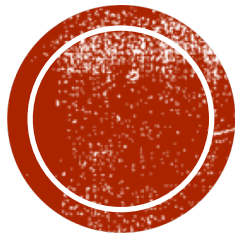
IBM APPLIED DATA SCIENCE CAPSTONE

Ivy

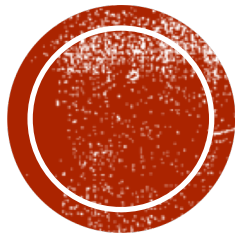
09/02/2020



INTRODUCTION

A description of the problem and a discussion of the background

- Suppose you are driving in a terrible weather condition. On the way, you come across a car accident on the other side of the highway, they must be in critical condition for all of this to be happening since police cars start appearing and there's a helicopter transporting the ones involved in the crash. You don't want those horrible things to happen to you so if there's a way that can warn you ahead then you would drive more carefully or even change your driving plan if you're able to.
- So the problem becomes: given the weather, the road conditions and other features about the possibility of you getting into an accident, to predict the severity of an accident.



DATASET

A description of the data and how it will be used to solve the problem

- I will use dataset of collisions of Seattle for this problem, it includes all types of collisions. Timeframe: 2004 to Present.
- In total, there are 37 attributes (columns) in this dataset, some have missing data. The first column **SEVERITYCODE** is the labeled data, which describes the fatality of an accident. The remaining columns have different types of attributes, such as **ADDRTYPE** stands for Collision address type of Alley, Blok, or intersection; **PERSONCOUNT** means the total number of people involved in the collision, etc., which can be used to train the model. Since it has unbalanced labels, should balance the data first to avoid a biased ML model.

```
df = pd.read_csv('https://s3.us.cloud-object-storage.appdomain.cloud/cf-courses-data/CognitiveClass/DP0701EN/version-2/Data-Collisions.csv')
df.head()
```

```
/opt/conda/envs/Python36/lib/python3.6/site-packages/IPython/core/interactiveshell.py:3020: DtypeWarning: Columns (33) have mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

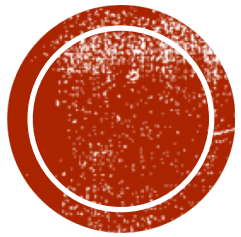
3]:

	SEVERITYCODE	X	Y	OBJECTID	INCKEY	COLDETKEY	REPORTNO	STATUS	ADDRTYPE	INTKEY	...	ROADCOND	LIGHTCOND	PEDROWNOTGRNT	SDOTCOLNUM	SPEEDING	ST_COLCODE	ST_COLDESC	SEGLANEKEY	CROSSWALKKEY	HITPARKEDCAR
0	2	-122.323148	47.703140	1	1307	1307	3502005	Matched	Intersection	37475.0	...	Wet	Daylight	NaN	NaN	NaN	10	Entering at angle	0	0	N
1	1	-122.347294	47.647172	2	52200	52200	2607959	Matched	Block	NaN	...	Wet	Dark - Street Lights On	NaN	6354039.0	NaN	11	From same direction - both going straight - bo...	0	0	N
2	1	-122.334540	47.607871	3	26700	26700	1482393	Matched	Block	NaN	...	Dry	Daylight	NaN	4323031.0	NaN	32	One parked--one moving	0	0	N
3	1	-122.334803	47.604803	4	1144	1144	3503937	Matched	Block	NaN	...	Dry	Daylight	NaN	NaN	NaN	23	From same direction - all others	0	0	N
4	2	-122.306426	47.545739	5	17700	17700	1807429	Matched	Intersection	34387.0	...	Wet	Daylight	NaN	4028032.0	NaN	10	Entering at angle	0	0	N

5 rows × 38 columns

```
df.shape
```

4]: (194673, 38)



METHODOLOGY

Discuss and describe exploratory data analysis, inferential statistical testing, and what machine learnings were used and why

FEATURE SELECTION

- **Training features:**
  - **WEATHER**: A description of the weather conditions during the time of the collision.
  - **LIGHTCOND**: light condition - The light conditions during the collision.
  - **ROADCOND**: road condition - The condition of the road during the collision.
  - **JUNCTIONTYPE**: Category of junction at which collision took place.
  - **PERSONCOUNT**: The total number of people involved in the collision.
  - **VEHCOUNT**: vehicle count - The number of vehicles involved in the collision.
- **Label data:**
  - **SEVERITYCODE**: A code that corresponds to the severity of the collision:
    - 3—fatality
    - 2b—serious injury
    - 2—injury
    - 1—prop damage
    - 0—unknown

CONTINUED

