Joshua Kobuskie
Professor Yasser Abduallah
CS-634-101 Data Mining
17 November 2024

# Final Project Report - Supervised Binary Classification and Metrics

**Abstract:**

Performance metrics play a key role in determining the success of most data mining models. Without ways to accurately judge the performance of a model, it becomes impossible to determine which models perform best at a given task. For many applications, multiple models could be well suited for the job at hand, but different models may produce more favorable and insightful results. Performance may not be strictly based on accuracy, and can include a multitude of different factors and metrics that influence the strength of a model in a given situation. Throughout this project, I develop and investigate three different data mining models attempting to solve a real-world problem by applying binary classification data mining techniques. I utilize the Random Forest Classifier, Convolutional 1D Network, and Naive Bayes Algorithm to perform supervised learning on the KDDCup99 dataset, which contains TCP network connection data for identifying network attacks. The performance for each model is evaluated using 10-fold cross validation, and the average performance is reviewed to determine the strength of the classifiers in this specific scenario. The implementation and evaluation of these various data mining models helps to establish a deeper understanding of the real-world applications for these concepts and demonstrates the importance of performance metrics when comparing solutions.

## Introduction:

Performance metrics are an important part of the data mining process. Performance metrics enable a reasonable comparison of different data mining models, with unique insights into more than just the model's accuracy. Accuracy can be a misleading statistic, as it may not capture the true performance of the model. For example, on a binary classification dataset containing 900 true values and 100 false values, a model that always predicts true no matter the input will have an accuracy of 90%. While the accuracy of this model is high, the model is behaving poorly when predicting the values of the false class. In these scenarios, balanced accuracy could be used to better reflect the performance of the model across both classes. The balanced accuracy in this situation would be only 50%, as compared to the accuracy of 90%, demonstrating the performance of the model in a more realistic way.

Class imbalances are only one scenario where more advanced performance metrics can provide unique insights into the model's true performance. In a scenario where medical scans are being evaluated to detect cancer, the cost for a false positive diagnosis results in additional testing and no harm to the patient. However, the cost for a false negative diagnosis means that

the cancer will go unnoticed and could potentially worsen, endangering the patient's life. In this scenario, the cost for a false negative is significantly higher than the cost for a false positive. Accuracy alone cannot capture this impact, but the recall performance metric could be used to measure the power of a data mining model in this scenario and provide more meaningful insights to compare with. In some situations, the cost of a false positive could be higher than the cost of a false negative. An example of this scenario exists with spam filters. If an email is incorrectly classified as spam, the cost to the user could be high due to missed communications including bills, appointment confirmations, or meaningful conversations. However, if spam is accidentally allowed to enter into the mailbox, the cost to the user is very low. In this scenario, the cost for a false positive identification of spam is much greater than the cost of a false negative identification, and thus a performance metric such as precision could be used to evaluate the efficacy of the data mining model. In cases where both precision and recall are important, the use of an F1 score can help to balance the cost of both false positive and false negative results.

Depending on the application of data mining, different performance metrics may be more appropriate when evaluating the models. By possessing a thorough understanding of a diverse collection of performance metrics, it enables data mining models to be compared effectively against one another. This project explores 18 performance metrics used to evaluate three data mining models from differing points of view and determine the best model for network attack detection in the KDDCup99 dataset. This project utilizes the data mining process by performing data preprocessing including data cleaning and reduction, pattern discovery with three different models, pattern evaluation with the use of performance metrics, and knowledge presentation in the conclusion of this report. The performance metrics explored throughout this project can be leveraged to support real-world applications and accurately evaluate models for features beyond just accuracy.

Key steps in my implementation include:

- Loading the KDDCup99 Dataset
- Preprocessing the data for binary classification
- Modifying dimensional representation to prevent ordinality and multicollinearity
- Defining performance metrics
- Performing 10-fold cross validation with the Random Forest Classifier, Convolutional 1D Network, and Naive Bayes Algorithm
- Evaluating model performance and average performance metrics

**Core Concepts and Principles:**

A well-defined understanding of performance metrics enables the models implemented within this project to be evaluated effectively against one another. The performance metrics evaluated throughout this project are defined below.

**True Positive**

Correctly predicted positive values
True Positive = TP

**True Negative**
Correctly predicted negative values
True Negative = TN

**False Positive**
Incorrectly predicted positive values that are actually negative values
False Positive = FP

**False Negative**
Incorrectly predicted negative values that are actually positive values
False Positive = FN

**Positive**
Total positive values
Positive = P = TP + FN

**Negative**
Total negative values
Negative = N = TN + FP

**True Positive Rate**
The proportion of positive values correctly predicted as positive values
True Positive Rate = TPR = $\frac{TP}{P}$

**True Negative Rate**
The proportion of negative values correctly predicted as negative values
True Negative Rate = $\frac{TN}{N}$

**False Positive Rate**
The proportion of negative values incorrectly predicted as positive values
False Positive Rate = FPR = $\frac{FP}{N}$

**False Negative Rate**
The proportion of positive values incorrectly predicted as negative values
False Negative Rate = FNR = $\frac{FN}{P}$

**Recall/Sensitivity**

The proportion of positive values correctly predicted as positive values

Recall = $\frac{TP}{P}$

**Precision**

The proportion of predicted positive values correctly predicted as positive values

Precision = $\frac{TP}{TP + FP}$

**F1 Score**

The harmonic mean of precision and recall

F1 Score = $2 \times \frac{Precision \times Recall}{Precision + Recall}$

**Accuracy**

The proportion of correct predictions out of all predictions

Accuracy = $\frac{TP + TN}{P + N}$

**Error Rate**

The proportion of incorrect predictions out of all predictions

Error Rate = $\frac{FP + FN}{P + N}$

**Balanced Accuracy**

The average of the true positive rate and true negative rate

Balanced Accuracy = $\frac{TPR + TNR}{2}$

**True Skill Statistics**

A measure of a model's ability to distinguish between positive and negative values

True Skill Statistics = TPR - FPR

**Heidke Skill Score**

A measure of a model's accuracy as compared to a standard model with randomly selected classification

Heidke Skill Score = $\frac{2 \times (TP \times TN - FP \times FN)}{(TP + FN) \times (FN + TN) + (TP + FP) \times (FP + TN)}$

**Specificity**

The proportion of negative values correctly predicted as negative values

Specificity = $\frac{TN}{N}$

**Negative Predictive Value**
The proportion of predicted negative values that are correctly predicted as negative values

Negative Predictive Value = $\frac{TN}{TN + FN}$

**False Discovery rate**
The proportion of predicted positive values that are incorrectly predicted as positive values

False Discovery Rate = $\frac{FP}{TP + FP}$

**Brier Score**
A measure equivalent to the mean squared error of the model in binary classification

Brier Score = $\frac{1}{N}\sum_{i=1}^{N}(p_i - y_i)^2$

**Brier Skill Score**
A measure of the model's Brier Score as compared to a standard model with randomly selected classification

