Cancer Gene Expression Classification

SPRINGBOARD

Capstone Project 3

2023 | MAY

TAMARA HORNE

Table of Contents

Introduction

Medical providers and researchers can diagnose the cancer type of tumors more quickly and accurately with the assistance of machine learning. However, developing a meaningful machine learning method is difficult because of the curse of dimensionality. The available dataframes are regularly wide and short because of the vast number of genes and the relatively small number of samples. The dataframe I'll use in this project is no different.

The gene expression cancer RNA_Seq data (https://archive-beta.ics.uci.edu/dataset/401/gene+expression+cancer+rna+seq) is part of the larger RNA_Seq (HiSeq) PANCAN data set. (PANCAN is the Pancreatic Cancer Action Network). This database consists of samples representing 5 different cancer types. This project seeks to find a model which will accurately classify samples in all five classes.

Using six models, some linear and some non-linear, I achieved f1 macro scores ranging from 0.979-0.994. KNeighborsClassifier, Logistic Regression, and Support Vector Classification all tied for the highest score and would be excellent models for predicting future samples. Using SHAP with XGBoostClassifier, I was able to show which features were most important in predicting each cancer type. For additional details, please visit my GitHub repository folder: https://github.com/tamarahorne/Springboard/tree/main/Capstone%20Project%202.

Data Acquisition & Wrangling

The gene expression cancer RNA_Seq data consists of 801 rows representing samples and 20531 features. The features are RNA_seq gene expression levels which were measured using the illumina HiSeq platform. The data contained no missing values and their were no exceptionally high or low values in columns to indicate a mistake (figure 1); the maximum value in the dataframe was 20.778 and the minimum value was 0. So, I turned my attention to other concerns.

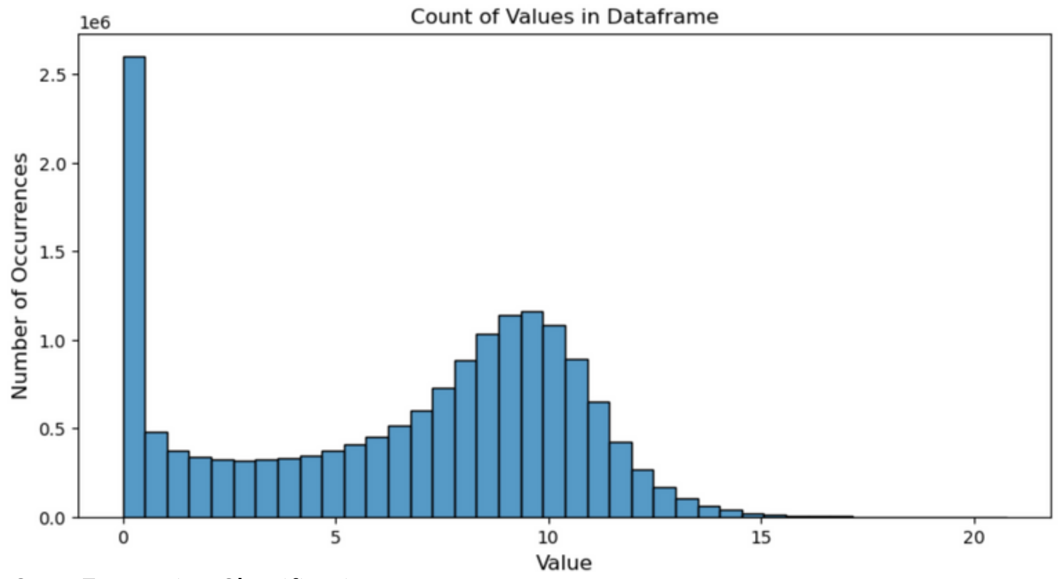
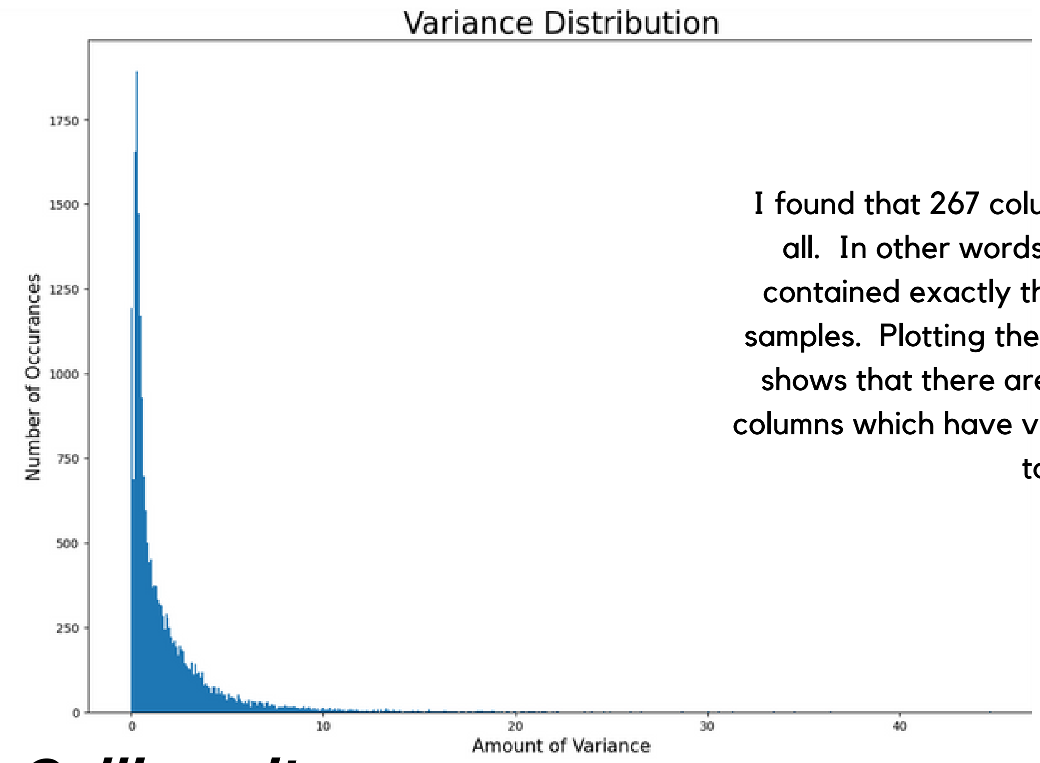


