DS 5110: Big Data Systems | Final Project

State of Virginia Traffic Reliability – MAP21

Christian Schroeder (dbn5eu), Timothy Tyree (twt6xy), Colin Warner (ynq9ya)

1. **Abstract**
2. **Introduction**

*Primary Objective:* Use actual Virginia highway traffic data from 2017-2020 to accurately predict the reliability of the state’s traffic projections. If that model is found, we can use the state’s forecasted metrics through 2024 to classify *future* unreliable highway segments.

*Context:* In 2012, President Obama signed into law the Moving Ahead for Progress in the 21st Century Act (MAP-21). Among other initiatives, this act transforms the process used for allocating funds towards the improvement of highway, transit, bike, and pedestrian programs - allowing a programmatic framework to inform whether or not a road is in need of transformation.

As part of an ongoing project at the Virginia Department of Transportation (VDOT), our team has been asked to explore more advanced classification models to predict if a MAP-21 reporting segment is reliable.

1. **Overview of Process**

Below we provide a quick description of each stage in our process. Please see specified sections in parentheses for a more thorough description of each stage.

*Data Import and Preprocessing (section ii.a.):* Write Preprocessing Class that (a) handles the import of packages and initializes the Spark Session, and then (b) reads, combines, and transforms data from 12 separate csv files to a workable format.

*Data Splitting (section ii.b.):* Split combined data into ‘actual’ and ‘forecasted’ data sets prior to Exploratory Data Analysis. Split the ‘actual’ data into train (90%) and test (10%) segments. Forecasted data is held out to use for classifying future unreliable segments.

*Exploratory Data Analysis (section ii.c.):* Evaluate distributions of numeric variables to determine necessary transformations. Three numeric variables benefit from log transformations. Also determined it necessary to drop geographic categorical variables due to certain instances of these variables not having examples of unreliable segments. Finally, we constructed a visualization to display highway segment data on a map of Virginia.

*Model Construction (section iii.a.):* Built pipeline for treating categorical variables as factors using StringIndexer and OneHotEncoder, as well as adding independent variables for modeling to features vector variable. Constructed Logistic Regression, Random Forest, and Support Vector Machine models.

*Model Evaluation (section iii.b.):* After evaluating our three model types on the basis of AUROC, TP, TN, FP, FN, and Accuracy, we ultimately decided that the Support Vector Machine with RegParam equal to \_\_\_\_\_\_ and Max Iterations equal to 10.

1. **Summary of Finding**
2. **Data and Methods**
3. **Data Import and Preprocessing**

*Write Preprocessing Class that (a) handles the import of packages and initializes the Spark Session, and then (b) reads, combines, and transforms data from 12 separate csv files to a workable format:*

The data import and preprocessing was by far the most complex portion of this project. We were provided 12 separate csv files from the Virginia Department of Transportation, with each file representing between one and six variables related to highway segments.

Beyond the fact that data was spread across 12 csvs, complexity was introduced by the way in which observations were grouped – whether it be by only highway segment (TMC), both TMC and Year, or by TMC, Year, and Period.

In order to accomplish all of the necessary preprocessing, we developed a preprocessing class. Below is an outline of the steps included in a readAndCombineData() function that is part of the preprocessing class:

1. Create a dictionary of directories with the directory name as key (ex. TMC/) and empty lists as values. This will hold dataframes that can be joined on shared unique identifiers.
2. Gets the full path to the input directories and uses a formatted string to get get the other directories in a loop. (looping through the directory name keys)
3. Creates a nested list of lists that define the type of joins each directory will be performing. Ordered the same as the directories.
4. Joins all data as follows;
   * a) Outer loop through each directory
   * b) Inner loop through the files in each directory and read the file into a Spark dataframe.
   * c) Append the dataframe to the values list within its respective directory (key/outer loop)
   * d) Get out of the inner loop, pop the last data frame out of this list, and save it to a temporary variable. This will be the df that starts joining on each directories respective join identifiers.
   * e) Join dataframes within each directory into one. Results in 3 dataframes after starting with 12. The logic is similar to sorting algorithms. Within another inner loop Start with the dataframe that was popped, set a temp df as that df, create a joined df with the temp df and the current loops df, on the columns specified within the current iterations index location of the join list, then set the temp df as the joined df, and set the start df back to temp. This will successfully join each dataframe within the list on their respective identifiers without repeating or missing dataframes.
   * f) Outside the previous loop, append the start df (which is now the full joined df) to the end of the list, and drop every other df in the list.
   * g) As a sanity check, loop through all columns in the joined df and drop any that may be duplicate.
5. Sequentially join the final three dfs into one, making sure to join on the df that had more previous identifiers so there's no data loss.
6. Create trainable and forecasted datasets by filtering on year (trainable < 2021), (forecasted > 2020)
7. Save the data.
8. **Data Splitting**

