Capstone Project-Car accident severity

Introduction/Business Understanding

In this project we are going to predict the severity of an accident given the current weather, road and visibility conditions.Our predictor or target variable will be 'SEVERITYCODE' because it is used measure the severity of an accident from 0 to 5 within the dataset. Attributes used to weigh the severity of an accident are 'WEATHER', 'ROADCOND' and 'LIGHTCOND' .

DATA

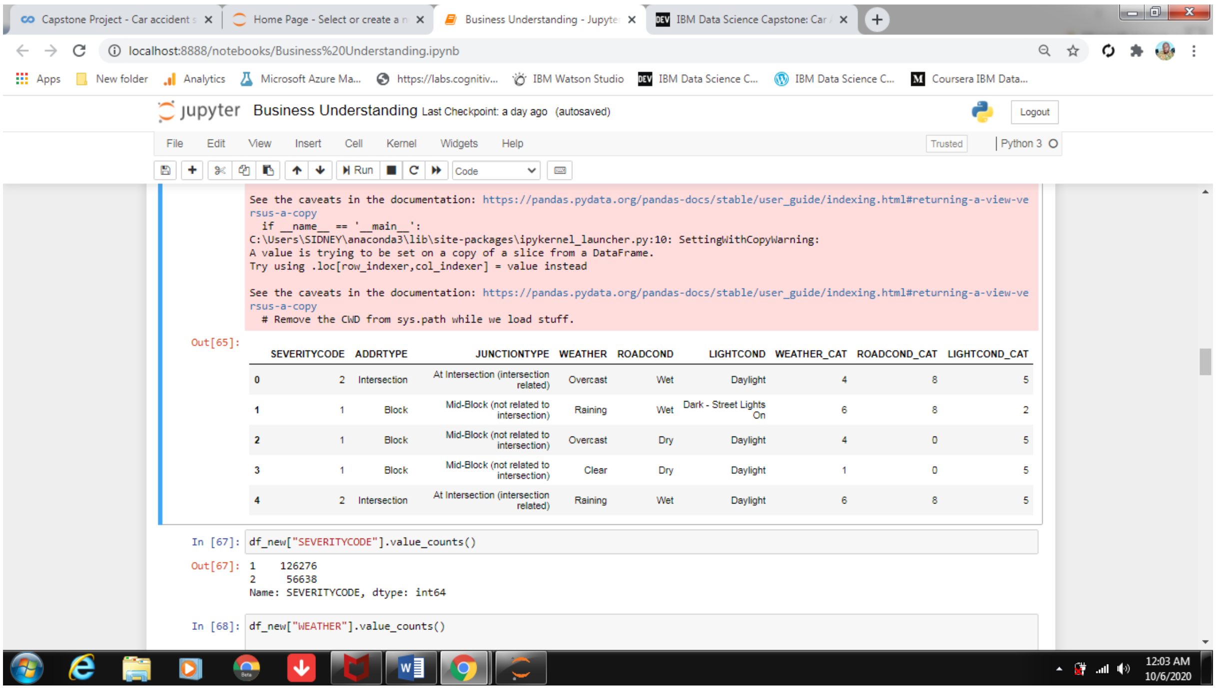
The predictor variable will be 'SEVERITYCODE' because it is used measure the severity of an accident within the dataset. Attributes used to weigh the severity of an accident are 'WEATHER', 'ROADCOND' and 'LIGHTCOND'.The key features used for analysis are
'SEVERITYVODE','ADDRTYPE', JUNCTIONTYPE','WEATHER', 'ROADCOND','LIGHTCOND'
'PERSONCOUNT','PEDCOUNT' 'PEDCYLCOUNT','VEHCOUNT'

Extract Dataset & Convert

In it's original form, this data is not fit for analysis. For one, there are many columns that we will not use for this model. Also, most of the features are of type object, when they should be numerical type.

We must use label encoding to covert the features to our desired data type.

```
# Label Encoding

# Convert column to category

df_new["WEATHER"] = df_new["WEATHER"].astype('category')

df_new["ROADCOND"] = df_new["ROADCOND"].astype('category')

df_new["LIGHTCOND"] = df_new["LIGHTCOND"].astype('category')


# Assign variable to new column for analysis

df_new["WEATHER_CAT"] = df_new["WEATHER"].cat.codes

df_new["ROADCOND_CAT"] = df_new["ROADCOND"].cat.codes

df_new["LIGHTCOND_CAT"] = df_new["LIGHTCOND"].cat.codes


df_new.head(5)
```



Balancing the Dataset

Our target variable SEVERITYCODE is only 42% balanced. In fact, severitycode in class 1 is nearly three times the size of class 2.