figure 1

Storytelling

With a wide and short dataframe, the first concern is going to be the number of dimensions. I wanted to gain a better understanding of the amount of variance within each column, the frequency of high collinearity between pairs of columns, and the balance of classes in order to choose an appropriate path for dimensional reduction and/or feature selection.



Variance

I found that 267 columns had no variance at all. In other words, each of those columns contained exactly the same value for all 801 samples. Plotting the distribution of variance shows that there are also a large number of columns which have very low variance - close to but not equal to zero.

figure 2

Collinearity

Next, I checked for collinearity between pairs of features since feature pairs with high collinearity cannot independently predict the value of the target. I discovered 2254 pairs of features with a correlation coefficient of over 90%. So, one way to the reduce the number of features could be to remove one feature from each of the highly collinear pairs. Six genes in the dataframe showed high collinearity with 50 or more genes each, and another 13 genes paired with 40 or more genes. So, these would be ideal candidates for removal.

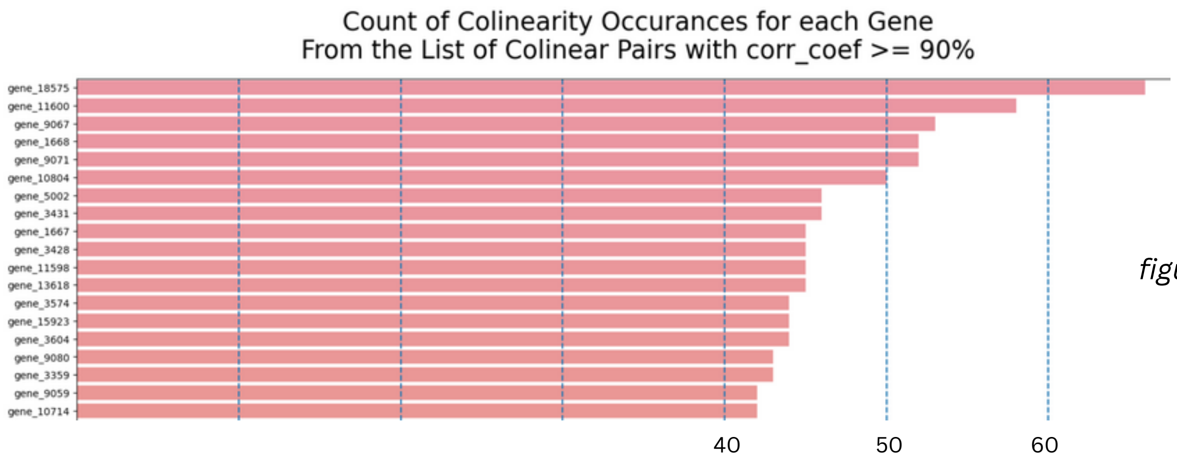
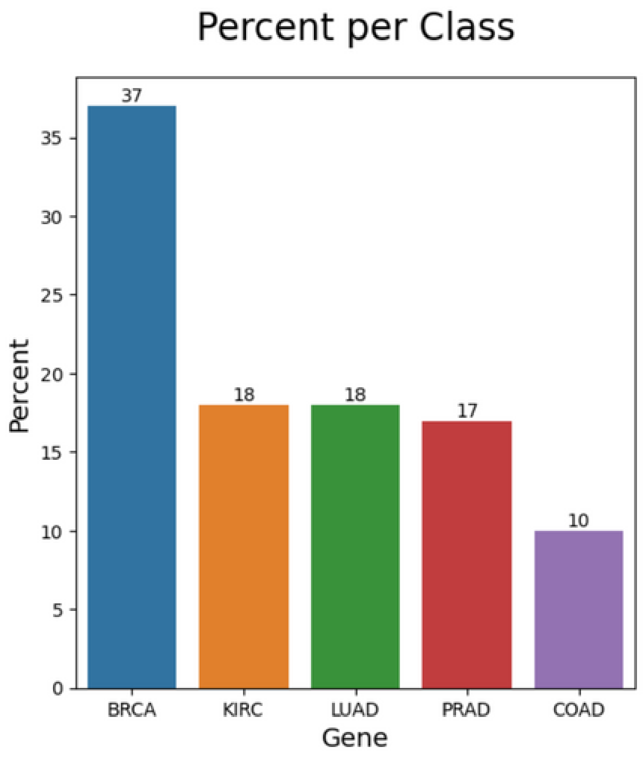


figure 3

Storytelling (continued)

