classification_01

November 22, 2020

- @Author : Pramil Paudel, Sumit Bhattarai

- Development Env : Jupyter Lab
- Module : Preprocessing
- Summary : This module will create a classification based on the data created from pre-processing

Total Sample Data Size Considered : 499999

Total Number of Columns analysed : 38

0.1 1. Model Description

- We are using Decsion Tree Classifier and KNN Classifier

- Both Gini and Entropy are selected as DT criteria

- Depth Limited Decision Tree / Prunning is done for better modeling

- KNN is run with different neighboures number

[1]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from pylab import rcParams
from mpl_toolkits.mplot3d import Axes3D
from pandas.plotting import scatter_matrix
from IPython.display import Image
import pydotplus
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import preprocessing
from sklearn import svm
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
```

```
from sklearn import preprocessing
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.ensemble import RandomForestRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.datasets import make_regression
from fbprophet import Prophet
from sklearn.metrics import accuracy_score
print("Loaded Successfully -- -- -- -- -- -- -- -- -- -- -- -- --")
```

```
Loaded Successfully -- -- -- -- -- -- -- -- -- -- -- -- --

/Users/patthar/opt/anaconda3/lib/python3.7/site-
packages/sklearn/externals/six.py:31: DeprecationWarning: The module is
deprecated in version 0.21 and will be removed in version 0.23 since we've
dropped support for Python 2.7. Please rely on the official version of six
(https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", DeprecationWarning)
```

Data is read after preprocessing is done. The output of preprocessing is directly read and started working on that.

[2]:
```
source_path = "../../data/pre_processing/"
us_road_accident_df_0 = pd.read_csv(source_path+"xaa", index_col=0)
us_road_accident_df_0.head()
```

[2]:
```
    ID  Severity           Start_Time             End_Time  Start_Lat  \
0  A-1         3  2016-02-08 05:46:00  2016-02-08 11:00:00  39.865147
1  A-2         2  2016-02-08 06:07:59  2016-02-08 06:37:59  39.928059
2  A-3         2  2016-02-08 06:49:27  2016-02-08 07:19:27  39.063148
3  A-4         3  2016-02-08 07:23:34  2016-02-08 07:53:34  39.747753
4  A-5         2  2016-02-08 07:39:07  2016-02-08 08:09:07  39.627781

   Start_Lng                     Street Side          City     County  …  \
0 -84.058723                     I-70 E    R        Dayton  Montgomery  …
1 -82.831184                   Brice Rd    L  Reynoldsburg    Franklin  …
2 -84.032608            State Route 32    R  Williamsburg    Clermont  …
3 -84.205582                     I-75 S    R        Dayton  Montgomery  …
4 -84.188354  Miamisburg Centerville Rd    R        Dayton  Montgomery  …

  Roundabout Station   Stop Traffic_Calming Traffic_Signal Turning_Loop  \
0      False   False  False           False          False        False
1      False   False  False           False          False        False
```

```
2      False    False   False             False            True          False
3      False    False   False             False           False          False
4      False    False   False             False            True          False

   Sunrise_Sunset             countyState  State_FIPS_Code  County_FIPS_Code
0           Night  Montgomery County, OH               39               113
1           Night    Franklin County, OH               39                49
2           Night    Clermont County, OH               39                25
3           Night  Montgomery County, OH               39               113
4             Day  Montgomery County, OH               39               113