Brier Skill Score = $1 - \frac{Brier\ Score\ of\ Model}{Brier\ Score\ of\ Standard\ Model}$

**Area under ROC Curve**
A measure of a model's ability to distinguish between classes

Area under ROC Curve = $\int_0^1 TPR(FPR)\ d(FPR)$ or the sum of the area of trapezoids under the curve

## Project Workflow

**Dependencies:**
In order for the program to run successfully, the following libraries must be installed prior to running the program:

- Keras
- Matplotlib
- Numpy
- Pandas
- Scikit-learn
- Seaborn

To install these libraries, run the following commands from the command line:

```
python3 -m pip install keras
python3 -m pip install matplotlib
python3 -m pip install numpy
python3 -m pip install pandas
python3 -m pip install scikit-learn
python3 -m pip install seaborn
```

To run the program, run the following command from the command line:

```
python3 kobuskie_joshua_finalproject.py
```

Please note that this program may take up to 2 minutes to execute. A detailed description of running this program is provided in the Tutorial section of this report.

**Project Background:**

The classification algorithms that I have chosen to implement and evaluate in this project are the Random Forest Classifier, the Convolutional 1D network, and the naive Bayes algorithm. The dataset I have chosen to use in this project is the KDDCup99 dataset from the Scikit-learn real world datasets. This dataset contains TCP network connection data with about 5 million training records and 2 million test records. Each record is labeled as normal, or as a specific attack type. To ensure that this data will meet the criteria for binary classification used in this project, the data will be separated into 2 classes of either normal or attack, rather than the specific classes of attacks detailed within the dataset. This also models the intention of the dataset by prioritizing attack detection rather than attack type identification. This is evident as the training data contains 24 types of attack, but the test data contains an additional 14 types of attacks not previously seen in the training data. By attempting to identify attacks rather than attack types, this will test the models' ability to detect known, and unknown attacks based on their understanding of normal network data. The models I will implement will be able to classify records as either normal TCP network connection data or abnormal TCP network connection data indicative of an attack.

The Scikit-learn software can be downloaded at the following link:
https://scikit-learn.org/stable/install.html

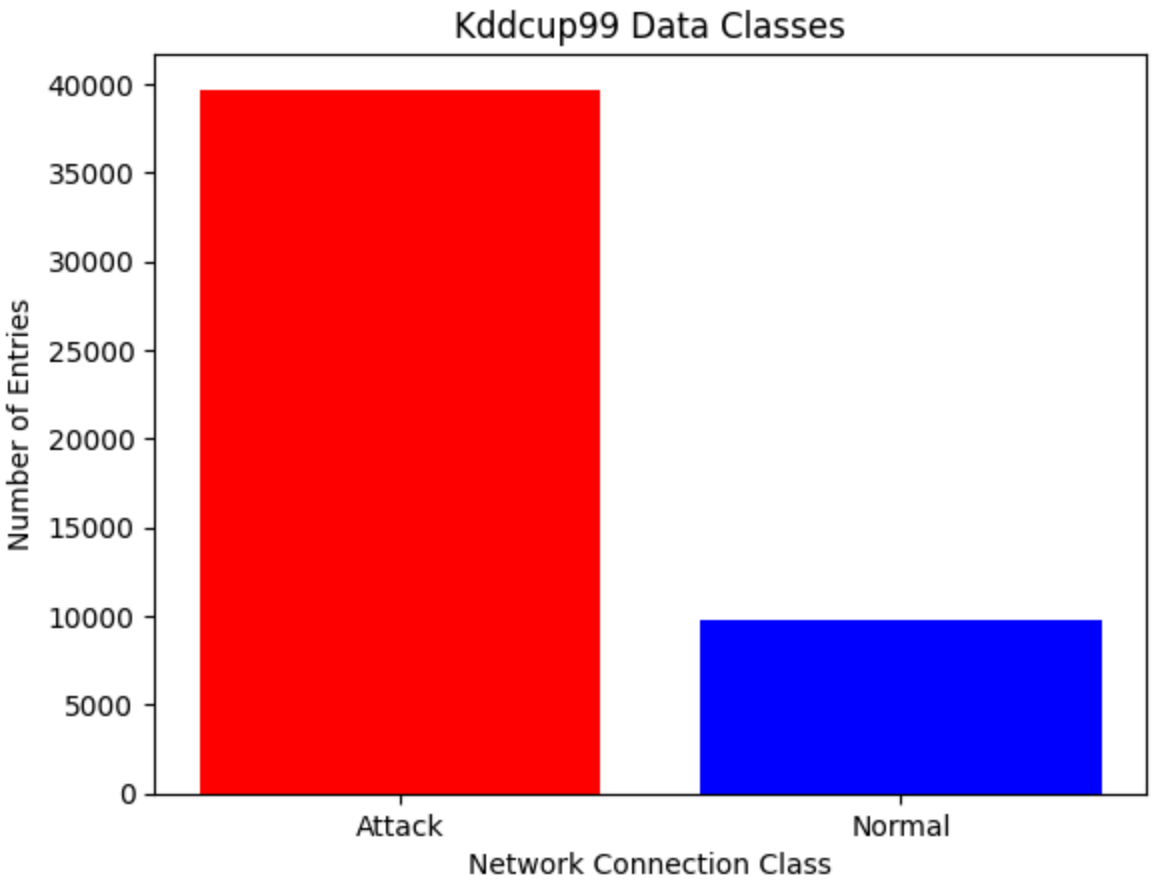
The Keras software can be downloaded at the following link:
https://keras.io/

The KDDCup99 dataset can be downloaded in its entirety at the following link:
https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

**Data Import:**
The KDDCup99 dataset is first imported from Scikit-learn and inspected. All values in the dataset are stored as objects and will be converted into integers, floats, or strings based on the appropriate data type for easier processing.

