**Can we predict if a person will be killed by a train?**

# Background

In the year 2017, accidents were the 4th leading cause of death in the United States. Out of these 136,053 deaths, 889 were caused from railroad accidents. Yes, a seemingly small number, but certainly 889 too high. We are living in the 21st century, but yet people are still getting killed by trains.

Our goal is to use the dataset provided by the Federal Railroad Administration, and predict if a person will be killed in a train-related accident.

# Preprocessing

Our dataset was generously provided to us by the Federal Railroad Administration’s Office of Safety Analysis. Regarding our goal, the dataset is quite rich with a total of 75,485 entries and 50 different variables. The dataset is collected from 2010 to 2017.

Unfortunately, the dataset provided was not very clean. It contained missing values and extraneous columns. So the process we followed to clean our data is the following:

1. We removed columns that did not contain any values:
   1. Old Casual Occurrence Code, Old Equipment movement Indicator, Dummy, Dummy1, Dummy2, Dummy3
2. We removed columns that would have had little impact on our analysis:
   1. Narrative1, Narrative2, Narrative3, Narrative Length, Accident/Incident Number
   2. Although these narrations would have provided context and it would have been interesting to mine the sentiment, most of the entries had no narrations provided.
3. We removed columns that provided the same information as other columns but in a different format.
   1. Year of Incident, FIPS & County Code, County Code, Location of Injury on Body
4. We removed the Longitude & Latitude columns because of a significant amount of entries were missing (40%) & because we already had the state and county information. We were willing to make a trade off in the granularity of the data.
5. We encoded all the categorical features using the get\_dummies function from the Scikit-Learn package.
   1. The resulted in a dataframe with 75,485 entries and 3,488 features.

## Test/Train Split

Before we could go any further, we had to determine the best way to split our data into training and test sets. Unfortunately, we could not rely on normal k-fold validation, because we of the time series nature of our data. If we were to use k-fold cross validation, then we would have fallen into the trap of thinking that our model generalizes better than it really does. We would effectively be ‘peeking’ into the future to train our models.

So we decided to use the TimeSeriesSplit in Scikit-learn. This would allow us to perform walk forward testing in our models. Going forward, we would be only use our training set for outlier detection and PCA.

## Outlier Detection

Before we could reduce the dimensionality of our data by performing PCA, we had to detect and remove the outliers from the data. This is because Principal Components Analysis is sensitive to outliers.

We compared two methods for detecting outliers: Isolation Forests and Local Outlier Factors. We choose these methods due to the high dimensionality of our data.

However, the problem we faced was a lack of RAM. We were unable to use Isolation Forests and Local Outlier Factors without running out of memory. Even after turning the dataset into a sparse dataframe, it was an error that we could not overcome.

So we had to think about the question that we wanted to answer. Could we predict if a person would be killed in a railroad-related accident?

Keeping that question in mind, we were able to reduce the number of categorical values to the following features:

Month of Incident, Railroad Reporting, Type of Person, Job Code, Age of Person Reported, FIPS State Code, Railroad Class, FRA Designated Region, Form F6180-54 Filled?, Form F6180-57 Filled?, Day of Incident, Year of Incident – 4 Digits, Hour of Incident, Minute of Incident, AM or PM Indicator, County, Number of Positive Alcohol Tests, Number of Positive Drug Tests, Physical Act Circumstance Code, General Location of Person at Time of Injury, On-track Equipment Involved, Specific Location of Person at Time of Injury, Event Code, Cause Code, Hazmat Exposure?

This reduced the resulting encoded dataframe from 3,488 features to 2,797 features. But again, we run into the memory error.

In order to reduce the number of features, we had to determine which features we could avoid creating dummies for.

* Age of Person Reported, Day of Incident, Year of Incident – 4 Digits, Hour of Incident, Minute of Incident, Month of Incident

Since time can be considered both a categorical or quantitative value, we decided to treat it as a quantitative value. The number of features drops to 2,580 but we still encounter a memory error.

Thus, we had to perform feature selection before using our outlier detection methods.

### Feature Selection

We had to investigate why our initial number of 25 variables would balloon to a significantly higher number after encoding. Thus, we counted the number of unique values for each column.

|  |  |  |  |
| --- | --- | --- | --- |
| County | 1460 | Hour of Incident | 12 |
| Railroad Reporting | 562 | Type of Person | 10 |
| Job Code | 119 | FRA Designated Region | 8 |
| Age of Person Reported | 99 | Year of Incident - 4 Digits | 8 |
| Physical Act Circumstance Code | 95 | Railroad Class | 7 |
| Event Code | 82 | Number of Positive Drug Tests | 5 |
| Minute of Incident | 60 | Number of Positive Alcohol Tests | 4 |
| Cause Code | 54 | Hazmat Exposure? | 3 |
| FIPS State Code | 51 | Form F6180-54 Filled? | 2 |
| Specific Location of Person at Time of Injury | 50 | Form F6180-57 Filled? | 2 |
| On-track Equipment Involved | 39 | AM or PM Indicator | 2 |
| Day of Incident | 31 | Accident? | 1 |
| General Location of Person at Time of Injury | 23 |  |  |

We see that the county variable, with 1460 unique values, was the reason for the explosion in dimensionality. Since we do have a FRA Designated Region, we decided to forgo the granularity of the county variable. This would drop the dummified number of features to 1,120. But yet again, another memory error.

We can trim the dataset some more by dropping the Railroad Reporting since we have the Railroad Class variable. Again, we will be making a trade off in the granularity of the data. The number of dummified features drops to 558. Same error again.