[5 rows x 39 columns]
```

[23]:
```
us_road_accident_df_0.shape
```

[23]:
```
(499999, 38)
```

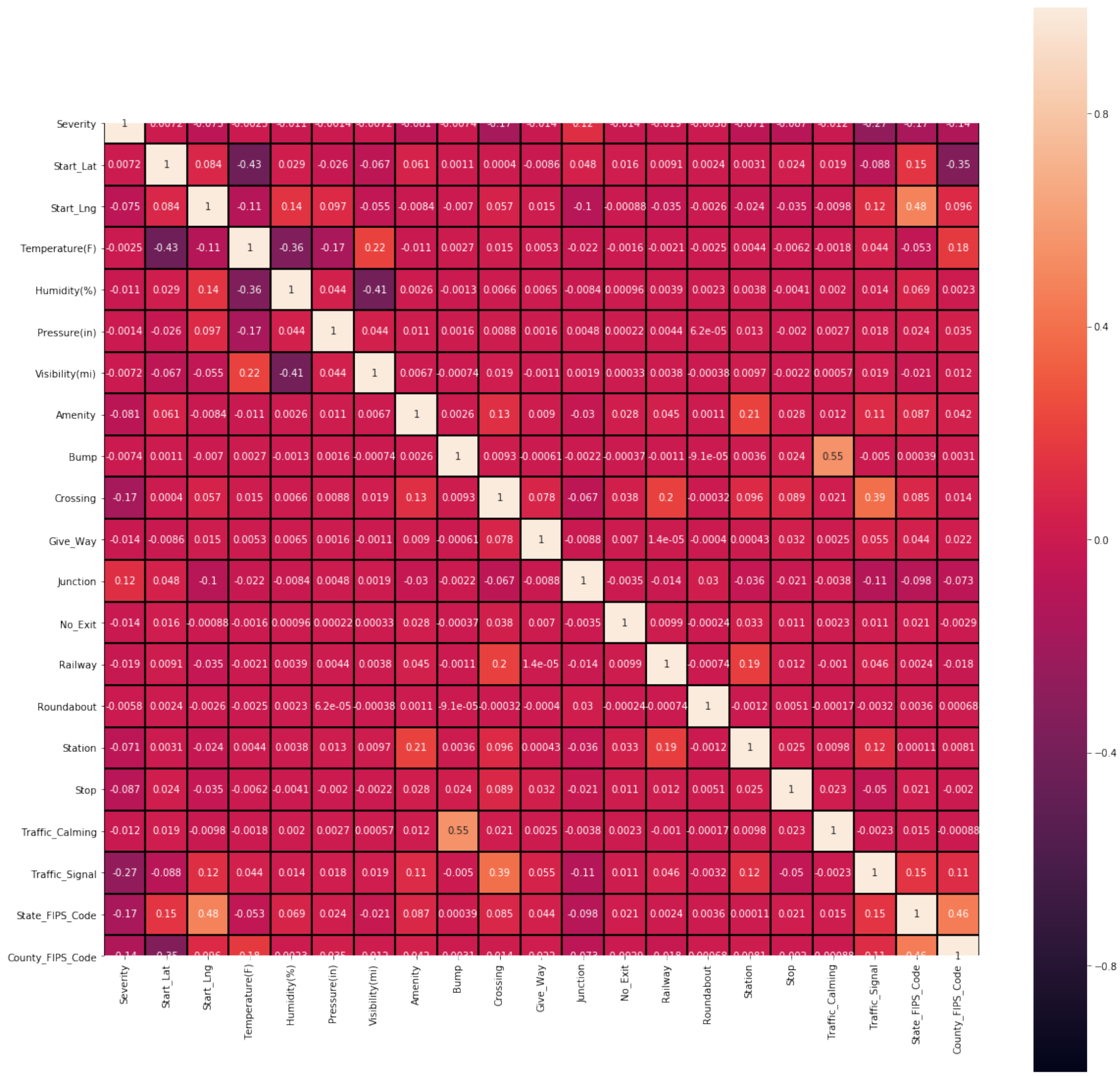
0.1.1 We found there is no variance based on Turning Loop so this attribute is droped in the beigning of the process itself.

[3]:
```
### Every 'Turning_Loop' value is False so lets drop it
us_road_accident_df_0 = us_road_accident_df_0.drop(['Turning_Loop'],axis=1)
```

0.2 2. Finding the corelation in the data/features

On of the idea to select features that are relevent to the classification is selecting features based on the correlation with target variable. So a heatmap of correlation values is plotted.

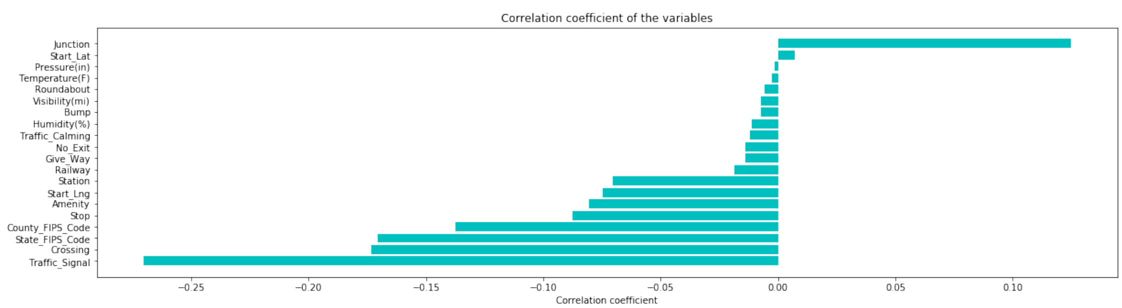
[4]:
```
fig=plt.gcf()
fig.set_size_inches(20,20)
fig=sns.heatmap(us_road_accident_df_0.
 ↪corr(),annot=True,linewidths=1,linecolor='k',square=True,mask=False,␣
 ↪vmin=-1, vmax=1,cbar_kws={"orientation": "vertical"},cbar=True)
```



The heatmap provided correlation values to all of the features which are either boolen, float or int type. We didn't converted some of the features to these datatypes as many of them share almost no relation with the target value. Selecting large number of features might not be the good idea always so selecting few and best features is best approach. We decided to plot correlation values in a graph and calculate most significant attributes for the classfication

[5]:
```
x_cols = [col for col in us_road_accident_df_0.columns if col not in␣
↪['Severity'] if
          (us_road_accident_df_0[col].dtype=='float64' or␣
↪us_road_accident_df_0[col].dtype=='bool' or us_road_accident_df_0[col].
↪dtype=='int64')
         ]
```

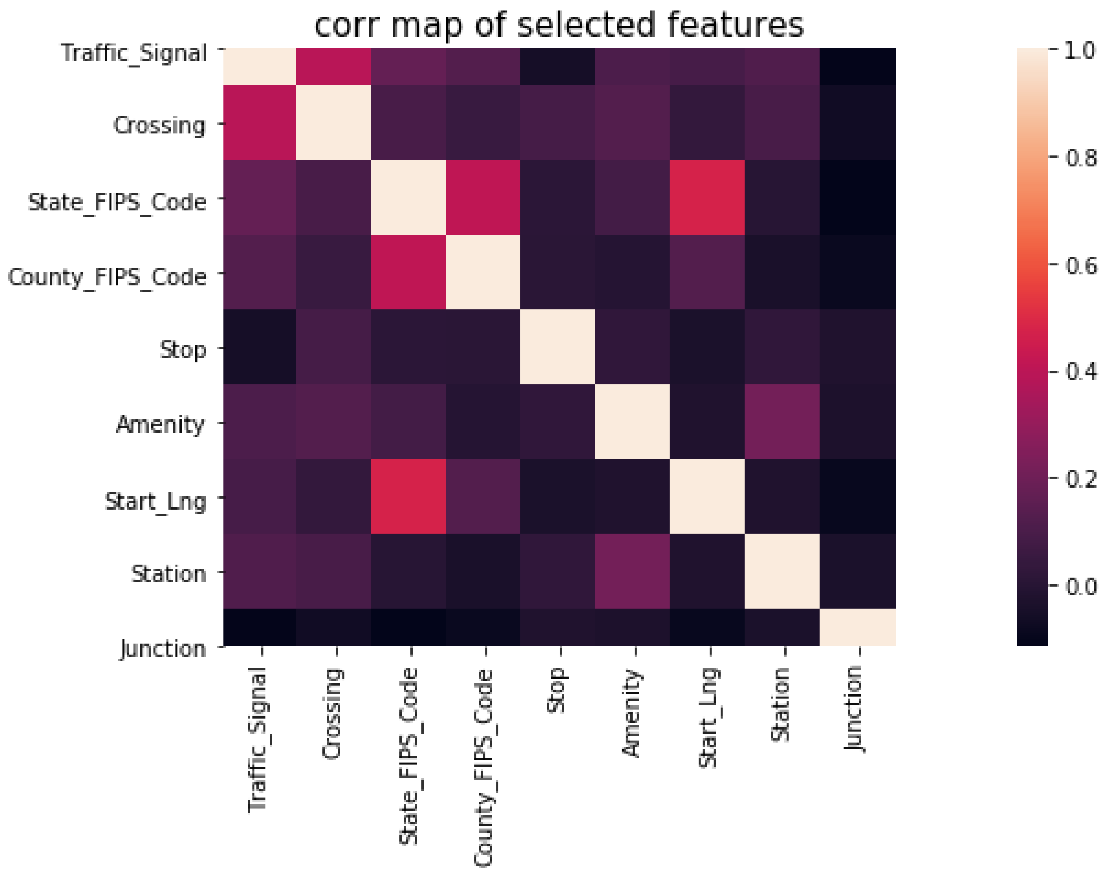
```python
labels = []
values = []
for col in x_cols:
    labels.append(col)
    values.append(np.corrcoef(us_road_accident_df_0[col].values, us_road_accident_df_0.Severity.values)[0,1])
corr_df = pd.DataFrame({'col_labels':labels, 'corr_values':values})
corr_df = corr_df.sort_values(by='corr_values')

