Supervised Learning Models Report

Wing Yin Andrew Sit

Parth Patel

Manvir Kaur

Rajiv Ian Aseron

Chung Hin Lam

COMP247: Supervised Learning

Merlin James Rukshan Dennis

11 April 2022

Executive Summary

For this project, we are presented with bicycle theft data from the City of Toronto. The dataset contains details of offences wherein a bicycle theft is included, the date when the theft occurred and reported, the location details, and the bicycle information. Using these features, the members of the group were able to train several models with new data. These models were the Decision Tree, Linear Regression, Neural Network, Random Forest, and Support Vector Machine.

The models aim to predict whether a bicycle will be recovered or not given different combinations of data. Using the feature selection technique of chi-square, we were able to determine which features we are going to keep for the model training. Also, as the data is imbalanced between the recovered and unrecovered classes, the technique of up-sampling has been applied. Lastly, a grid search was used to determine the best parameters for each of the models and the training dataset.

Based on the scores of the generated models, the group has determined that the Decision Tree algorithm is best to be used for the prediction as it has an 89% accuracy score. Other models tested either overfitting (SVM – 100%; Random Forest – 99%; Neural Networks – 91%) or underfitting (Linear Regression – 77%). Therefore, the other models generated was not a good model to be used for the given dataset.

Overview of Solution

1. Loading the dataframe

```
import os
from pathlib import Path as path
from sklearn.model_selection import StratifiedShuffleSplit

dataFilePath = path.cwd()
dataFilename = 'Bicycle_Thefts.csv'
dataFullPath = os.path.join(dataFilePath,dataFilename)
data_theft = pd.read_csv(dataFullPath)
```

2. Carry out initial investigation

- Data Types

```
# a. Check the names and types of columns.
print('\nData Types:')
print(data_theft.dtypes)
```

The dataset consists of either numerical or categorical data. Each row of data gives information on the location of the theft, the primary offence where the theft was involved, date and time the offence occurred, the reported date, and the stolen bike information.

```
Data Types:
X                         float64
Y                         float64
OBJECTID_1                  int64
OBJECTID                    int64
event_unique_id            object
Primary_Offence            object
Occurrence_Date            object
Occurrence_Year             int64
Occurrence_Month           object
Occurrence_DayOfWeek       object
Occurrence_DayOfMonth       int64
Occurrence_DayOfYear        int64
Occurrence_Hour             int64
Report_Date                object
Report_Year                 int64
Report_Month               object
Report_DayOfWeek           object
Report_DayOfMonth           int64
Report_DayOfYear            int64
Report_Hour                 int64
Division                   object
City                       object
Hood_ID                    object
NeighbourhoodName          object
Location_Type              object
Premises_Type              object
Bike_Make                  object
Bike_Model                 object
Bike_Type                  object
Bike_Speed                  int64
Bike_Colour                object
Cost_of_Bike              float64
Status                     object
Longitude                 float64
Latitude                  float64
dtype: object
```

3. Visualization

Plot of theft occurrence on Toronto map

```
# plot a map of geographic locations of the theft occurrence
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
minmax_scale = preprocessing.MinMaxScaler(feature_range=(0,800))
map_coord = data_theft.iloc[:,0:2]
scaler = StandardScaler()
df = pd.DataFrame(scaler.fit_transform(map_coord),columns=['z','t'])

im = plt.imread("map.png")
implot = plt.imshow(im,extent=[-9.2,10,-9,10], aspect='auto')

plt.scatter(df['z'],df['t'],s=0.5,c=data_theft["Occurrence_Year"],
            cmap=plt.get_cmap("jet"))
plt.title('Geographic Representation of Bike Theft Occurrence in Toronto')
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.colorbar()
plt.show()
```

Plot of recovered bicycles on Toronto map

```
# plot a map of geographyic locations of the recoveries
df["Status"] = data_theft["Status"]
df_reco = df[df.Status == 'RECOVERED']

im = plt.imread("map.png")
implot = plt.imshow(im,extent=[-9.2,10,-9,10], aspect='auto')
plt.scatter(df_reco['z'],df_reco['t'],s=0.5)
plt.title("Geographic Representation of Stolen Bike Recovery in Toronto")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.show()
```

Numeric Histogram

```
# Histogram of numeric data
data_theft.hist(bins=50, figsize=(15,15))
```

Correlation of Numeric Data

```
# Correlation Plot of numeric data
import seaborn as sns
data_copy = data_theft[list(data_theft.select_dtypes(include=numerics).columns)]
cor= data_copy.corr(method='pearson')
print(cor)
```

```
fig, ax =plt.subplots(figsize=(8, 6))
plt.title("Correlation Plot")
sns.heatmap(cor, mask=np.zeros_like(cor, dtype=np.bool),
            cmap=sns.diverging_palette(220, 10, as_cmap=True),
            square=True, ax=ax)
plt.show()
```

- Please see Visualization section of document for plot output.