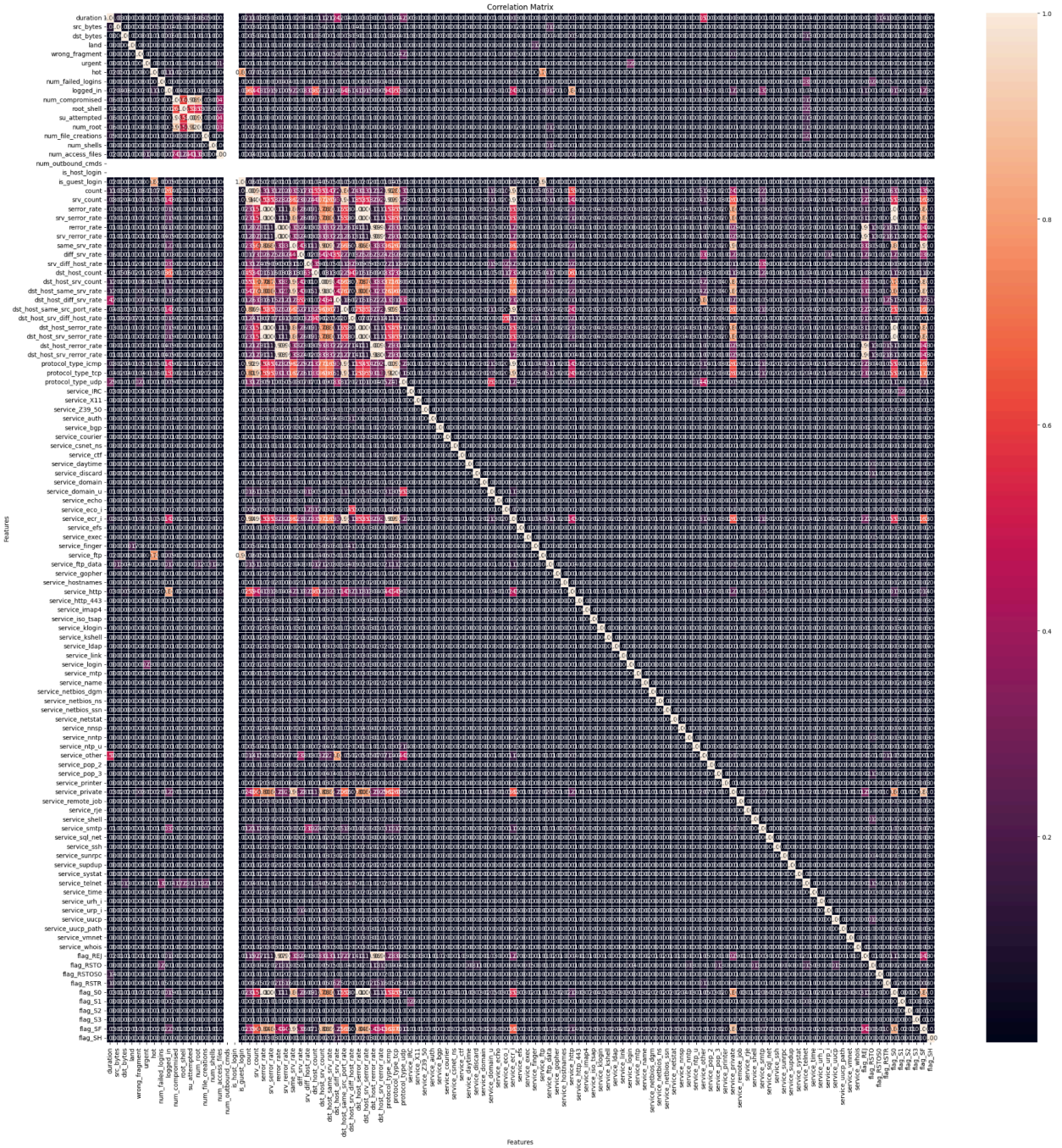
**Data Preprocessing:**
While the entire dataset provides valuable information to improve the accuracy of the models, it is too large and slows down training significantly. To improve training times, the dataset will be reduced to 10% of the size while preserving the class proportions through stratification based on the labels. This dataset contains all the data needed for classification, but currently stores the protocol_type, service, flag, and labels as strings. These values will be one-hot encoded rather than being preserved as categorical variables to prevent ordinal implications in the models. This data also presents a multi-classification problem rather than a binary classification. To transform this into a binary classification problem, I will use two class labels of normal and attack and convert the current dataset labels accordingly. All non-normal labels will be considered an attack for training and testing purposes. Now that the data has been transformed, I prepare this dataset for training by separating the X dimensions and the Y labels.

**Class Balance Inspection:**
In the KDDCup99 dataset, there is an imbalance in the number of normal connections and attack connections with a bias toward attack connection data. This bias is then visualized. To overcome this imbalance in data, either the normal data could be over-sampled or the attack data could be under-sampled. While these options would balance the dataset, the data mining techniques used in this project are able to effectively work with small imbalances in data and neither of these approaches are necessary to achieve high model performance. In the real world, it is likely that an imbalance in the data would exist as well, and thus the small imbalance has been disregarded. To ensure that the classes are proportionally utilized during training and testing, a stratified sampling approach will be used during the k-fold cross validation.

