CSE:587 Data Intensive Computing Phase II Report

Raakhal Rapolu (raakhalr) | Kajol (kajol)
Department of Computer Science and Engineering
University at Buffalo, Buffalo, NY 14260
raakhalr@buffalo.edu | kajol@buffalo.edu

1 US Accidents Severity Prediction (2016-2021)

Inferring the insights from Phase-1 pre-processing and exploratory data analysis, we have detailed the distribution and insights gathered for every feature by performing univariate and multivariate analysis. In this phase of the project, we will be using the processed data from phase 1 and with the processed data we will be doing the needful tweaking in the data and performing various Machine learning models, and predicting the severity.

As described in the US accidents dataset from 2016-2021, where it has 46 columns or feature variables for 28,45,342 rows. Let's derive a new feature variable time duration, which tells how much time is the traffic impacted due to the accident, Maximum time to clear an accident is 12424.0 minutes or 207 hours or 9 days, Minimum to clear an accident time difference is 2.0 minutes.

Understanding the data size and computational cost associated with running the data as a whole, we try to select the most important features from the correlation matrix and various other factors, now we scale down the features and select those that are helpful in predicting our target variable severity. The selected features are shown in Figure 1. Here we can see that all our data features are in the float64 data type.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2829624 entries, 0 to 2845341
Data columns (total 14 columns):
 #   Column              Dtype
---  ------              -----
 0   Severity            float64
 1   Traffic_Signal      float64
 2   Hour                float64
 3   Crossing            float64
 4   Temperature(F)      float64
 5   Wind_Chill(F)       float64
 6   Wind_Speed(mph)     float64
 7   Junction            float64
 8   Humidity(%)         float64
 9   Pressure(in)        float64
 10  Visibility(mi)      float64
 11  Weekday             object
 12  Year                float64
 13  Time_Duration(min)  float64
dtypes: float64(13), object(1)
memory usage: 323.8+ MB
```

Figure 1: Feature variables - data types

The population is a complete set of traces/data points. The sample is a subset of the complete set (or population) how we select the sample introduces biases into the data. Considering the limited availability of computational resources we will be taking the sample from our whole dataset, we sample the data year-wise[12]. We get six samples from the population, data for 2016, 2017, 2018, 2019, 2020, and 2021, showing the distribution in the below Figure 2. In order to maintain the variance in the data among the variables, we do the feature scaling using scikit-learn MinMaxScaler, StandardScaler [9].

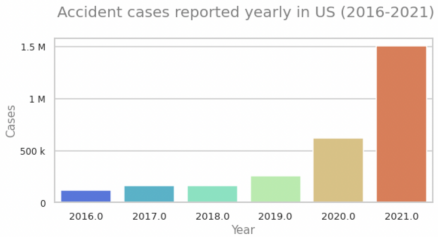


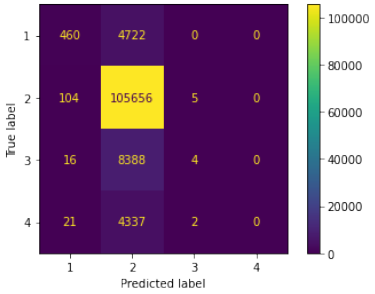
Figure 2: Accidents by Year

1.1 Logistic Regression

Logistic regression is the best-suited technique to model the dependent variable's variation given a set of independent variables. Here we will be performing Logistic regression on our dataset, predicting the target variable is severity, which can be classified into 4 severity indexes. Firstly we perform the training on the sample data collected for the year **2020**. The data has 6,18,572 entries and 14 columns.

We perform a train and test data split using a split ratio of 0.2 and a random state of 21. Logistic regression when ran on the above parameters it does not converge,[10] as the data is very huge and the inbuilt maximum iteration (**max_iter)** parameter is set to **100**, so we increase the number of iterations in the iterations to **10000**.

[Logistic regression algorithm for the year 2016] accuracy_score: 0.717.
[Logistic regression algorithm for the year 2017] accuracy_score: 0.684.
[Logistic regression algorithm for the year 2018] accuracy_score: 0.665.
[Logistic regression algorithm for the year 2019] accuracy_score: 0.793.
[Logistic regression algorithm for the year 2020] accuracy_score: 0.858.
[Logistic regression algorithm for the year 2021] accuracy_score: 0.984.




