ML_algorythms

April 29, 2022

0.1 Download and Load the Data

We download the data directly from the DropBox link and load them in the Jupyter workspace as Pandas Dataframe. We than call the .head() method to check the result.

[ ]:
```
import os
import urllib
import pandas as pd

DOWNLOAD_URL = 'https://www.dropbox.com/s/7nwimmta836si5f/churn.csv?dl=1'
CHURN_PATH = os.path.join("dataset", "churn")

# Download data directly from Dropbox
def fetch_data(download_url=DOWNLOAD_URL, path=CHURN_PATH):
    os.makedirs(path, exist_ok=True)
    csv_path = os.path.join(path, "churn.csv")
    urllib.request.urlretrieve(download_url, csv_path)

# Load data
def load_data(path=CHURN_PATH):
    csv_path = os.path.join(path, "churn.csv")
    return pd.read_csv(csv_path)

fetch_data()
churn = load_data()

# Check result
churn.head(3)
```

0.2 Assemble Datasets

Here we assemble three datasets of different size:

- the data_full dataset includes all the variables present in the original dataset except for the CLIENTNUM identifier
- the data_best includes only the ten variables that we selected as most correlated with Attrition_Flag from preliminary analysis

- the data_mini only includes Total_Trans_Amt and Total_Trans_Ct as they proved the most predictive for the target

We also decided to keep an or_data version of the full original set keeping all the unkown rows

[ ]:
```
# Compute 3 different datasets
or_data=churn.drop(["Unnamed: 0", "CLIENTNUM"], axis=1)
data_full=churn.drop(["Unnamed: 0", "CLIENTNUM"], axis=1)
data_best=churn[["Attrition_Flag","Gender","Income_Category","Total_Relationship_Count","Month
data_mini=churn[["Attrition_Flag","Total_Trans_Amt","Total_Trans_Ct"]]
```

0.3 Delete "Unknown" rows

Here we define a function that first checks the presence of the Education_Level, Marital_Status, Income_Category in the datasets. It replaces the "Unknown" values with Nan and returns the dataset dropping all the NaN values. Notice that this operation was perfomed exclusively on the data_full and data_best datasets as the data_mini does not include any categorical attribute. Moreover this operation was performed only on the Income_Category for the data_best in order to retain as many data points as possible. We also had to reset the index.

Than we check wheter the "Unknown" values have been correctly removed for both sets and if the Attrition Flag Proportions between Existing and Attriting customers have been retained after the removal.

[ ]:
```
# Replace Unkown with NAN and drop Nan to delete rows
import numpy as np

unkown_vars = ['Education_Level', 'Marital_Status', 'Income_Category']

def replace_unkown(dataset):
    for var in unkown_vars:
        if var in dataset.columns:
            dataset[var] = dataset[var].replace("Unknown", np.NaN)
    return dataset.dropna()

data_full = replace_unkown(data_full).reset_index(drop=True)
data_best = replace_unkown(data_best).reset_index(drop=True)
```

[ ]:
```
# Double-check for results with values_counts()
def check_replace_unkown(dataset):
    for var in unkown_vars:
        if var in dataset.columns:
            print(dataset[var].value_counts())
            print('\n')

check_replace_unkown(data_best)
print("DATAFULL#############\n")
check_replace_unkown(data_full)
```

```
[ ]: # Check if proportions for Attrition_Flag actually resemble those of the␣
      ↪original dataset after dropping Unknown
     prop = data_full["Attrition_Flag"].value_counts() / len(data_full)
     proptot = churn["Attrition_Flag"].value_counts() / len(churn)

     print(prop)     # data_full
     print("\n")
     print(proptot)  # original dataset
```