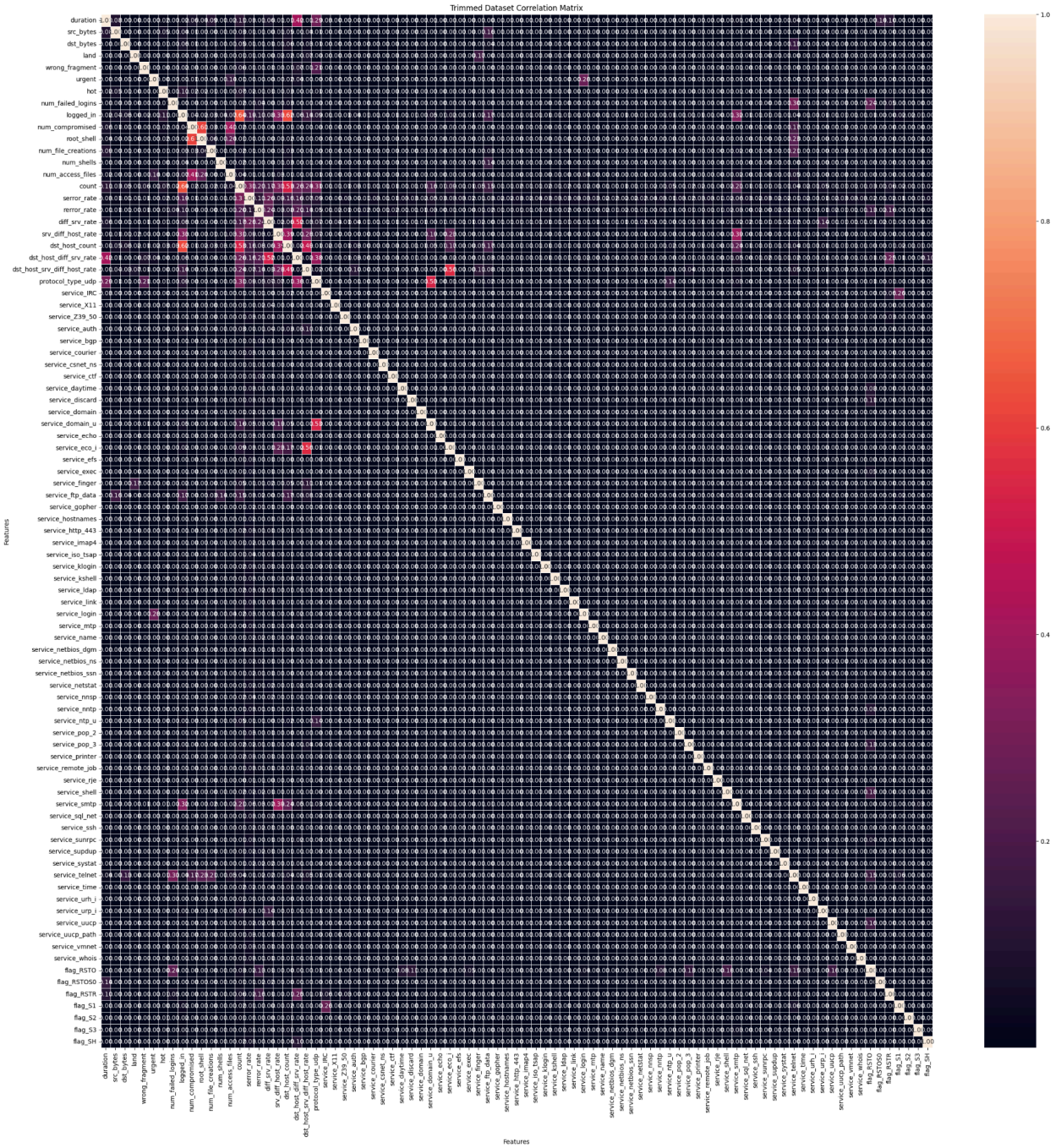



**Dimensional Analysis:**

To visualize the correlation of the variables, a correlation matrix is used. This will help to identify data for dimensional reduction.




Variables where the value does not change, resulting in white lines within the correlation matrix, are dropped because they provide no value to the models. Variables with highly correlated values are also dropped to prevent multicollinearity. The new correlation matrix is created with the reduced dimensionality dataset.




**Performance Metrics Definition:**

The following performance metrics are calculated by the getMetrics function: True Positive, True Negative, False Positive, False Negative, Positive, Negative, True Positive Rate, True Negative Rate, False Positive Rate, False Negative Rate, Recall/ Sensitivity, Precision, F1 Score, Accuracy, Error Rate, Balanced Accuracy, True Skill Statistics, Heidke Skill Score, Specificity, Negative Predictive Value, False Discovery Rate, Brier Score, Brier Skill Score, and the Area Under ROC Curve. Each performance metric is manually calculated using only the True

Positive, True Negative, False Positive, and False Negative values from the confusion matrix, except for the Brier Score and the AUC which utilize a library for calculation. The getMetrics function will be called to evaluate the performance of each model after training in the k-fold cross validation.

**Model Implementation and 10-Fold Cross Validation:**
I have implemented the Random Forest Classifier using the Scikit-learn library. I have selected and implemented the Convolutional 1D network using the Keras library. I have selected and implemented the naive Bayes algorithm using the Scikit-learn library. This implementation comes as a result of previous attempts with an SVM taking multiple hours to train and being unable to complete training effectively. The getMetrics function I have written is used to calculate the statistics discussed in the "Evaluating Classifiers" module for each run of the 10 folds and also for the average of the 10 folds. Each of the three models identified are run on the same dataset. The experimental results for each of the three classification methods used in this project are captured in their respective data frames for comparison.

**Performance Metric Evaluation:**
The statistics calculated have been stored in the data frames for each fold and respective model, and the average for each is now calculated. The counts of positive and negative identifications should be whole numbers represented as integers, and are updated as such. The three sections below display the results of the metrics in a tabular format listing all details for each fold and the average for each model.

> **Random Forest Classifier Performance Metrics:**
> This section provides a tabular view of the metrics from the Random Forest Classifier.
>
> **Convolutional 1D Network Performance Metrics:**
> This section provides a tabular view of the metrics from the Convolutional 1D network.
>
> **Naive Bayes Algorithm Performance Metrics:**
> This section provides a tabular view of the metrics from the naive Bayes algorithm.

**Identification of Best Performance:**
The best performance metrics are identified as well as the model from which these results originated. Some metrics are optimized by selecting the maximum score, and others are optimized by selecting the minimum score. The results for all metrics are displayed for easy evaluation of the best models. A discussion of the models and results is also provided.

**Conclusion:**

Based on the average results from the 10-fold cross validation testing, it is clear that the Random Forest Classifier stands out as the best algorithm due to its superior performance. The Random Forest Classifier was able to achieve the highest true positive rate, recall/sensitivity, precision, F1 score, accuracy, balanced accuracy, true skill statistics, Heidke skill score, negative predictive value, Brier skill score, and area under the ROC curve. The Random Forest Classifier was also able to achieve the lowest false negative rate, error rate, and Brier score. Based on these metrics, the Random Forest Classifier was able to outperform both the Convolutional 1D network and the naive Bayes algorithm. Within these categories, the precision, recall, F1 score, and balanced accuracy stand out by offering deeper insights into the strength of the model. The Random Forest Classifier achieved a precision of 99.89%, a recall of 99.06%, an F1 score of 99.47%, and a balanced accuracy of 99.32%. The precision of the model indicates its ability to accurately predict positive results. With this dataset, this means that fewer false positives will be generated, and thus network administrators would be more well-equipped to respond effectively and take the detected attacks seriously. If the precision were too low, network administrators may not respond with urgency to the attacks because of a high likelihood that it is a false alarm. The recall of the model indicates its ability to identify attacks and not miss threats to the network. Since the cost of missing an attack could be high, the recall gives valuable insight into the ability of the model to accurately detect if an attack is occurring within the network. The F1 score is the harmonic mean of precision and recall and provides an effective way to determine if the model is a good fit when both false positives and false negatives have a high cost. In this scenario, false positives cause network administrators to become overburdened with false alerts, resulting in a loss in ability to respond to real threats and a decreased trust in the system. False negatives have a high cost as well, making the network susceptible to exploitation and leaving critical resources vulnerable to attack. The F1 score provides valuable insight into the ability of the model to perform while both of these concerns are present. The balanced accuracy provides information about the model's ability to correctly classify data into normal and attack traffic. While the accuracy of this model is also high, the balanced accuracy provides a better understanding of the accuracy because an imbalance in the classes exists within the data, which the balanced accuracy can adjust for. The balanced accuracy of this model shows that the Random Forest Classifier is able to correctly identify attacks in the overwhelming majority of cases. These four metrics can provide a good measure of the model's performance, and help to demonstrate why the Random Forest Classifier is the best choice of the three models evaluated.

