



# ACCIDENT SEVERITY PREDICTION

19.09.2020

---

PRIYANKA JAIN

## Problem Understanding

To reduce the number of vehicle collisions in a community, we have to predict the severity of an accident given variable features like weather, road and visibility conditions.

The analysis is to predict the severity of accident occurrence using data provided with features like vehicle count, weather conditions and road conditions in order to inform the public over probability of occurrence of severity of an accident on roads to take well informed measures beforehand. The analysis would help the department of transportation to improve upon the ways and minimize accidents and the public for the purpose of increasing the quality of life for the residents, increasing transparency, accountability and comparability, promoting economic development and research, and improving internal performance management.

## Data Understanding

I have used the data provided in the course. The dataset is available as comma-separated values (CSV) files. The data is also available from RESTful API services in formats such as GeoJSON. Our predictor or target variable will be "SEVERITYCODE" which is used to measure the severity of an accident from 1 to 2 in the dataset. Attributes used to weigh the severity of an accident are "WEATHER", "ROADCOND", "LIGHTCOND" and "VEHCOUNT". The data set consists of 37 features from which weather, road condition, light condition and vehicle count were selected based on correlation analysis of each feature to the target variable "SEVERITYCODE".

## Data Preparation

The data first needs to be converted to the desired data frame which includes all the meaningful features and target set. Also, the type of some of the features are object which need to be converted to desired types for analysis. I have used label encoding to convert the features to our desired data type. The initial data table has features as shown in *table 1*.

|        | SEVERITYCODE | COLLISIONTYPE | VEHCOUNT | WEATHER  | ROADCOND | LIGHTCOND               |
|--------|--------------|---------------|----------|----------|----------|-------------------------|
| 0      | 2            | Angles        | 2        | Overcast | Wet      | Daylight                |
| 1      | 1            | Sideswipe     | 2        | Raining  | Wet      | Dark - Street Lights On |
| 2      | 1            | Parked Car    | 3        | Overcast | Dry      | Daylight                |
| 3      | 1            | Other         | 3        | Clear    | Dry      | Daylight                |
| 4      | 2            | Angles        | 2        | Raining  | Wet      | Daylight                |
| ...    | ...          | ...           | ...      | ...      | ...      | ...                     |
| 194668 | 2            | Head On       | 2        | Clear    | Dry      | Daylight                |
| 194669 | 1            | Rear Ended    | 2        | Raining  | Wet      | Daylight                |
| 194670 | 2            | Left Turn     | 2        | Clear    | Dry      | Daylight                |
| 194671 | 2            | Cycles        | 1        | Clear    | Dry      | Dusk                    |
| 194672 | 1            | Rear Ended    | 2        | Clear    | Wet      | Daylight                |

194673 rows × 6 columns

*Table 1*

Here the target variable is "SEVERITYCODE" which is integer type and vehicle count is integer type and the rest of the features are object type as shown Fig 2.

```
.2]: data.dtypes
ut[42]: SEVERITYCODE      int64
        COLLISIONTYPE    object
        VEHCOUNT          int64
        WEATHER           object
        ROADCOND          object
        LIGHTCOND         object
        dtype: object
```

*Fig 2*

First I have checked for all the null values in the data set and dropped all the rows that contain a minimum of one null value. As the original data set consists of 194673 rows shown in *Table 1*, after dropping the number of rows reduces to 189316 shown in *Fig 3*.

```
]:
```

```
new_data = data.dropna(axis = 0, how = 'any')
```

```
]:
```

```
m=new_data.isnull()
for col in m.columns.values.tolist():
    print(m[col].value_counts())
```

```
False    189316
Name: SEVERITYCODE, dtype: int64
False    189316
Name: COLLISIONTYPE, dtype: int64
False    189316
Name: VEHCOUNT, dtype: int64
False    189316
Name: WEATHER, dtype: int64
False    189316
Name: ROADCOND, dtype: int64
False    189316
Name: LIGHTCOND, dtype: int64
```