ind = np.arange(len(labels))
width = 1
fig, ax = plt.subplots(figsize=(20,5))
rects = ax.barh(ind, np.array(corr_df.corr_values.values), color='c')
ax.set_yticks(ind)
ax.set_yticklabels(corr_df.col_labels.values, rotation='horizontal')
ax.set_xlabel("Correlation coefficient")
ax.set_title("Correlation coefficient of the variables")
plt.show()
```



0.3 3. Selected Features

After plotting corelation values of different features we decided to pull those features which are significant to the classification. So based on the above calculation those features having either positive or negative >0.05 are selected as features for the classification. Heatmap of only these features is also created.

[6]:
```python
corr_df_sel = corr_df.loc[(corr_df['corr_values']>0.05) | (corr_df['corr_values'] < -0.05)]
corr_df_ = corr_df_sel.col_labels.tolist()
tem_df = us_road_accident_df_0[corr_df_]
corrmat = tem_df.corr(method='spearman')
fig,ax= plt.subplots(figsize=(20,5))
sns.heatmap(corrmat,vmax=1,square = True)
plt.title('corr map of selected features',fontsize=15)
plt.show()
```



0.4 4. Filtering the data from Dataframe

Those features selected from step 3 are now filtered from the original dataset

[7]:
```
us_accident_features = ["Traffic_Signal",
                         "Crossing",
                         "State_FIPS_Code",
                         "County_FIPS_Code",
                         "Stop",
                         "Amenity",
                         "Start_Lng",
                         "Station",
                         "Junction",
                         "Severity"
                        ]

us_road_accident_df_1 = us_road_accident_df_0.filter(us_accident_features)
us_road_accident_df_1.head()
```

[7]:

```
   Traffic_Signal  Crossing  State_FIPS_Code  County_FIPS_Code   Stop  \
0           False     False               39               113  False
1           False     False               39                49  False
2            True     False               39                25  False
3           False     False               39               113  False
4            True     False               39               113  False

   Amenity  Start_Lng  Station  Junction  Severity
0    False -84.058723    False     False         3
1    False -82.831184    False     False         2
2    False -84.032608    False     False         2
3    False -84.205582    False     False         3
4    False -84.188354    False     False         2
```

0.5 5. Level Encoding

Some of the values are boolean True/False, some are lattitue with float value. To make them uniform we did LabelEncoding using preprocessing from sklearn. All boolean variables are converted to 0/1 and severity to [0,1,2,3] and Lattitude to some unique positive integer value

[8]:

```python
le = preprocessing.LabelEncoder()
us_road_accident_df_1 = us_road_accident_df_1.apply(le.fit_transform)
us_road_accident_df_1.head()
```

[8]:

