Exploration into Arthritis Classification Efficacy of Various ML Algorithms using 2018 BRFSS Survey Data prepared by CDC

CS699 Term Project

Cody Thayer | 8.20.2023

Objective

The goal of this project assignment is to give students an opportunity to build and test classifier models using real-world data. The data to be used is a part of the 2018 BRFSS Survey Data prepared by CDC. You must go to the website from which the dataset was downloaded and read the description of the dataset. The website is:

https://www.cdc.gov/brfss/annual_data/annual_2018.html

The primary objective of this project is for you to (1) research attribute selection methods, (2) apply attribute selection methods to your data, and (3) successfully apply the classification models you have learned in class. The secondary objective of this project is to expose you to the complexities of working with messy data. This is very common in many academic research and corporate research work. The intention is for you to experience some of the real-world challenges. However, please do not spend too much time on preprocessing and attribute selection. It is not expected for you to solve all of the real-world challenges in this project.

Additional Resources

Python Notebook:
https://colab.research.google.com/drive/1tQ-wMfn7hGRHrPTdgnlcIwx2rbwElqxe#scrollTo=CZyR5yqmKXf3

- Python notebook in google collab with notes on how to run each feature selection and classification model used in this paper.

Feature Summary Excel file:
https://docs.google.com/spreadsheets/d/1MSb4dchlg3C6rQuwD025oBFvJj-RbyGIidXaikzBBno/edit#gid=1983465839

- Feature Summary
- 5 Features Selections Subset
- Feature Correlation Matrix
- Feature Gini Importance Values

Preprocessing

I did my attribute selection before I processed my data. Attribute selection > Processing > Data Split. I hope this isn't an issue. I just found this a lot easier to do it in this order because some preprocessing steps were unique to the attribute selection methods I decided to use. So processing my data before attribute selection didn't make sense, especially in python. This just means that my yourname-project-initial.csv file does not have all 108 features in it. Just the 31 I used for Feature Set 1. If this is a big deal let me know and I can adjust the file.

Normalization
For all 5 of my feature sets, I normalized all attributes to land between 0 and 1. Even though all of the dataset's features were numeric, and many were categorical, they all had very different ranges. Some were binary, others ranged from 0-7, others 0-3, etc. Additionally, features like height and weight were in different units, so normalizing all of my attributes allowed me to retain variance, while placing all features on the same scale. Furthermore, normalization helped me:

1. **Reduce Bias**: Normalization helps prevent certain features from dominating the learning process simply because of their larger scales or ranges. By bringing all my features to a similar scale, I decreased bias towards features with higher magnitudes. This ensures that each feature contributes proportionally to each of my model learning processes.
2. **Improve Data Interpretability**: Having features with the same magnitude enabled you to compare feature coefficients or weights to understand their relative influence on the classification decision. Specifically, I used my Random Forest model's to assign gini importance scores to each of my input features. This allowed me to identify relevant features for future feature sets.
3. **Distance-Based Algorithms:** Many classification algorithms rely heavily on distance calculations, such as the K-nearest neighbors (KNN) I used in this project. Normalizing the dataset ensures that the distance metrics are not dominated by a single feature. If some features have larger scales than others, they may disproportionately affect the distance calculations and lead to biased results.

Missing Data Handling
Most of the missing data was replaced with the median value of the feature. Missing data includes everything from blanks ("?"), refused to answer, and. I chose to fill the missing data with median over mean as median is a lot more robust to outliers than mean.

Various other python data wrangling techniques were used for organizing and manipulating the data. For example, type adjustments for the data was done frequently, as some of the data was found as strings, while others as integers. Additionally, I renamed all the column names to better identify features. For example, changing *htm4* to *height*, made it a lot clearer what the feature was. This allowed me to better explore and interpret the dataset, and understand correlations between features. These small data processing steps are pretty unnoteworthy, but are laid out in the accompanying python notebook file. I found that data encoding techniques were not necessary with this dataset, as the CDC data was already numerically encoded. I refrained from doing any using any instance reduction techniques (except for attribute reduction of course) as I wanted to retain as much data as possible in hopes it would provide my models with more data to train on.

Feature Selection

All Feature Sets used for each of the 5 attribute selection methods are shown in full in the accompanying excel file.