4. Drop unnecessary columns

- Drop id columns

```
#drop ID columns
to_drop = ['OBJECTID','OBJECTID_1','event_unique_id']
data_theft = data_theft.drop(columns=to_drop)
```

- Generate chi-square for feature selection

```
from scipy.stats import chi2_contingency
data_copy = data_theft.copy(deep=True)
data_copy['Status'].replace(['STOLEN','UNKNOWN','RECOVERED'],
                        [0, 0, 1], inplace=True)

for i in list(data_copy.select_dtypes(include=categorical).columns):
    csq=chi2_contingency(pd.crosstab(data_copy['Status'], data_copy[i]))
    print("P-value ["+ i +"]: ",csq[1])
```

- Determine elapsed days before theft was reported

```
# Both occurence date and reported date is relevant to the target
# to simplify this data, we determine number of days before a theft is reported
# and drop the date-related columns
data_theft['Report_Date'] = pd.to_datetime(data_theft['Report_Date'].str.slice(0,10))
data_theft['Occurrence_Date'] = pd.to_datetime(data_theft['Occurrence_Date'].str.slice(0,10))
data_theft.insert(0,'Elapsed_Days_Before_Reported',
        (data_theft["Report_Date"] - data_theft["Occurrence_Date"]).dt.days)
```

- Simplify bike model information to “KNOWN” and “UNKNOWN”

```
# since the bike model is relevant but has multitude of values
# that can put stress to the machine when transforming
# reduce the values from the bike model to 'KNOWN' and 'UNKNOWN'
# and check that it is still relevant to the target variable
data_copy = data_theft.copy(deep=True)
data_copy['Bike_Model'] = np.where(data_copy['Bike_Model'].isnull(), 'UNKNOWN',data_copy['Bike_Model'])
data_copy['Bike_Model'] = data_copy['Bike_Model'].str.replace('\W', '')
data_copy['Bike_Model'] = np.where(data_copy['Bike_Model']=='', 'UNKNOWN',data_copy['Bike_Model'])
data_copy['Bike_Model'] = np.where(data_copy['Bike_Model'].str.contains("UNK"), 'UNKNOWN', 'KNOWN')
data_theft['Bike_Model'] = data_copy['Bike_Model']
```

- Remove remaining columns

```python
# remove irrelevant columns
# exception/s:
#  NeighbourhoodName: removed as it is the same data as Hood ID (same relevanc as well)
#  Date related columns: simplified to the elapsed days reported column
to_drop = ['X','Y','Occurrence_Date','Occurrence_Month',
           'Occurrence_DayOfMonth','Report_Date','Report_Year','Report_Month',
           'Report_DayOfWeek','Report_DayOfMonth','Report_Hour',
           'Report_DayOfYear','NeighbourhoodName','Location_Type',
           'Bike_Speed','Bike_Colour','Longitude','Latitude'
           ]
data_theft = data_theft.drop(columns=to_drop)
```

5. Separate the features from the target variable (class)

```python
features = data_theft.iloc[:,0:13]
target_variable = data_theft["Status"]
```

6. Create the transformer pipeline

```python
target_variable.replace(['STOLEN','UNKNOWN','RECOVERED'],
                        [0, 0, 1], inplace=True)


num_features = list(features.select_dtypes(include=numerics).columns)
cat_features = list(features.select_dtypes(include=categorical).columns)

from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer

num_pipeline = Pipeline([
    ('num_imputer', SimpleImputer(missing_values=np.NAN, strategy='median'))
])

cat_pipeline = Pipeline([
    ('cat_imputer', SimpleImputer(missing_values=np.nan,
                                  strategy='constant', fill_value='UNKNOWN')),
    ('cat_selector', OneHotEncoder(sparse=False,handle_unknown='ignore')),
])

transformer = ColumnTransformer(transformers=[
                    ("numerics", num_pipeline, num_features),
                    ("categorical", cat_pipeline, cat_features)
                    ], remainder='passthrough')
```

7. Create test split

```
splitter=StratifiedShuffleSplit(test_size=0.2,random_state=25)
for train,test in splitter.split(features,target_variable):
    X_train_df = features.iloc[train]
    y_train_df= target_variable.iloc[train]
    X_test_df = features.iloc[test]
    y_test_df = target_variable.iloc[test]
```

8. Upsample the training data

```
from sklearn.utils import resample
data_uncovered = X_train_df[X_train_df.Status==1]
data_stolen = X_train_df[X_train_df.Status==0]
data_theft_upsampled = resample(data_uncovered, replace=True,
                                n_samples=len(data_stolen),
                                random_state=40)
X_train_df = pd.concat([data_stolen, data_theft_upsampled])
y_train_df = X_train_df['Status']
X_train_df = X_train_df.drop(columns='Status')
```

9. Create the model

```
## Random Forest
from sklearn.ensemble import RandomForestClassifier
rnd_clf = RandomForestClassifier(n_estimators=10,
                                 max_depth = 10,
                                 criterion = 'gini',
                                 random_state=40)

#from sklearn.pipeline import Pipeline
pipeline = Pipeline(steps=[
                           ('transformer', transformer),
                           ('rand_clf', rnd_clf)
                          ])
```

10. Fit the data

```
fitted_train = pipeline.fit(X_train_df,y_train_df)
```

11. Perform cross validation and print the test scores

```
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
scores= cross_val_score(fitted_train, X_train_df,y_train_df,
                        cv=cv, scoring="accuracy")

print('\n10-fold cross validation (stratified) test results')
i = 1;
for sc in scores:
    print('Test '+ str(i) +': ' + str(sc))
    i=i+1
print("Mean of scores: {:.3f} ".format(scores.mean()), end="\n\n" )
```