```
# features that can be used for ML model (including labels)
features = ['WEATHER','LIGHTCOND','ROADCOND','JUNCTIONTYPE','PERSONCOUNT','VEHCOUNT','SEVERITYCODE']
df_collision = df[features]
df_collision.head()
```

6]:

	WEATHER	LIGHTCOND	ROADCOND	JUNCTIONTYPE	PERSONCOUNT	VEHCOUNT	SEVERITYCODE
0	Overcast	Daylight	Wet	At Intersection (intersection related)	2	2	2
1	Raining	Dark - Street Lights On	Wet	Mid-Block (not related to intersection)	2	2	1
2	Overcast	Daylight	Dry	Mid-Block (not related to intersection)	4	3	1
3	Clear	Daylight	Dry	Mid-Block (not related to intersection)	3	3	1
4	Raining	Daylight	Wet	At Intersection (intersection related)	2	2	2

DATA PREPROCESSING

- **Weather**

Convert categorical variables to numerical variables and combine similar weathers into one

- Clear : 1
- Raining, Snowing, Sleet/Hail/Freezing Rain : 2
- Overcast, Partly Cloudy, Fog/Smog/Smoke : 3
- Blowing Sand/Dirt, Severe Crosswind : 4
- Unknown, Other: 0

```
1    109163
2     33672
3     27771
0     12516
4        74
Name: WEATHER, dtype: int64
```

DATA PREPROCESSING

- **Light Condition**

Convert categorical variables to numerical variables and combine similar light condition into one

- Daylight : 1
- Dark - Street Lights On, Dark - No Street Lights, Dark - Street Lights Off, Dark - Unknown Lighting : 2
- Dusk : 3
- Dawn : 4
- Unknown, Other : 0

```
1    113972
2     50225
0     10764
3      5781
4      2454
Name: LIGHTCOND, dtype: int64
```

DATA PREPROCESSING

- **Junction Condition**

Convert categorical variables to numerical variables

- Mid-Block (not related to intersection) : 1
- Mid-Block (but intersection related) : 2
- At Intersection (intersection related) : 3
- At Intersection (but not related to intersection) : 4
- Driveway Junction : 5
- Ramp Junction :6
- Unknown : 0

```
1     86856
3     61241
2     22353
5     10520
4      2057
6       162
0         7
Name: JUNCTIONTYPE, dtype: int64
```

DATA PREPROCESSING

- **Balancing Data**

The data set is unbalanced - too many 'good condition' collisions, like Weather is clear. We should get rid of some of them to make the data set contain around half good and half bad condition.

The simplest method is to randomly drop half data with weather condition = clear (code:1) or road condition = dry (code:1)

2]:

	WEATHER	LIGHTCOND	ROADCOND	JUNCTIONTYPE	PERSONCOUNT	VEHCOUNT	SEVERITYCODE
0	3	1	2	3	2	2	2
1	2	2	2	1	2	2	1
2	3	1	1	1	4	3	1
3	1	1	1	1	3	3	1
4	2	1	2	3	2	2	2

```
df_collision.shape
```

3]: (119656, 7)

DATA PREPROCESSING

- Define X, and y
- X[0:5]:

```
: array([[3, 1, 2, 3, 2, 2],
         [2, 2, 2, 1, 2, 2],
         [3, 1, 1, 1, 4, 3],
         [1, 1, 1, 1, 3, 3],
         [2, 1, 2, 3, 2, 2]])
```

- y[0:5]

```
: array([2, 1, 1, 1, 2])
```

DATA PREPROCESSING

- **Normalize the dataset**
- **X[0:5]:**

```
array([[ 1.68825901, -0.43454235,  1.10892205,  0.82365956, -0.34246566,
         0.05632737],
       [ 0.55710619,  0.93513258,  1.10892205, -0.88838936, -0.34246566,
         0.05632737],
       [ 1.68825901, -0.43454235, -0.46768917, -0.88838936,  1.14905617,
         1.83937161],
       [-0.57404663, -0.43454235, -0.46768917, -0.88838936,  0.40329526,
         1.83937161],
       [ 0.55710619, -0.43454235,  1.10892205,  0.82365956, -0.34246566,
         0.05632737]])
```

TRAIN/TEST DATASET

- Split dataset into train and test set 70% ~ 30%