```
   Traffic_Signal  Crossing  State_FIPS_Code  County_FIPS_Code  Stop  Amenity  \
0               0         0               30                59     0        0
1               0         0               30                26     0        0
2               1         0               30                14     0        0
3               0         0               30                59     0        0
4               1         0               30                59     0        0

   Start_Lng  Station  Junction  Severity
0     110535        0         0         2
1     117888        0         0         1
2     110621        0         0         1
3     108808        0         0         2
4     109232        0         0         1
```

0.6 6. Conversion into test/train data set

The sample data frame is now converted to train/test set of 0.7:0.3

[9]:

```python
#Separating target column at first
feature_columns = ["Traffic_Signal",
                   "Crossing",
                    "State_FIPS_Code",
                    "County_FIPS_Code",
```

```
                     "Stop",
                     "Amenity",
                     "Start_Lng",
                     "Station",
                     "Junction"]
target_columns=['Severity']

# Retaining required columns in each DF
x_us_accident_df = us_road_accident_df_1[feature_columns]
y_us_accident_df = us_road_accident_df_1[target_columns]

x_train,x_test,y_train,y_test=train_test_split(x_us_accident_df,y_us_accident_df,train_size=0.
↪7,test_size=0.3,random_state=123)
```

[10]:
```
x_train.head()
```

[10]:
```
        Traffic_Signal  Crossing  State_FIPS_Code  County_FIPS_Code  Stop  \
16774                0         0                3                52     0
476904               0         0                3                 0     0
452961               1         0                7                54     0
493161               0         0               37               164     0
128846               1         1                7                50     0

        Amenity  Start_Lng  Station  Junction
16774         0      24955        0         0
476904        0      17337        0         0
452961        0     119802        0         0
493161        0      70916        0         0
128846        0     135145        0         0
```

0.7 6. Decision Tree Classifier

There are two best known decision tree classification based on criteria.

- Entropy
- Gini #### We applied either of them checked the accuracy of the classification. Since we have large dataset we decided to use depth limited decision tree prunning the branches. For this purpose two methods are written with max_depth as a parameter to check the decision tree behaviour of the data.

[11]:
```
feature_columns = ["Traffic_Signal",
                   "Crossing",
                    "State_FIPS_Code",
                    "County_FIPS_Code",
                    "Stop",
                    "Amenity",
                    "Start_Lng",
                    "Station",
```

```
                    "Junction"]
target_columns=['Severity']

# By default decision tree classifier is of gini type in SK-learn so gini is␣
↪used.
def gini_classifier_road_accident(max_depth):
    gini_classifier = DecisionTreeClassifier(max_depth=max_depth)
    gini_classifier = gini_classifier.fit(x_train, y_train)

    # generating images only for depth less than 5 as data contains to many␣
↪features and its hard to add add decision tree in the
    # o/p image file

    if max_depth <5:
        dot_file = StringIO()
        export_graphviz(gini_classifier, filled=True,
                        rounded=True,
                        special_characters=True,
                        feature_names=feature_columns,
                        out_file=dot_file)
        graph = pydotplus.graph_from_dot_data(dot_file.getvalue())
        png_name = "gini_tree_with_depth_{}_and_with_training_{}_.png".
↪format(max_depth, 100-100*0.3)
        graph.write_png(png_name)
        Image(graph.create_png())


    # predict the response for the test data set
    y_pred = gini_classifier.predict(x_test)
    # Evaluating model
    # Checking agnaist real data
    result = "Accuracy of Gini Classifier using {} % training and depth {}: : :␣
↪: ".format(100 - 100*0.3, max_depth)
    print(result, metrics.accuracy_score(y_test, y_pred))

    if max_depth == 20 :
        m = confusion_matrix(y_test, y_pred)
        print(m)

# Here decision tree classifier is selected as entropy. Gini is supposed to be␣
↪better than entropy.
def entropy_classifier_road_accident(max_depth):
    gini_classifier = DecisionTreeClassifier(criterion="entropy",␣
↪max_depth=max_depth)
    gini_classifier = gini_classifier.fit(x_train, y_train)
```

```
    # generating images only for depth less than 5 as data contains to many␣
↪features and its hard to add add decision tree in the
   # o/p image file    dot_file = StringIO()
   if max_depth < 5 :
       dot_file = StringIO()
       export_graphviz(gini_classifier, filled=True,
                       rounded=True,
                       special_characters=True,
                       feature_names=feature_columns,
                       out_file=dot_file)
       graph = pydotplus.graph_from_dot_data(dot_file.getvalue())
       png_name = "insurance_entropy_tree_with_depth_{}_and_with_training_{}_.
↪png".format(max_depth, 100-100*0.3)
       graph.write_png(png_name)
       Image(graph.create_png())
   # predict the response for the test data set
   y_pred = gini_classifier.predict(x_test)
   # Evaluating model
   # Checking agnaist real data
   result = "Accuracy of Entropy Classifier using {} % training and depth {}: :
↪ : : ".format(100 - 100*0.3, max_depth)
   print(result, metrics.accuracy_score(y_test, y_pred))

   if max_depth == 20:
       m = confusion_matrix(y_test, y_pred)
       print(m)
```

The output from 'DT classification using GINI' appeared like below. The accuracy kept increasing with increment of the depth. Which usually doesn't happen. The confusion matrix for depth 20 is also plotted

[12]:
```
for i in range(21):
        if i > 0:
            gini_classifier_road_accident(i)
```