Fig 3

Each of the features is described with its unique no of counts and the totals counts as in fig 4.

```
] new_data.describe(include=[ 'object' ])
```

[9]:

|               | COLLISIONTYPE | WEATHER | ROADCOND | LIGHTCOND |
|---------------|---------------|---------|----------|-----------|
| <b>count</b>  | 189316        | 189316  | 189316   | 189316    |
| <b>unique</b> | 10            | 11      | 9        | 9         |
| <b>top</b>    | Parked Car    | Clear   | Dry      | Daylight  |
| <b>freq</b>   | 47815         | 111002  | 124294   | 116064    |

Fig 4

## Data Visualization

To visualize our data we select all features individually and plot bar graph and histogram to see the variability of counts.

For the feature "VEHCOUNT" histogram is plotted showing the number of counts with respect to number of vehicles as shown in Fig 5.

```

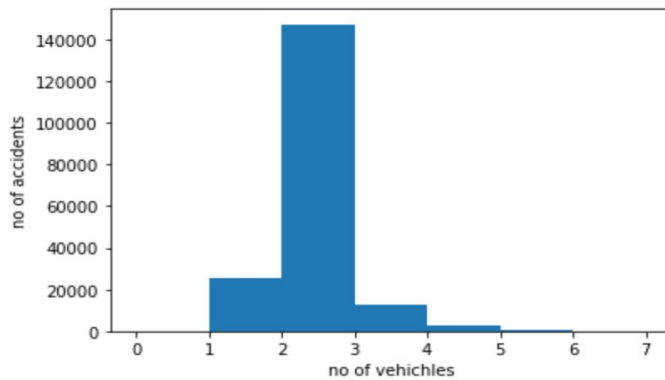
In [ ]: bins=np.arange(new_data['VEHCOUNT'].min(),8,1)
plt.hist(new_data['VEHCOUNT'],bins=bins)
plt.xlabel('no of vehichles')
plt.ylabel('no of accidnets')

```

```

In [ ]: Text(0, 0.5, 'no of accidnets')

```



*Fig 5*

For the feature “ROADCOND”, a bar graph is plotted showing the number of counts with respect to number of vehicles as shown in Fig 6.

```

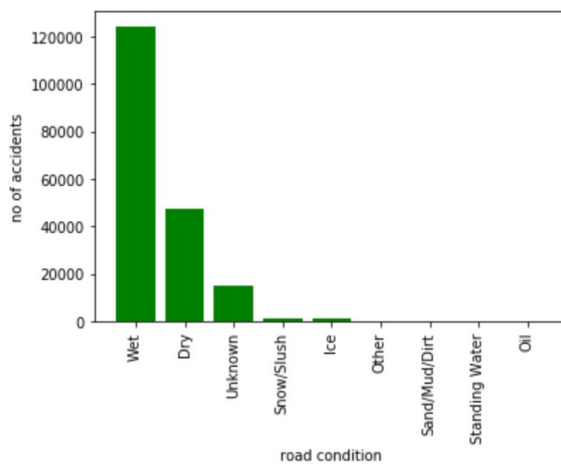
In [ ]: group_rc = new_data['ROADCOND'].unique()
data=new_data['ROADCOND'].value_counts()
plt.bar(x = group_rc,height = data,color = 'g')
plt.xlabel('road condition')
plt.ylabel('no of accidnets')
plt.xticks(rotation = 90)

```

```

In [ ]: ([0, 1, 2, 3, 4, 5, 6, 7, 8], <a list of 9 Text xticklabel objects>)