12. Run predict on test data and check metrics

```
y_pred = fitted_train.predict(X_test_df)

from sklearn import metrics
print("\n\nAccuracy:",metrics.accuracy_score(y_test_df, y_pred))
print("Precision:",metrics.precision_score(y_test_df, y_pred))
print("Recall:",metrics.recall_score(y_test_df, y_pred))
print("f1 score:",metrics.f1_score(y_test_df, y_pred))

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test_df, y_pred, labels=fitted_train.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=fitted_train.classes_)
disp.plot()
plt.title('Confusion Matrix')
plt.show()

y_pred_proba = fitted_train.predict_proba(X_test_df)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test_df,  y_pred_proba)
plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC curve')
plt.show()
```

13. Perform Grid Search

```
from sklearn.model_selection import GridSearchCV

parameters = {'rand_clf__n_estimators' : range(10,30,10),
              'rand_clf__criterion': ['gini', 'entropy'],
              'rand_clf__max_depth': range(50,100,15),
              'rand_clf__max_features':['sqrt', 'log2']}

grid_search = GridSearchCV(estimator=pipeline, param_grid=parameters,
                           scoring='accuracy',refit = True,
                           verbose = 3)

grid_search.fit(X_train_df, y_train_df)
```

14. Get the best parameters

```
grid_search.best_params_
```

15. Get best Estimator and fit to the fine-tuned model

```
grid_search.best_estimator_

fine_tuned_model = grid_search.best_estimator_
fine_tuned_model.fit(X_train_df,y_train_df)
ft_y_pred = fine_tuned_model.predict(X_test_df)
```

16. Print fine-tuned model scores

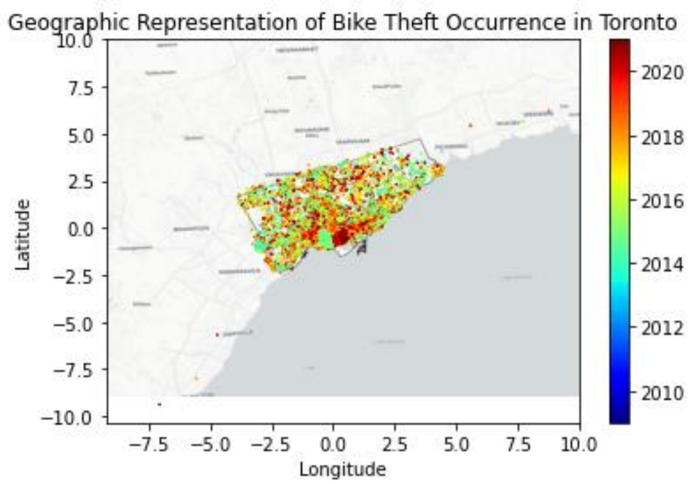
```
print("\nAccuracy:",metrics.accuracy_score(y_test_df, ft_y_pred))
print("Precision:",metrics.precision_score(y_test_df, ft_y_pred))
print("Recall:",metrics.recall_score(y_test_df, ft_y_pred))
print("f1 score:",metrics.f1_score(y_test_df, y_pred))
```

17. Save the fine-tuned model using the joblib

```
import joblib
joblib.dump(fine_tuned_model, 'theft_rf_model.pkl')
```

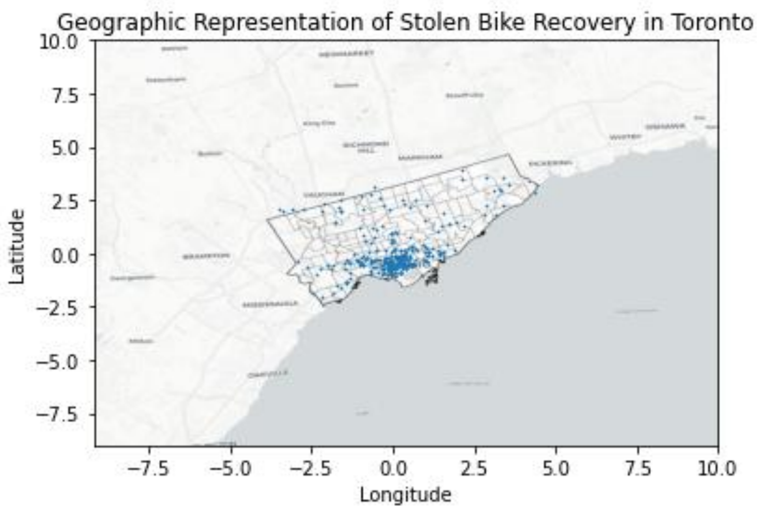
Data Exploration

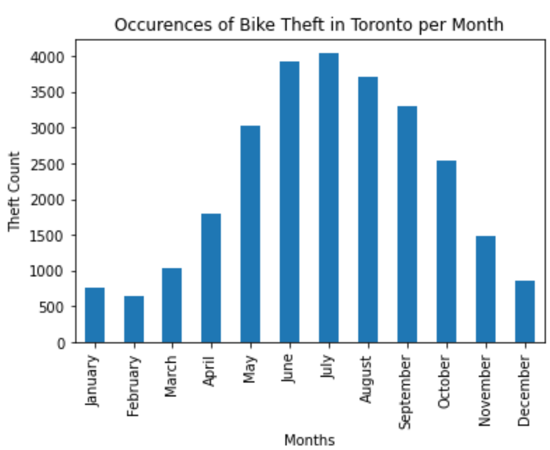
Visualization



A geographic representation of bike theft occurrence in Toronto with each dot representing a theft and its approximate location on the Toronto map. Each dot is also colored based on the date of occurrence of the theft as indicated by the color bar on the right.

