Fetal Cardiotocograph (CTG) Classification

Kate Stadelman

Introduction

Background and Project Significance

Used during pregnancy to monitor fetal heart rate and uterine contractions, cardiotocography (CTG) tests "allow early detection of fetal distress" (Potter, 2021). Typically, CTGs are manually reviewed by obstetricians and midwives (Figure 1). One criticism of this diagnostic tool is the tendency for the same CTG to receive different interpretations depending on the reviewer (high inter-observer variability rate) (Prior & Lees, 2019). An automated, reliable method for reading CTGs would empower expectant mothers with the knowledge they need while also reducing burden on their clinical teams.

Project Purpose and Research Question

The purpose of this research project is to leverage machine learning to interpret CTGs. We explore feature extraction methods and classification algorithms in pursuit of our research question: How do Random Forest, Logistic Regression, and K-Nearest Neighbors compare in predicting normal vs. abnormal CTG results given test metrics?

Dataset Description

This research project utilizes a dataset of 2,126 CTGs. Test measurements form 21 numeric input features related to uterine contractions, fetal movements, as well as several fetal heart rate metrics: baseline beats per minute, accelerations and decelerations, short- and long-term variability, and histogram descriptive statistics. Three expert obstetricians reviewed these CTGs, providing an output label of normal, suspect, or pathological for each test.

Data Preprocessing

Exploratory Data Analysis and Visualization

During the data preprocessing phase, we confirmed the dataset has no missing values, as

well as generated descriptive statistics for each input variable. The distribution of output classifications was visually inspected. Due to extreme imbalance, suspect and pathological CTGs were mapped to a single abnormal class, Class 1, and normal CTGs were mapped to Class 0. While improved, the final output classes still suffer from imbalance (Figure 2). For every input variable, boxplots were created to identify features with significant outliers (Figure 3), and histograms were used to visualize feature distributions to determine the most appropriate scaler (Figure 4). After scaling, a correlation heatmap was produced to uncover highly correlated features that might negatively impact Logistic Regression (Figure 5).

Data Preparation and Splitting

Outliers were identified and removed using the interquartile range method, leaving 1,349 observations. Additionally, all input variables were scaled using min-max normalization. Due to several highly correlated features, principal component analysis was performed. Seven principal components were extracted from the original 21 features, maintaining 99.2% of the explained variance (Figure 6). These seven principal components were mapped back to their most correlated feature for later analysis of feature importance. To complete data preprocessing, the final dataset was randomly split into training (70%) and test (30%) sets.

Model Building and Evaluation

Model Building and Optimization

Initial Random Forest, Logistic Regression, and K-Nearest Neighbor models were constructed, and baseline scores were generated for both training and test datasets (Tables 1 & 2). Like other medical diagnostic tools, false negatives are far more harmful than false positives, as false negatives prevent patients from receiving the care they need. Because of this, we placed far more emphasis on recall and ROC AUC scores during model evaluation. Prior to tuning,

Random Forest was quite overfitted, and all three models had low recall scores for test data.

For hyperparameter tuning, we utilized grid search k-fold cross-validation. Max depth and out-of-bag scoring eliminated Random Forest overfitting. Lasso regularization was applied to Logistic Regression, and the number of neighbors, leaf size, and distance metric parameters were tuned for K-Nearest Neighbors. Of the many parameters tested, class weight had the greatest impact for both Random Forest and Logistic Regression in improving model outcomes. Code snippets for model building and tuning are found in Figures 7 – 9.

Model Comparison

Final models for Random Forest and Lasso Logistic Regression performed similarly, with Random Foresting achieving slightly higher recall and ROC AUC scores for both test and training (Tables 3 & 4). Tuning did not appear to improve K-Nearest Neighbors, which maintained high accuracy but ultimately had poor recall. In addition to scoring, comparison ROC curves (Figure 10) and confusion matrices (Figures 11 – 13) were generated, confirming Random Forest as the best model with 94.4% recall and 89.5% ROC AUC on test data.

Conclusion

In our efforts to compare Random Forest, Logistic Regression, and K-Nearest Neighbors in predicting normal vs. abnormal CTG results, we found Random Forest was the best model. Furthermore, CTG fetal heart rate histogram mode, abnormal short-term variability, and percentage of time with long-term variability were the most influential features (Figure 14). Tuning for recall increased the number of false positives, decreasing the overall accuracy. A high number of outliers significantly decreased our sample size, and we recommend gathering additional CTGs readings to determine if a larger sample would improve results.

