Predicting Car Crash Fatalities with Machine Learning

Zach Chase

University of Denver

COMP 4448: Data Science Tools 2

Neba Nfonsang

August 22, 2022

Car Crash Predictions

Recently the National Highway Traffic Safety Administration published a study in which they estimated 42,915 people died in motor vehicle traffic crashes nationwide in the year 2021. Not only is this number quite extreme, it is also a 16 year high. With thousands of families affected by such tragic experiences, there is hope that advances in technology and knowledge in the field will help decrease the needless deaths each and every year.

Purpose

One such way of helping decrease the number of motor vehicle deaths is by better understanding the data. This project aims to to do this by comparing the performance of the algorithms Logistic Regression, Support Vector Machine, Decision Tree, Random Forest, and K-Nearest Neighbor in predicting deaths from car accidents using car crash data.

Significance

In addition to the overall large number of fatalities from car crashes nationwide, there is also a very specific demographic that's leading cause of death is car crashes. That demographic, is children and adolescents. In 2016 *The New England Journal of Medicine* found that motor vehicle crashes were the leading cause of death in children and adolescents, representing 20% of all deaths. Ever since discovering this statistic, and ever since my wife and I have found out that she is pregnant, this subject has become of upmost importance to us.

Research Question

The significance of this topic, along with the readily available data detailing car crashes has led to a very natural research question for this project: How do the algorithms of Logistic Regression, Support Vector Machine, Decision Tree, Random Forest, and K-Nearest Neighbor

compare in predicting vehicle fatalities using known car accident information such as year of the model, if an airbag is deployed, etc.? This research question will ideally lead to better understanding about car crash fatalities, and knowledge of ways to reduce unnecessary deaths.

Description of Data

The data was gathered from Kaggle at the following site: **https://www.kaggle.com/datasets/prasannakm/car-crash-dataset.** From this data, there are 9 input variables which are described as the following with it's accompanying variable name:

Variable Name	Variable Description	Variable Type
seatbelt	If seatbelt was being worn	Category
frontal	If accident was frontal collision	Category
sex	Sex of occupant	Category
ageOcc	Age of occupant	Int
yearAcc	Year of accident	Int
yearVeh	Year of vehicle	Int
airbag	If airbag was deployed	Category
speed	Speed group of vehicle	Category
occRole	If occupant was driver or not	Category

As for the response variable, the variable name is 'dead' and it reports if the specific observation resulted in the death of the occupant.

Data Preprocessing

Exploratory Data Analysis and Visualizations

Before implementing the algorithms, it is first best to understand the data. Using the pandas profiling library, it appears that the data has 26,217 observations, one missing row, and

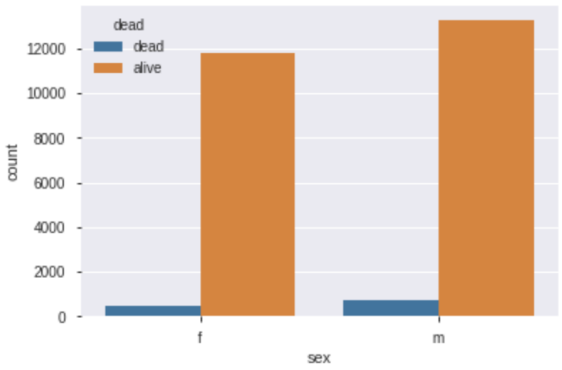
no duplicate entries. Additionally, a count of specific values for the output variable results in the following:

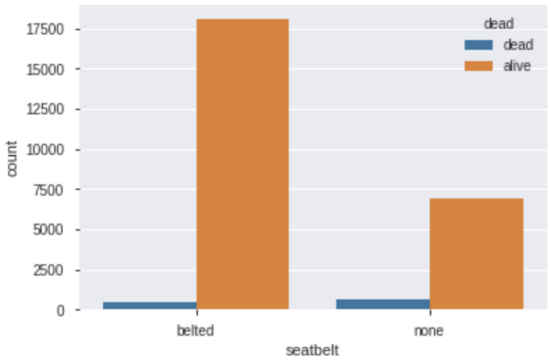
```
df['dead'].value_counts()
```

```
0    25036
1     1180
Name: dead, dtype: int64
```

With the response of “alive” having about 25 times more responses than that of “dead”, this results in an imbalanced dataset.