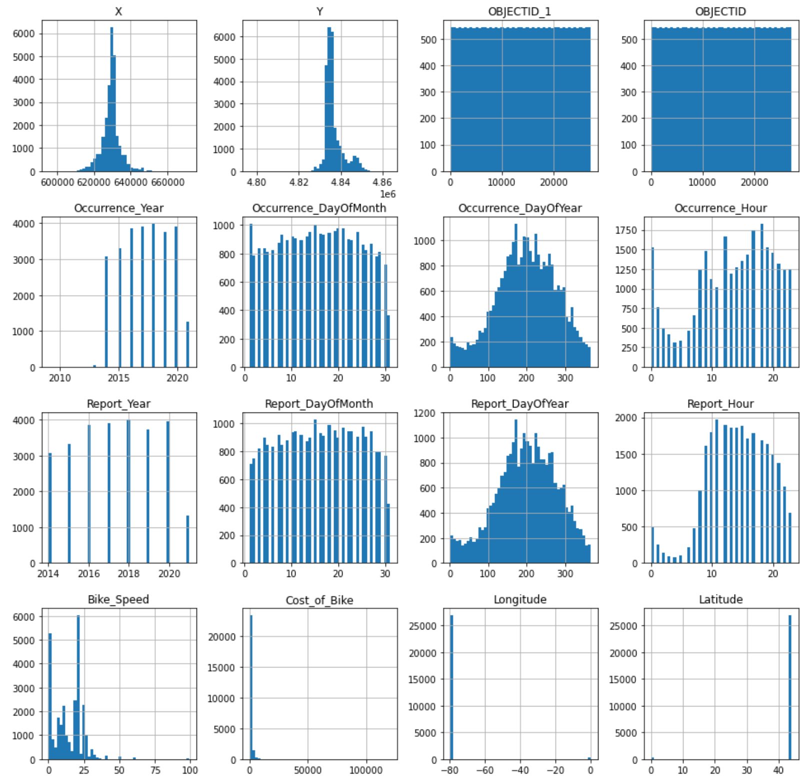
Another helpful visualization for users is this geographic depiction of the stolen bike recoveries in Toronto. This visualization shows how only a small percentage of bike thefts are resolved and the actual bikes are returned to the owners.

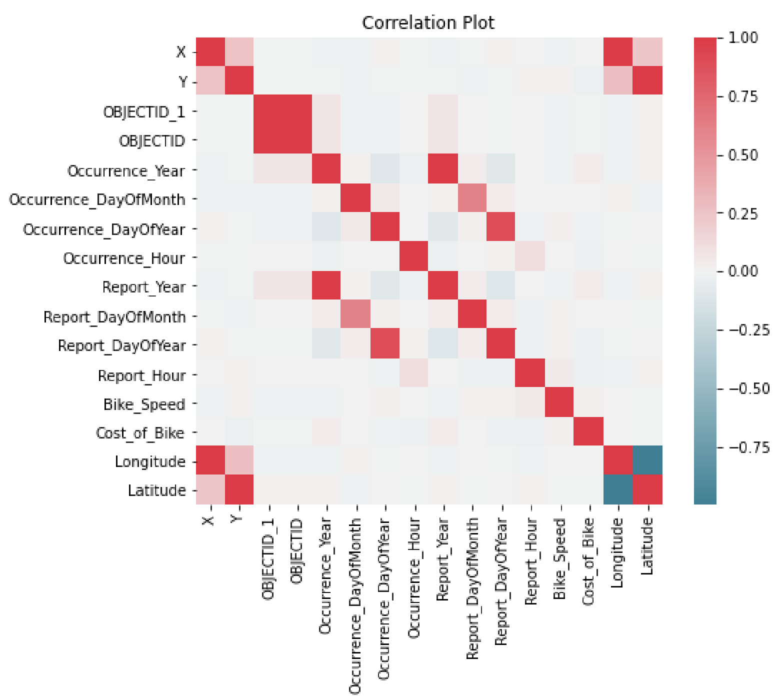




This bar chart depicts the frequency of bike theft for each month regardless of the year. The graph shows a normal distribution, with most of the thefts found in July.

Another visualization generated was a histogram of the numerical data of the dataset. This graph was also used to determine the important numerical features of the dataset. The numerical columns with normal distribution are important for the model.





The correlation plot shows the relationship of a numeric column with other numeric columns. An example is that the “X”, “Y” columns have a strong correlation with each other as these are the coordinates where the bike theft has occurred.

Feature Selection

Chi Square Testing

The Chi Square testing is a statistical test of independence to get the dependency of two variables. By using this testing on the feature columns and the target variable, we can observe how strong the relationship of each feature column is to the target variable. If the feature column is independent from the target variable (lower score) then we can discard or drop this column from the data frame. On the other hand, if the feature column is dependent to the target variable (higher score) then this column is important and should remain on the data frame.