Balance

The dataset has a class imbalance but it is not severe. The majority class, Class 0: BRCA ((breast), accounts for 37% of the samples. The smallest minority class, Class 1: COAD (colon), accounts for 10% of the samples. Oversampling and undersampling are common ways to treat data imbalance, but both methods have a downside. Oversampling can cause overfitting because it works by duplicating existing data. Undersampling reduces the number of samples in the larger classes yielding a smaller number of total samples available to train and test the model.



Decisions

After considering the number of dimensions, the variance, and the collinearity, I decided PCA (principal component analysis) would be a good option to try. PCA takes all of those elements into consideration when choosing what comprises each principal components.

In regards to the data imbalance, I decided to try modeling without utilizing either method (oversampling or undersampling) to see how well the baseline model worked without such intervention. I stratified the data when I created the train/test split in order to preserve the ratio of samples per class.

Baseline Modeling

I used logistic regression for a baseline model. Before I could move forward with modeling, though, I needed to choose the landing dimension for PCA. For a multi-class classification problem, the rule of thumb is that a dataframe should have at least 10 samples per feature for the smallest class. In other words, there should be at least ten times more rows than columns. The dataframe I am using has 801 samples, but the smallest class only has 78. So, according to the rule of thumb, the number of features should be restricted to 7 or, if I round, 8.

Baseline Modeling (continued)

Plotting the individual and cumulative variance explained by the first seven principal components (figure 1) gave me confidence that following the rule of thumb would be sufficient for this project. Further, when we look at a plot of just the first two principal components (figure 2), it seems entirely likely an even smaller number of principal components would be sufficient. An extension of this project would be to fine tune the number of principal components by comparing computation time against model effectiveness.

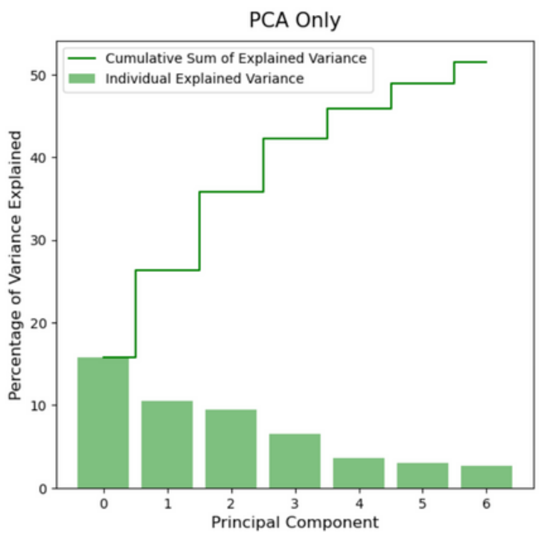


figure 1

Plot of First Two Principal Components

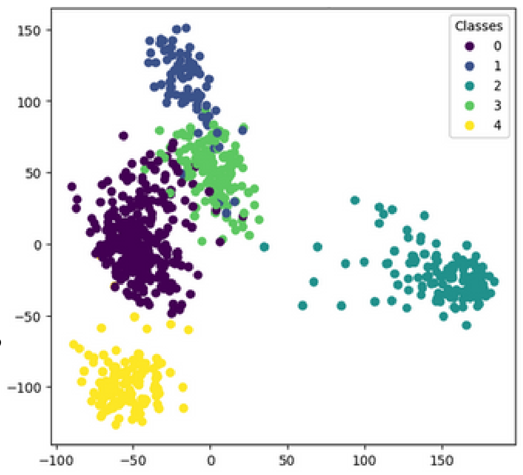


figure 2

I explored the idea of scaling the data, since that is a typical step to take before using PCA. However, I found that scaling made the amount of variance explained by the first seven principal components go down (figure 3), it made the classes less defined when plotting the first two principal components (figure 4), and the confusion matrix for the baseline linear regression model showed an increase in the number of errors (figure 5). So, I made the decision to proceed with unscaled data.