0.4 Train-Test Splits

Here we define two functions to implement both a traditional train_test_split and a Stratified-ShuffleSplit split for Attrition Flag of the datasets. Both the methods were .imported from the model_selection module of scikit-learn. We will call the functions in the section "Select and Train models" below. We also check the results of the stratified split on the data_full computing the the Attrition Flag Proportions between Existing and Attriting customers on the strat_test_set

```
[ ]: # Classic sklearn split
     from sklearn.model_selection import train_test_split

     def split(dataset, test_size):
         train_set, test_set = train_test_split(dataset, test_size=test_size,␣
      ↪random_state=42)
         print("\033[1mTrain:\033[0m", len(train_set), "\t\033[1mTest:\033[0m",␣
      ↪len(test_set))
         return train_set, test_set

     train_set, test_set = split(data_full, 0.2) # FULL
```

```
[ ]: # Stratified Split based on Attrition flag
     from sklearn.model_selection import StratifiedShuffleSplit

     def strat_split(dataset, test_size):
         split = StratifiedShuffleSplit(n_splits=1, test_size=test_size,␣
      ↪random_state=42)

         for train_index, test_index in split.split(dataset,␣
      ↪dataset["Attrition_Flag"]):
             strat_train_set = dataset.loc[train_index]
             strat_test_set = dataset.loc[test_index]
         print("\033[1mTrain:\033[0m", len(strat_train_set), "\t\033[1mTest:
      ↪\033[0m", len(strat_test_set))
         return strat_train_set, strat_test_set

     strat_train_set, strat_test_set = strat_split(data_full, 0.2)
```

[ ]:
```
# Check if proportions in test_set actually resemble those of the full dataset
prop = strat_test_set["Attrition_Flag"].value_counts() / len(strat_test_set)
proptot = churn["Attrition_Flag"].value_counts() / len(churn)

print(prop)
print("\n")
print(proptot)
print("\nThe proportions between Attrited and Existing costumers are respected")
```

0.5 Prepare Data for ML Models

In this section we define a few functions to prepare the data for the Machine Learning models:

- the sep_pred_target takes as input the train and test sets and splits them both in X (predictors) and y (labels)
- the tranformation_pipeline takes as input only the train set of the predictors (X), splits numerical and categorical variables and via the ColumnTransformer applies Standard Scaling to numerical variables and One Hot encoding on categorical variables. It returns the prepared train set. To perform this operations we imported the ColumnTransformer from the compose module and the StandardScaler and the OneHotEncoder from the preprocessing module of scikit-learn.

[ ]:
```
# Lets separate the predictors and target value PLAIN

def sep_pred_target(train_set, test_set):
    X = train_set.drop("Attrition_Flag", axis=1)
    y = train_set["Attrition_Flag"].copy()

    X_test = test_set.drop("Attrition_Flag", axis=1)
    y_test = test_set["Attrition_Flag"].copy()

    return X, y, X_test, y_test

# Lets separate the predictors and target value STRATIFIED
X, y, X_test, y_test = sep_pred_target(train_set, test_set)
X_strat_train, y_strat_train, X_strat_test, y_strat_test  =␣
 ↪sep_pred_target(strat_train_set, strat_test_set)
X.shape
```

[ ]:
```
# TRASFORMATION PIPELINE
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer

# Define variables lists
cat_vars = ['Attrition_Flag', 'Gender', 'Education_Level', 'Marital_Status',␣
 ↪'Income_Category', 'Card_Category']
```

```
num_vars = ["Customer_Age", "Dependent_count", "Months_on_book",␣
↪"Total_Relationship_Count",
              "Months_Inactive_12_mon", "Contacts_Count_12_mon",␣
↪"Credit_Limit", "Total_Trans_Amt",
             "Total_Trans_Ct", "Avg_Utilization_Ratio"]

def tranformation_pipeline(X):

    lst=[]
    for var in cat_vars:
        if var in X.columns:
            lst.append(var)
    #print(lst)

    # Split cat and num attributes
    X_num = X.drop(lst, axis=1)
    X_cat = X.drop(list(X_num.columns), axis=1)

    num_attribs = list(X_num.columns)
    cat_attribs = list(X_cat.columns)

    # Separate col transformations for num and cat
    full_pipeline = ColumnTransformer([("num", StandardScaler(), num_attribs), ␣
↪ # STD SCALING for numerical
                                      ("cat", OneHotEncoder(), cat_attribs),])␣
↪ # ONE-HOT for categorical

    # Final TRAIN dataset (without labels)
    X_prep = full_pipeline.fit_transform(X)

    return X_prep

X_prep=tranformation_pipeline(X)
X_prep_test=tranformation_pipeline(X_test)
X_prep_test.shape
```

0.6 SMOTE

Synthetic Minority Oversampling Technique(SMOTE) is an oversampling technique and widely used to handle the imbalanced dataset. This technique synthesizes new data points for minority class (Attrited Customers) and oversample that class. Unfortunately although we were able to run the SMOTE on the prepared train set and the train labels we were not able to feed the resampled data to our machine learning models.

