cor, xticklabels=cor.columns,yticklabels=cor.columns)
```

```
def AvgMinutes(Count, Duration):
    if Duration == 0:
```

AvgMinutes function is used to calculate the average time spent by a customer on the given page. It is obtained by dividing the “Count” by “Duration”

```
        output = 0
    elif Duration != 0:
        output = float(Duration)/float(Count)
    return output
```

Three new column features hence get added and six columns can now be dropped

```
Columns = [['Administrative', 'Administrative_Duration'], ['Informational

X['AvgAdministrative'] = X.apply(lambda x: AvgMinutes(Count = x['Administ
X['AvgInformational'] = X.apply(lambda x: AvgMinutes(Count = x['Informati
X['AvgProductRelated'] = X.apply(lambda x: AvgMinutes(Count = x['ProductR
X.drop(['Administrative', 'Administrative_Duration','Informational', 'Inf
```

Correlation matrix is plotted again using sns heatmap to check if the correlation between the above dropped six features has been dealt with

```
cor = X.corr()
sns.heatmap(cor, xticklabels=cor.columns,yticklabels=cor.columns)
```

Quick overview of features

Histogram of all features

```
for idx,column in enumerate(X.columns):
    plt.figure(idx)
    X.hist(column=column,grid=False)
```

Checking for NA values

```
for i in X.columns:
    print('Feature:',i)
    print('# of N/A:',X[i].isna().sum())
```

Visualising no of unique values and the unique values in each column of the training dataset

```
for i in X_train.columns:
    print('####################')
    print('COLUMN TITLE:',i)
    print('# UNIQUE VALUES:',len(X_train[i].unique()))
    print('UNIQUE VALUES:',X_train[i].unique())
    print('####################')
    print()
```

Scaling to normalize data

```
X_copy = X.copy()
rc = RobustScaler()
X_rc=rc.fit_transform(X_copy)
X_rc=pd.DataFrame(X_rc,columns=X.columns)
```

Plotting the histogram obtained post above processing functions

```
for idx,column in enumerate(X_rc.columns):
    plt.figure(idx)
    X_rc.hist(column=column,grid=False)
```

Linear Model with All Features

```
from sklearn import linear_model
from sklearn import metrics
X_train, X_test, y_train, y_test = train_test_split(X_rc,Y,test_size=.2)
```

Linear model

```
model = linear_model.SGDClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Accuracy score imported to calculate accuracy

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

roc_auc_score imported to calculate accuracy

```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test, y_pred)
```

It illustrates in a binary classifier system the discrimination threshold created by plotting the true positive rate vs false positive rate

Random Forest with all Features

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(max_depth=17, random_state=0)
clf.fit(X_train, y_train)
y_pred1 = clf.predict(X_test)

accuracy_score(y_test, y_pred1)

roc_auc_score(y_test, y_pred1)
```

Finding Important Features then Removing from Dataframe

SelectKBest to obtain a list of importance of each feature column

```
from sklearn import svm
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
list_one =[]


feature_ranking = SelectKBest(chi2, k=5)
fit = feature_ranking.fit(X, Y)

fmt = '%-8s%-20s%s'

for i, (score, feature) in enumerate(zip(feature_ranking.scores_, X.colum
    list_one.append((score, feature))

dfObj = pd.DataFrame(list_one)
dfObj.sort_values(by=[0], ascending = False)
```

On seeing the list, we drop the ones which have a very low weightage and less importance

```
X_rc.drop(['Aug','TrafficType','OperatingSystems','Other','Jul'],axis=1,i

X_train1, X_test1, y_train1, y_test1 = train_test_split(X_rc,Y,test_size=
```

Random Forest Classifier with Feature Selection Dataframe

Now once again we run Random Forest Classifier, but after retaining only the important features as determined by SelectKBest

```
clf1 = RandomForestClassifier(n_estimators= 200, max_depth = 30 )
clf1.fit(X_train1, y_train1)
y_pred2 = clf1.predict(X_test1)

accuracy_score(y_test1, y_pred2)

roc_auc_score(y_test1, y_pred2)
```

XGBoost Classifier with Feature Selection Dataframe

```
model = XGBClassifier(learning_rate = 0.1, n_estimators=150, min_child_we
model.fit(X_train1, y_train1)
y_pred3 = model.predict(X_test1)

accuracy_score(y_test1, y_pred3)

roc_auc_score(y_test1, y_pred3)
```

LogisticRegression with Feature Selection Dataframe

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
log_reg = LogisticRegression(solver='lbfgs',multi_class='multinomial',max
log_reg.fit(X_train1,y_train1)
y_pred4 = log_reg.predict(X_test1)
print(accuracy_score(y_pred4,y_test1))
print(roc_auc_score(y_test1, y_pred4))
```

Gaussian Naive Bayes with Feature Selection Dataframe

```
from sklearn.naive_bayes import GaussianNB
GNB = GaussianNB()
GNB.fit(X_train1,y_train1)
y_pred5 = GNB.predict(X_test1)
print(accuracy_score(y_pred5,y_test1))
print(roc_auc_score(y_test1, y_pred5))
```

KNN classifier with Feature Selection Dataframe

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 6)
knn.fit(X_train1,y_train1)
y_pred6 = knn.predict(X_test1)
print(accuracy_score(y_pred6,y_test1))
print(roc_auc_score(y_test1, y_pred6))
```

SVM Classification with PCA feature reduction technique

```
from sklearn.decomposition import PCA
pca = PCA(n_components=15)
d=pca.fit_transform(X_train1)
e=pca.fit_transform(X_test1)
print(pca.explained_variance_ratio_.sum())


from sklearn.svm import SVC
svm = SVC()
svm.fit(d,y_train1)
y_pred7 = svm.predict(e)
print(accuracy_score(y_pred7,y_test1))
print(roc_auc_score(y_test1, y_pred7))


from sklearn.svm import SVC
svm = SVC()
```

SVM Classification with Feature Selection Dataframe

```
svm.fit(X_train1,y_train1)
y_pred8 = svm.predict(X_test1)
print(accuracy_score(y_pred8,y_test1))
print(roc_auc_score(y_test1, y_pred8))
```

Neural Network Classifier With Feature Selection Dataframe

```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(19,19,19), activation='relu', sol
mlp.fit(X_train1,y_train1)
y_pred9= mlp.predict(X_test1)
print(accuracy_score(y_pred9,y_test1))
print(roc_auc_score(y_test1, y_pred9))
```