Chi-Square code

```
# understand relationship between categorical variables and target variable
from scipy.stats import chi2_contingency
data_copy = data_theft.copy(deep=True)
data_copy['Status'].replace(['STOLEN','UNKNOWN','RECOVERED'],
                            [0, 0, 1], inplace=True)

for i in list(data_copy.select_dtypes(include=categorical).columns):
    csq=chi2_contingency(pd.crosstab(data_copy['Status'], data_copy[i]))
    print("P-value ["+ i +"]: ",csq[1])
```

Result:

```
P-value [Primary_Offence]:  0.0
P-value [Occurrence_Date]:  2.04040667447134e-32
P-value [Occurrence_Month]:  0.07777510424960905
P-value [Occurrence_DayOfWeek]:  0.3929295487005147
P-value [Report_Date]:  6.840414771030162e-23
P-value [Report_Month]:  0.033871988731702905
P-value [Report_DayOfWeek]:  0.19655287179311243
P-value [Division]:  0.16599689133003628
P-value [City]:  0.5937204643188589
P-value [Hood_ID]:  0.07874597637273274
P-value [NeighbourhoodName]:  0.07874597637273306
P-value [Location_Type]:  1.1651526842885868e-15
P-value [Premises_Type]:  0.009912699649141054
P-value [Bike_Make]:  3.4951185391350815e-39
P-value [Bike_Model]:  0.07775293236007914
P-value [Bike_Type]:  0.2693460208852993
P-value [Bike_Colour]:  2.9170641083871403e-56
```

Model Building

Stratified Shuffle Split

For this project, we opted to split our data to the training and testing data sets using the Stratified Shuffle Split function. According to the skLearn documentation, stratified shuffle split is a merge of StratifiedKFold and ShuffleSplit, which returns stratified randomized folds. The folds are made by preserving the percentage of samples for each class.

```
# Split your data into train 80% train and 25% test
splitter=StratifiedShuffleSplit(test_size=0.2,random_state=25)
for train,test in splitter.split(features,target_variable):
    X_train_df = features.iloc[train]
    y_train_df= target_variable.iloc[train]
    X_test_df = features.iloc[test]
    y_test_df = target_variable.iloc[test]
```

Sampling

The sample data have extremely uneven classes. The data with a status set to RECOVERED is only 328 out of 27128 samples (1.2%). Therefore, the data is resampled using an up-sampling method. The samples with class RECOVERED are duplicated to match the number of the samples in the other class. The data size has nearly doubled after the process.

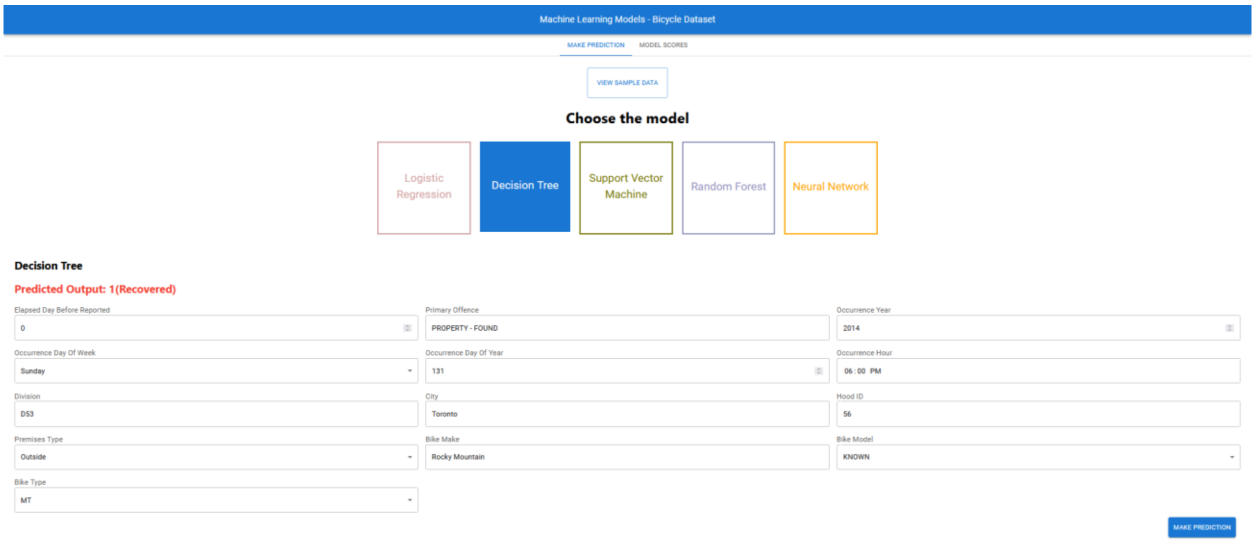
```
from sklearn.utils import resample
data_uncovered = X_train_df[X_train_df.Status==1]
data_stolen = X_train_df[X_train_df.Status==0]
data_theft_upsampled = resample(data_uncovered, replace=True,
                                n_samples=len(data_stolen),
                                random_state=40)
X_train_df = pd.concat([data_stolen, data_theft_upsampled])
y_train_df = X_train_df['Status']
X_train_df = X_train_df.drop(columns='Status')
```