[ ]:
```
from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state=0)
```

```
# Train
X_resampled, y_resampled = sm.fit_resample(X_prep, y)
y_resampled.value_counts()
```

0.7 Metrics

Here we define a function that takes X_train, y_train, y_test and y_pred as input, compute all the metrics to evaluate our model and return them in a ordered list. We computed the following metrics:

- Confusion matrix
- Accuracy, Precision, Sensitivity, Specificity (manually computed form CM), Precision and Recall were recomputed with the precision_score and recall_score from scikit-learn just to double-check our results.
- Cross Validation (cv=3) scores, mean and standard deviation of the scores.
- f1 score

```
[ ]: from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.model_selection import cross_val_score

# Compute function to analyze models performance
def metrics(X_train, y_train, y_test, y_pred):

    lst=[]

    # Confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    accuracy=(cm[1,1]+cm[0,0])/(cm[0,0]+cm[0,1]+cm[1,0]+cm[1,1])
    precision=(cm[1,1]/(cm[1,1]+cm[0,1]))
    sensitivity=cm[1,1]/(cm[1,1]+cm[1,0])
    specificity=cm[0,0]/(cm[0,0]+cm[0,1])

    # Precision and Recall
    prec = round(precision_score(y_test, y_pred, pos_label='Existing␣
↪Customer'),3)
    recall = round(recall_score(y_test,y_pred, pos_label='Existing Customer'),3)

    # Accuracy with crossval
    cv_scores=cross_val_score(classifier, X_train, y_train, cv=3,␣
↪scoring="accuracy")
    mean=round(cv_scores.mean(),3)
    std=round(cv_scores.std(),3)

    # F1 score
```

```
    f1 = round(f1_score(y_test, y_pred, pos_label='Existing Customer'),3)

    # ROC and AUC

    # Put it all in a list
    lst.append(f"test_size={size}"+"\n"+"Accuracy: "+str(round(accuracy*100,2))+
                    " Precision: "+str(round(precision*100,2))+
                    " Sensitivity: "+str(round(sensitivity*100,2))+
                    " Specificity: "+str(round(specificity*100,2))+"\n"
                    "CrossVal scores: "+str(cv_scores)+
                    " Mean e std: "+str(mean)+"\t"+str(std)+"\n"
                    "Prec and recall: "+str(prec)+"\t"+str(recall)+
                    "\tF1 score: "+str(f1))
    return lst
```

0.8 Select and Train Models

Here we run a for loop to fit the models with three different test sizes (0.2, 0.25, 0.30). We ran the for loop on the data_full set but to run it on the data_best,the data_mini or on the original dataset without the unknown values removal, it would be sufficient to substitute the dataset name where highlighted in comment.

- We first split the dataset with boht the plain and the stratified splits, we prepare the data by calling the transformation pipeline function defined above (Standard Scaling, One Hot) and define the final prepared variables.
- Than we feed four different models from scikit-learn with the prepared data: Logistic Regression, Support Vector Machines, Decision Trees, Random Forest. We fit the models wiht both the plain and stratified data
- We compute the metrics for each model by calling the metrics function defined above and print the perfomance measures.