```
Severity
1.0     25911
2.0    528823
3.0     42040
4.0     21798
dtype: int64
```

Figure 3: Left: Confusion Matrix year 2020 | Right: Severity distribution values

The testing accuracy seems to be good, but when evaluating the confusion matrix as shown in left Figure 3, It is observed that for the class 4 severity index there are no predictions and it can also be seen that the model is overfitting on one class that is on severity index 2. Understanding the distribution of the data it is seen that the distribution is unbalanced, in order to make the distribution balance we need to balance the data distribution, so here we use the technique called Synthetic Minority Over-sampling Technique (SMOTE)[11].

Using SMOTE, we now up-sample the data sample based on the severity index feature, we up sample the severity index 1 by 1200%, severity index 3 by 925%, and severity index 4 by 1000% using the imblearn[3] package by scikit-learn. and then again we do the train test split, and train the upsampled dataset on the logistic regression model.

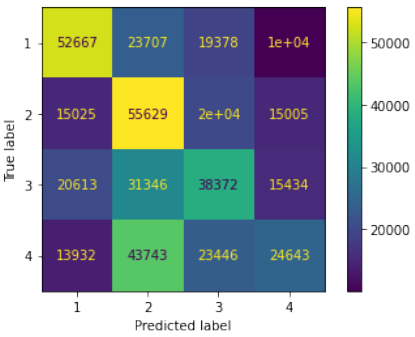


Figure 4: Confusion matrix - 2020

Resampled dataset shape Counter(2.0: 528823, 4.0: 220423, 1.0: 325863, 3.0: 389623). Now after performing the model fit on the resampled data, the testing accuracy is 66.5% and the model is now learning all the features as the data is oversampled and the model gets to learn more features. From figure 4 we can see that there is certain learning for the logistic regression model for each of the class distributions. Fitting the model on the other sample data with the resultant SMOTE upsampled dataset, we achieve accuracies of 45% for the year 2016, 51.8 %for the year 2017, 48.9% for the year 2018, 56.9% for the year 2019, 82% for the year 2021 respectively.

1.2 K Nearest Neighbors

The k-nearest neighbors algorithm (k-NN) is a non-parametric supervised learning method used for classification and regression. In both cases, the input consists of the k closest training examples in a data set. The output depends on whether k-NN is used for classification or regression.[2]. Here in our case, we have seen the up-sampling done on the above dataset samples year-wise, using scikit-learn's KNeighborsClassifier[7] package we will be training our KNN classifier upon the upsampled data.

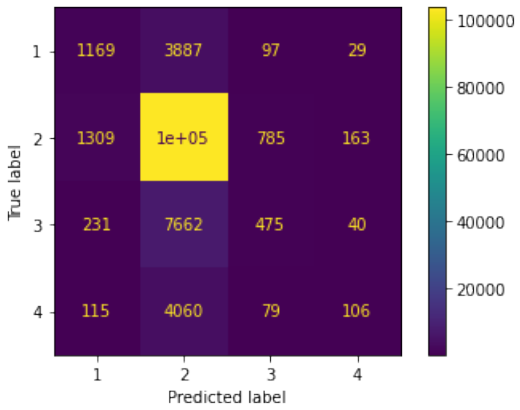


Figure 5: Confusion matrix

After performing the KNN classifier with n - value 7 for the initial step, we have got a testing accuracy of 85.6%, it is also observed from the above figure 5 there is overfitting on severity index 2 and thus we need to oversample and nominally balance the data as done in LR. when over-sampling is done on the data, and KNN accuracy trained on the over-sampled dataset is 52%. It is also observed that the learning is much better after balancing the dataset distribution which has improved the per-class accuracy.

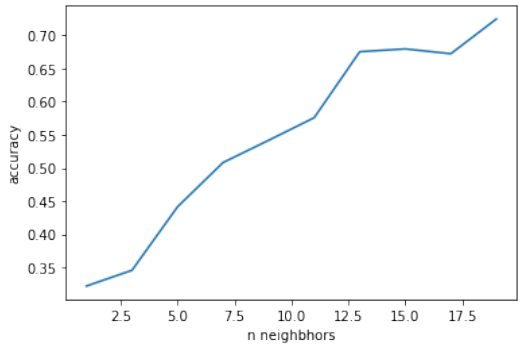


Figure 6: Plot for n value Vs the accuracy for the year 2020 data sample

We know that the n-value in KNN is one of the main factors to be considered, in order to find the best fit n value we try running on a series of n values and see how the accuracy is performing with the change in n value. From figure 6 it is observed that with the increase in n value that is the number of neighbors in the KNN model we can see from the plot that the testing accuracy of the model is performing better, and thus the same is performed on other data samples of other years.