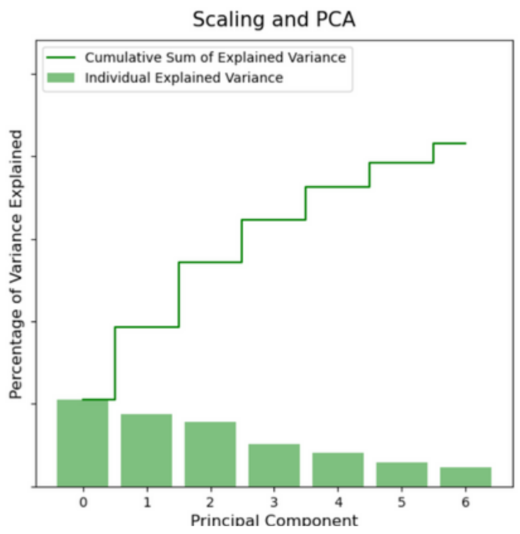


figure 3

Plot of First Two Principal Components

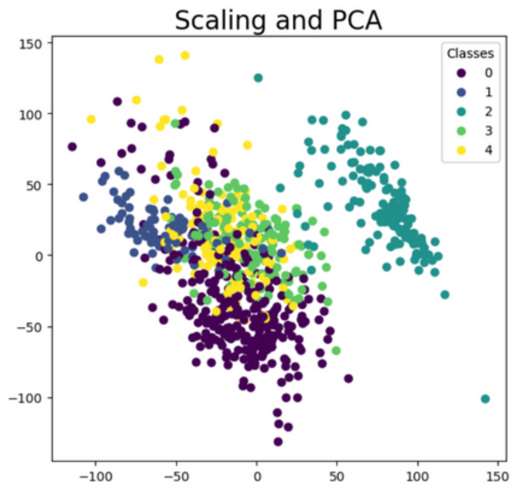
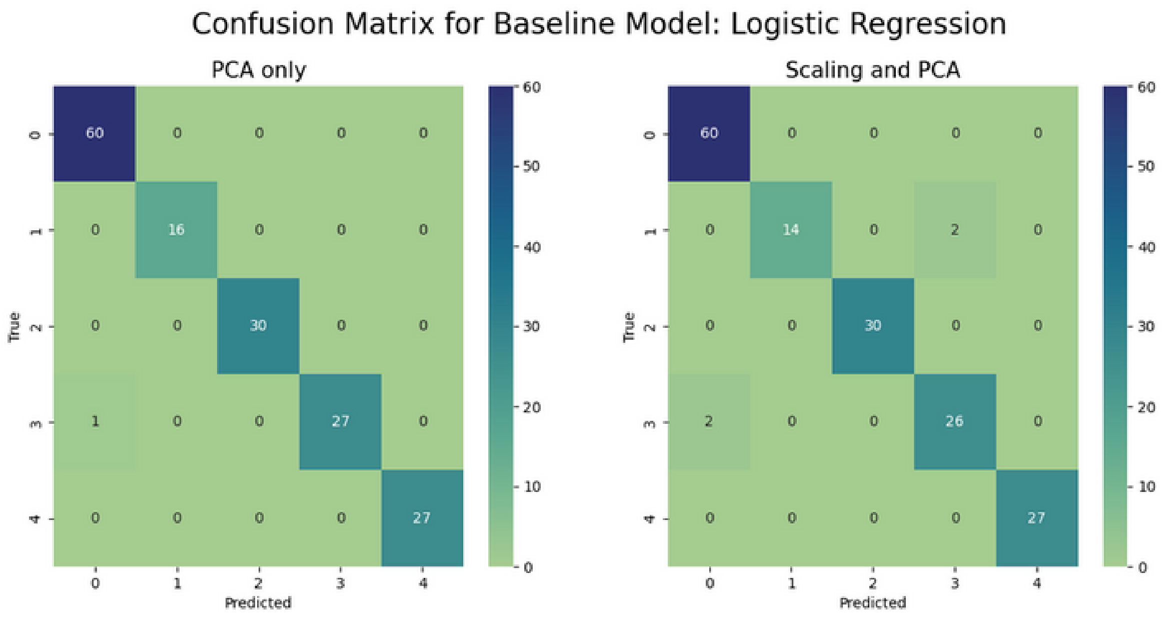


figure 4

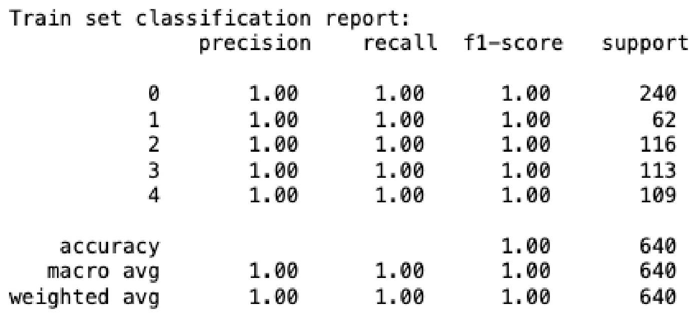
Baseline Modeling (continued)



figures 5

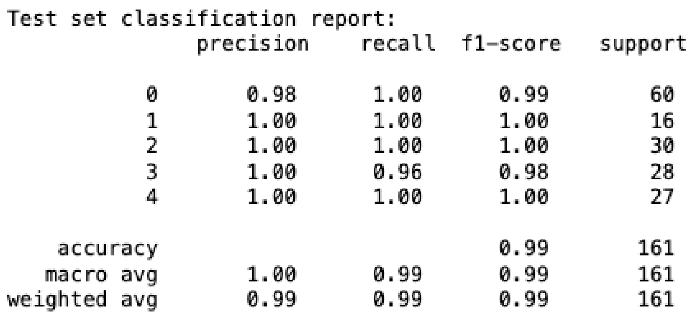
The logistic regression model with PCA performs very well for all classes so further action to balance the classes is unnecessary. The success of the baseline models also confirms that the landing dimension is sufficient. I used KStratifiedFold to validate the success of the model on the training data, just as an added check. KStratifiedFold preserves the class balance in each fold. The scores were consistent enough to give me confidence the model would generalize well, and indeed it did as we can see below with the classification report for the testing data.