References

Ayres-de-Campos, D., Bernardes, J., Garrido, A., Marques-de-Sá, J., & Pereira-Leite, L. (2000). SisPorto 2.0: A Program for Automated Analysis of Cardiotocograms. *The Journal of Maternal-Fetal Medicine*. *9*(5), 311-318. https://doi.org/10.1002/1520-6661(200009/10)9:5%3C311::AID-MFM12%3E3.0.CO;2-9

Prior, L., & Lees, C. (2019). Control and Monitoring of Fetal Growth. *Encyclopedia of Endocrine Diseases (Second Edition)*. https://www.sciencedirect.com/topics/nursing-and-health-professions/cardiotocograph

Potter, L. (2021, November 12). How to Read a CTG. *Geeky Medics*. https://geekymedics.com/how-to-read-a-ctg/

Table 1

Training Data Baseline Scores

Model	Accuracy	Precision	Recall	F1-Score	ROC AUC
Random Forest	1.000	1.000	1.000	1.000	1.000
Logistic Regression	0.912	0.647	0.427	0.515	0.699
K-Nearest Neighbors	0.932	0.767	0.544	0.636	0.762

Table 2

Test Data Baseline Scores

Model	Accuracy	Precision	Recall	F1-Score	ROC AUC
Random Forest	0.948	0.759	0.611	0.677	0.796
Logistic Regression	0.928	0.630	0.472	0.540	0.723
K-Nearest Neighbors	0.926	0.600	0.500	0.545	0.734

Table 3

Training Data Final Scores

Model	Accuracy	Precision	Recall	F1-Score	ROC AUC
Random Forest	0.857	0.427	0.913	0.582	0.881
Logistic Regression	0.885	0.484	0.874	0.623	0.880
K-Nearest Neighbors	0.933	0.744	0.592	0.659	0.784

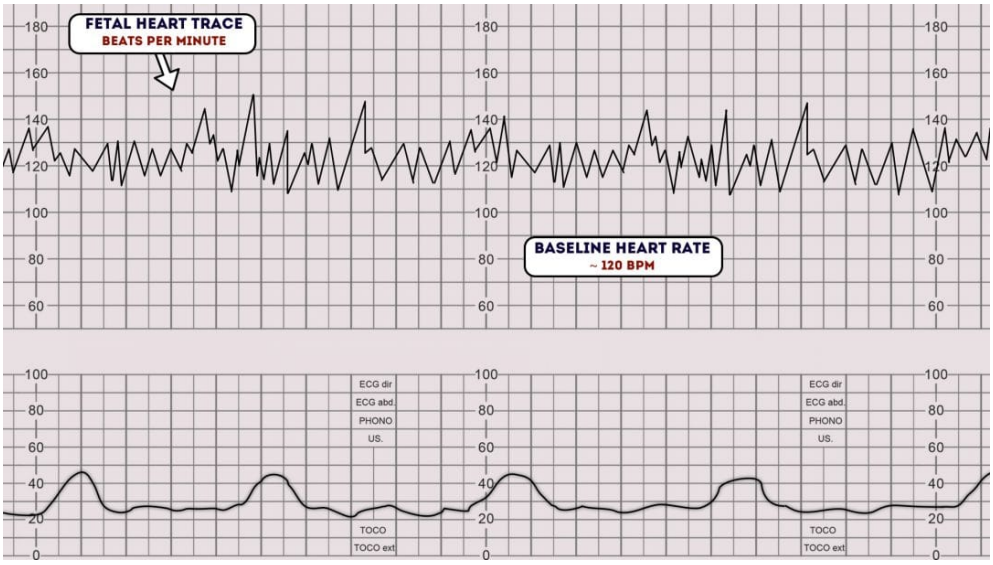
Table 4

Test Data Final Scores

Model	Accuracy	Precision	Recall	F1-Score	ROC AUC
Random Forest	0.854	0.374	0.944	0.535	0.895
Logistic Regression	0.877	0.410	0.889	0.561	0.882
K-Nearest Neighbors	0.931	0.667	0.444	0.533	0.711