From the plot, we can see the improved results. We have achieved an accuracy of 0.71 at K=19. As we already derived the error plot and got the minimum error at k=19, so we will get better efficiency at that K value.

1.3 Decision Trees

Decision trees are one of the most intuitive and interpretable tools for data classification, Decision trees partition the input space hierarchically till a certain subspace is assigned to a class label [13]. These are easy to use, and implement and are highly interpretable.

We perform a decision tree using the scikit learn DecisionTreeClassifier package[8], to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “log_loss” and “entropy” both for the Shannon information gain, we use random split as our splitting strategy with a maximum depth of 8.

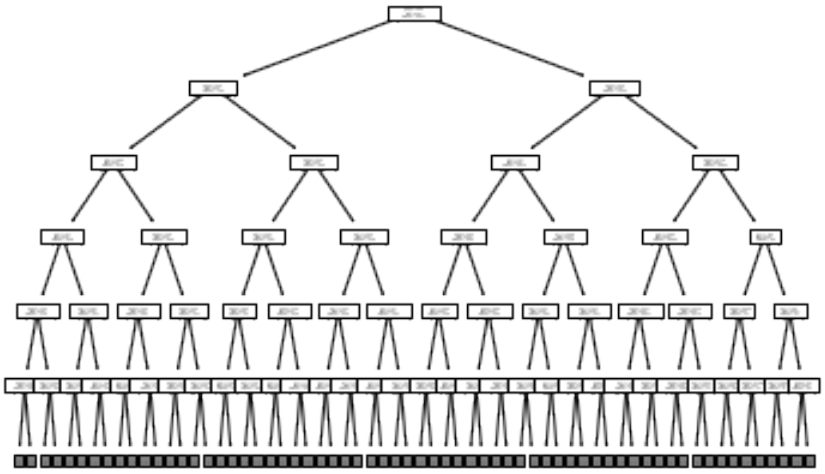


Figure 7: tree plot for our sample with depth 5

From the above figure 7 we can visualize the Gini impurity values and based on what factors the decision is made at every node. Fitting the decision tree classifier on our data sample using entropy as our criteria we have achieved an accuracy_score: 0.770 and with Gini criteria, we have got an accuracy_score: 0.871.

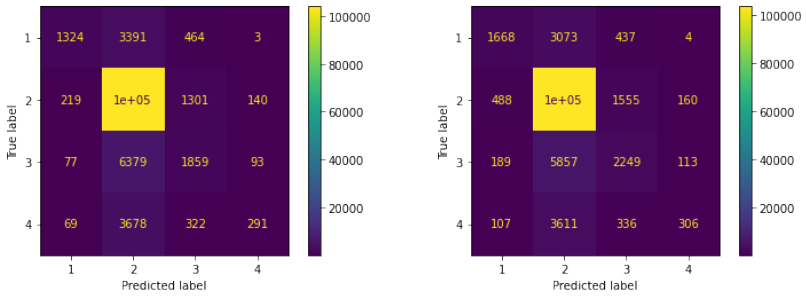


Figure 8: Left: Confusion Matrix year 2020 | Right: Severity distribution values

Figure 8 we can see that the left matrix is for entropy and the right one is for the Gini index, the values show that the model fits well on our data and gives balanced results and per class accuracy is also learned well, we have also performed the same for the other data samples, for the year 2021 we have got testing accuracy of 0.85. Overall the decision tree classifier is performing better when compared with Logistic regression and KNN models.

1.4 Random Forest Classifier

A random forest is a meta-estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree[6].

The higher, the more important the feature. The importance of a feature is computed as the (normalized) total reduction of the criterion brought by that feature. It is also known as the Gini importance.