We will also drop the job code variable. This variable describes the specific type of employee involved in the accident. But we have the Type of Person feature in our dataset. This type of person variable would indicate if a the person involved was an employee or non-employee. The # of features drops to 440. Finally, we don’t get an out of memory error.

### Outlier Detection Results

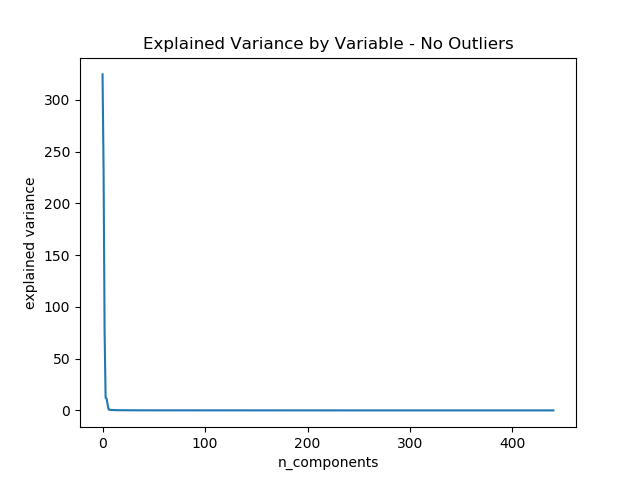
After finally reducing the number of features to a manageable result, we were able to apply the isolation forests and local outlier factor methods to detect outliers.

|  |  |
| --- | --- |
| **Outlier Detection Method** | **# Of Outliers** |
| Local Outlier Factor | 6710 |
| Isolation Forests | 6710 |

We see that the two methods agree upon the number of outliers. However, upon further investigation, only 1160 data points are flagged as outliers by both methods. Since both methods agree upon the number of outliers, but not quite which points should be outliers, we decided that we would just pick one method. In this case, we picked the isolation forests method. If the # of outliers differed, then we would have just picked the data points that both methods agreed were outliers.

## Dimensionality Reduction

### PCA

Now that we have removed the outliers from our data, we can finally perform our PCA analysis.

|  |  |
| --- | --- |
| **Variable** | **Explained Variance Ratio** |
| Month of Incident | 48% |
| Age of Person Reported | 35% |
| Day of Incident | 11% |
| Year of Incident - 4 Digits | 2% |
| Hour of Incident | 2% |
| Minute of Incident | 1% |
| Type of Person\_A | 0% |

The top 6 variables explain 99% of the variability in the data. So we slim our data from 441 features to 6.

### Manifold Learning

Unfortunately, PCA is a linear method and we’re not sure if this is the best way to reduce the dimensionality of the data. However, we were not able to perform any manifold learning techniques, due to the out of memory error. But this would be an interesting topic to explore in the future.

## Unbalanced Classes

Whenever we do classification, we need to make sure that our classes are not too unbalanced. In our situation, 5962 entries out of 75,486 resulted in a fatality. If our algorithms naively predicted that every event would not result in a death, then it would have a 92% accuracy rating! Clearly, we would have to pick an appropriate performance metric and balance the classes.

We decided to attack this problem from two directions. One direction would be use machine learning algorithms that are resistant to imbalanced datasets. For example, we would use things like bagging or boosting. The other direction would be to balance out the data itself by using oversampling.

### Random Over-Sampling

Given the number of entries that we had (75,486), we were not comfortable under-sampling the data. We didn’t feel that we had enough data to discard potentially useful information.

However, we were aware of the fact that over-sampling could lead to over-sampling. That is why we decided to choose specific algorithms that could potentially minimize that issue.

Before we oversample our data, it’s important to note the timing of the application of this technique. If we oversample too early in the process, then we will think that a model will generalize better than it actually does. Thus, the proper way is to split our data into test & training datasets, and then oversample only our training dataset. We must remember that the point of model validation is to estimate how the model will generalize to new data. It’s critical that oversampling is done correctly.­

# Machine Learning

We used the following classification techniques to predict our target variable of Fatality:

1. Logistic Regression
2. Decision Trees
3. Random Forests Classifier

The following are the results of the models using the Area Under Receiver Operator Curve as the performance metric.

Note that the results are an average of 3 runs

|  |  |  |
| --- | --- | --- |
| **Logistic Regression** | **Decision Tree** | **Random Forest** |
| **Non PCA'd Data & No Oversampling** | | |
| 0.7416 | 0.7088 | 0.6852 |
| **Non PCA'd Data & Oversampling** | | |
| 0.8995 | 0.6732 | 0.7241 |
| **PCA'd Data & No Oversampling** | | |
| 0.5000 | 0.5314 | 0.5042 |
| **PCA'd Data & Oversampling** | | |
| 0.5746 | 0.5231 | 0.5071 |

As we can see, something interesting happened. When we performed PCA on the training set, we ended up with performance scores worse than if we didn’t perform PCA at all.

It could possibly be that Manifold Learning would result in better scores than PCA, but alas, that would be something to test in the future.

# Conclusion

Can we predict if a person will be killed by a train? Based on our performance scores, the answer is a resounding yes. We can certainly do a better job than flipping a coin randomly.

With some more rigorous hyperparameter tuning, we might be able to squeeze some more performance out of the Logistic Regression Model. But an AUROC score of .90 seems to be pretty good.

Using these models will allow decision-makers to focus their time and effort on improving the factors of a person’s train-related death. For those in a regulatory role, reducing deaths is always a top priority. For those in the executive suites, preventing deaths limits the time spent with regulators and all the possible bad PR. Preventing deaths allows the system to keep on rolling and not grind to a halt.