Figure 1

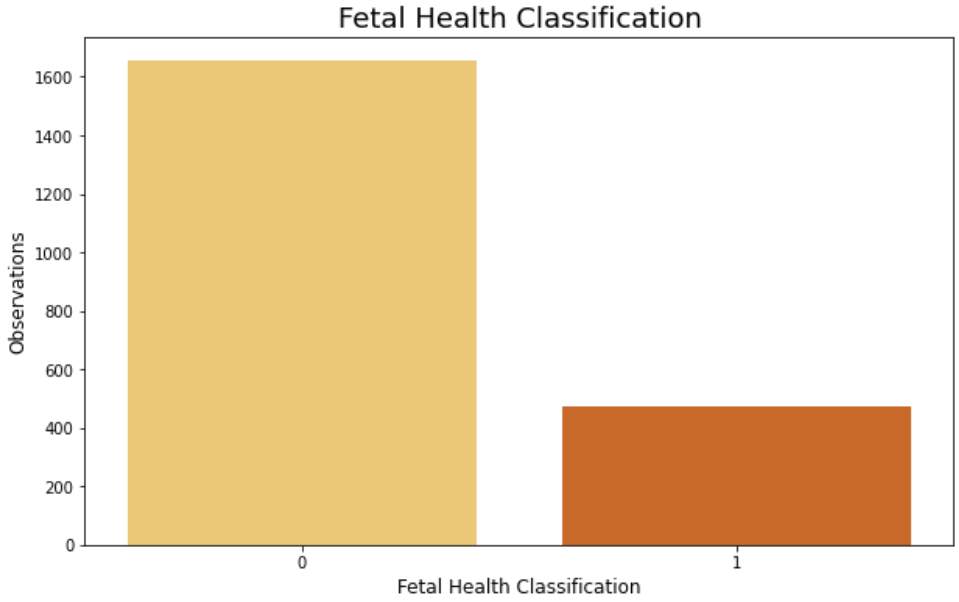
CTG: Baseline Heart Rate



Note. Reprinted from "How to Read a CTG" (Potter, 2021).

Figure 2

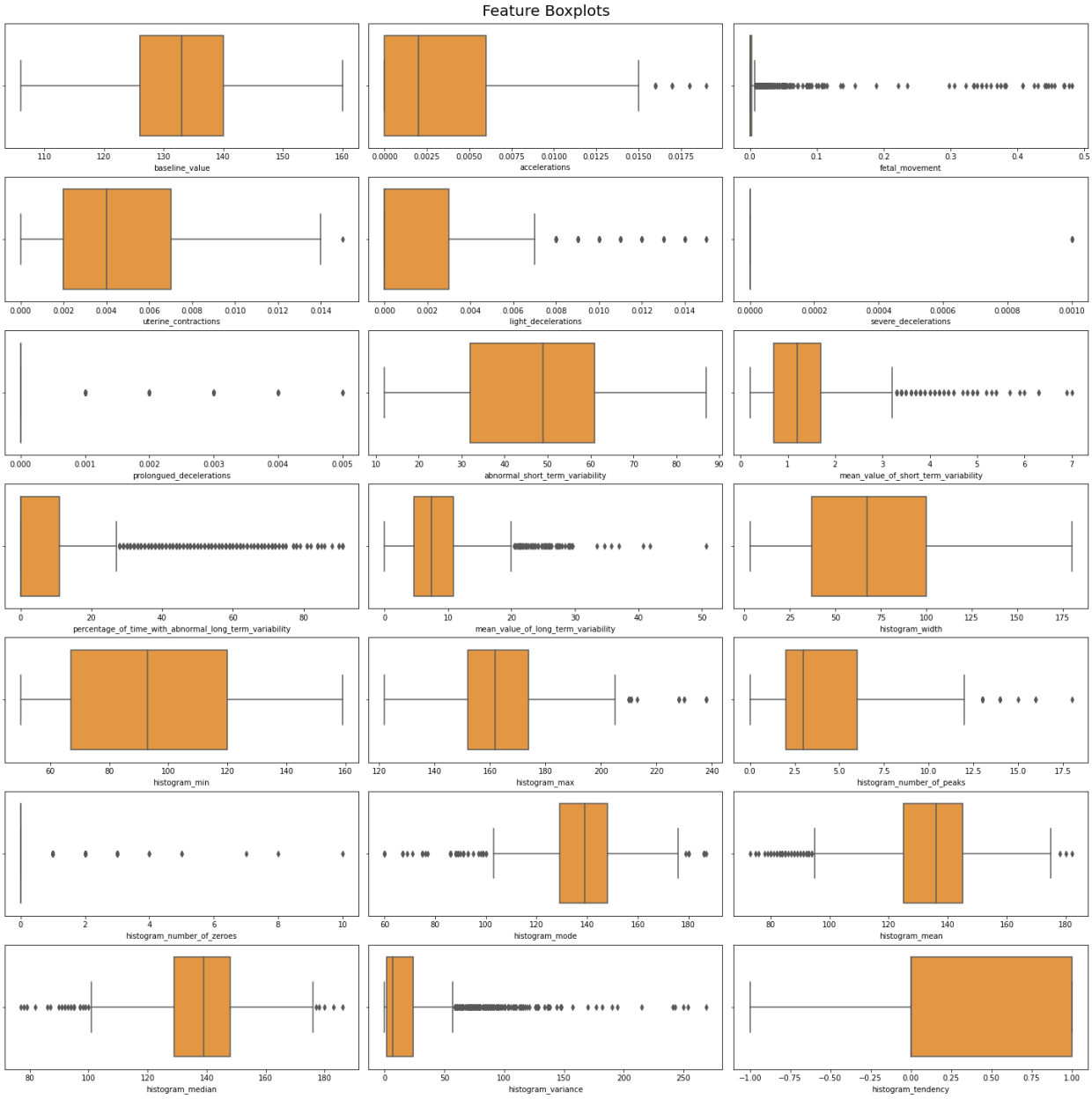
Distribution of Output Classifications



Note. Suspect and pathological CTGs were mapped to Class 1. Normal CTGs were mapped to Class 0. Output classes are imbalanced.