```

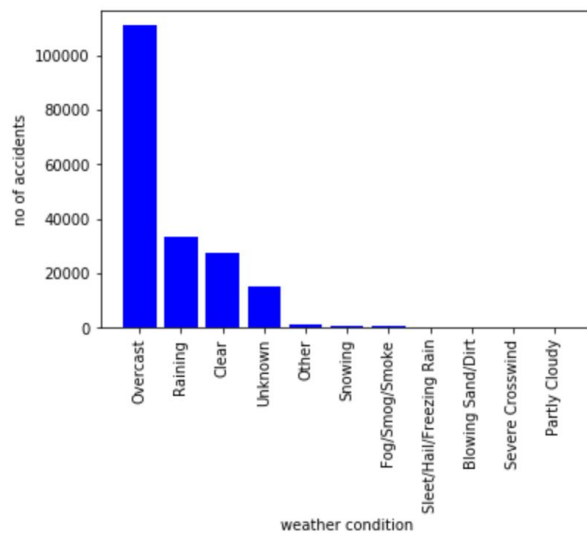


*Fig 6*

For the feature “WEATHER”, a bar graph is plotted showing the number of counts with respect to number of vehicles as shown in *Fig 7*.

```
group_w = new_data['WEATHER'].unique()
data_2=new_data['WEATHER'].value_counts()
plt.bar(x = group_w,height = data_2,color = 'b')
plt.xlabel('weather condition')
plt.ylabel('no of accidents')
plt.xticks(rotation = 90)
```

13]: ([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10], <a list of 11 Text xticklabel objects:



*Fig 7*

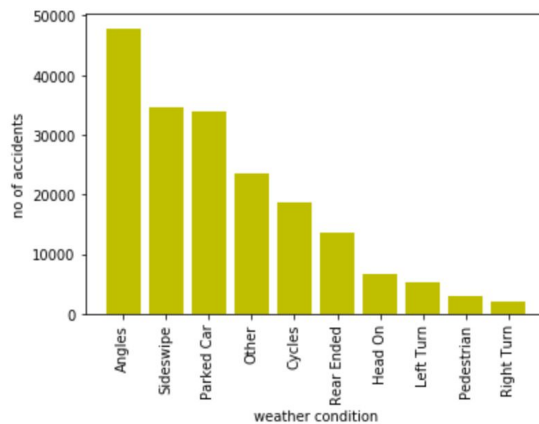
For the feature “COLLISIONTYPE”, a bar graph is plotted showing the number of counts with respect to number of vehicles as shown in *Fig 8*.

```

: ▶ group_ct = new_data['COLLISIONTYPE'].unique()
group_ct
data_3 = new_data['COLLISIONTYPE'].value_counts()
plt.bar(x = group_ct,height = data_3, color = 'y')
plt.xlabel('weather condition')
plt.ylabel('no of accidents')
plt.xticks(rotation = 90)

[14]: ([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], <a list of 10 Text xticklabel objects>)

```



*Fig 8*

For the feature “LIGHTCOND”, a bar graph is plotted showing the number of counts with respect to number of vehicles as shown in Fig 9.

```

group_lt = new_data['LIGHTCOND'].unique()
group_lt
data_4 = new_data['LIGHTCOND'].value_counts()
plt.bar(x = group_lt,height = data_4, color = 'b')
plt.xlabel('light condition')
plt.ylabel('no of accidents')
plt.xticks(rotation = 90)

```

5]: ([0, 1, 2, 3, 4, 5, 6, 7, 8], <a list of 9 Text xticklabel objects>)