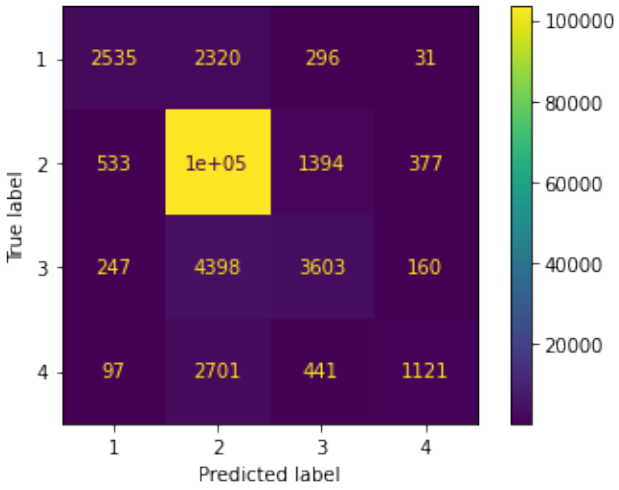


Figure 9: Confusion matrix

We build the Random forest algorithm with 100 estimators and we are able to achieve an accuracy_score: 0.825, Confusion matrix from figure 9 shows the metrics on which the model was evaluated, training was performed on the over-sampled dataset for the year 2020 data. Comparatively, the Random forest classifier has performed best when compared to decision trees, and also we can see that there is a better fit for a balanced dataset than that of the unbalanced data sample.

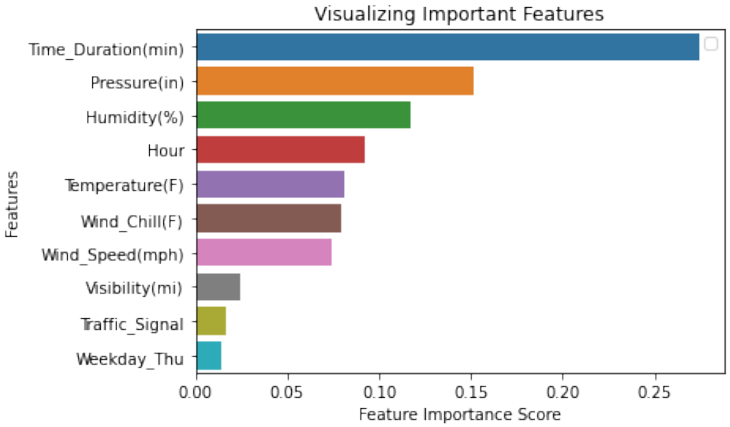


Figure 10: Feature importance plot

Random Forests are often used for feature selection, Feature importance's provided by the fitted attribute feature_=_importances_and they are computed as the mean and standard deviation of accumulation of the impurity decrease within each tree[5]. From the figure 10 we are able to visualize that the Time duration feature is shown as a more important feature in predicting the severity index using the Random Forest classifier.

1.5 Artificial Neural Networks

Neural networks learn features by training the network with known examples that have labeled data with input and target output forming probability-weighted associations between the two, which are stored within the data structure of the net itself. The training of a neural network from a given example is usually conducted by determining the difference between the processed output of the network (often a prediction) and the target output. This difference is the error. The network then adjusts its weighted associations according to a learning rule and uses this error value. Successive adjustments will cause the neural network to produce output that is increasingly similar to the target output. After a sufficient number of these adjustments, the training can be terminated based on certain criteria. This is known as supervised learning [1].

Using the TensorFlow package and Keras package to build the network. We have built a sequential neural network with an input layer with nodes of input shape, five dense layers with nodes 256,128,64,32,32 and the output layer with 4 nods, intermediate dense layers we use 'relu' as our activation function, and in the output layer, we use softmax as our activation function since this is a classification task[4]. We trained the model for 100 epochs with a batch size of 20, with a validation split of 0.2, and monitor the performance of the training using validation loss.