```
Accuracy of Gini Classifier using 70.0 % training and depth 1: : : :
0.6360133333333333
Accuracy of Gini Classifier using 70.0 % training and depth 2: : : :
0.6360133333333333
Accuracy of Gini Classifier using 70.0 % training and depth 3: : : :
0.6663866666666667
Accuracy of Gini Classifier using 70.0 % training and depth 4: : : :
0.6819066666666667
Accuracy of Gini Classifier using 70.0 % training and depth 5: : : :  0.6943
Accuracy of Gini Classifier using 70.0 % training and depth 6: : : :
0.7009333333333333
Accuracy of Gini Classifier using 70.0 % training and depth 7: : : :
```

```
0.7124733333333333
Accuracy of Gini Classifier using 70.0 % training and depth 8: : : :
0.7339066666666667
Accuracy of Gini Classifier using 70.0 % training and depth 9: : : :
0.7475333333333334
Accuracy of Gini Classifier using 70.0 % training and depth 10: : : :
0.7635866666666666
Accuracy of Gini Classifier using 70.0 % training and depth 11: : : :  0.7734
Accuracy of Gini Classifier using 70.0 % training and depth 12: : : :
0.7911733333333333
Accuracy of Gini Classifier using 70.0 % training and depth 13: : : :
0.8015533333333333
Accuracy of Gini Classifier using 70.0 % training and depth 14: : : :
0.8101066666666666
Accuracy of Gini Classifier using 70.0 % training and depth 15: : : :  0.81742
Accuracy of Gini Classifier using 70.0 % training and depth 16: : : :
0.8262866666666666
Accuracy of Gini Classifier using 70.0 % training and depth 17: : : :  0.833
Accuracy of Gini Classifier using 70.0 % training and depth 18: : : :
0.8433533333333333
Accuracy of Gini Classifier using 70.0 % training and depth 19: : : :
0.8517666666666667
Accuracy of Gini Classifier using 70.0 % training and depth 20: : : :  0.86004
[[    1    79    37     0]
 [   23 84921 10449     9]
 [    3 10217 44083    29]
 [    0    37   111     1]]
```

The output from "DT Classification using Entropy" appread like below. There is very minor difference between them in terms of accuracy.

[13]:
```
for i in range(21):
        if i > 0:
            entropy_classifier_road_accident(i)
```

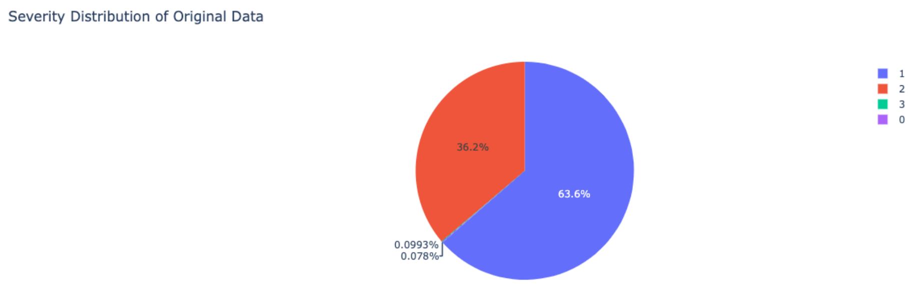
```
Accuracy of Entropy Classifier using 70.0 % training and depth 1: : : :
0.6360133333333333
Accuracy of Entropy Classifier using 70.0 % training and depth 2: : : :
0.6360133333333333
Accuracy of Entropy Classifier using 70.0 % training and depth 3: : : :  0.6519
Accuracy of Entropy Classifier using 70.0 % training and depth 4: : : :
0.6543533333333333
Accuracy of Entropy Classifier using 70.0 % training and depth 5: : : :  0.66366
Accuracy of Entropy Classifier using 70.0 % training and depth 6: : : :
0.6882533333333334
Accuracy of Entropy Classifier using 70.0 % training and depth 7: : : :
0.7021666666666667
Accuracy of Entropy Classifier using 70.0 % training and depth 8: : : :
```

```
0.7262066666666667
Accuracy of Entropy Classifier using 70.0 % training and depth 9: : : :
0.7318066666666667
Accuracy of Entropy Classifier using 70.0 % training and depth 10: : : :
0.74606
Accuracy of Entropy Classifier using 70.0 % training and depth 11: : : :
0.7554733333333333
Accuracy of Entropy Classifier using 70.0 % training and depth 12: : : :  0.7715
Accuracy of Entropy Classifier using 70.0 % training and depth 13: : : :
0.7841733333333333
Accuracy of Entropy Classifier using 70.0 % training and depth 14: : : :
0.7954333333333333
Accuracy of Entropy Classifier using 70.0 % training and depth 15: : : :
0.8052333333333334
Accuracy of Entropy Classifier using 70.0 % training and depth 16: : : :
0.8128733333333333
Accuracy of Entropy Classifier using 70.0 % training and depth 17: : : :
0.82224
Accuracy of Entropy Classifier using 70.0 % training and depth 18: : : :
0.8303133333333333
Accuracy of Entropy Classifier using 70.0 % training and depth 19: : : :
0.8402933333333333
Accuracy of Entropy Classifier using 70.0 % training and depth 20: : : :
0.8497533333333334
[[    1    80    36     0]
 [   33 83861 11496    12]
 [    6 10691 43600    35]
 [    0    36   112     1]]
```

0.8 7. Analysing the DT O/P using Visualization

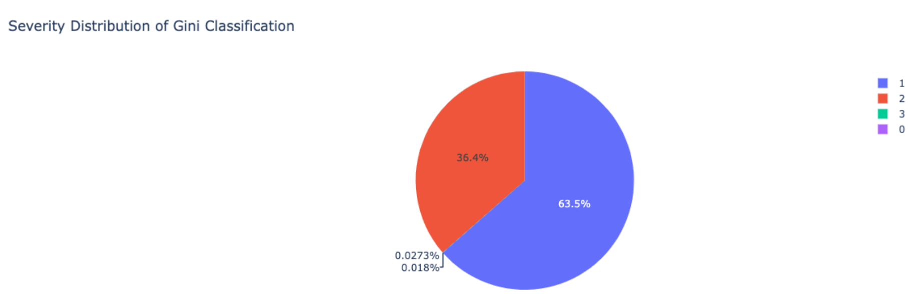
0.8.1 7.1 Severity Distribution of Original Data Set

[14]:
```python
import plotly.express as px
group_by_severity = y_test.groupby(['Severity'])['Severity'].count().
    ↪to_frame(name="Count").reset_index()
fig = px.pie(group_by_severity, values='Count', names='Severity',␣
    ↪title='Severity Distribution of Original Data')
fig.show()
```



0.8.2 7.2 Severity Distribution of Classified data using gini classsification