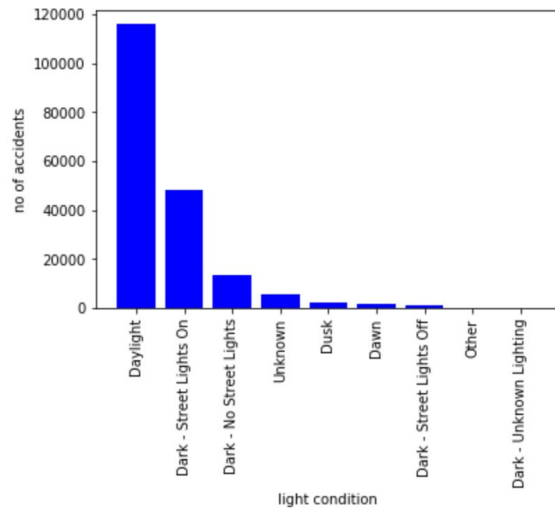


Fig 9

Now , I have converted all the object types to category type before changing data to numerical type using label encoding as in Fig 10.



```

▶ vc=new_data['VEHCOUNT'].unique()
vc
l6]: array([ 2,  3,  1,  4,  0,  7,  5,  6,  8, 11,  9, 10, 12], dtype=int64)

▶ wc=new_data['WEATHER'].unique()
wc
l7]: array(['Overcast', 'Raining', 'Clear', 'Unknown', 'Other', 'Snowing',
          'Fog/Smog/Smoke', 'Sleet/Hail/Freezing Rain', 'Blowing Sand/Dirt',
          'Severe Crosswind', 'Partly Cloudy'], dtype=object)

▶ new_data['SEVERITYCODE'].unique()
l8]: array([2, 1], dtype=int64)

▶ lc=new_data['LIGHTCOND'].unique()
lc
new_data["LIGHTCOND"] = new_data["LIGHTCOND"].astype('category')

rc=new_data['ROADCOND'].unique()
rc
new_data["ROADCOND"] = new_data["ROADCOND"].astype('category')

wc=new_data['WEATHER'].unique()
wc
new_data["WEATHER"] = new_data["WEATHER"].astype('category')

ct=new_data['COLLISIONTYPE'].unique()
ct
new_data["COLLISIONTYPE"] = new_data["COLLISIONTYPE"].astype('category')

```

*Fig 10*

Now converting all types to numerical types using label encoding as in Fig 11. "nWEATHER", "nROADCOND", "nLIGHTCOND" are now converted to integer type from the category type.

## Label Encoding

```

In [20]: data = new_data.drop(columns=['COLLISIONTYPE'])
data["nLIGHTCOND"] = data["LIGHTCOND"].cat.codes
data["nROADCOND"] = data["ROADCOND"].cat.codes
data["nWEATHER"] = data["WEATHER"].cat.codes
data["nWEATHER"] = data["WEATHER"].cat.codes
data.dtypes

```

```

Out[20]: SEVERITYCODE      int64
VEHCOUNT      int64
WEATHER      category
ROADCOND      category
LIGHTCOND      category
nLIGHTCOND      int8
nROADCOND      int8
nWEATHER      int8
dtype: object

```

*Fig 11*

The new data set consists of numerical values. Now the data is ready for building models and evaluation as in *Fig 12*.

```

In [22]:

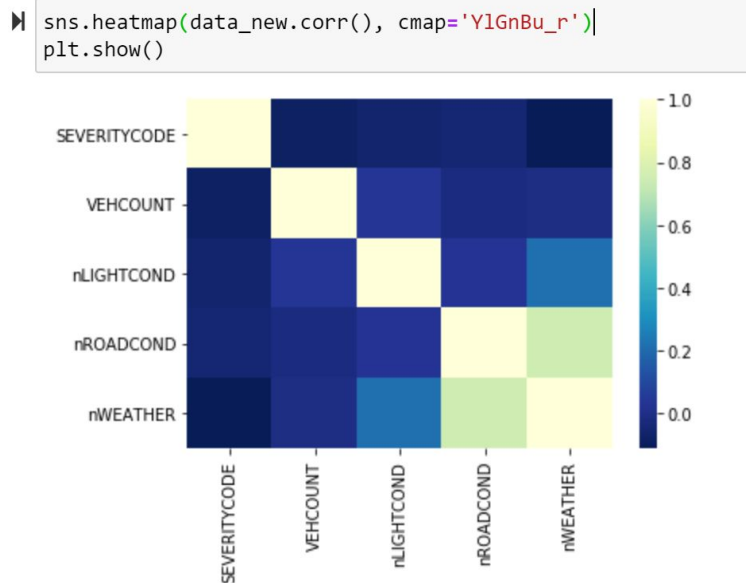
```