Figure 3

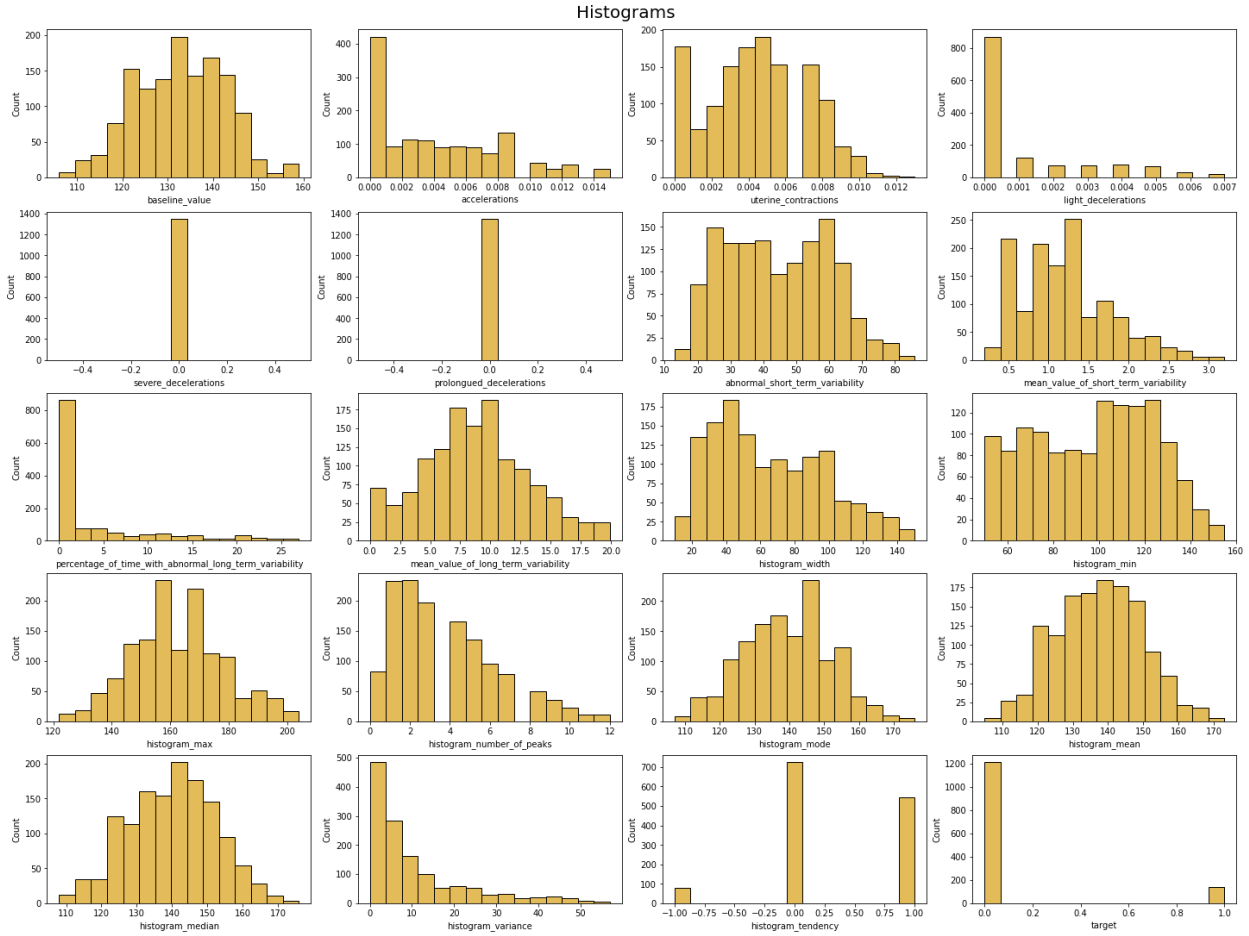
Identification of Features with Extreme Outliers



Note. Several features suffer from significant outliers.