```
[15]: feature_columns = ["Traffic_Signal",
                    "Crossing",
                     "State_FIPS_Code",
                     "County_FIPS_Code",
                     "Stop",
                     "Amenity",
                     "Start_Lng",
                     "Station",
                     "Junction"]
target_columns=['Severity']
gini_classifier = DecisionTreeClassifier(max_depth=20)
gini_classifier = gini_classifier.fit(x_train, y_train)
y_pred = gini_classifier.predict(x_test)
predicated_df = pd.DataFrame(data=y_pred, columns=["Severity"])
group_by_severity = predicated_df.groupby(['Severity'])['Severity'].count().
 ↪to_frame(name="Count").reset_index()
fig = px.pie(group_by_severity, values='Count', names='Severity',␣
 ↪title='Severity Distribution of Gini Classification')
fig.show()
```



0.9 8. Why depth is continuouly increasing accuracy ?

[16]:
```
#Separating target column at first
feature_columns = ["Traffic_Signal",
                   "Crossing",
                   "Stop",
                   "Amenity",
                   "Station",
                   "Junction"]
target_columns=['Severity']

# Retaining required columns in each DF
x_us_accident_df_1 = us_road_accident_df_1[feature_columns]
y_us_accident_df_1 = us_road_accident_df_1[target_columns]

x_train_1,x_test_1,y_train_1,y_test_1=train_test_split(x_us_accident_df_1,y_us_accident_df_1,t
 ↪7,test_size=0.3,random_state=123)

# By default decision tree classifier is of gini type in SK-learn so gini is␣
 ↪used.
def gini_classifier_road_accident_checking_fitting(max_depth):
    gini_classifier = DecisionTreeClassifier(max_depth=max_depth)
    gini_classifier = gini_classifier.fit(x_train_1, y_train_1)

    # generating images only for depth less than 5 as data contains to many␣
 ↪features and its hard to add add decision tree in the
    # o/p image file

    if max_depth <5:
        dot_file = StringIO()
        export_graphviz(gini_classifier, filled=True,
                        rounded=True,
                        special_characters=True,
                        feature_names=feature_columns,
                        out_file=dot_file)
        graph = pydotplus.graph_from_dot_data(dot_file.getvalue())
        png_name =␣
 ↪"updated_gini_tree_with_depth_{}_and_with_training_{}_removing_geo_information.
 ↪png".format(max_depth, 100-100*0.3)
        graph.write_png(png_name)
        Image(graph.create_png())


    # predict the response for the test data set
    y_pred_1 = gini_classifier.predict(x_test_1)
```

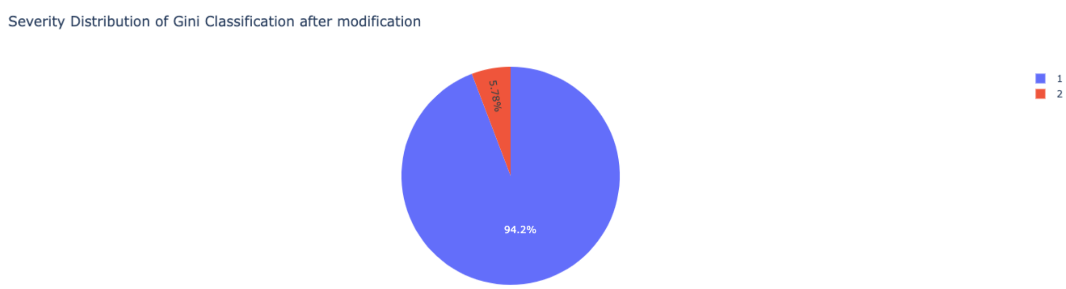
```python
    # Evaluating model
    # Checking agnaist real data
    result = "Accuracy using Gini Classifier using {} % training and depth {}: :␣
↪: : ".format(100 - 100*0.3, max_depth)
    print(result, metrics.accuracy_score(y_test_1, y_pred_1))

    if max_depth == 20 :
        m = confusion_matrix(y_test_1, y_pred_1)
        print(m)
```

[17]:
```python
for i in range(21):
        if i > 0:
            gini_classifier_road_accident_checking_fitting(i)
```