Metrics

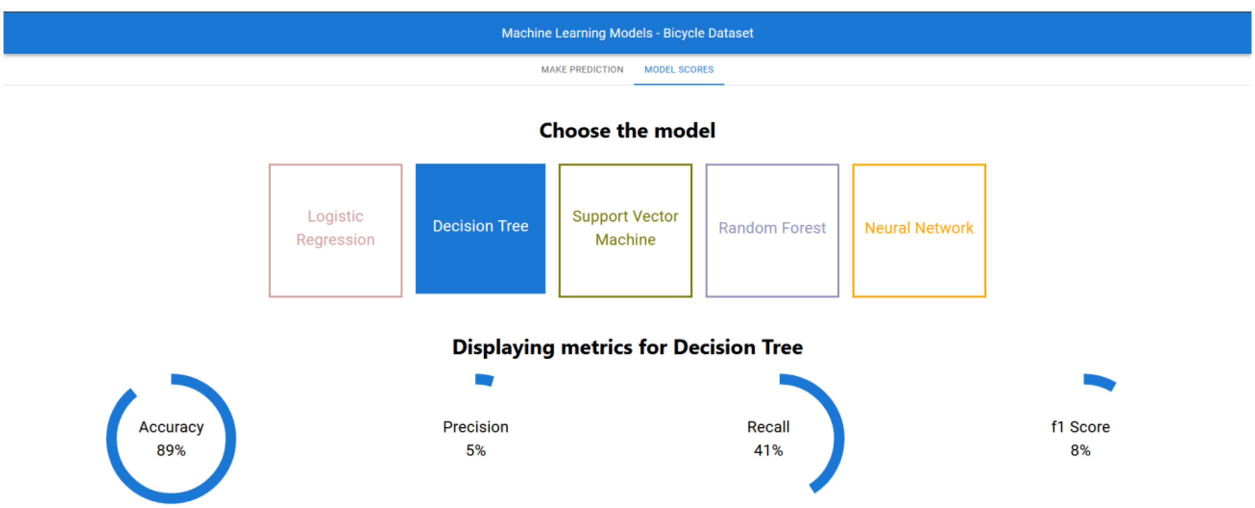
	Neural Network	Random Forest	Decision Tree	Logistic Regression	SVM
Accuracy	0.9093	0.9885	0.8916	0.775	0.9968
Precision	0.0448	0.7000	0.0468	0.031	0.9803
Recall	0.3181	0.1060	0.4090	0.576	0.7575
F1 score	0.0786	0.1842	0.0841	0.059	0.8547
Confusion Matrices	[[4913 447] [45 21]]	[[5357 3] [59 7]]	[[4811 549] [39 27]]	[[4200 1200] [28 38]]	[[5359 1] [64 2]]