Figure 4

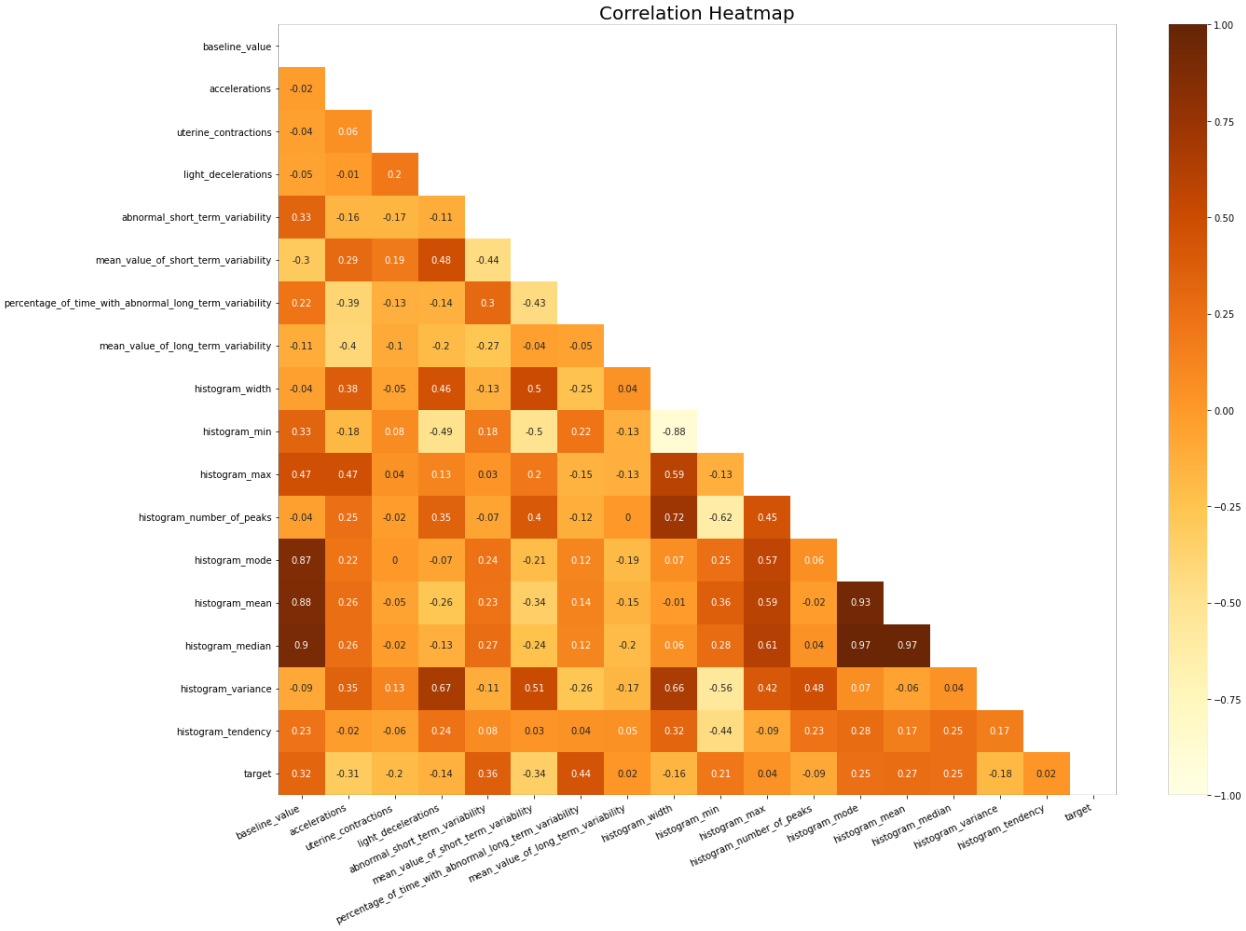
Review of Feature Distributions



Note. Min-max normalization was selected to scale data due to non-normal feature distributions.