```
Train set: (83759, 6) (83759,)
Test set: (35897, 6) (35897,)
```

MODELING (LOGISTIC REGRESSION WITH SCIKIT-LEARN)

- It is a classification problem, so i decided to use Logistic Regression as the model.
- This function implements logistic regression and can use different numerical optimizers to find parameters, including ‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’ solvers.

- The version of Logistic Regression in Scikit-learn, support regularization. Regularization is a technique used to solve the overfitting problem in machine learning models. C parameter indicates inverse of regularization strength which must be a positive float. Smaller values specify stronger regularization.

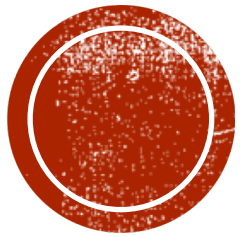
CONTINUED

- Trained 3 models using solvers of 'liblinear', 'sag' , and 'lbfgs' respectively

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='liblinear',
          tol=0.0001, verbose=0, warm_start=False)
```

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='sag',
          tol=0.0001, verbose=0, warm_start=False)
```

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='lbfgs',
          tol=0.0001, verbose=0, warm_start=False)
```



PREDICTION & RESULTS

Discuss results

PREDICTION

- Predict of three models using test set

```
yhat1 = LR1.predict(X_test)
yhat1
```

```
array([1, 1, 1, ..., 1, 1, 1])
```

```
yhat2 = LR2.predict(X_test)
yhat2
```

```
array([1, 1, 1, ..., 1, 1, 1])
```

```
yhat3 = LR3.predict(X_test)
yhat3
```

```
array([1, 1, 1, ..., 1, 1, 1])
```

- **predict_proba** returns estimates for all classes, ordered by the label of classes. So, the first column is the probability of class 1, P(Y=1|X), and second column is probability of class 2, P(Y=2|X):

```
yhat_prob1 = LR1.predict_proba(X_test)
yhat_prob1
```

```
array([[0.57988457, 0.42011543],
       [0.72205162, 0.27794838],
       [0.81121264, 0.18878736],
       ...,
       [0.72205162, 0.27794838],
       [0.62394179, 0.37605821],
       [0.63607072, 0.36392928]])
```

```
yhat_prob2 = LR2.predict_proba(X_test)
yhat_prob2
```

```
array([[0.58100226, 0.41899774],
       [0.72319889, 0.27680111],
       [0.81223843, 0.18776157],
       ...,
       [0.72319889, 0.27680111],
       [0.62499534, 0.37500466],
       [0.6372225 , 0.3627775 ]])
```

```
yhat_prob3 = LR3.predict_proba(X_test)
yhat_prob3
```

```
array([[0.5810072 , 0.4189928 ],
       [0.72319963, 0.27680037],
       [0.81224092, 0.18775908],
       ...,
       [0.72319963, 0.27680037],
       [0.6250018 , 0.3749982 ],
       [0.63722916, 0.36277084]])
```

EVALUATION

- **Jaccard Index**

We can define jaccard as the size of the intersection divided by the size of the union of two label sets. If the entire set of predicted labels for a sample strictly match with the true set of labels, then the subset accuracy is 1.0; otherwise it is 0.0.

```
jaccard_similarity_score(y_test, yhat1)
```

```
39]: 0.707496448171563
```

```
|: jaccard_similarity_score(y_test, yhat2)
```

```
40]: 0.7074407332089032
```

```
|: jaccard_similarity_score(y_test, yhat3)
```

```
41]: 0.7074407332089032
```

EVALUATION

- **Confusion Matrix**

Based on the count of each section, we can calculate precision and recall of each label:

Precision is a measure of the accuracy provided that a class label has been predicted. It is defined by: precision = TP / (TP + FP)

Recall is true positive rate. It is defined as: Recall = TP / (TP + FN)