Training Data



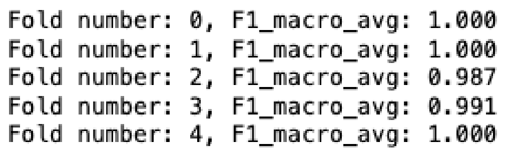
```
Train set classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       240
           1       1.00      1.00      1.00        62
           2       1.00      1.00      1.00       116
           3       1.00      1.00      1.00       113
           4       1.00      1.00      1.00       109

    accuracy                           1.00       640
   macro avg       1.00      1.00      1.00       640
weighted avg       1.00      1.00      1.00       640
```

Testing Data



```
Test set classification report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        60
           1       1.00      1.00      1.00        16
           2       1.00      1.00      1.00        30
           3       1.00      0.96      0.98        28
           4       1.00      1.00      1.00        27

    accuracy                           0.99       161
   macro avg       1.00      0.99      0.99       161
weighted avg       0.99      0.99      0.99       161
```

figure 6



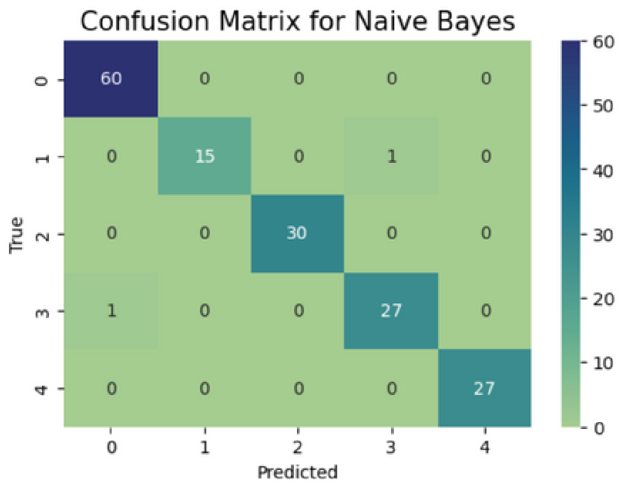
```
Fold number: 0, F1_macro_avg: 1.000
Fold number: 1, F1_macro_avg: 1.000
Fold number: 2, F1_macro_avg: 0.987
Fold number: 3, F1_macro_avg: 0.991
Fold number: 4, F1_macro_avg: 1.000
```

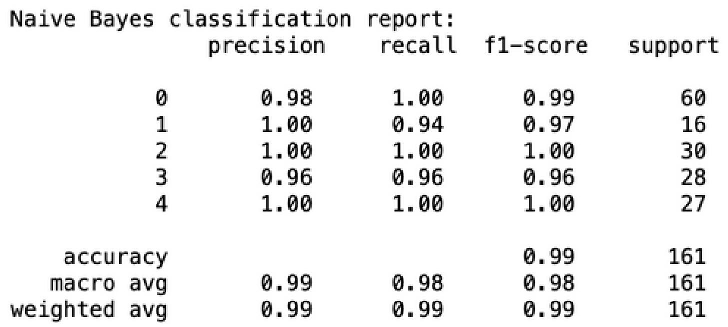
figure 7

Extended Modeling

After seeing the success of the logistic regression model, I was curious to see how other models, both linear and non-linear, would compare. I ran the linear models, Naive Bayes and Support Vector Classifier, in a pipeline with PCA. I then used StratifiedKFold for cross validation so the class balance would be preserved. Hyper-parameter tuning was not necessary since overfitting was not a concern. Finally, I collected the f1 macro score for each model and viewed the classification report and confusion matrix for each.