Web Interface using React

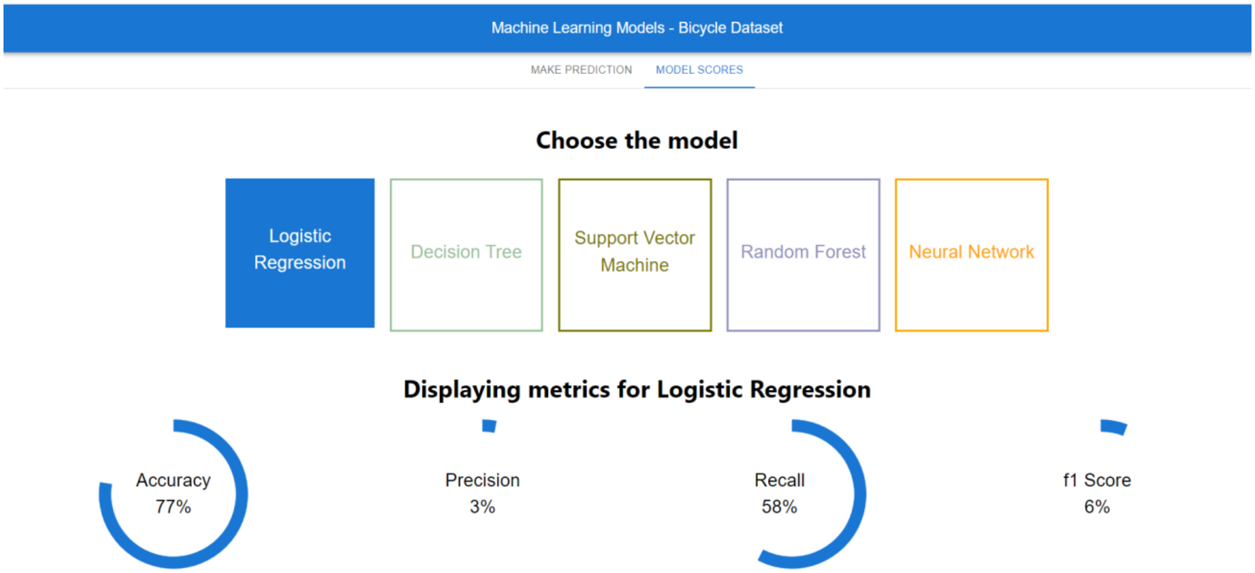
Prediction



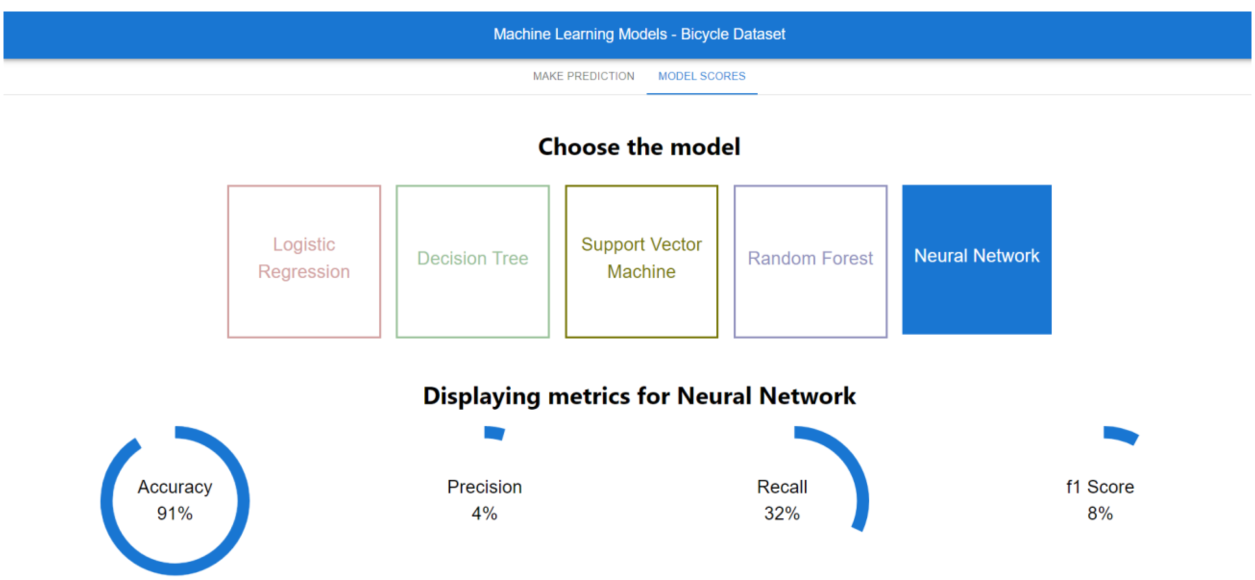
Decision Tree Scores



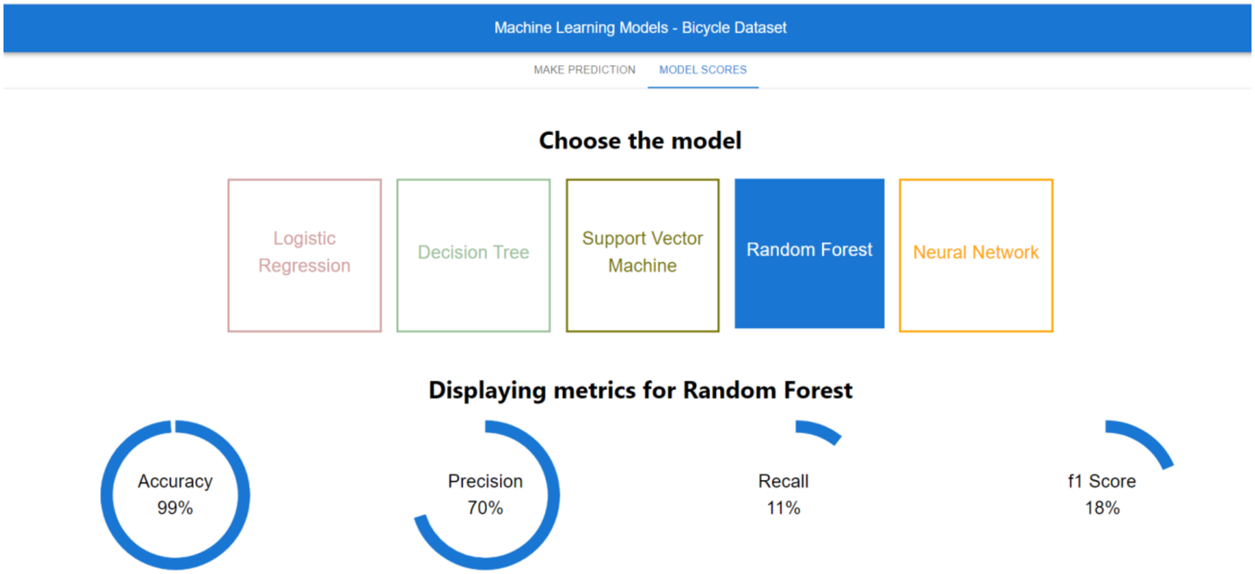
Logistic Regression Scores



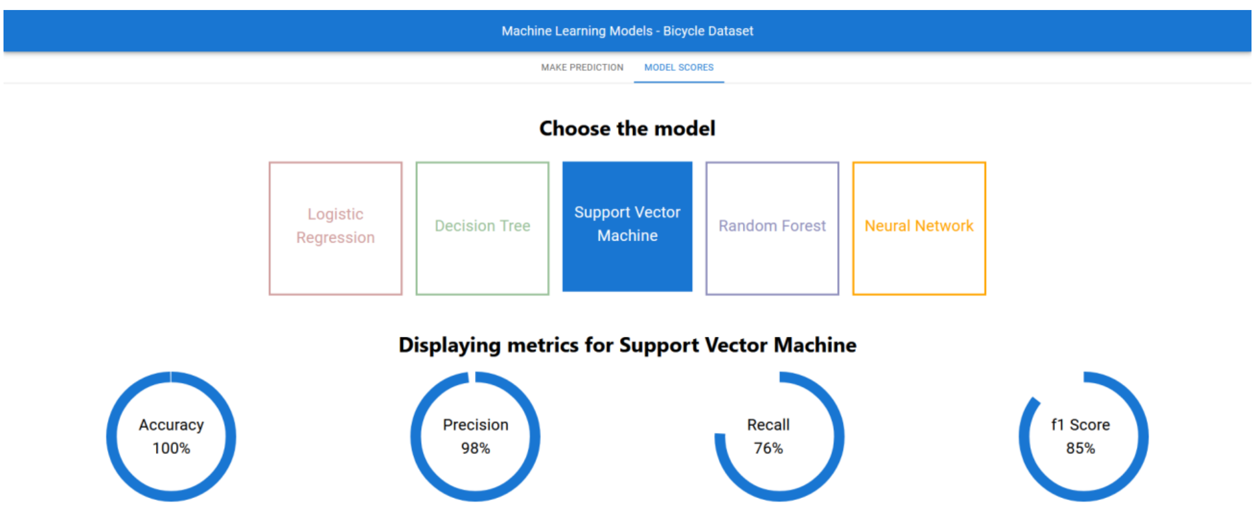
Neural Network Scores



Random Forest Scores



Support Vector Machine Scores



Backend Implementation using Flask

1. Load relevant libraries

```
from flask import Flask, request, jsonify
import traceback
import pandas as pd
#from sklearn import preprocessing
# import pickle
import joblib
import sys
from os import path
from sklearn import metrics
from flask_cors import CORS
```

2. Define pickled models and other files for loading

```
project_folder = r'C:\Projects\COMP247\Final_Project\_deploy'

models = {
          "Random_Forest": "group4_rf_fullpipe_rajiv.pkl"
         ,"Neuro_Network": "group4_nn_fullpipe_v7_andrew.pkl"
         ,"Decision_Tree": "group4_dt_fullpipeline_manvir.pkl"
         ,"Logistic_Regression": "LR_Model_Chung.pkl"
         ,"SVM": "SVM_model_parth.pkl"
         }

cols_pkl = 'group4_model_columns.pkl'

X_train_df = pd.read_csv(path.join(project_folder,"x_train_data.csv"))
y_train_df = pd.read_csv(path.join(project_folder,"y_train_data.csv"))
X_test_df = pd.read_csv(path.join(project_folder,"x_test_data.csv"))
y_test_df = pd.read_csv(path.join(project_folder,"y_test_data.csv"))
```

3. Define flask application

```
# Your API definition
app = Flask(__name__)
CORS(app)
```

4. Create route for generating prediction using selected model

```
@app.route("/predict/<model_name>", methods=['GET','POST'])
def predict(model_name):
    if loaded_model:
        try:
            json_ = request.json
            print('JSON: \n', json_)
            query = pd.DataFrame(json_, columns=model_columns)
            prediction = list(loaded_model[model_name].predict(query))
            print(f'Returning prediction with {model_name} model:')
            print('prediction=', prediction)