|        | SEVERITYCODE | VEHCOUNT | nLIGHTCOND | nROADCOND | nWEATHER |
|--------|--------------|----------|------------|-----------|----------|
| 0      | 2            | 2        | 5          | 8         | 4        |
| 1      | 1            | 2        | 2          | 8         | 6        |
| 2      | 1            | 3        | 5          | 0         | 4        |
| 3      | 1            | 3        | 5          | 0         | 1        |
| 4      | 2            | 2        | 5          | 8         | 6        |
| ...    | ...          | ...      | ...        | ...       | ...      |
| 194668 | 2            | 2        | 5          | 0         | 1        |
| 194669 | 1            | 2        | 5          | 8         | 6        |
| 194670 | 2            | 2        | 5          | 0         | 1        |
| 194671 | 2            | 1        | 6          | 0         | 1        |
| 194672 | 1            | 2        | 5          | 8         | 1        |

189316 rows × 5 columns

*Fig 12*

The heatmap shows the correlation of data among various variables and how they are correlated to other variables. This further helps in identifying features which are highly correlated to the target variable as in *Fig 13*.



*Fig 13*

## Methodology (Modelling and Evaluation)

Now the data is used to build models. The data is labeled and uses supervised learning models to predict the category of data. We will use the following models :

### Linear Regression Model

A linear regression line has an equation of the form  $Y = a + bX$ , where  $X$  is the explanatory variable and  $Y$  is the dependent variable. The slope of the line is  $b$ , and  $a$  is the intercept (the value of  $y$  when  $x = 0$ ).

The model predicts the values of independent variables and the residual sum of square error and variance score as shown in *Fig 14*.

## Linear regression model

```
[24]: ▶ x = np.asarray(data_new[['VEHCOUNT', 'nLIGHTCOND', 'nROADCOND', 'nWEATHER']])
      y = np.asarray(data_new[['SEVERITYCODE']])
      regr=linear_model.LinearRegression()
      regr.fit(x,y)
      print(regr.coef_)

      [[-0.06593003 -0.00696535  0.00845026 -0.02511717]]
```

ordinary least square prediction

```
[25]: ▶ y_hat = regr.predict(data_new[['VEHCOUNT', 'nLIGHTCOND', 'nROADCOND', 'nWEATHER']])
      print("residual sum of square: %.2f" % np.mean(y_hat-y)**2)
      print("variance score: %.2f" % regr.score(x,y))

      residual sum of square:0.00
      variance score:0.02
```

Fig 14

## K-Nearest Neighbor (KNN)

K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure .For my model I have used k =20.

## knn algorithm

```

74]: ➤ from sklearn.model_selection import train_test_split

75]: ➤ x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.2,random_state = 4)
      print('train set',x_train.shape,y_train.shape)
      print('test set',x_test.shape,y_test.shape)

      train set (151452, 4) (151452, 1)
      test set (37864, 4) (37864, 1)

76]: ➤ from sklearn.neighbors import KNeighborsClassifier

77]: ➤ k=20
      neigh = KNeighborsClassifier(n_neighbors = k).fit(x_train,y_train)
      neigh

C:\Users\priya\anaconda3\lib\site-packages\ipykernel_launcher.py:2: DataConversionWarning:
a 1d array was expected. Please change the shape of y to (n_samples, ), for example using r

Out[77]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                             metric_params=None, n_jobs=None, n_neighbors=20, p=2,
                             weights='uniform')

[*]: ➤ y_hat = neigh.predict(x_test)
      y_hat[0:5]

```

The predicted accuracy for the train and test set is shown below in *Fig 15*.