Naive Bayes - test data results

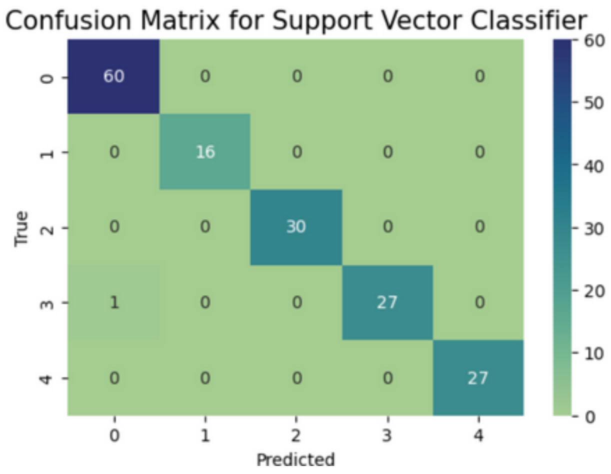


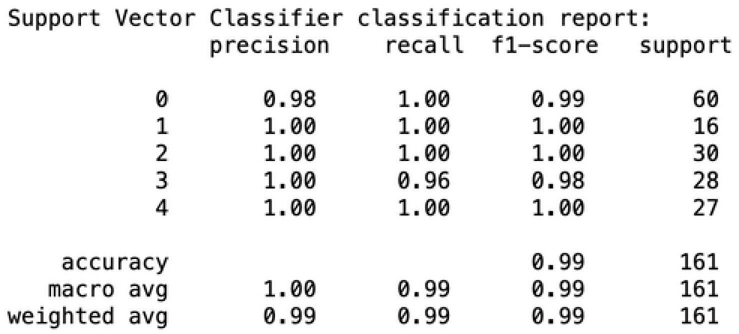

```
Naive Bayes classification report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        60
           1       1.00      0.94      0.97        16
           2       1.00      1.00      1.00        30
           3       0.96      0.96      0.96        28
           4       1.00      1.00      1.00        27

    accuracy                           0.99       161
   macro avg       0.99      0.98      0.98       161
weighted avg       0.99      0.99      0.99       161
```

figures 4, and 5

Like logistic regression, the Naive Bayes model makes one mistake for class three, classifying it as class 0, but Naive Bayes makes one additional mistake - misclassifying one sample of class 1 as class 3. The classification report still shows very good results with an f1 macro score of 0.98.

Support Vector Classifier - test data results




```
Support Vector Classifier classification report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        60
           1       1.00      1.00      1.00        16
           2       1.00      1.00      1.00        30
           3       1.00      0.96      0.98        28
           4       1.00      1.00      1.00        27

    accuracy                           0.99       161
   macro avg       1.00      0.99      0.99       161
weighted avg       0.99      0.99      0.99       161
```

figures 4, and 5

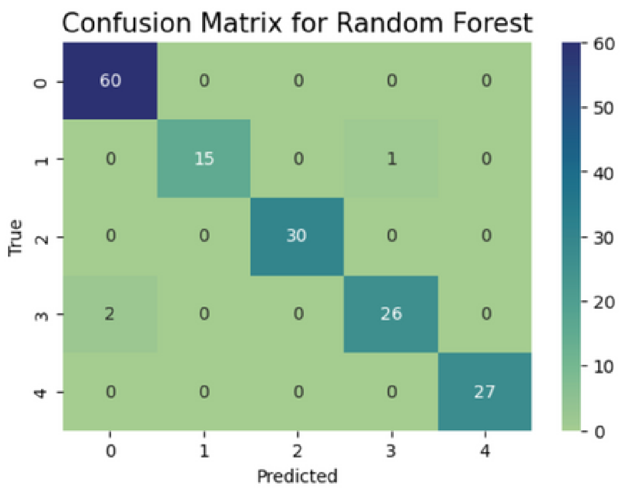
Support Vector Classifier mirrors logistic regression with the single misclassification of class 3 as class 0. The f1 macro average score is 0.99.

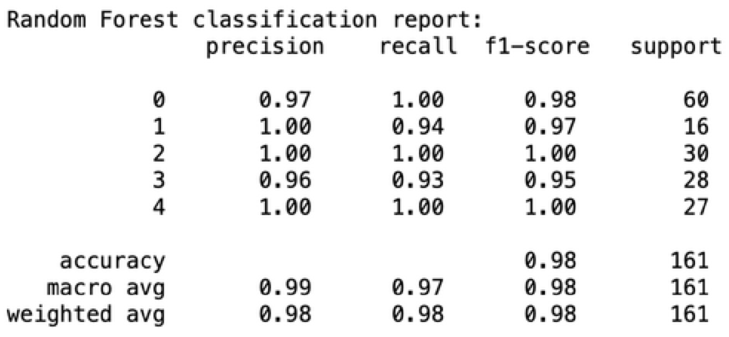
Extended Modeling (continued)

Next I tried a sampling of non-linear models including Random Forest Classifier, XGBoost Classifier, and K Neighbors Classifier. These models do require hyper-parameter tuning.

Random Forest Classifier

To find the best parameters for the Random Forest model, I tried a variety of values for the number of estimators, the max features, max depth, minimum sample leaf, minimum sample split, and the criterion. I then judged each combination by it's out-of-bag score (1 - oob_score). I then ran the model with the best parameters to find the f1 macro score (0.979).