While the Random Forest Classifier performed the best, the Convolutional 1D network also performed incredibly well. For most metrics, the performance of the Random Forest Classifier and the Convolutional 1D network was within less than 1% difference. The Convolutional 1D network achieved a similar precision of 99.79%, a recall of 98.68%, an F1 score of 99.21%, and a balanced accuracy of 98.92%. With this dataset and the current hyperparameters for these models, the Random Forest Classifier was able to perform slightly better, but it is possible that both of these models could perform well predicting additional data from this network. Further evaluation could be done to test how well each model can generalize

to the larger population, and hyperparameter optimization could enable either of these models to improve performance. These models both performed well due to the extensive features present within the data and their ability to detect complex patterns. The Random Forest Classifier and the Convolutional 1D network could be used together within the network to produce a more robust attack detection system.

The naive Bayes algorithm performed comparatively worse than the other two models, with a precision of 58.93%, a recall of 17.79%, an F1 score of 20.56%, and a balanced accuracy of 58.72%. This model exhibited a significantly faster training and prediction time as compared to the other two models, but struggled with certain folds of the data. This could be caused in part by the examples available in the training data within each fold and the assumption that all features are independent, which is not necessarily true with this dataset. While the correlation of the data was minimized, some network conditions could be dependent on each other. The naive Bayes model likely struggles to understand the complex patterns within the dataset due to this assumption of conditional independence.

**Github Project:**

The code created for this project can be accessed using the following link:
https://github.com/joshuakobuskie/FA24-CS634-101-Final

This repository belongs to my GitHub account joshuakobuskie, which is associated with both my NJIT and personal emails. My emails are jsk47@njit.edu and joshkobuskie@gmail.com respectively.

**Tutorial:**

Open the command line and navigate to the directory in which the file is saved. In this example, the file is saved in kobuskie_joshua_finalproject and the file is named kobuskie_joshua_finalproject.py.

You should see something similar to the following:

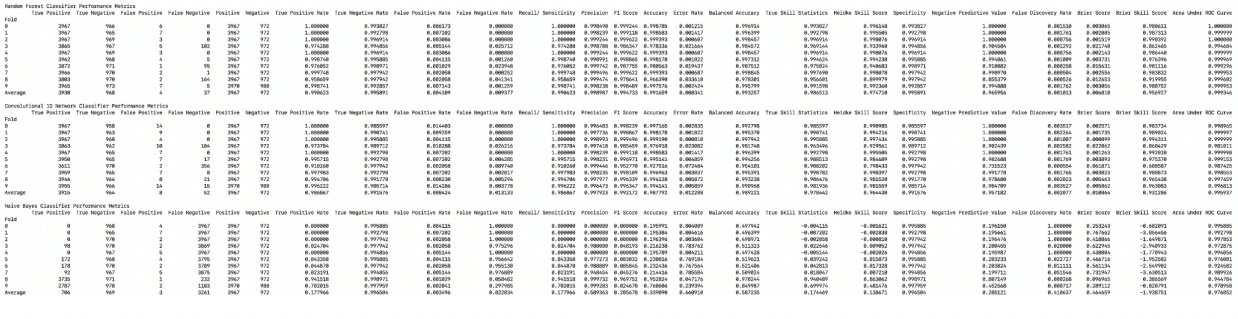
```
[joshuakobuskie@JSK64 ~ % cd Desktop
[joshuakobuskie@JSK64 Desktop % cd CS634
[joshuakobuskie@JSK64 CS634 % cd kobuskie_joshua_finalproject
 joshuakobuskie@JSK64 kobuskie_joshua_finalproject %
```

Run the following command:
python3 kobuskie_joshua_finalproject.py

```
[joshuakobuskie@JSK64 ~ % cd Desktop
[joshuakobuskie@JSK64 Desktop % cd CS634
[joshuakobuskie@JSK64 CS634 % cd kobuskie_joshua_finalproject
[joshuakobuskie@JSK64 kobuskie_joshua_finalproject % python3 kobuskie_joshua_finalproject.py
```