            res = jsonify({"prediction": str(prediction)})
            res.headers.add('Access-Control-Allow-Origin', '*')
            return res
        except:
            return jsonify({'trace': traceback.format_exc()})
    else:
        return ('No model available.')
```

5. Create route for generating scores using selected model

```
@app.route("/scores/<model_name>", methods=['GET','POST'])
def scores(model_name):
    if loaded_model:
        try:
            y_pred = loaded_model[model_name].predict(X_test_df)
            print(f'Returning scores for {model_name}:')
            accuracy = metrics.accuracy_score(y_test_df, y_pred)
            precision = metrics.precision_score(y_test_df, y_pred)
            recall = metrics.recall_score(y_test_df, y_pred)
            f1 = metrics.f1_score(y_test_df, y_pred)
            res = jsonify({"accuracy": accuracy,
                           "precision": precision,
                           "recall":recall,
                           "f1": f1
                          })
            res.headers.add('Access-Control-Allow-Origin', '*')
            return res
        except:
            return jsonify({'trace': traceback.format_exc()})
    else:
        return ('No model available.')
```

6. Main driver logic

```
if __name__ == '__main__':
    try:
        port = int(sys.argv[1]) # This is for a command-line input
    except:
        port = 12345

    # load all models:
    loaded_model = {}
    for model_name in (models):
        loaded_model[model_name] = joblib.load(path.join(project_folder, models[model_name]))
        print(f'Model {model_name} loaded')

    model_columns = ['Elapsed_Days_Before_Reported', 'Primary_Offence', 'Occurrence_Year',
          'Occurrence_DayOfWeek', 'Occurrence_DayOfYear', 'Occurrence_Hour',
          'Division', 'City', 'Hood_ID', 'Premises_Type', 'Bike_Make',
          'Bike_Model', 'Bike_Type']

    app.run(port=port, debug=True)
```