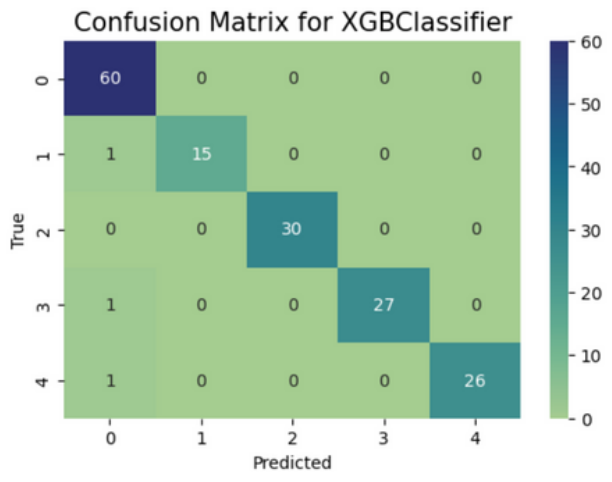


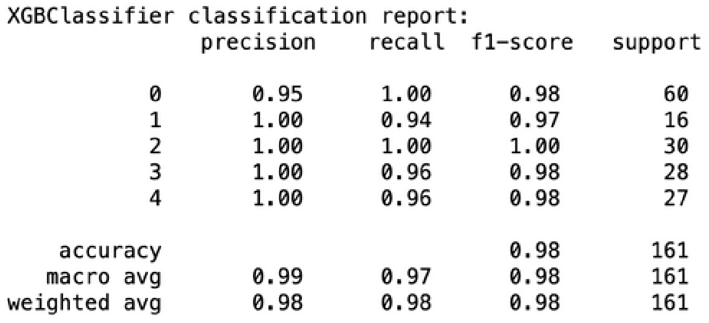
```
Random Forest classification report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.98        60
           1       1.00      0.94      0.97        16
           2       1.00      1.00      1.00        30
           3       0.96      0.93      0.95        28
           4       1.00      1.00      1.00        27

    accuracy                           0.98       161
   macro avg       0.99      0.97      0.98       161
weighted avg       0.98      0.98      0.98       161
```

figures 4, and 5

XGBoost Classifier

I used hyper-parameter tuning again for the XGBoost Classifier but, since it does not have an out-of-bag error option, I had to go about it in a different way. I used early stopping with 'merror' as the evaluation metric, and I used the accuracy score as the measure for determining the best parameters. I then ran the model with the best parameters and found the f1 macro score to be 0.981.





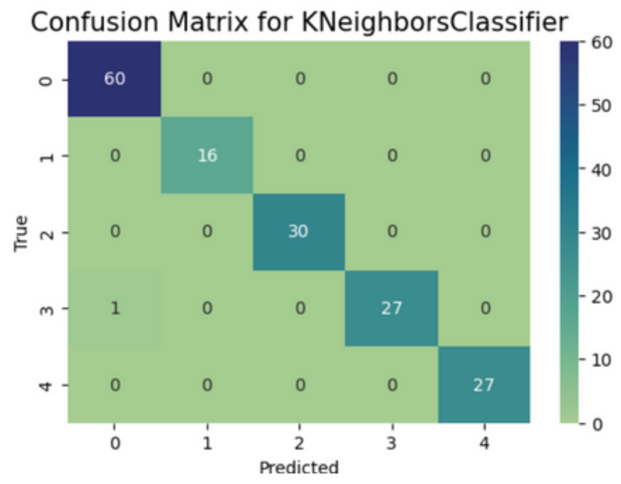
```
XGBClassifier classification report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.98        60
           1       1.00      0.94      0.97        16
           2       1.00      1.00      1.00        30
           3       1.00      0.96      0.98        28
           4       1.00      0.96      0.98        27

    accuracy                           0.98       161
   macro avg       0.99      0.97      0.98       161
weighted avg       0.98      0.98      0.98       161
```

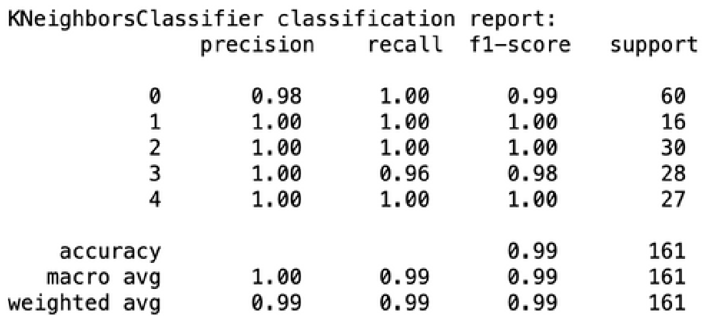
figures 4, and 5

Extended Modeling (continued)

K Neighbors Classifier

For the K Neighbors Classifier hyper-parameter tuning, I used Randomized Search CV to try a variety of values for the number of neighbors, the weights, the leaf size, and the value of p and scored the results using best_score. The model run with the best parameters yielded an f1 macro score of 0.9947.