Figure 5

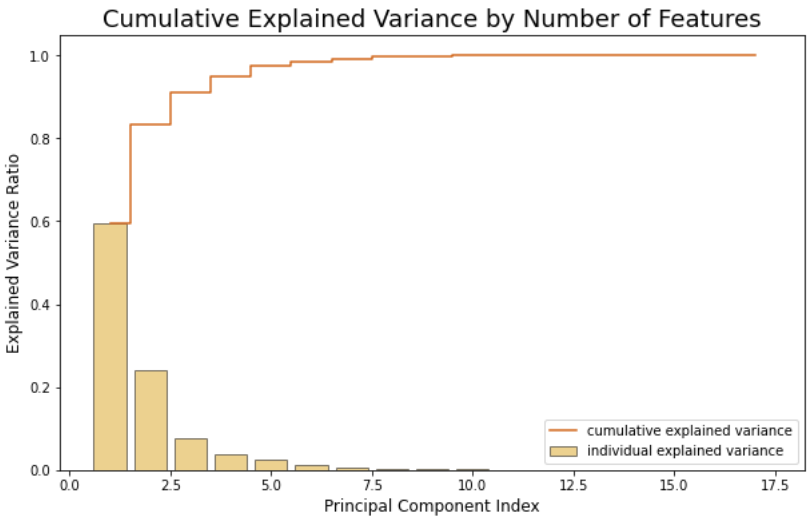
Feature Correlations



Note. Several features are highly correlated with each other.

Figure 6

Cumulative Explained Variance by Number of Features



Note. Seven principal components explain 99.2% of variance.

Figure 7

Random Forest Model Building & Hyperparameter Tuning Code Snippets

```
# Random Forest
rf_clf = RandomForestClassifier(random_state=36)
rf_clf.fit(X_train, y_train)
```

Random Forest

```
[29]: # Random Forest
rf = RandomForestClassifier(random_state=36)
rf_pg = {
    'n_estimators': [50, 70, 100],
    'max_depth': [2, 3, 5],
    'oob_score': [True, False],
    'class_weight': [{0:.09, 1:.91}, {0:.1, 1:.9}]
}
rf_grid = GridSearchCV(rf, param_grid=rf_pg, cv=10, scoring='recall')
with joblib.parallel_backend('threading', n_jobs=4):
    rf_grid.fit(X_train, y_train)

# Print best parameters.
rf_grid.best_params_
```

```
[29]: {'class_weight': {0: 0.09, 1: 0.91},
       'max_depth': 2,
       'n_estimators': 100,
       'oob_score': True}
```

Figure 8

Logistic Regression Model Building & Hyperparameter Tuning Code Snippets

```
# Logistic Regression
log_reg = LogisticRegression(solver='lbfgs', max_iter=1000, random_state=36)
log_reg.fit(X_train, y_train)
```

Logistic Regression

```
[31]: # Logistic Regression GridSearchCV
      lr = LogisticRegression(max_iter=5000, random_state=36)
      lr_pg = {
          'penalty': ['l1','l2'],
          'solver': ['liblinear','saga'],
          'class_weight': [{0:.01, 1:.99}, {0:.1, 1:.9}, {0:.15, 1:.85}],
          'C': [0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 1]
      }
      lr_grid = GridSearchCV(lr, param_grid=lr_pg, cv=10, scoring='roc_auc')
      lr_grid.fit(X_train, y_train)

      # Print best parameters.
      lr_grid.best_params_

[31]: {'C': 1,
       'class_weight': {0: 0.15, 1: 0.85},
       'penalty': 'l2',
       'solver': 'liblinear'}
```

Figure 9

K-Nearest Neighbors Model Building & Hyperparameter Tuning Code Snippets

```
# K-Nearest Neighbors
knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_train)
```

K-Nearest Neighbors

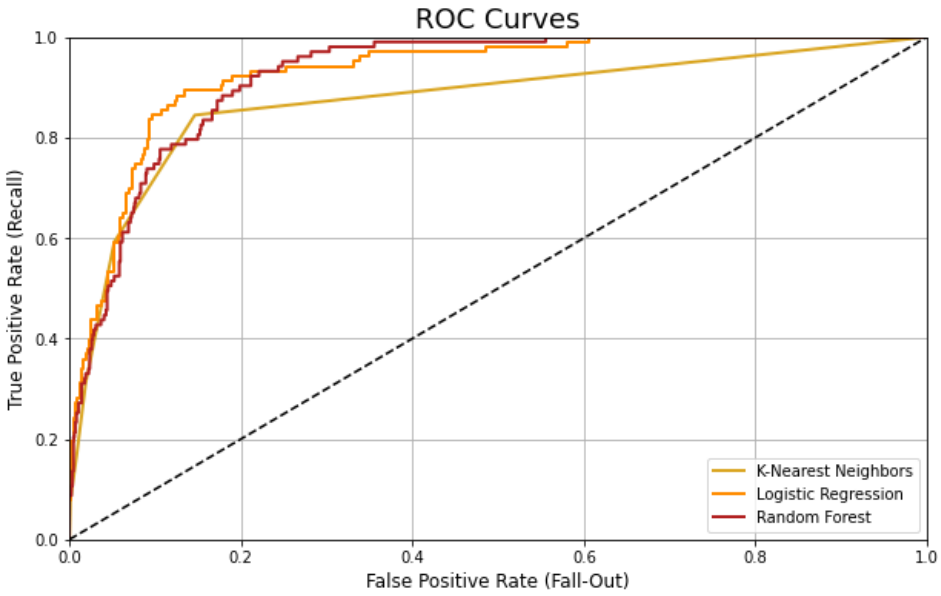
```
[33]: # KNN
      knn = KNeighborsClassifier()
      knn_pg = {
          'n_neighbors': [ i for i in range(5,15) ],
          'leaf_size': [10,30,50],
          'algorithm': ['ball_tree','kd_tree'],
          'p': [1,2],
          'metric':['minkowski','chebyshev']
      }
      knn_grid = GridSearchCV(knn, param_grid=knn_pg, cv=10, scoring='recall')
      with joblib.parallel_backend('threading', n_jobs=4):
          knn_grid.fit(X_train, y_train)

      # Print best parameters.
      knn_grid.best_params_

[33]: {'algorithm': 'ball_tree',
       'leaf_size': 10,
       'metric': 'minkowski',
       'n_neighbors': 5,
       'p': 1}
```