[ ]:
```
###### Model Fitting and Testing

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

test_sizes=[0.20,0.25,0.30]


for size in test_sizes:

    ############################################### DATA PREPARATION
    # Split
    train_set, test_set = split(data_best, size)                            #␣
 ↪SUBSTITUTE DATASET with DATA_BEST, MINI, OR
```

```python
    strat_train_set, strat_test_set = strat_split(data_best, size)    # SUBSTITUTE DATASET with DATA_BEST, MINI, OR

    # Lets separate the predictors and target value
    X, y, X_test, y_test = sep_pred_target(train_set, test_set)
    X_strat, y_strat, X_strat_test, y_strat_test  = sep_pred_target(strat_train_set, strat_test_set)

    # Transformation Pipeline
    X_prep = tranformation_pipeline(X)
    X_prep_test = tranformation_pipeline(X_test)

    X_prep_strat = tranformation_pipeline(X_strat)
    X_prep_test_strat = tranformation_pipeline(X_strat_test)

    # FINAL renaming
    X_train = X_prep
    y_train = y
    X_test = X_prep_test
    y_test = y_test

    X_train_strat = X_prep_strat
    y_train_strat = y_strat
    X_test_strat = X_prep_test_strat
    y_test_strat = y_strat_test

    ############################### LOGISTIC REGRESSION
    # Fitting Logistic to Train set
    classifier = LogisticRegression(random_state = 0)
    classifier.fit(X_train, y_train)
    # Predicting the Test set results
    y_pred = classifier.predict(X_test)

    # Metrics
    print("\nLOGISTIC\n")
    logistic = metrics(X_train, y_train, y_test, y_pred)
    for x in logistic:
        print(x)
    print("\n")

    ############################### LOGISTIC REGRESSION STRATIFIED
    # Fitting Logistic to Train set
    classifier = LogisticRegression(random_state = 0)
    classifier.fit(X_train_strat, y_train_strat)
    # Predicting the Test set results
    y_pred_strat = classifier.predict(X_test_strat)
```

```python
    # Metrics
    logisticstr = metrics(X_train_strat, y_train_strat, y_test_strat, y_pred_strat)
    for x in logisticstr:
        print(x)
    print("\n")


    #################################### SVM
    # Fitting SVM to the Training set
    classifier = SVC(kernel = 'linear', random_state = 0)
    classifier.fit(X_train, y_train)
    # Predicting the Test set results
    y_pred_svm = classifier.predict(X_test)

    # Metrics
    print("\nSVM\n")
    svm = metrics(X_train, y_train, y_test, y_pred_svm)
    for x in svm:
        print(x)
    print("\n")


    #################################### SVM STRATIFIED
    # Fitting Logistic to Train set
    classifier = SVC(kernel = 'linear', random_state = 0)
    classifier.fit(X_train_strat, y_train_strat)
    # Predicting the Test set results
    y_pred_strat = classifier.predict(X_test_strat)

    # Metrics
    svm_str = metrics(X_train_strat, y_train_strat, y_test_strat, y_pred_strat)
    for x in svm_str:
        print(x)
    print("\n")


    #################################### DECISION TREE CLASSIFICATION
    classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
    # entropy for homogenous node split
    classifier.fit(X_train, y_train)
    # Predicting the Test set results
    y_pred_dt = classifier.predict(X_test)

    print("\nDECISION TREE\n")
    dt = metrics(X_train, y_train, y_test, y_pred_dt)
    for x in dt:
```

```python
        print(x)
    print("\n")

    #################################### DECISION TREE STRATIFIED
    # Fitting Logistic to Train set
    classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
    classifier.fit(X_train_strat, y_train_strat)
    # Predicting the Test set results
    y_pred_strat = classifier.predict(X_test_strat)

    # Metrics
    dt_str = metrics(X_train_strat, y_train_strat, y_test_strat, y_pred_strat)
    for x in dt_str:
        print(x)
    print("\n")

    #################################### RANDOM FOREST CLASSIFICATION
    classifier = RandomForestClassifier(n_estimators = 300, criterion =␣
↪'entropy', random_state = 0)
    classifier.fit(X_train, y_train)
    y_pred_rf = classifier.predict(X_test)

    print("\nRANDOM FOREST\n")
    rf = metrics(X_train, y_train, y_test, y_pred_rf)
    for x in rf:
        print(x)
    print("\n")

    #################################### RANDOM FOREST STRATIFIED
    # Fitting Logistic to Train set
    classifier = RandomForestClassifier(n_estimators = 300, criterion =␣
↪'entropy', random_state = 0)
    classifier.fit(X_train_strat, y_train_strat)
    # Predicting the Test set results
    y_pred_strat = classifier.predict(X_test_strat)

    # Metrics
    rf_str = metrics(X_train_strat, y_train_strat, y_test_strat, y_pred_strat)
    for x in rf_str:
        print(x)
    print("\n")
```

0.9 ROC curves

Here we define a function to plot the ROC curve of the classifier based on the confusion matrix

```
[ ]: # ROC e AUC
from sklearn.metrics import precision_recall_curve, roc_curve
import matplotlib.pyplot as plt

# Precision Recall curve
def plot_prec_rec(classifier):
    y_pred = classifier.predict(X_test)
    y_pred_prob = classifier.predict_proba(X_test)[:,0]

    precisions, recalls, thresholds = precision_recall_curve(y_test,␣
 ↪y_pred_prob, pos_label='Existing Customer')

    def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
        plt.figure(figsize=(10, 8))
        plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
        plt.plot(thresholds, recalls[:-1], "g-", label="Recall")

    plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
    plt.legend()
    plt.show()

# ROC
def plot_roc(classifier):
    y_pred = classifier.predict(X_test)
    y_pred_prob = classifier.predict_proba(X_test)[:,0]

    fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob, pos_label="Existing␣
 ↪Customer")

    def plot_roc(fpr, tpr, thresholds):
        plt.figure(figsize=(10, 8))
        plt.plot(fpr, tpr, linewidth=2)
        plt.plot([0, 1], [0, 1], 'k--')

    plot_roc(fpr, tpr, thresholds)
    plt.legend()
    plt.show()
```