We can fix this by downsampling the majority class.

```
2    58188
1    58188
Name: SEVERITYCODE, dtype: int64
```

Perfectly balanced.

Methodology

Our data is now ready to be fed into machine learning models.

We will use the following models:

K-Nearest Neighbor (KNN)

KNN will help us predict the severity code of an outcome by finding the most similar to data point within k distance.

Decision Tree

A decision tree model gives us a layout of all possible outcomes so we can fully analyze the concequences of a decision. It context, the decision tree observes all possible outcomes of different weather conditions.

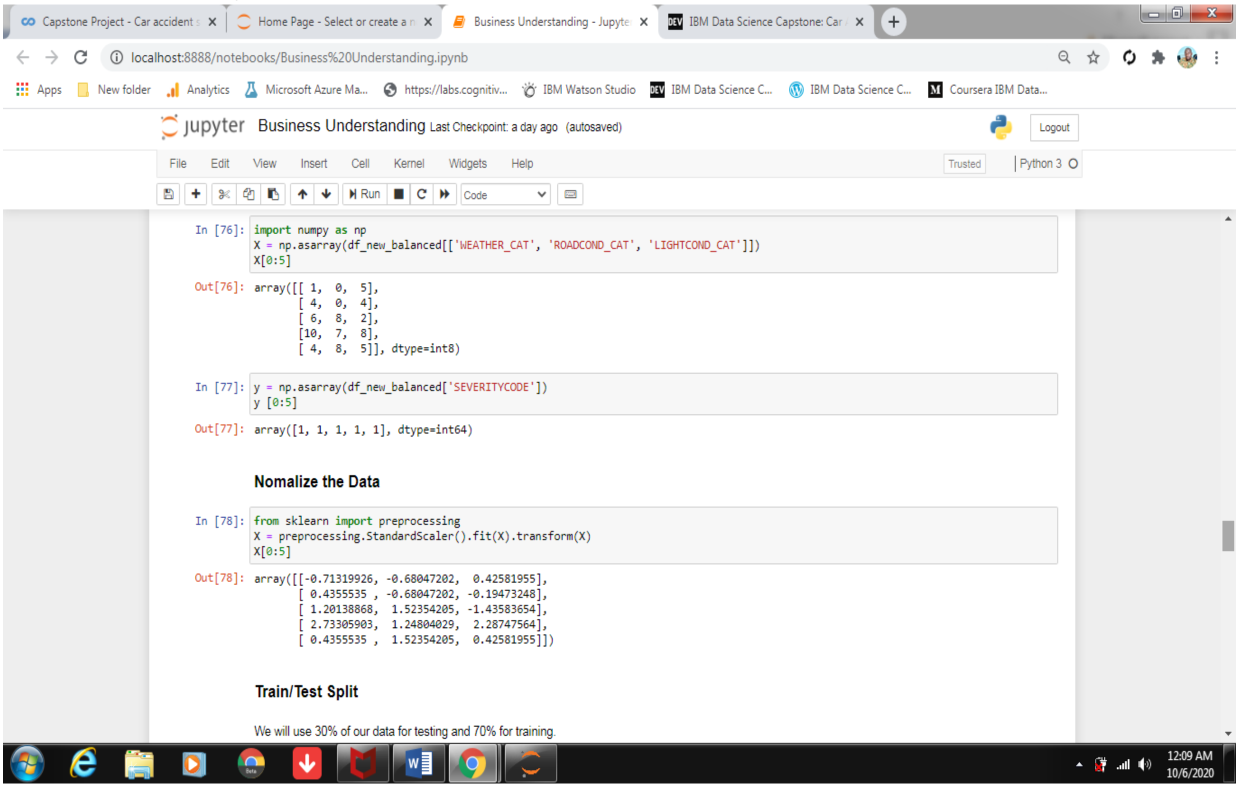
Logistic Regression

Because our dataset only provides us with two severity code outcomes, our model will only predict one of those two classes. This makes our data binary, which is perfect to use with logistic regression.