### accuracy evaluation

```

0]: ➤ from sklearn import metrics

0]: ➤ print("train set accuracy",metrics.accuracy_score(y_train,neigh.predict(x_train)))

      train set accuracy 0.7016942661701397

1]: ➤ print("test set accuracy",metrics.accuracy_score(y_test,y_hat))

      test set accuracy 0.6993450242974857

```

*Fig 15*

For the model I have calculated the jacquard similarity score and f1 score as shown in *fig 16*.

```

> from sklearn.metrics import jaccard_similarity_score

> j_score_knn = jaccard_similarity_score(y_test, yhat)
j_score_knn

C:\Users\priya\anaconda3\lib\site-packages\sklearn\metrics
as been deprecated and replaced with jaccard_score. It will
ehavior for binary and multiclass classification tasks.
FutureWarning)

: 0.5557785759560533

> from sklearn.metrics import f1_score
f1_score_knn = f1_score(y_test, yhat, average='weighted')
f1_score_knn

: 0.5645435085163047

```

*Fig 16*

## Decision Tree

Decision Tree a tree based classifier, which splits the sample data into two or more sets. To choose a differentiator (predictor), the algorithm considers all features and does a binary split on them (for categorical data, split by category; for continuous, pick a cut-off threshold). It will then choose the one with the least cost (i.e. highest accuracy), and repeats recursively, until it successfully splits the data in all leaves. Information gain for a decision tree classifier can be calculated Entropy measure gain.

We perform a train/test split on data set and fit the data using decision tree classifier. The predicted values of test data are checked for accuracy as shown in *Fig 17*.

## decision tree

```
[82]: ▶ from sklearn.tree import DecisionTreeClassifier
```

```
[83]: ▶ x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.2,random_state = 4)
      ▶ print('train set',x_train.shape,y_train.shape)
      ▶ print('test set',x_test.shape,y_test.shape)
```

```
train set (151452, 4) (151452, 1)
test set (37864, 4) (37864, 1)
```

```
[84]: ▶ dtree = DecisionTreeClassifier(criterion ="entropy", max_depth = 6)
      ▶ dtree.fit(x_train,y_train)
```

```
Out[84]: DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                                max_depth=6, max_features=None, max_leaf_nodes=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, presort='deprecated',
                                random_state=None, splitter='best')
```

### prediction

```
[85]: ▶ ptree = dtree.predict(x_test)
      ▶ print(ptree[0:5])
```

```
[1 1 1 2 1]
```

### evaluation

```
[86]: ▶ print("DT accuracy",metrics.accuracy_score(y_test,ptree))
```

```
DT accuracy 0.7193112190999366
```

*Fig 17*

The final decision tree for the model is shown below in *Fig 18*-

```
out=tree.export_graphviz(dtree,feature_names=featureNames, out_file=dot_data, class_names= list(map(str, np.unique(y_train)))
```

```
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
```

```
graph.write_png(filename)
img = mpimg.imread(filename)
plt.figure(figsize=(300, 500))
plt.imshow(img,interpolation='nearest')
```

```
41]: <matplotlib.image.AxesImage at 0x2bb123384c8>
```



```
plt.savefig("C:/Users/priya/Desktop/dtree.png", dpi=300)
```

```
<Figure size 432x288 with 0 Axes>
```

Fig 18

## Logistic Regression

Logistic Regression is a classifier that estimates discrete values (binary values like 0/1, yes/no, true/false) based on a given set of independent variables. It basically predicts the probability of occurrence of an event by fitting data to a logistic function. Hence it is also known as logistic regression. The values obtained would always lie within 0 and 1 since it predicts the probability. The chosen dataset has more than two target categories in terms of the accident severity code assigned.