*Split combined data into ‘actual’ and ‘forecasted’ data sets prior to Exploratory Data Analysis:*

As briefly mentioned in the outline of our Preprocessing Class, the single resulting dataframe from preprocessing contains ‘actual’ data from 2017-2020, as well as the Virginia Department of Transportations ‘forecasted’ data through 2024.

The complete dataset with both ‘actual’ and ‘forecasted’ observations equates to 21 columns and 54,624 observations. We have elected to split the actual and forecasted data into two separate dataframes, using the actual data as our ‘trainable’ data set. This trainable data set has 22 columns and 27,312 observations.

*Split the ‘actual’ data into train (90%) and test (10%) segments:*

We then take the trainable data set and split that further into both train and test segments. We elected to use a 90%/10% split between the two segments given our relatively small data set after holding out the forecasted data. We use these train and test sets to evaluate several model types and determine whether there is a model that can accurately identify unreliable highway segments.

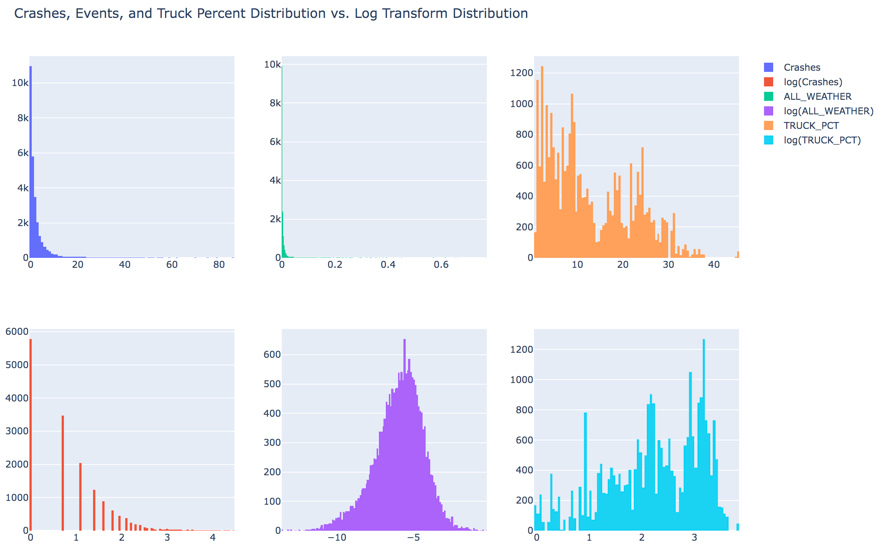
*Forecasted data is held out to use for classifying future unreliable segments:*

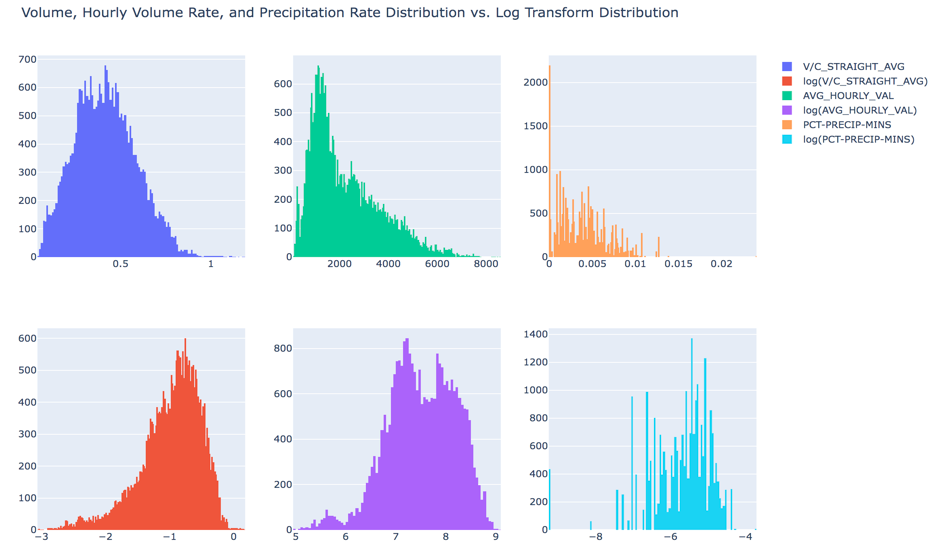
The forecasted data holds the identical shape as the trainable data, with 22 columns and 27,312 observations. If we are able to find a model that accurately identifies unreliable highway segments, we will use that model to classify these forecasted observations as unreliable or reliable.