Figure 10

Receiver Operating Characteristic (ROC) Curves for Final Models



Note. Random Forest has the highest ROC Area Under Curve at 88.1% for training data and 89.5% for test data.

Figure 11

Random Forest Final Model Confusion Matrices

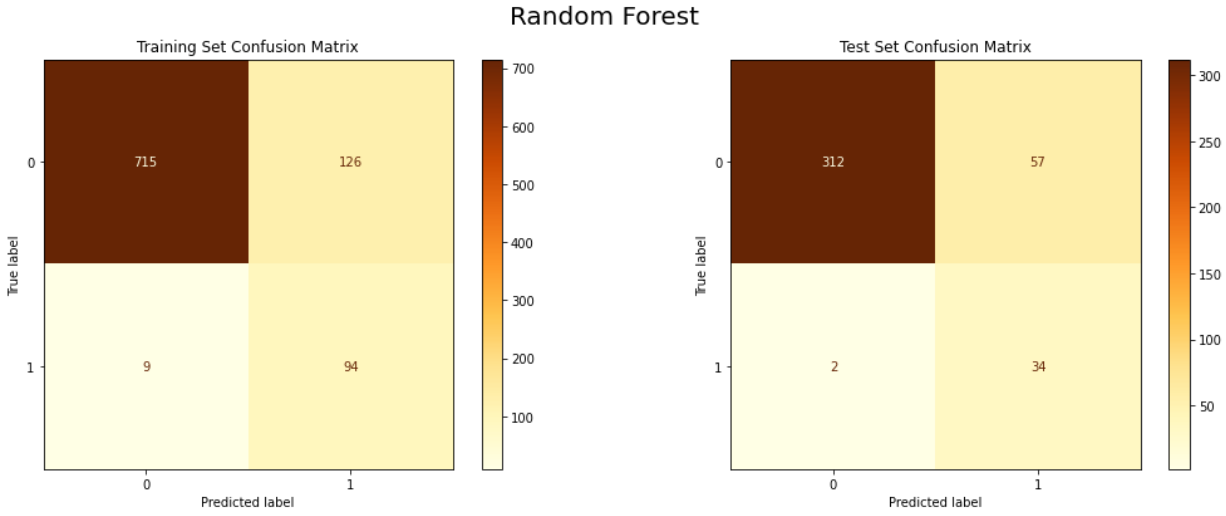


Figure 12

Logistic Regression Final Model Confusion Matrices