```
Accuracy using Gini Classifier using 70.0 % training and depth 1: : : :
0.6360133333333333
Accuracy using Gini Classifier using 70.0 % training and depth 2: : : :
0.6359333333333334
Accuracy using Gini Classifier using 70.0 % training and depth 3: : : :  0.64788
Accuracy using Gini Classifier using 70.0 % training and depth 4: : : :
0.6478933333333333
Accuracy using Gini Classifier using 70.0 % training and depth 5: : : :  0.6479
Accuracy using Gini Classifier using 70.0 % training and depth 6: : : :  0.6479
Accuracy using Gini Classifier using 70.0 % training and depth 7: : : :  0.6479
Accuracy using Gini Classifier using 70.0 % training and depth 8: : : :  0.6479
Accuracy using Gini Classifier using 70.0 % training and depth 9: : : :  0.6479
Accuracy using Gini Classifier using 70.0 % training and depth 10: : : :  0.6479
Accuracy using Gini Classifier using 70.0 % training and depth 11: : : :  0.6479
Accuracy using Gini Classifier using 70.0 % training and depth 12: : : :  0.6479
Accuracy using Gini Classifier using 70.0 % training and depth 13: : : :  0.6479
Accuracy using Gini Classifier using 70.0 % training and depth 14: : : :  0.6479
Accuracy using Gini Classifier using 70.0 % training and depth 15: : : :  0.6479
Accuracy using Gini Classifier using 70.0 % training and depth 16: : : :  0.6479
Accuracy using Gini Classifier using 70.0 % training and depth 17: : : :  0.6479
Accuracy using Gini Classifier using 70.0 % training and depth 18: : : :  0.6479
Accuracy using Gini Classifier using 70.0 % training and depth 19: : : :  0.6479
Accuracy using Gini Classifier using 70.0 % training and depth 20: : : :  0.6479
[[    0   115     2     0]
 [    0 91962  3440     0]
 [    0 49109  5223     0]
 [    0   140     9     0]]
```

0.9.1 8.1 Severity distribution after modification

[18]:
```
feature_columns = ["Traffic_Signal",
                   "Crossing",
                    "State_FIPS_Code",
                    "County_FIPS_Code",
                    "Stop",
                    "Amenity",
                    "Start_Lng",
                    "Station",
                    "Junction"]
target_columns=['Severity']
gini_classifier = DecisionTreeClassifier(max_depth=20)
gini_classifier = gini_classifier.fit(x_train_1, y_train_1)


y_pred_1 = gini_classifier.predict(x_test_1)
predicated_df = pd.DataFrame(data=y_pred_1, columns=["Severity"])
group_by_severity = predicated_df.groupby(['Severity'])['Severity'].count().
 ↪to_frame(name="Count").reset_index()
fig = px.pie(group_by_severity, values='Count', names='Severity',␣
 ↪title='Severity Distribution of Gini Classification after modification')
fig.show()
```



Thus it implies that, the 'Severity' is to be analysed in terms of geographical information too.

0.10 9 KNN Classifier

After Decision Tree Classification we used K-NN as another classification approach. For this a generic method doing prediction based on number of neighbours is written as below

[19]:
```
```

```
def road_accident_knn_classifier(neighbours):
    knn = KNeighborsClassifier(n_neighbors = neighbours).fit(x_train, y_train.
 ↪values.ravel())

    # accuracy on X_test
    accuracy = knn.score(x_test, y_test)
    result = "Accuracy Using KNN-Classifier with {} % training and {}␣
 ↪neighbours is {}:".format(100 - 100*0.3, neighbours,accuracy)
    print(result)
    # creating a confusion matrix
    knn_predictions = knn.predict(x_test)
#     cm = confusion_matrix(y_test, knn_predictions)
#     print(cm)
```

[20]:
```
for i in range(15):
        if i > 0:
            road_accident_knn_classifier(i)
```