## logistic regression model

```
[ ]: ➤ import scipy.optimize as opt
import pylab as pl

[ ]: ➤ from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split( x, y, test_size=0.2, random_state=4)
print ('Train set:', x_train.shape, y_train.shape)
print ('Test set:', x_test.shape, y_test.shape)

[ ]: ➤ from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.01, solver='liblinear').fit(x_train,y_train)
LR

[ ]: ➤ yhat = LR.predict(x_test)
yhat

[ ]: ➤ yhat_prob = LR.predict_proba(x_test)
yhat_prob
```

The predicted values are shown below-

```

                                warning:
[ ]: ➤ yhat = LR.predict(x_test)
yhat

2]: array([1, 1, 1, ..., 1, 1, 1], dtype=int64)

[ ]: ➤ yhat_prob = LR.predict_proba(x_test)
yhat_prob

3]: array([[0.72076512, 0.27923488],
          [0.7413265 , 0.2586735 ],
          [0.7413265 , 0.2586735 ],
          ...,
          [0.65178535, 0.34821465],
          [0.67634483, 0.32365517],
          [0.85176631, 0.14823369]])
```

I have calculated the jaccard similarity score for evaluation of the logistic regression model in Fig 19.

**evaluation**

```

: ➤ from sklearn.metrics import jaccard_similarity_score
jaccard_similarity_score(y_test, yhat)

C:\Users\priya\anaconda3\lib\site-packages\sklearn\metrics\c
as been deprecated and replaced with jaccard_score. It will b
ehavior for binary and multiclass classification tasks.
FutureWarning)
[94]: 0.6969416860342278

```

*Fig 19*

Another way to evaluate is using confusion matrix which calculates the accuracy of the model using precision and recall values. I have used f1 score to check for best precision and recall value and log loss to measure the performance of the classifier where the predicted output is a probability between 0 and 1.

```

from sklearn.metrics import classification_report, confusion_matrix
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
print(confusion_matrix(y_test, yhat, labels=[1,0]))

```

```

[[26352  0]
 [  0  0]]

```

```

❏ cnf_matrix = confusion_matrix(y_test, yhat, labels=[1,0])
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['churn=1','churn=0'],normalize= False,  title='Confusion matrix')

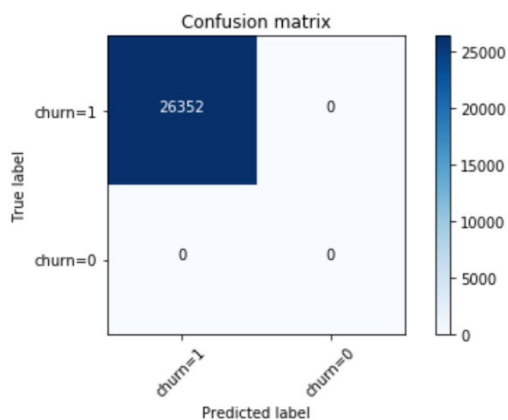
```

Confusion matrix, without normalization

```

[[26352   0]
 [   0   0]]

```



```

❏ print(classification_report(y_test, yhat))

```

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 1            | 0.70      | 1.00   | 0.82     | 26395   |
| 2            | 0.46      | 0.00   | 0.01     | 11469   |
| accuracy     |           |        | 0.70     | 37864   |
| macro avg    | 0.58      | 0.50   | 0.41     | 37864   |
| weighted avg | 0.63      | 0.70   | 0.57     | 37864   |

```

: ❏ from sklearn.metrics import log_loss
log_loss(y_test, yhat_prob)

```

```

[98]: 0.6014771549621085

```

## SVM Model

Support Vector Machine is a linear model for classification and regression problems. The algorithm creates a line or a hyperplane which separates the data into classes. I have used feature set and target set and have trained the data on half the size of original data as it took too long to train the data. The model is shown in *Fig 20-*

### SVM model

```
[8]: ▶ x = np.asarray(data_new[['VEHCOUNT', 'nLIGHTCOND', 'nROADCOND', 'nWEATHER']])
      y = np.asarray(data_new[['SEVERITYCODE']]).ravel()

[ ]: ▶

[9]: ▶ x_train, x_test, y_train, y_test = train_test_split( x, y, test_size=0.5, random_state=4)
      print ('Train set:', x_train.shape, y_train.shape)
      print ('Test set:', x_test.shape, y_test.shape)

      Train set: (94658, 4) (94658,)
      Test set: (94658, 4) (94658,)

[0]: ▶ from sklearn import svm

[1]: ▶ clf = svm.SVC(kernel='rbf', gamma='scale', probability=True)

      ▶ clf.fit(x_train, y_train)

[ ]: SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
      decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
      max_iter=-1, probability=True, random_state=None, shrinking=True, tol=0.001,
      verbose=False)

      ▶ yhat = clf.predict(x_test)
      yhat [0:5]

[ ]: array([1, 1, 1, 2, 1], dtype=int64)
```