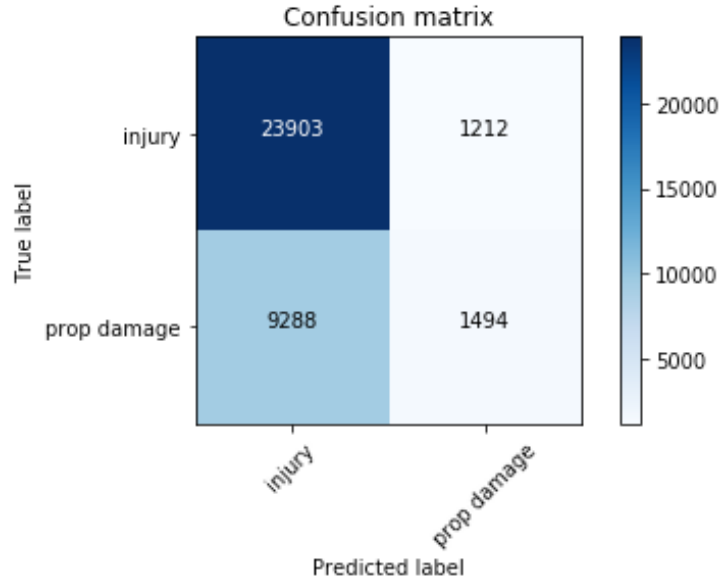
So, we can calculate precision and recall of each class.

F1 score: Now we are in the position to calculate the F1 scores for each label based on the precision and recall of that label.

The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0. It is a good way to show that a classifer has a good value for both recall and precision.

And finally, we can tell the average accuracy for this classifier is the average of the F1-score for both labels, which is 0.64 in this case (all three solvers have the same F1-score).

```
Confusion matrix, without normalization
[[23903  1212]
 [ 9288  1494]]
```




```
print (classification_report(y_test, yhat1))
```

	precision	recall	f1-score	support
1	0.72	0.95	0.82	25115
2	0.55	0.14	0.22	10782
micro avg	0.71	0.71	0.71	35897
macro avg	0.64	0.55	0.52	35897
weighted avg	0.67	0.71	0.64	35897

EVALUATION

- **Log Loss**

Now, will try **log loss** for evaluation. In logistic regression, the output can be the probability of injury (or equals to 1). This probability is a value between 0 and 1. Log loss(Logarithmic loss) measures the performance of a classifier where the predicted output is a probability value between 0 and 1.

```
:  from sklearn.metrics import log_loss
   log_loss(y_test, yhat_prob1)

7]:  0.5776349992894849

:  log_loss(y_test, yhat_prob2)

8]:  0.5776252908717573

:  log_loss(y_test, yhat_prob3)

9]:  0.5776252595502196
```

DISCUSSION

discuss any observations noted and any recommendations can make based on the results.

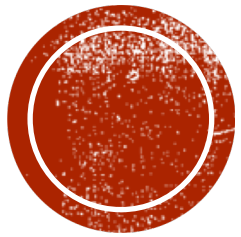
- As you can see from the above evaluation of the model, i used three different numerical optimizers to find parameters: ‘liblinear’, ‘sag’ and 'lbfgs'. The results show that all three have a same f1-score while a slightly different jaccard similarity and log loss. That means for each y_test data, it is considered as the same category by these three solvers, though with a different probabilities.

  While with a F1-score of 0.64 means this model is not a 'perfect' one, there should be some ways to imporve the accuracy. For example, if i use 80% as training data and 20% as testing, i assume it would have a higher score; and for feature selection and data preprocessing, if i don't combine some similar condition to one group, perhaps the results would be different. But maybe worse?

  But one thing is for sure, this data set is not a pretty good one for trainning model since for the labeled data (SEVERITYCODE), it only has 2 categories, which means it's unable to predict whether there could be a more sever collisions like fatality or serious injury, for a sever condition (e.g. weather is raining and road condition is wet, etc.), since the model failed to train for that.

RECOMMENDATION

- Though there's insufficient proof that sever weather and road condition will cause a terrible collision, most predicitons are 1s, it still means somehow an accident may occur and people will get injured. So it's always good to be careful when driving during bad conditions.



CONCLUSION

Conclude the report

- For this capstone, I'm using the shared dataset of collisions of Seattle. I selected **WEATHER**, **LIGHTCOND**, **ROADCOND**, **JUNCTIONTYPE**, **PERSONCOUNT** and **VEHCOUNT** as the training features and **SEVERITYCODE** as the labeled data. For data preprocessing, I removed the ~5000 rows of missing data and only kept the rows with all data available; then I converted the string values of some features to numerical ones, by first combing similar conditions to one group, such as wet and snow of road condition. I split 70% of the data to training set and the rest 20% as testing set. For modeling, i used Logistic Regression as the training method, applied three solvers to it and got an F1-score of 0.64, which seems not as good as expected. For improvements, I'll try to use 80% for training and see the results. Or use decision tree instead. But the dataset itself does have drawbacks, since the data is unbalanced and only contains 2 category of labeled data.