```
Epoch 1/100
19795/19795 [==============================] - 67s 3ms/step - loss: 0.4509 - accuracy: 0.8564 - val_loss: 0.4362 - val_accuracy: 0.8595 - lr: 0.0010
Epoch 2/100
19795/19795 [==============================] - 67s 3ms/step - loss: 0.4305 - accuracy: 0.8593 - val_loss: 0.4245 - val_accuracy: 0.8606 - lr: 0.0010
Epoch 3/100
19795/19795 [==============================] - 67s 3ms/step - loss: 0.4257 - accuracy: 0.8602 - val_loss: 0.4259 - val_accuracy: 0.8600 - lr: 0.0010
Epoch 4/100
19795/19795 [==============================] - 69s 3ms/step - loss: 0.4218 - accuracy: 0.8606 - val_loss: 0.4263 - val_accuracy: 0.8609 - lr: 0.0010
Epoch 5/100
19795/19795 [==============================] - 70s 4ms/step - loss: 0.4198 - accuracy: 0.8607 - val_loss: 0.4182 - val_accuracy: 0.8608 - lr: 0.0010
Epoch 6/100
19795/19795 [==============================] - 67s 3ms/step - loss: 0.4162 - accuracy: 0.8610 - val_loss: 0.4144 - val_accuracy: 0.8617 - lr: 0.0010
Epoch 7/100
19795/19795 [==============================] - 69s 3ms/step - loss: 0.4135 - accuracy: 0.8615 - val_loss: 0.4121 - val_accuracy: 0.8619 - lr: 0.0010
Epoch 8/100
19795/19795 [==============================] - 65s 3ms/step - loss: 0.4127 - accuracy: 0.8612 - val_loss: 0.4188 - val_accuracy: 0.8592 - lr: 0.0010
Epoch 9/100
19795/19795 [==============================] - 67s 3ms/step - loss: 0.4175 - accuracy: 0.8611 - val_loss: 0.4113 - val_accuracy: 0.8615 - lr: 0.0010
Epoch 10/100
19795/19795 [==============================] - 69s 3ms/step - loss: 0.4137 - accuracy: 0.8616 - val_loss: 0.4125 - val_accuracy: 0.8627 - lr: 0.0010
Epoch 11/100
19795/19795 [==============================] - 64s 3ms/step - loss: 0.4095 - accuracy: 0.8623 - val_loss: 0.4049 - val_accuracy: 0.8621 - lr: 0.0010
Epoch 12/100
19795/19795 [==============================] - 64s 3ms/step - loss: 0.4077 - accuracy: 0.8622 - val_loss: 0.4027 - val_accuracy: 0.8624 - lr: 0.0010
Epoch 13/100
19795/19795 [==============================] - 68s 3ms/step - loss: 0.4107 - accuracy: 0.8626 - val_loss: 0.4001 - val_accuracy: 0.8633 - lr: 0.0010
Epoch 14/100
19795/19795 [==============================] - 68s 3ms/step - loss: 0.4116 - accuracy: 0.8618 - val_loss: 0.4021 - val_accuracy: 0.8630 - lr: 0.0010
Epoch 15/100
19795/19795 [==============================] - 71s 4ms/step - loss: 0.4080 - accuracy: 0.8617 - val_loss: 0.4164 - val_accuracy: 0.8617 - lr: 0.0010
Epoch 16/100
19795/19795 [==============================] - 66s 3ms/step - loss: 0.4106 - accuracy: 0.8618 - val_loss: 0.4041 - val_accuracy: 0.8625 - lr: 0.0010
```

Figure 11: Training history showing training accuracy, epoch, validation loss, and learning rate

From Figure 11 we can see the training metrics, where we achieved 0.86 training accuracy and validation accuracy of 0.8625, Figure 12 shows us the plot visualizing the training and validation performance metrics. Neural networks performed better when compared with decision trees and other models performed earlier.

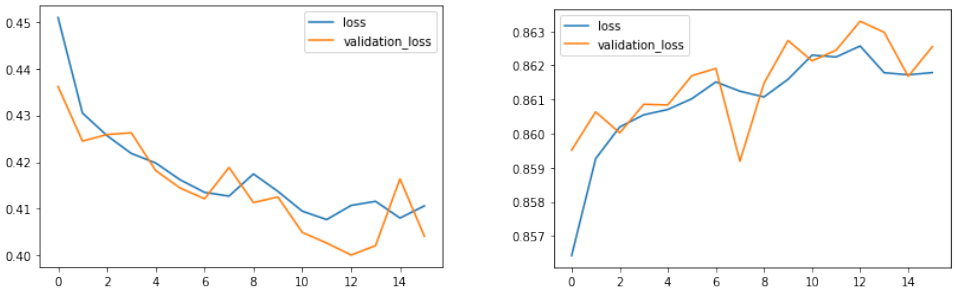


Figure 12: Left: Plot - Loss Vs Validation loss | Right: Plot - accuracy Vs Validation accuracy

References

[1] https://en.wikipedia.org/wiki/Artificial_neural_network.

[2] https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.

[3] https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html.

[4] https://keras.io/guides/sequential_model/.

[5] https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html.

[6] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[7] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html.

[8] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.

[9] https://scikit-learn.org/stable/modules/preprocessing.html.

[10] https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

[11] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, June 2002.

[12] Rachel Schutt and Cathy O'Neil. Doing data science, 2014.

[13] Peter Norvig Stuart Russell. *Artificial Intelligence : A modern approach*, pages 212–227. Pierrson, Boston, MA, 2010.