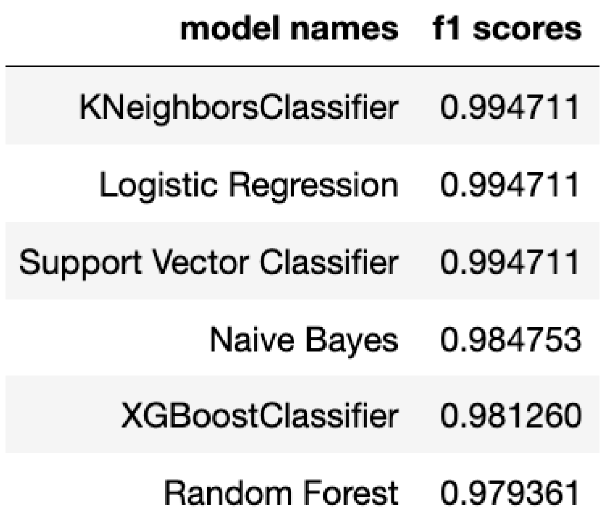


```
KNeighborsClassifier classification report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        60
           1       1.00      1.00      1.00        16
           2       1.00      1.00      1.00        30
           3       1.00      0.96      0.98        28
           4       1.00      1.00      1.00        27

    accuracy                           0.99       161
   macro avg       1.00      0.99      0.99       161
weighted avg       0.99      0.99      0.99       161
```

figures 4, and 5

Findings

In the end, three models tied for the highest f1 macro score. KNeighborsClassifier, Logistic Regression, and Support Vector Classifier all achieved a score of 0.994711. Moreover, the lowest f1 score of the whole group was only 0.01535 below the leaders. So, all six models performed very well.



model names	f1 scores
KNeighborsClassifier	0.994711
Logistic Regression	0.994711
Support Vector Classifier	0.994711
Naive Bayes	0.984753
XGBoostClassifier	0.981260
Random Forest	0.979361

Recommendations Section:

If clients needed a way to choose between the models, they might look to the confusion matrices. For example, All six models misclassified one class 3 as a class 0, but random forest misclassified a second one.

**Make a grid showing model, class x misclassifications, class y misclassification, etc.

Conclusions:

Tree based models worked the best for all three aggregations in terms of R squared values but K Neighbors was successful where MAE needed to be considered. XGBoost was the used for modeling_df and modeling_df3, whereas Random Forest was used for modeling_df2. The first aggregation predicts the safety risk for the nation as a whole. The second allows as the opportunity to see how the features impacting Nashville's safety risk compare to those affecting the nation. The third aggregation allows us examine the safety risk for the individual modes of transportation so Nashville can better assess the safety risk of adding a new mode of transportation and compare that to the risk associated with increasing service for modes of transportation which are already in present in the area.

Future Work:

- Discover the reason for the shifted data in modeling_df2.
- Explore the two low ridership values which has the highest SHAP values for modeling_df2.
- Use K Neighbors to further explore modeling_df2 and modeling_df3.
- Import the data about failures (needed for System Reliability) which is stored in separate annual Vehicle Maintenance xlsx files available through the NTD. Then incorporate the new data into the definition of the target to better align with the MTA's safety performance measures.
- Perform time series analysis on each aggregation to better understand the relationship of each over time. Included in this might be gaining an understanding of how the pandemic affected public transportation and whether or not the post-pandemic rebound is happening in an expected way. Are we returning to the same balance of feature importance post pandemic or are we on a new path? Is the small subset of post pandemic data a better predictor of future safety risk or is that predicting capability equivalent to the predicting capability of the whole dataset?

Recommendations for the Clients:

- Adding light rail would increase the overall safety risk for Nashville. I would not recommend recommend adding this mode of transportation.
- Adding service to an existing mode of transportation would be advisable.
  - Caution: The added vehicle revenue miles would be similar depending on the mode of transportation chosen, but not exactly the same. For example, rail would take a different path than a bus. The added vehicle revenue miles will alter the safety risk since it is part of the definition of the target.
  - The existing data shows that the safety risk for Nashville's current modes in ascending order is as follows: CR, VP, CB, DR, MB. So, Nashville should consider adding one of the modes earlier on that list to minimize safety risk.

Consulted Resources:

- National Transit Database Monthly Modal Time Series Data: https://datahub.transportation.gov/Public-Transit/Monthly-Modal-Time-Series/5ti2-5uiv/data
- National Transit Database Monthly Modal Time Series Information Page: https://datahub.transportation.gov/Public-Transit/Monthly-Modal-Time-Series/5ti2-5uiv
- Federal Transit Administration Glossary: https://www.transit.dot.gov/ntd/national-transit-database-ntd-glossary
- National Transit Database Mode Definitions: https://www.ftis.org/iNTD-Urban/modes.pdf
- Guidance for Type of Service "TN": https://www.transit.dot.gov/sites/fta.dot.gov/files/docs/NTD%202108%20FRN%20Webinar%20Presentation.pdf
- Metropolitan Transit Authority Safety Performance Measures: https://www.wegotransit.com/file.aspx?DocumentId=102
- Federal Transit Administration Safety Performance Target Guide: https://www.transit.dot.gov/sites/fta.dot.gov/files/2021-06/SPTs-Guide-v2-20210629.pdf

Packages:

- pandas
- matplotlib
- seaborn
- numpy
- sklearn
- scipy
- xgboost
- SHAP

Appreciation:

Special thanks to my mentor, AJ Sanchez, for all of his time, guidance, and encouragement. We do not walk alone.