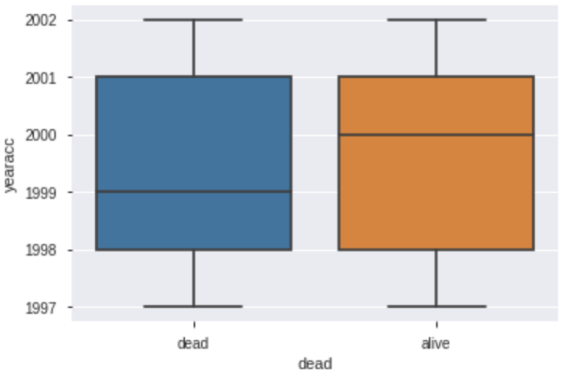
Included within the pandas-profiling report were several key visualizations that are useful in more exploratory data analysis. First, note the following seaborn countplots:

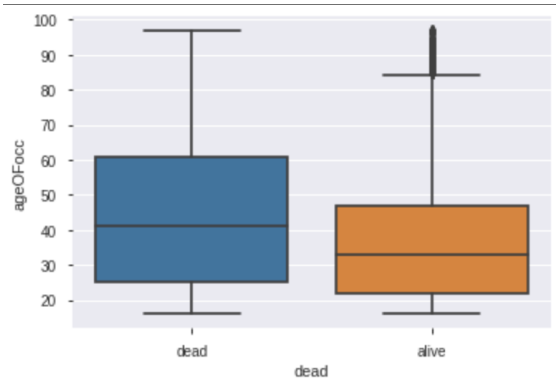




These plots excellently display just how imbalanced the data truly is, by having such a large amount of responses for “alive” compared to “dead”, this will severely impact the results of the algorithms.

Other additional box plots looking at the yearacc and ageOFocc variables, while grouped by if they died is also insightful:

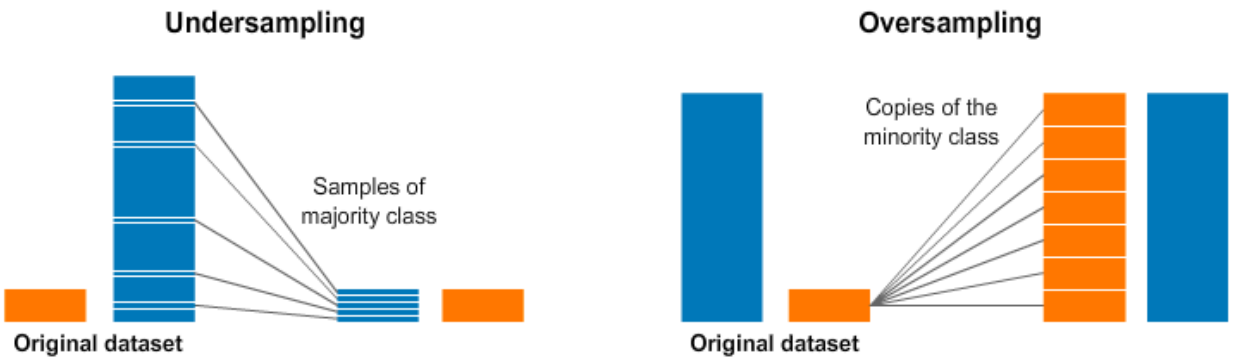




Note that it does appear that the median values for “alive” are different than the median values for “dead” in both of these plots. This indicates that there is potential in using the algorithms explained above in producing if a fatality occurs from a vehicle accident.

Data Preparation

The primary data preparation needed is the dealing of the imbalanced dataset. Note that there are two main ways of dealing with imbalanced data: undersampling and oversampling.



While undersampling balances out the data by deleting examples in the majority class, oversampling either duplicates or creates synthetic replications of the minority class.

Since undersampling would involve removing about 24,000 data entries, in this case it is best to perform oversampling. Within oversampling there are two main techniques used to create copies of the minority class: random oversampling and SMOTE. Random oversampling supplements the data with random copies of the minority class, until both class sizes equal each other.

SMOTE (Synthetic Minority Over-Sampling Technique) creates synthetic data by selecting pairs of minority points and creating a synthetic point on a line that connects these observations. While both have their uses, for this particular project the data will be balanced using random oversampling.

As for the other data preparation needed, the binary responses need to be replaced with 1s and 0s in most of the columns, column names need to be renamed, unused columns can be removed, and dummy variables need to be created for each categorical feature.

```
df['dead'].replace({'dead':1,'alive':0},inplace=True)
df['seatbelt'].replace({'belted':1,'none':0},inplace=True)
df['sex'].replace({'f':1,'m':0},inplace=True)
df['occRole'].replace({'driver':1,'pass':0},inplace=True)

df = df.drop(['weight', 'caseid', 'airbag', 'deploy'],axis = 1)
df.columns = ['speed', 'dead', 'seatbelt', 'frontal', 'sex', 'ageOcc', 'yearAcc', 'yearVeh', 'airbag', 'occRole']
df = pd.get_dummies(df, columns = ['speed', 'airbag'])
```

Data Splitting

Now the data can be split using train_test_split. Additionally, this step also includes using the imblearn library to perform random oversampling.

```
from imblearn.over_sampling import RandomOverSampler

ro = RandomOverSampler()
X_ro, y_ro = ro.fit_resample(X, y)

X_train_ro, X_test_ro, y_train_ro, y_test_ro = train_test_split(X_ro, y_ro, test_size=.33)
```