Allow the program to run, which can take up to 2 minutes. At the completion of the program's execution, observe the output generated that includes the tabular performance metrics for each of the three models during 10-fold cross validation, as well as the best model and best average value for each of the performance metrics. Due to the width of the tables, the results may be difficult to view from the command line. Adjusting your zoom settings on the command line enables the table to fit the content and formatting properly. Reviewing the Jupyter Notebook enables closer inspection of these tables and provides additional graphical representations of the data during preprocessing.

```
joshuakobuskie@JSK64 ~ % cd Desktop
joshuakobuskie@JSK64 Desktop % cd CS634
joshuakobuskie@JSK64 CS634 % cd kobuskie_joshua_finalproject
joshuakobuskie@JSK64 kobuskie_joshua_finalproject % python3 kobuskie_joshua_finalproject.py
CS634–101 Data Mining: Binary Classification Final Project – Joshua Kobuskie

#################### Checking for Data Imbalance ####################
Attack Entries: 39673, Normal Entries: 9728, Total Entries: 49401
Attack Entries: 80.31%, Normal Entries: 19.69%

2024-11-11 10:41:58.741564: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Fold 1
Testing on Normal values in range(0, 972)
Testing on Attack values in range(9728, 13695)
Epoch 1/2
278/278 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step – accuracy: 0.9248 – loss: 0.3578 – val_accuracy: 0.9919 – val_loss: 0.0389
Epoch 2/2
278/278 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step – accuracy: 0.9898 – loss: 0.0360 – val_accuracy: 0.9936 – val_loss: 0.0238
155/155 ━━━━━━━━━━━━━━━━━━━━ 0s 762us/step
155/155 ━━━━━━━━━━━━━━━━━━━━ 0s 468us/step

Fold 2
Testing on Normal values in range(972, 1944)
Testing on Attack values in range(13695, 17662)
Epoch 1/2
278/278 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step – accuracy: 0.9358 – loss: 0.3342 – val_accuracy: 0.9918 – val_loss: 0.0305
Epoch 2/2
278/278 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step – accuracy: 0.9914 – loss: 0.0346 – val_accuracy: 0.9951 – val_loss: 0.0178
155/155 ━━━━━━━━━━━━━━━━━━━━ 0s 765us/step
155/155 ━━━━━━━━━━━━━━━━━━━━ 0s 464us/step

Fold 3
Testing on Normal values in range(1944, 2916)
Testing on Attack values in range(17662, 21629)
Epoch 1/2
278/278 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step – accuracy: 0.9130 – loss: 0.3635 – val_accuracy: 0.9916 – val_loss: 0.0359
Epoch 2/2
278/278 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step – accuracy: 0.9896 – loss: 0.0361 – val_accuracy: 0.9938 – val_loss: 0.0237
155/155 ━━━━━━━━━━━━━━━━━━━━ 0s 752us/step
155/155 ━━━━━━━━━━━━━━━━━━━━ 0s 473us/step

Fold 4
Testing on Normal values in range(2916, 3888)
Testing on Attack values in range(21629, 25596)
Epoch 1/2
278/278 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step – accuracy: 0.9430 – loss: 0.3262 – val_accuracy: 0.9952 – val_loss: 0.0306
Epoch 2/2
278/278 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step – accuracy: 0.9952 – loss: 0.0258 – val_accuracy: 0.9957 – val_loss: 0.0193
155/155 ━━━━━━━━━━━━━━━━━━━━ 0s 739us/step
155/155 ━━━━━━━━━━━━━━━━━━━━ 0s 467us/step

Fold 5
Testing on Normal values in range(3888, 4860)
Testing on Attack values in range(25596, 29563)
Epoch 1/2
278/278 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step – accuracy: 0.9141 – loss: 0.3688 – val_accuracy: 0.9908 – val_loss: 0.0417
Epoch 2/2
278/278 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step – accuracy: 0.9897 – loss: 0.0400 – val_accuracy: 0.9944 – val_loss: 0.0227
155/155 ━━━━━━━━━━━━━━━━━━━━ 0s 811us/step
155/155 ━━━━━━━━━━━━━━━━━━━━ 0s 564us/step

Fold 6
Testing on Normal values in range(4860, 5832)
Testing on Attack values in range(29563, 33530)
Epoch 1/2
278/278 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step – accuracy: 0.9476 – loss: 0.3505 – val_accuracy: 0.9921 – val_loss: 0.0388
Epoch 2/2
278/278 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step – accuracy: 0.9895 – loss: 0.0423 – val_accuracy: 0.9946 – val_loss: 0.0222
155/155 ━━━━━━━━━━━━━━━━━━━━ 0s 774us/step
155/155 ━━━━━━━━━━━━━━━━━━━━ 0s 537us/step

Fold 7
Testing on Normal values in range(5832, 6804)
Testing on Attack values in range(33530, 37497)
Epoch 1/2
278/278 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step – accuracy: 0.9549 – loss: 0.3819 – val_accuracy: 0.9956 – val_loss: 0.0289
Epoch 2/2
278/278 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step – accuracy: 0.9949 – loss: 0.0248 – val_accuracy: 0.9974 – val_loss: 0.0209
155/155 ━━━━━━━━━━━━━━━━━━━━ 0s 784us/step
155/155 ━━━━━━━━━━━━━━━━━━━━ 0s 481us/step

Fold 8
Testing on Normal values in range(6804, 7776)
Testing on Attack values in range(37497, 41464)
Epoch 1/2
278/278 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step – accuracy: 0.9239 – loss: 0.3710 – val_accuracy: 0.9888 – val_loss: 0.0417
Epoch 2/2
278/278 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step – accuracy: 0.9915 – loss: 0.0291 – val_accuracy: 0.9926 – val_loss: 0.0275
155/155 ━━━━━━━━━━━━━━━━━━━━ 0s 823us/step
155/155 ━━━━━━━━━━━━━━━━━━━━ 0s 485us/step

Fold 9
Testing on Normal values in range(7776, 8748)
Testing on Attack values in range(41464, 45431)
Epoch 1/2
278/278 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step – accuracy: 0.9467 – loss: 0.3455 – val_accuracy: 0.9882 – val_loss: 0.0379
Epoch 2/2
278/278 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step – accuracy: 0.9911 – loss: 0.0328 – val_accuracy: 0.9922 – val_loss: 0.0259
155/155 ━━━━━━━━━━━━━━━━━━━━ 0s 792us/step
155/155 ━━━━━━━━━━━━━━━━━━━━ 0s 482us/step

Fold 10
Testing on Normal values in range(8748, 9728)
Testing on Attack values in range(45431, 49401)
Epoch 1/2
278/278 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step – accuracy: 0.9453 – loss: 0.3192 – val_accuracy: 0.9867 – val_loss: 0.0516
Epoch 2/2
278/278 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step – accuracy: 0.9934 – loss: 0.0298 – val_accuracy: 0.9929 – val_loss: 0.0389
155/155 ━━━━━━━━━━━━━━━━━━━━ 0s 789us/step
155/155 ━━━━━━━━━━━━━━━━━━━━ 0s 474us/step
```