```
Accuracy Using KNN-Classifier with 70.0 % training and 1 neighbours is 0.87514:
Accuracy Using KNN-Classifier with 70.0 % training and 2 neighbours is
0.8740466666666666:
Accuracy Using KNN-Classifier with 70.0 % training and 3 neighbours is
0.8833266666666667:
Accuracy Using KNN-Classifier with 70.0 % training and 4 neighbours is 0.882:
Accuracy Using KNN-Classifier with 70.0 % training and 5 neighbours is 0.87944:
Accuracy Using KNN-Classifier with 70.0 % training and 6 neighbours is 0.87708:
Accuracy Using KNN-Classifier with 70.0 % training and 7 neighbours is
0.8743666666666666:
Accuracy Using KNN-Classifier with 70.0 % training and 8 neighbours is
0.8728266666666666:
Accuracy Using KNN-Classifier with 70.0 % training and 9 neighbours is
0.8688533333333334:
Accuracy Using KNN-Classifier with 70.0 % training and 10 neighbours is 0.86782:
Accuracy Using KNN-Classifier with 70.0 % training and 11 neighbours is 0.86432:
Accuracy Using KNN-Classifier with 70.0 % training and 12 neighbours is
0.8632066666666667:
Accuracy Using KNN-Classifier with 70.0 % training and 13 neighbours is
0.8596533333333334:
Accuracy Using KNN-Classifier with 70.0 % training and 14 neighbours is 0.85814:
```

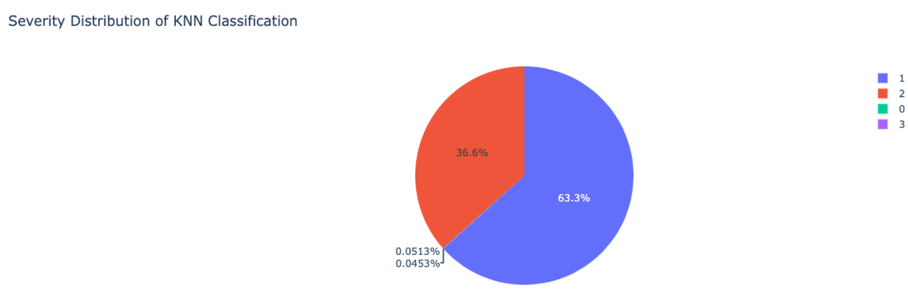
From above execution we found 3 is the best no of neighbours to be considered.

0.10.1 9.1 Confusion matrix and Distribution graph for KNN

As from above code number of neighbours are calculted now confusion matrix and distribution chart is ploted for neighbours = 3

[21]:
```python
    knn = KNeighborsClassifier(n_neighbors = 3).fit(x_train, y_train.values.ravel())
# accuracy on X_test
    accuracy = knn.score(x_test, y_test)
    result = "Accuracy Using KNN-Classifier with {} % training and {} neighbours is {}:".format(100 - 100*0.3, 3,accuracy)
    print(result)
    # creating a confusion matrix
    knn_predictions = knn.predict(x_test)
    cm = confusion_matrix(y_test, knn_predictions)
    print(cm)


    predicated_df = pd.DataFrame(data=knn_predictions, columns=["Severity"])
    group_by_severity = predicated_df.groupby(['Severity'])['Severity'].count().to_frame(name="Count").reset_index()
    fig = px.pie(group_by_severity, values='Count', names='Severity', title='Severity Distribution of KNN Classification')
    fig.show()
```

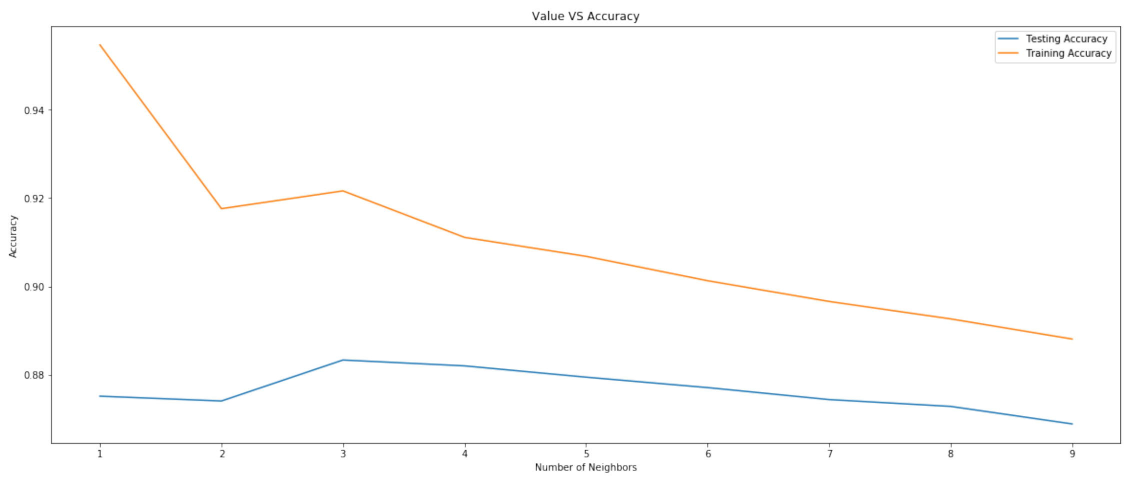
```
Accuracy Using KNN-Classifier with 70.0 % training and 3 neighbours is
0.8833266666666667:
[[    0    90    27     0]
 [   47 86538  8816     1]
 [   30  8284 45956    62]
 [    0    35   109     5]]
```



Apart from distribution graph a graph for test/train accuracy is also plotted as below

[22]:
```python
neighbors = np.arange(1, 10)
train_accuracy, test_accuracy = list(), list()

for iterator, kterator in enumerate(neighbors):
```

```
    knn = KNeighborsClassifier(n_neighbors=kterator)
    knn.fit(x_train, y_train.values.ravel())
    train_accuracy.append(knn.score(x_train, y_train))
    test_accuracy.append(knn.score(x_test, y_test))

plt.figure(figsize=[20, 8])
plt.plot(neighbors, test_accuracy, label="Testing Accuracy")
plt.plot(neighbors, train_accuracy, label="Training Accuracy")
plt.legend()
plt.title("Value VS Accuracy")
plt.xlabel("Number of Neighbors")
plt.ylabel("Accuracy")
plt.xticks(neighbors)
plt.savefig("knn_accuracies.png")
plt.show()

print("Best Accuracy is {} with K={}".format(np.max(test_accuracy), 1 +␣
↪test_accuracy.index(np.max(test_accuracy))))
```




```
Best Accuracy is 0.8833266666666667 with K=3
```

0.11 Conclusion

- Decision Tree and KNN classification both worked almost with equal accuracy.
- Accuracy of around 88% is achieved.
- For proper classification geo information is important.