Let's get started!

Initialization

Define X and y



Train/Test Split

We will use 30% of our data for testing and 70% for training.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4)
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
Train set: (80378, 3) (80378,)
Test set: (34448, 3) (34448,)
```

K-Nearest Neighbors (KNN)

```
# Building the KNN Model
from sklearn.neighbors import KNeighborsClassifier

k = 25
#Train Model & Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh

Kyhat = neigh.predict(X_test)
Kyhat[0:5]
array([2, 1, 2, 2, 2], dtype=int64)
```

Decision Tree

```
#Building the Decision Tree
from sklearn.tree import DecisionTreeClassifier
df_newTree = DecisionTreeClassifier(criterion="entropy", max_depth = 7)
df_newTree
df_newTree.fit(X_train,y_train)
# Train Model & Predict
DTyhat = df_newTree.predict(X_test)
```

Logistic Regration

```
# Building the LR Model
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=6, solver='liblinear').fit(X_train,y_train)
LR
```

Out[91]:

```
LogisticRegression(C=6, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbo
se=0,
```

```
                        warm_start=False)
# Train Model & Predicr
LRyhat = LR.predict(X_test)
LRyhat
array([1, 1, 2, ..., 1, 2, 2], dtype=int64)
yhat_prob = LR.predict_proba(X_test)
yhat_prob
array([[0.54940819, 0.45059181],
       [0.52458009, 0.47541991],
       [0.45649265, 0.54350735],
       ...,
       [0.50632209, 0.49367791],
       [0.48136103, 0.51863897],
       [0.45649265, 0.54350735]])
```

Results & Evaluation

Now we will check the accuracy of our models.

```
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
```

K-Nearest Neighbor

In [95]:

```
# Jaccard Similarity Score
jaccard_similarity_score(y_test, Kyhat)

C:\Users\SIDNEY\anaconda3\lib\site-packages\sklearn\metrics\_classification
.py:664: FutureWarning: jaccard_similarity_score has been deprecated and re
placed with jaccard_score. It will be removed in version 0.23. This impleme
ntation has surprising behavior for binary and multiclass classification ta
sks.
  FutureWarning)
0.5328611240130051
# F1-SCORE
f1_score(y_test, Kyhat, average='macro')

0.4921075848964933
```

Model is most accurate when k is 25.

Decision Tree

```
# Jaccard Similarity Score
jaccard_similarity_score(y_test, DTyhat)
0.5465048769159313
# F1-SCORE
f1_score(y_test, DTyhat, average='macro')

0.5203507728191309
```

Model is most accurate with a max depth of 7.

Logistic Regression

```python
# Jaccard Similarity Score
jaccard_similarity_score(y_test, LRyhat)
```

```
0.5215977705527172
```

```python
# F1-SCORE
f1_score(y_test, LRyhat, average='macro')
```

```
0.5097825024463251
```

```python
# LOGLOSS
yhat_prob = LR.predict_proba(X_test)
log_loss(y_test, yhat_prob)
```

```
0.6868866924905638
```

Model is most accurate when hyperparameter C is 6.

Discussion

In the beginning of this notebook, we had categorical data that was of type 'object'. This is not a data type that we could have fed through an algorithm, so label encoding was used to created new classes that were of type int8; a numerical data type.

After solving that issue we were presented with another - imbalanced data. As mentioned earlier, class 1 was nearly three times larger than class 2. The solution to this was downsampling the majority class with sklearn's resample tool. We downsampled to match the minority class exactly with 58188 values each.

Once we analyzed and cleaned the data, it was then fed through three ML models; K-Nearest Neighbor, Decision Tree and Logistic Regression. Although the first two are ideal for this project, logistic regression made the most sense because of its binary nature.

Evaluation metrics used to test the accuracy of our models were jaccard index, f-1 score and logloss for logistic regression. Choosing different k, max depth and hyperamater C values helped to improve our accuracy to be the best possible.

Conclusion

Based on historical data from weather conditions pointing to certain classes, we can conclude that particular weather conditions have a somewhat impact on whether or not travel could result in property damage (class 1) or injury (class 2).

Type *Markdown* and LaTeX: α^2