Model Building and Evaluation

Model Building

With the data now fully prepared, the next step is to actually input the data into the algorithms to create models. This is done by creating a function that iterates through each model, and calculating key metrics of performance.

```
table = PrettyTable()

table.field_names = ["Model", "Accuracy","Precision","F1"]

models = [LogisticRegression(), SVC(), DecisionTreeClassifier(), RandomForestClassifier(), KNeighborsClassifier()]

best = []

for model in models:

    model.fit(X_train_ro,y_train_ro)
    y_hat = model.predict(X_test_ro)
    accuracy = accuracy_score(y_test_ro,y_hat)
    prec = precision_score(y_test_ro, y_hat)
    f1 = f1_score(y_test_ro,y_hat)

    table.add_row([type(model).__name__, format(accuracy, '.2f'),format(prec, '.2f'),format(f1,'.2f')])

print(table)
```

The resulting output is an easy to read table that displayed various performance metrics for each algorithm.

Model Optimization and Model Selection

Building off of the above function, the next step is to tune for hyperparameters. This is conducted by building a list of hyperparameters for each algorithm, and then using the GridSearchCV tool to create a grid search or each algorithm with each combination of hyperparameters to optimize the accuracy metric.

```
table = PrettyTable()

table.field_names = ["Model", "Accuracy","Precision","F1"]

models = [LogisticRegression(), SVC(), DecisionTreeClassifier(), RandomForestClassifier(), KNeighborsClassifier()]
parameters = [{'penalty':('l2', 'none'), 'solver':('newton-cg', 'lbfgs')},
          {'C':[.1, 1, 2], 'kernel':('linear', 'poly', 'rbf')},
          {'criterion':('gini', 'entropy'), 'max_depth':[3, 5, 10]},
          {'n_estimators':[10,50,100,200], 'criterion':('gini', 'entropy', 'log_loss')},
          {'n_neighbors':[3, 5, 8], 'algorithm':('auto', 'ball_tree', 'kd_tree')}]

best = []

for model, params in zip(models, parameters):

    print(model, params)
    clf = GridSearchCV(model, params)
    clf.fit(X_train_ro,y_train_ro)
    best_model = clf.best_estimator
    y_hat = best_model.predict(X_test_ro)
    accuracy = accuracy_score(y_test_ro,y_hat)
    prec = precision_score(y_test_ro, y_hat)
    f1 = f1_score(y_test_ro,y_hat)
    best.append(clf.best_estimator_)

    table.add_row([type(model).__name__, format(accuracy, '.2f'),format(prec, '.2f'),format(f1,'.2f')])
```

Model Comparison

The final results of the algorithms is as follows:

Model	Accuracy	Precision	F1
LogisticRegression	0.81	0.81	0.81
SVC	0.58	0.65	0.45
DecisionTreeClassifier	0.97	0.95	0.98
RandomForestClassifier	0.98	0.97	0.98
KNeighborsClassifier	0.94	0.89	0.94

First, note that the best performing algorithm is that of the Random Forest, with an accuracy of 98%, a precision of 97%, and an F1 score of 98%. Additionally, Decision Tree also performed very well with metrics similar to that of the Random Forest. On the other end, the algorithm that performed the worst was the Support Vector Machine with an accuracy of 58%, a precision of 65%, and an F1 score of 45%.

In addition to the above performances, the algorithms were also run using the data that wasn't imbalanced, just to compare the results to the dataset that was balanced. The results are displayed below:

Model	Accuracy	Precision	F1
LogisticRegression	0.95	0.51	0.18
SVC	0.95	0.00	0.00
DecisionTreeClassifier	0.93	0.29	0.31
RandomForestClassifier	0.95	0.50	0.25
KNeighborsClassifier	0.95	0.69	0.14

Firstly, note that the accuracy of each of these models basically represents the percent of "alive" responses in the output variable. Essentially, this means that if a model predicts "alive" every single time, it will still result in an accuracy of about 95%. This can be seen by looking at the very poor precision and F1 metrics, which demonstrate that these models aren't actually very good when only using imbalanced data.

Conclusion

Conclusion and Lessons Learned

There are several key takeaways from this project. First off, the primary takeaway is the importance of dealing with imbalanced data. Many datasets deal with this crucial issue, from health data to credit fraud, and must be dealt with properly in order to create an accurate model.

Next, this project also taught the importance of using various performance metrics, not just the typical, standard ones. By simply looking at one metric, true understanding of model performance may not be understood, as seen when looking at just accuracy for the imbalanced data. More metrics lead to more understanding of performance.

Lastly, is that it is possible to predict deaths in car accident with significant accuracy using Random Forests. This was part of the purpose of this project, and it's good to know that this aspect of the project succeeded.

Recommendations/Up Next

The next steps to proceed now that this aspect of the project is done is to do further analysis into car crash fatalities. These steps may take many forms. It could involve looking at feature importance. It could look like conducting statistical tests on the individual features. Ultimately, the main goal is to figure out ways to make cars safer, and better understanding of what causes fatalities in crashes can help assist in this goal.