Random Forest Classifier Performance Metrics

| Fold | True Positive | True Negative | False Positive | False Negative | Positive | Negative | True Positive Rate | True Negative Rate | False Positive Rate | False Negative Rate | Recall/ Sensitivity | Precision | F1 Score | Accuracy | Error Rate | Balanced Accuracy | True Skill Statistics | Heidke Skill Score | Specificity | Negative Predictive Value | False Discovery Rate | Brier Score | Brier Skill Score | Area Under ROC Curve |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3967 | 966 | 6 | 0 | 3967 | 972 | 1.000000 | 0.993827 | 0.006173 | 0.000000 | 1.000000 | 0.996990 | 0.998486 | 0.998725 | 0.001275 | 0.996914 | 0.993827 | 0.996148 | 0.993827 | 1.000000 | 0.003150 | 0.003065 | 0.988611 | 1.000000 |
| 1 | 3967 | 966 | 7 | 0 | 3967 | 972 | 1.000000 | 0.992798 | 0.007202 | 0.000000 | 1.000000 | 0.998239 | 0.999118 | 0.998583 | 0.001417 | 0.996399 | 0.992798 | 0.995985 | 0.992798 | 1.000000 | 0.001761 | 0.002085 | 0.987353 | 0.999999 |
| 2 | 3967 | 969 | 3 | 0 | 3967 | 972 | 1.000000 | 0.996914 | 0.003086 | 0.000000 | 1.000000 | 0.999244 | 0.999622 | 0.999393 | 0.000607 | 0.998457 | 0.996914 | 0.998876 | 0.996914 | 1.000000 | 0.000756 | 0.001519 | 0.990392 | 1.000000 |
| 3 | 3865 | 967 | 5 | 102 | 3967 | 972 | 0.974288 | 0.994854 | 0.005144 | 0.025712 | 0.974288 | 0.998708 | 0.986347 | 0.978334 | 0.021664 | 0.984572 | 0.969144 | 0.933968 | 0.994856 | 0.904584 | 0.001292 | 0.021748 | 0.862460 | 0.994684 |
| 4 | 3967 | 969 | 3 | 0 | 3967 | 972 | 1.000000 | 0.996914 | 0.003086 | 0.000000 | 1.000000 | 0.999244 | 0.999622 | 0.999393 | 0.000607 | 0.998457 | 0.996914 | 0.998876 | 0.996914 | 1.000000 | 0.000756 | 0.002143 | 0.986448 | 0.999999 |
| 5 | 3962 | 968 | 4 | 5 | 3967 | 972 | 0.998740 | 0.995885 | 0.004115 | 0.001260 | 0.998740 | 0.998991 | 0.998865 | 0.998178 | 0.001822 | 0.997312 | 0.994624 | 0.994238 | 0.995885 | 0.995861 | 0.001009 | 0.003731 | 0.976396 | 0.999969 |
| 6 | 3872 | 971 | 1 | 95 | 3967 | 972 | 0.976052 | 0.998971 | 0.001029 | 0.023948 | 0.976052 | 0.999742 | 0.987755 | 0.980563 | 0.019437 | 0.987512 | 0.975024 | 0.960683 | 0.998971 | 0.910882 | 0.000258 | 0.010431 | 0.903116 | 0.999216 |
| 7 | 3966 | 970 | 2 | 1 | 3967 | 972 | 0.999748 | 0.997942 | 0.002058 | 0.000252 | 0.999748 | 0.999496 | 0.999622 | 0.999393 | 0.000607 | 0.998845 | 0.997690 | 0.998078 | 0.997942 | 0.999738 | 0.000504 | 0.002556 | 0.983832 | 0.999953 |
| 8 | 3883 | 970 | 2 | 164 | 3967 | 972 | 0.958659 | 0.997942 | 0.002058 | 0.041341 | 0.958659 | 0.999479 | 0.978644 | 0.966398 | 0.033618 | 0.978301 | 0.956601 | 0.899729 | 0.997942 | 0.855179 | 0.000526 | 0.012653 | 0.919958 | 0.999682 |
| 9 | 3965 | 973 | 7 | 0 | 3970 | 980 | 0.998741 | 0.992857 | 0.007143 | 0.001259 | 0.998741 | 0.998238 | 0.998489 | 0.997576 | 0.002424 | 0.995799 | 0.991598 | 0.992368 | 0.992857 | 0.998488 | 0.001762 | 0.003856 | 0.988753 | 0.999953 |
| Average | 3938 | 968 | 4 | 37 | 3967 | 972 | 0.990619 | 0.996091 | 0.003909 | 0.009377 | 0.990619 | 0.998987 | 0.994733 | 0.991689 | 0.008311 | 0.993357 | 0.986713 | 0.973718 | 0.996091 | 0.963956 | 0.001813 | 0.006018 | 0.956927 | 0.999346 |

Convolutional 1D Network Classifier Performance Metrics

| Fold | True Positive | True Negative | False Positive | False Negative | Positive | Negative | True Positive Rate | True Negative Rate | False Positive Rate | False Negative Rate | Recall/ Sensitivity | Precision | F1 Score | Accuracy | Error Rate | Balanced Accuracy | True Skill Statistics | Heidke Skill Score | Specificity | Negative Predictive Value | False Discovery Rate | Brier Score | Brier Skill Score | Area Under ROC Curve |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3967 | 958 | 14 | 0 | 3967 | 972 | 1.000000 | 0.985597 | 0.014403 | 0.000000 | 1.000000 | 0.996483 | 0.998239 | 0.997165 | 0.002835 | 0.992798 | 0.985597 | 0.990985 | 0.985597 | 1.000000 | 0.003517 | 0.002571 | 0.983734 | 0.999965 |
| 1 | 3967 | 963 | 9 | 0 | 3967 | 972 | 1.000000 | 0.990741 | 0.009259 | 0.000000 | 1.000000 | 0.997736 | 0.998867 | 0.998178 | 0.001822 | 0.995370 | 0.990741 | 0.994216 | 0.990741 | 1.000000 | 0.002264 | 0.001735 | 0.989034 | 0.999997 |
| 2 | 3967 | 968 | 4 | 0 | 3967 | 972 | 1.000000 | 0.995885 | 0.004115 | 0.000000 | 1.000000 | 0.998993 | 0.999496 | 0.999393 | 0.000607 | 0.997942 | 0.995885 | 0.997434 | 0.995885 | 1.000000 | 0.001007 | 0.000979 | 0.994311 | 0.999999 |
| 3 | 3863 | 962 | 10 | 104 | 3967 | 972 | 0.973784 | 0.989712 | 0.010288 | 0.026216 | 0.973784 | 0.997418 | 0.985459 | 0.976958 | 0.023042 | 0.981748 | 0.963496 | 0.929561 | 0.989712 | 0.902439 | 0.002582 | 0.022862 | 0.868429 | 0.983811 |
| 4 | 3967 | 965 | 7 | 0 | 3967 | 972 | 1.000000 | 0.992798 | 0.007202 | 0.000000 | 1.000000 | 0.998239 | 0.999118 | 0.998583 | 0.001417 | 0.996399 | 0.992798 | 0.995985 | 0.992798 | 1.000000 | 0.001761 | 0.001263 | 0.992858 | 0.999998 |
| 5 | 3950 | 965 | 7 | 17 | 3967 | 972 | 0.995716 | 0.992798 | 0.007202 | 0.004285 | 0.995716 | 0.998231 | 0.996971 | 0.995141 | 0.004859 | 0.994256 | 0.988513 | 0.984689 | 0.992798 | 0.982688 | 0.001769 | 0.003893 | 0.975378 | 0.999153 |
| 6 | 3611 | 970 | 2 | 356 | 3967 | 972 | 0.910168 | 0.997942 | 0.002058 | 0.089748 | 0.910168 | 0.999446 | 0.952778 | 0.927556 | 0.072444 | 0.954281 | 0.908382 | 0.796433 | 0.997942 | 0.731033 | 0.000554 | 0.063871 | 0.680587 | 0.987425 |
| 7 | 3969 | 966 | 7 | 8 | 3967 | 972 | 0.997983 | 0.992798 | 0.007202 | 0.002017 | 0.997983 | 0.998236 | 0.998109 | 0.996963 | 0.003037 | 0.995391 | 0.990782 | 0.990397 | 0.992798 | 0.991778 | 0.001766 | 0.003023 | 0.988873 | 0.998503 |
| 8 | 3946 | 964 | 8 | 21 | 3967 | 972 | 0.994706 | 0.991770 | 0.008230 | 0.005294 | 0.994706 | 0.997977 | 0.996339 | 0.994128 | 0.005872 | 0.993238 | 0.986476 | 0.981028 | 0.991770 | 0.978680 | 0.002023 | 0.005443 | 0.965438 | 0.997409 |
| 9 | 3955 | 966 | 14 | 15 | 3970 | 980 | 0.996222 | 0.985714 | 0.014286 | 0.003778 | 0.996222 | 0.996473 | 0.996347 | 0.994141 | 0.005859 | 0.990968 | 0.981936 | 0.983899 | 0.985714 | 0.984709 | 0.003527 | 0.005862 | 0.963083 | 0.996813 |
| Average | 3913 | 966 | 8 | 52 | 3967 | 972 | 0.986867 | 0.990654 | 0.009346 | 0.013133 | 0.986867 | 0.997923 | 0.992372 | 0.987792 | 0.012208 | 0.988221 | 0.978442 | 0.964438 | 0.990654 | 0.961872 | 0.002077 | 0.018864 | 0.921286 | 0.998937 |