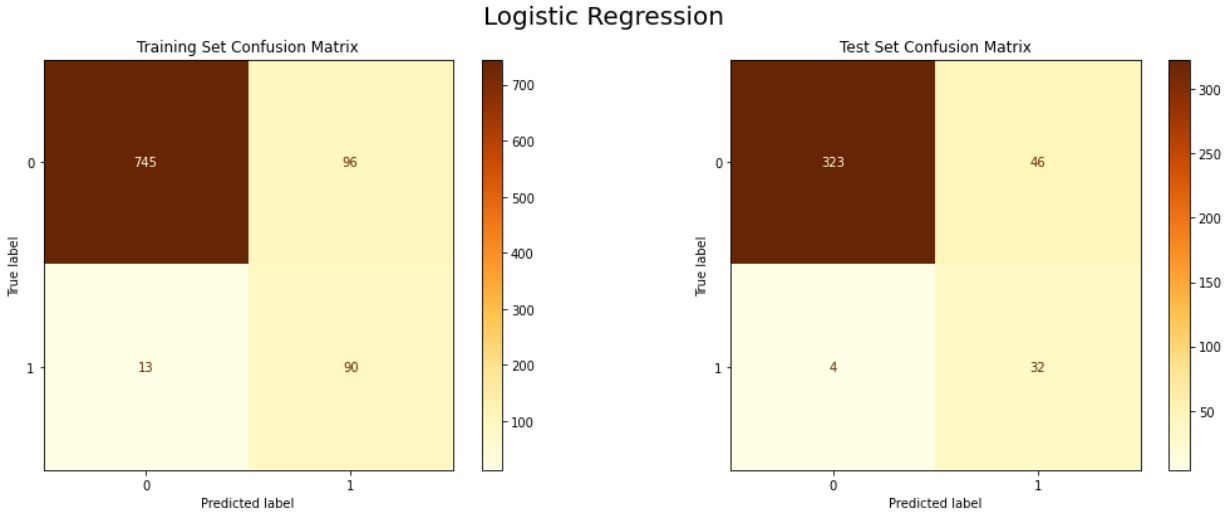


Figure 13

K-Nearest Neighbors Final Model Confusion Matrices

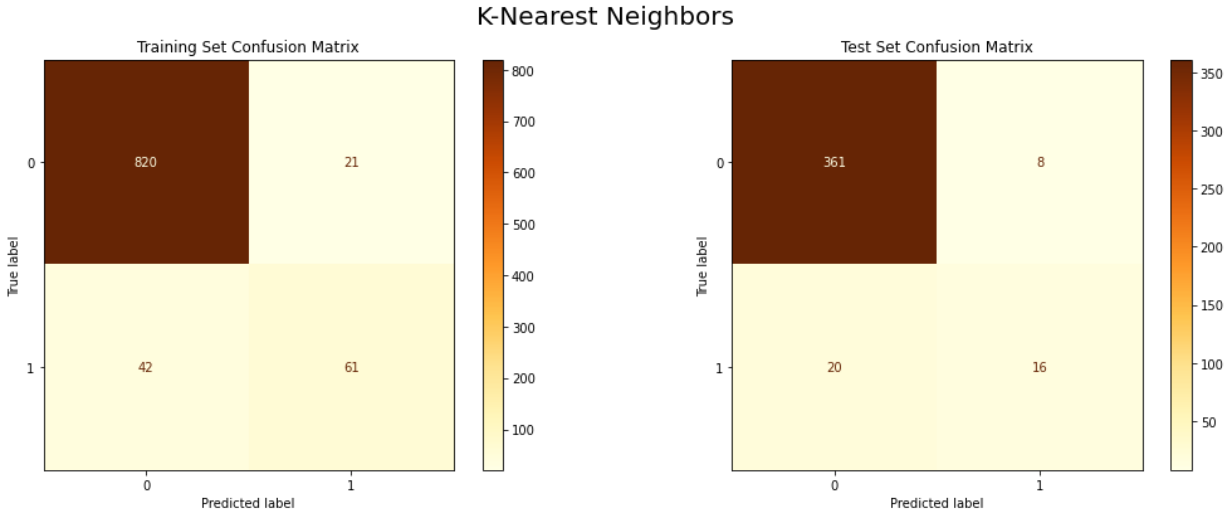
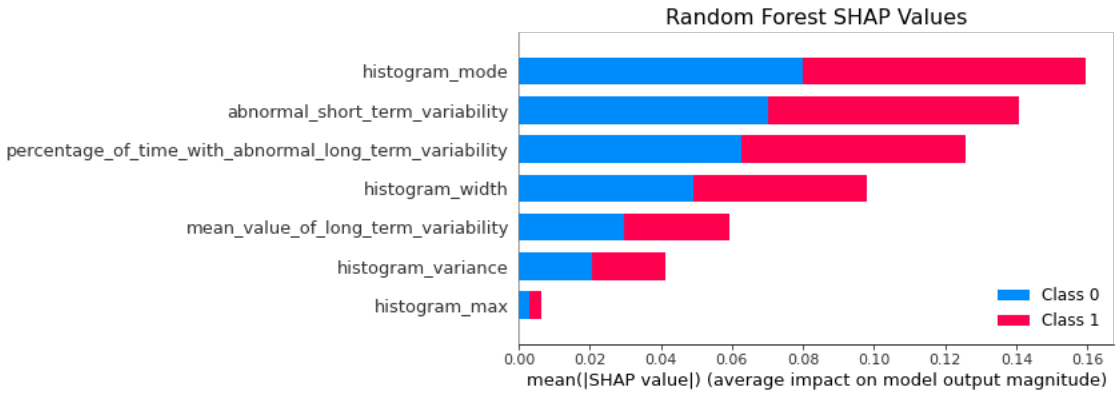


Figure 14

Feature Importance



Note. Principal components were mapped back to their most highly correlated input feature.