0.10 Hyperparameters Fine Tuning

We used the RandomizedSearchCV method from sklearn.model_selection module to fine tune the Random Forest classifier hyperparameters. Our goal was to to inspect whether we could obtain even better performance metrics.

```
[ ]: from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform
```

```
classifier = RandomForestClassifier(n_estimators = 300, criterion = 'entropy',␣
↪random_state = 0)

distributions = {'bootstrap': [True, False],
                 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110,␣
↪None],
                 'max_features': ['auto', 'sqrt'],
                 'min_samples_leaf': [1, 2, 4],
                 'min_samples_split': [2, 5, 10],
                 'n_estimators': [130, 180, 230, 300]}

clf = RandomizedSearchCV(classifier, distributions, random_state=0)
search = clf.fit(X_train, y_train)
search.best_params_
```

0.11 Final Model

Finally we fit the Randomized Search seach.best_estimator_ with the data_best dataset and analyze the final results and the ROC curve obtained.

[ ]:
```
train_set, test_set = split(data_best, 0.2)
# Separate the predictors and target value
X, y, X_test, y_test = sep_pred_target(train_set, test_set)
# Transformation Pipeline
X_prep = tranformation_pipeline(X)
X_prep_test = tranformation_pipeline(X_test)
# FINAL renaming
X_train = X_prep
y_train = y
X_test = X_prep_test
y_test = y_test

#########################################

classifier = search.best_estimator_
classifier.fit(X_train, y_train)
y_pred_rf = classifier.predict(X_test)

print("\nRANDOM FOREST\n")
rf = metrics(X_train, y_train, y_test, y_pred_rf)
for x in rf:
    print(x)
```

[ ]:
```
plot_prec_rec(classifier)
```

[ ]:
```
plot_roc(classifier)
```

0.12 Results Plots

To further visualize our classifiers performance we decided to plot decision boundaries for the two variables of Tot_Trans_Count over the Tot_Trans_Amt for both the training and the test set. We repeated this procedure for both the Decision Tree and the Random Forest classifier.

```
[ ]: import matplotlib.pyplot as plt

# Prepare data
data_best = data_best[["Attrition_Flag","Total_Trans_Amt","Total_Trans_Ct"]]
flag=[]
for x in data_best.iloc[:, 0].values:
    if x=="Existing Customer":
        flag.append(1)
    else:
        flag.append(0)

data_best["flag"]=flag

X = data_best.iloc[:,[1,2]].values
y = data_best.iloc[:, 3].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
 ↪random_state = 0)

# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:,␣
 ↪0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:,␣
 ↪1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).
 ↪reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
```

```
plt.title('Decision Tree (Training set)')
plt.xlabel('Total_Trans_Amt')
plt.ylabel('Total_Trans_Ct')
plt.legend()
plt.show()

# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:,␣
↪0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:,␣
↪1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).
↪reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Decision Tree (Test set)')
plt.xlabel('Total_Trans_Amt')
plt.ylabel('Total_Trans_Ct')
plt.legend()
plt.show()
```

[ ]:
```
import matplotlib.pyplot as plt

# Prepare data
data_best = data_best[["Attrition_Flag","Total_Trans_Amt","Total_Trans_Ct"]]
flag=[]
for x in data_best.iloc[:, 0].values:
    if x=="Existing Customer":
        flag.append(1)
    else:
        flag.append(0)

data_best["flag"]=flag

X = data_best.iloc[:,[1,2]].values
y = data_best.iloc[:, 3].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
↪random_state = 0)

# Feature Scaling
```

```python
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
classifier = search.best_estimator_
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:,␣
↪0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:,␣
↪1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).
↪reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Random Forest (Training set)')
plt.xlabel('Total_Trans_Amt')
plt.ylabel('Total_Trans_Ct')
plt.legend()
plt.show()

# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:,␣
↪0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:,␣
↪1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).
↪reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Random Forest (Test set)')
plt.xlabel('Total_Trans_Amt')
plt.ylabel('Total_Trans_Ct')
```

```
plt.legend()
plt.show()
```