Feature Set 1 (F1): ***Domain Knowledge***
For Feature Set 1, I thoroughly examined the 108 features present in the 2018 BRFSS Survey Data and carefully reviewed the variable descriptions provided in the LLCP 2018 Codebook Report. I analyzed the values associated with each variable and summarized my findings. After assessing all 108 variables, I identified several instances of seemingly irrelevant variables such as *interview yea*r and *file month*, as well as redundant variables like *x.age.g, x.age80, x.ageg5yr, and x.ageg5yr*, which are all variations of the age attribute.

To streamline the dataset and eliminate redundancy amongst features, I categorized all 108 variables into 35 distinct categories. Within each category, I selected a summary variable that would effectively "represent" that entire category. The objective was to reduce the overlap of variables, avoiding a situation where I would have a feature set with five age-related features. This significantly reduced a feature set with highly correlated features. For example, for the Lung category there were five features: *chccopd1, asthma3, x.asthms1, x.ltasth1*, and *x.casthm1.* I chose x.asthms1 to represent the Lung category as it was a calculated variable that summarized the answers from many asthma features in the dataset.
I did this for each of the 35 categories. In some cases, there was a clear feature that summarizes the entire category, in others this was not possible and each category member was unique and was therefore included in the final 31 features that were used for Feature Set 1 (F1).

Feature Set 2 (F2): *F1 + Design Weights*
After reading the 2018 BRFSS Survey Overview (https://www.cdc.gov/brfss/annual_data/2018/pdf/overview-2018-508.pdf) I was intrigued by the different, seemingly complex and extremely studied weighting systems the CDC used to adjust the survey data to better match the population. The technical term for this is raking or iterative proportional fitting (IPF) and is simply a system that adjusts sample weights so that the sample more accurately reflects the true population. I wanted to see how including these weights in my feature set would impact how my models perform. To test this, I kept all my features in Feature Set 2 the same as Feature Set 1 and just added the final design weights feature, *x.llcpwt*, so I could really understand the impact this one, seemingly very important and painstaking calculated by the CDC, would impact my metrics.

From what I read, *x.llcpwt,* was the penultimate weight for the sample and that is why I chose it as the singular weight I used in my project. Looking back, it would have been interesting to try out different weights, or combinations of weight to see which had the most classification significance. This would be a place of experimentation in the future.

Feature Set 3 (F3): *Correlation*
For Feature Set 3 I did two things. I created some new features, and removed highly correlated features from F2. Below is a table of the features from the 2018 BRFSS Survey Data I combined.

Combined Features			
Original Feature	**Summarizing Feature**	**New Feature Description**	**Gini Importance Score**

x.bmi5	**bmi**	Choose to use BMI instead of separate height and weight features which have a high correlation with both *sex* and each other.	0.12578
wtkg3			
htm4			
chcocncr	**cancer2**	Combining two cancer features into one. Yes (1) indicating the individual has had cancer. No (0) if they have not had cancer.	0.00491
chcscncr			
cvdstrk3	**heart**	Combining three heart disease features into one. Yes (1) indicating the individual has complications with their heart. No (0) if they have not.	0.00760
cvdinfr4			
x.michd			
diffwalk	**phy_diff**	Combining three physical difficulty features into one. Yes (1) the individual has difficulty with the physical requirements of everyday life. No (0) if they do not.	0.0051
diffdres			
diffalon			

Table 1.0. *2018 BRFSS Survey Data features I combined in Feature Set 3.*

These features I generated ended up having very little impact on my model performance. In fact all three had the lowest gini importance scores out of all features used in this feature selection method. Which was extremely disappointing considering how annoying it was to write the code to combine these variables. These scores are listed in **Table 1.0**. BMI was the only exception, having one of the highest gini importance scores, which was not a surprise, and both height and weight both had high gini importance scores.

Additionally, I built a correlation matrix for all 32 features used in Feature Set 2 in order to determine which features were linearly correlated with each other. Ideally you want a dataset with features who are independent from each other. Building a correlation matrix allowed me to check for any highly correlated features. I did not find any extremely strong correlations, |correlation coefficient| > 0.7. However, there were a couple with correlations between 0.4 and 0.7. These highly correlated feature pairs are listed below, along with the decision made about said features.