*Fig 20*

To calculate the accuracy of the model I have used a confusion matrix. The results are shown below-

evaluation

```

: ➤ from sklearn.metrics import classification_report, confusion_matrix
    import itertools

: ➤ def plot_confusion_matrix(cm, classes,
                             normalize=False,
                             title='Confusion matrix',
                             cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

```

```

# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, yhat, labels=[1,2])
np.set_printoptions(precision=2)

print (classification_report(y_test, yhat))

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['(1)', '(2)'], normalize= False, title='Confusion matrix')

```

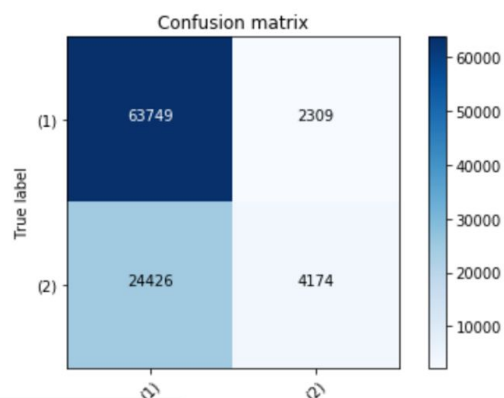
|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 1            | 0.72      | 0.97   | 0.83     | 66058   |
| 2            | 0.64      | 0.15   | 0.24     | 28600   |
| accuracy     |           |        | 0.72     | 94658   |
| macro avg    | 0.68      | 0.56   | 0.53     | 94658   |
| weighted avg | 0.70      | 0.72   | 0.65     | 94658   |

Confusion matrix, without normalization


```

[[63749 2309]
 [24426 4174]]

```



The accuracy of the model is calculated using F1 score and jacquard similarity score. F1 score is 0.648 and jacquard index is 0.717. This shows the model predicts fairly accurately. Since i have used test\_size as 0.5, a similar process can be repeated by decreasing the test size for more accurate results.



```

7] from sklearn.metrics import f1_score
   f1_score(y_test, yhat, average='weighted')
7]: 0.6487858243877695

8] from sklearn.metrics import jaccard_similarity_score
   jaccard_similarity_score(y_test, yhat)
C:\Users\priya\anaconda3\lib\site-packages\sklearn\metrics\jaccard_similarity_score.py:10: FutureWarning:
jaccard_similarity_score has been deprecated and replaced with jaccard_score. Its behavior for binary and multiclass
classification tasks will change in the future.
8]: 0.7175621711846859

```

*Fig 21*

The Jaccard similarity score and F1 score is calculated and shown above in *Fig 21*.

After the model evaluation we can experiment and adjust the model varying test and train size. In the SVM model increasing the train data set would give better accuracy results.

## Conclusion

For the linear regression model the residual sum of square and variance score is calculated. This model is a perfect fit for the data as the residual sum of square is zero and the variance is low.

For the k nearest neighbor algorithm the predicted train and test set accuracy is 0.701 and 0.699. This model is fairly good to predict the data outcome as the accuracy is good.

The decision tree model predicts with an accuracy score of 0.719.

The logistic model used for evaluation also indicates that the data fit the model fairly as Jaccard similarity score is 0.696 and the log loss calculated is 0.601.

The SVM model used took time to train the data as the data set is long. The accuracy of the model is calculated using f1 score and Jaccard similarity score. F1 score is 0.648 and Jaccard index is 0.717.