1. **Exploratory Data Analysis**

*Evaluate distributions of numeric variables to determine necessary transformations:*

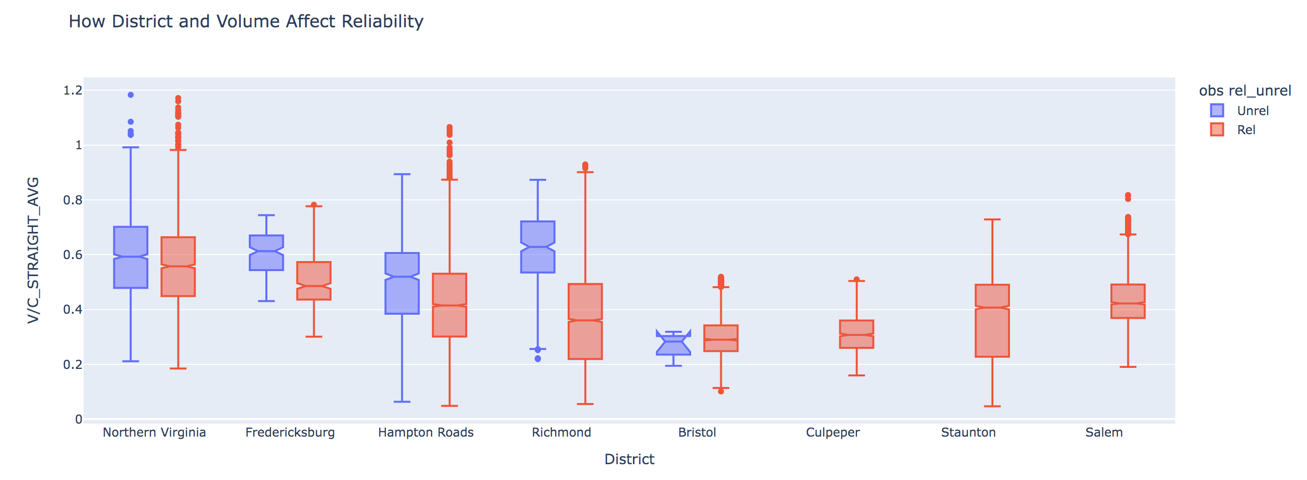
Upon this evaluation, it was determined that three numeric variables could benefit from a log transformation. These numeric variables included Weather, Precipitation, and Hourly Traffic Volume. Below are a few quick snapshots of the normal and log-transformed distribution of the numeric variables:





*Determined it necessary to drop geographic categorical variables due to certain instances of these variables not having examples of unreliable segments:*

One of the more interesting decisions we had to make during this project was how to treat categorical variables that describe geographic regions of Virginia. As you can see from the box plots below, the categorical variable of District contains several classes with zero examples of an ‘Unreliable’ highway segment. We noticed a similar trend in the variable ‘Road’.



In our evaluation, we believed there were two acceptable ways to move forward: (1) We could remove observations for classes that do not have an Unreliable instance, or (2) We could simply remove these geographic categorical variables altogether and keep all rows.

Ultimately, we elected to remove the categorical variables that describe geographic regions. This allowed us to maintain a larger trainable data set, as well as provided the possibility of our model ultimately flagging future unreliable segments in these areas where there are currently no examples of an unreliable segment.