Naive Bayes Classifier Performance Metrics

| Fold | True Positive | True Negative | False Positive | False Negative | Positive | Negative | True Positive Rate | True Negative Rate | False Positive Rate | False Negative Rate | Recall/ Sensitivity | Precision | F1 Score | Accuracy | Error Rate | Balanced Accuracy | True Skill Statistics | Heidke Skill Score | Specificity | Negative Predictive Value | False Discovery Rate | Brier Score | Brier Skill Score | Area Under ROC Curve |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 968 | 4 | 3967 | 3967 | 972 | 0.000000 | 0.995885 | 0.004115 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.196991 | 0.803009 | 0.497942 | -0.004115 | -0.003421 | 0.995885 | 0.196268 | 1.000000 | 0.203243 | -0.443891 | 0.995855 |
| 1 | 0 | 966 | 7 | 3967 | 3967 | 972 | 0.000000 | 0.992798 | 0.007202 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.196284 | 0.803716 | 0.496399 | -0.007202 | -0.005838 | 0.992798 | 0.196461 | 1.000000 | 0.207662 | -0.804406 | 0.992798 |
| 2 | 0 | 970 | 2 | 3967 | 3967 | 972 | 0.000000 | 0.997942 | 0.002058 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.196296 | 0.803704 | 0.498971 | -0.002058 | -0.000358 | 0.997942 | 0.196476 | 1.000000 | 0.168846 | -1.649815 | 0.997853 |
| 3 | 98 | 970 | 2 | 3869 | 3967 | 972 | 0.024784 | 0.997942 | 0.002058 | 0.975216 | 0.024784 | 0.980000 | 0.040293 | 0.216238 | 0.783762 | 0.511323 | 0.022666 | 0.009852 | 0.997942 | 0.200495 | 0.020000 | 0.622945 | -2.948933 | 0.972875 |
| 4 | 0 | 967 | 5 | 3967 | 3967 | 972 | 0.000000 | 0.994854 | 0.005144 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.196789 | 0.803211 | 0.497428 | -0.005144 | -0.002826 | 0.994856 | 0.196967 | 1.000000 | 0.438894 | -1.728963 | 0.994854 |
| 5 | 172 | 968 | 4 | 3795 | 3967 | 972 | 0.043358 | 0.995885 | 0.004115 | 0.956642 | 0.043358 | 0.977273 | 0.083032 | 0.238816 | 0.761184 | 0.519621 | 0.039242 | 0.018873 | 0.995885 | 0.203233 | 0.022727 | 0.466716 | -1.952082 | 0.976801 |
| 6 | 178 | 970 | 2 | 3789 | 3967 | 972 | 0.044878 | 0.997942 | 0.002058 | 0.955130 | 0.044878 | 0.988889 | 0.085943 | 0.232434 | 0.767566 | 0.521446 | 0.042813 | 0.017328 | 0.997942 | 0.203824 | 0.011111 | 0.561234 | -2.547992 | 0.976982 |
| 7 | 92 | 967 | 5 | 3875 | 3967 | 972 | 0.023191 | 0.994854 | 0.005144 | 0.976809 | 0.023191 | 0.948454 | 0.045276 | 0.214434 | 0.785566 | 0.509024 | 0.018047 | 0.007218 | 0.994856 | 0.199711 | 0.051546 | 0.731947 | -3.638533 | 0.989926 |
| 8 | 3735 | 971 | 1 | 232 | 3967 | 972 | 0.941518 | 0.998971 | 0.001029 | 0.058482 | 0.941518 | 0.999732 | 0.969752 | 0.951824 | 0.048176 | 0.970244 | 0.940489 | 0.863842 | 0.998971 | 0.807149 | 0.000268 | 0.090765 | 0.860669 | 0.964784 |
| 9 | 2787 | 970 | 2 | 1183 | 3970 | 980 | 0.702015 | 0.997959 | 0.002041 | 0.297985 | 0.702015 | 0.999283 | 0.824678 | 0.768004 | 0.231996 | 0.849987 | 0.699974 | 0.485476 | 0.997959 | 0.452568 | 0.000717 | 0.209112 | -0.828791 | 0.978958 |
| Average | 706 | 969 | 3 | 3261 | 3967 | 972 | 0.177966 | 0.996904 | 0.003096 | 0.822034 | 0.177966 | 0.989363 | 0.285678 | 0.339098 | 0.660902 | 0.587336 | 0.174469 | 0.138671 | 0.996904 | 0.285121 | 0.010637 | 0.464499 | -1.958751 | 0.976802 |

Best True Positive: Naive Bayes Algorithm = 706
Best True Negative: 1D Convolutional Network = 964
Best False Positive: Naive Bayes Algorithm = 3
Best False Negative: Random Forest Classifier = 37
Best Positive: Random Forest Classifier = 3967
Best Negative: Random Forest Classifier = 972
Best True Positive Rate: Random Forest Classifier = 0.9906227319974169
Best True Negative Rate: Naive Bayes Algorithm = 0.996503737297388
Best False Positive Rate: Naive Bayes Algorithm = 0.0034962627026119105
Best False Negative Rate: Random Forest Classifier = 0.009377268002583022
Best Recall/ Sensitivity: Random Forest Classifier = 0.9906227319974169
Best Precision: Random Forest Classifier = 0.9989866268653733
Best F1 Score: Random Forest Classifier = 0.9947326545390945
Best Accuracy: Random Forest Classifier = 0.9916587703313761
Best Error Rate: Random Forest Classifier = 0.008341229668623879
Best Balanced Accuracy: Random Forest Classifier = 0.9932566922773681
Best True Skill Statistics: Random Forest Classifier = 0.9865133845547363
Best Heidke Skill Score: Random Forest Classifier = 0.9747104939355259
Best Specificity: Naive Bayes Algorithm = 0.996503737297388
Best Negative Predictive Value: Random Forest Classifier = 0.9659563626242959
Best False Discovery Rate: Random Forest Classifier = 0.0010133731346267595
Best Brier Score: Random Forest Classifier = 0.006809912559897416
Best Brier Skill Score: Random Forest Classifier = 0.9569271570785339
Best Area Under ROC Curve: Random Forest Classifier = 0.9993456331922077