*Constructed a visualization to display highway segment data on a map of Virginia:*

1. **Results**
2. **Model Construction**

*Build Pipeline:*

Prior to running several model types on the train/test data, we used PySpark’s pipeline functionality to streamline the process of preparing our dataset for modeling. Three stages that we set up in our Pipeline were StringIndexer, OneHotEncoder, and Vector Assembler.

In order for our models to treat categorical variables as factor variables, we must first use StringIndexer to convert each class of the variable to an index value. The StringIndexer is a preliminary step to the OneHotEncoder, which takes the index variables from StringIndexer and stores them in a new vector variable. Finally, the VectorAssembler stores all of our independent variables for modeling into a vector variable that we call ‘features.’

*Logistic Regression:*

The first model that we tried was a Logistic Regression. To begin, we ran Logistic Regression without cross validation or setting too many tuning parameters. Out of the box, the PySpark LogisticRegression function performed very well – however, we felt it was important to implement the LogisticRegression with cross validation to prevent overfitting.

For our Logistic Regression model with cross validation, we created a Tuning Parameter Grid that tested a range of values for the tuning parameters of RegParam (lambda), MaxIter, and ElasticNetParam (alpha). Ultimately, the parameters that had the best performance against the training data turned out to be lambda = 0.01, MaxIter = 10, and alpha = 0.2. This model, when used to predict the test data, had an accuracy of 0.91 and AUROC of 0.66.

The ElasticNet tuning parameter in Logistic Regression is an interesting one to dive deeper into. Two common Logistic Regression methods are Lasso and Ridge Regression, which seek to add a penalty to the coefficients of less important features. In Lasso Regression, the penalty is seeking to lower the coefficient of less important features all the way to zero – eliminating their impact on the classification model. In this instance the ElasticNet parameter would be set equal to 1. Alternatively, the Ridge Regression’s penalty seeks to lessen the impact of less important features, while maintaining a coefficient greater than zero. In this instance, the ElasticNet parameter would be set equal to 0. The value of 0.2 that is used in our best Logistic Regression model suggests that our data benefits from a methodology closer to Ridge than Lasso. This may indicate that a large portion of our columns have an impact on the reliability of Virginia highways.

The AUROC of 0.66 was not very convincing, so we decided to further tune the model by testing different thresholds for classification. We found the highest AUROC of 0.87 with a threshold of 0.1, but that came with a large drop in accuracy. Ultimately, we believed a good balance of AUROC and accuracy was found at a threshold of 0.2, with 0.82 and 0.88 respectively.

Table

Description automatically generated

*Random Forest:*

The second model we trained was Random Forest. This model type creates an ensemble tree predictor using bagging. At each tree split, a random subset of the predictors is chosen. This method gives less strong variables more of a chance to have an influence. We used cross validation to select the best model given the tuning parameters of max depth, max bins, and number of trees. The best performing Random Forest model against the training data had a max depth of 10, max bins of 10, and number of trees at 5. This model, when used to predict the test data, had an accuracy of 0.90 and AUROC of 0.57.

This model performed worse on the test data than the logistic regression model, so we tested additional threshold values to try and improve the AUROC. The highest AUROC of 0.81 was found at a threshold of 0.1, which came with a very large drop in accuracy to 0.70. We believed the best threshold for this model was also 0.2, with an AUROC of 0.78 and accuracy of 0.88.

Table

Description automatically generated

*Linear Support Vector Machine (SVC):*

The final model we trained was a Linear SVC. This model type creates hyperplanes to divide between classifications. The tuning parameters used for this model were lambda and MaxIter. The best performing SVC model against the training data had lambda = 0.01, and MaxIter = 10. This model, when used to predict the test data, had an accuracy of 0.91 and AUROC of 0.73.

Because the LinearSVC() function does not return a “probability” column, we were unable to test varying thresholds to increase the AUROC.

1. **Model Evaluation**

Performance of the best models:

Table, calendar

Description automatically generated

ROC Curves:

|  |  |  |
| --- | --- | --- |
| Logistic Regression | Random Forest | Linear SVC |
|  |  | Pyspark’s LinearSVC model does not return a probability value. |

1. **Conclusions**

The conclusions section can include future work, if there was more time.