Correlation Pair		**Decision**
employ	age	Both have high importance. Keep both.
phys_health	gen_health	Both have high importance. Keep both.
race	language	Language was really low in importance in our RF analysis. Language removed. Race was kept
sex	height	Height and weight switched for BMI.
weight	height	Height and weight switched for BMI.

If the linearly correlated features were also of low importance, one of the features in the pair was removed. Lastly, a list of correlation coefficients for each feature with the class variable *havarth* from our created correlation matrix was compared to the features with the lowest importance scores. Features that were in the bottom 10 of both were removed.

In summary, Feature Set 3 was a variation of Feature Set 2 with 3 adjustments to reduce the feature pool:

1. New features were gendered by combining like features.
2. Highly linearly correlated features were removed.

3. Features that were both low in importance and not correlated with the class variable were removed.

A full list of all features in Feature Set 3 are in tab “F3” of the accompanying excel file, along with the correlation matrix, and gini importance scores in tabs of their namesake.

Feature Set 4 (F4): *Gini Importance*
The 10 features (out of the 31 from Feature Set 2) with the highest gini importance were selected for Feature Set 4.

The scikit-learn random forest classification model I used in my project uses “gini importance” or “mean decrease impurity” for feature selection. Specifically, the ‘gini importance” is the total decrease in node impurity averaged over all trees of the forest. These importance values for each feature are weighted by the probability of a sample in the population reaching that node in the random forest. Impurity-based feature selection like this is great for low cardinality features (few unique values). Although we had some continuous features like height and weight, the majority of our features are categorical, so this method was chosen over a permutation feature selector like a backwards or forward feature selection method.I used these gini importance coefficients or importance values for each feature as a metric to determine which features in our dataset were important for classification. The 10 features chosen are listed in the accompanying excel file in the tab “F4”, along with all the gini importance scores gathered for each feature from each round of feature selection in tab “gini importance”.

Feature Set 5 (F5): *PCA*
After seeing very little model performance improvements despite hours and hours of data wrangling and model parameter tweaking, I decided I need to look for a more elaborate attribute selection method. So for Feature Set 5 I decide on Principal Component Analysis (PCA) to summarize the original 31 features of F1 into four principal components.

PCA is both a dimensionality reduction method and a feature generator (generating principal components from a set of n_features). PCA removed me and my biases from the attribute selection process (somewhat) and instead used linear algebra to determine the latent features (principle components) of the dataset.

PCA involves standardizing the features, calculating the covariance matrix to analyze the relationships between variables, performing eigendecomposition to obtain eigenvectors and eigenvalues, and selecting the principal components with the highest eigenvalues. These chosen principle components capture the most variance in the dataset.

The hope was that this technique would improve the performance of my classification models by reducing the number of features, especially the unimportant features, while retaining the most important information latent in the dataset. This reduction simplifies computations and can help avoid overfitting by removing noise or irrelevant information which can improve the accuracy of classification models.

I experimented with different amounts of principal components and I found that retaining four principal components struck a perfect balance between dimensionality reduction and preserving important information. These four new features resulted in the largest model performance improvement with models improving from 70% precision to 96% precision. Overall, PCA proved to be the most valuable attribute

selection tool I used, greatly improving all of my classification model performance in every calculated metric.

Classification Models

Model 1: *Decision Tree*

Decision trees are an extremely intuitive way to classify unknown objects. You simply ask a series of questions designed to zero in on the class of the unknown object. It is like you are playing a game of 20 questions with your friend. For example, if you wanted to build a decision tree to classify an animal your friends saw on a hike, you might build a decision tree around the questions:

1. How big was the animal?
2. Did the animal have horns?
3. Did the animal have two legs?
4. Did the animal have wings?
5. Did the animal have fur?

The structure of a decision tree is that of a tree. You start with a root, and you grow a trunk, branches and finally ending with leaves. You begin with a single question (root node). This splits the data, forming two branches that each leading to a new question (internal node/decision node) that then splits again. This continues until you have no more questions. This branch that does not lead to another question is called a leaf node.

The binary splitting makes this extremely efficient: in a well-constructed tree, each question will cut the number of options by approximately half, very quickly narrowing the options even among a large number of classes. The trick, of course, comes in deciding which questions to ask at each step. In machine learning implementations of decision trees, the questions generally take the form of axis-aligned splits in the data: that is, each node in the tree splits the data into two groups using a cutoff value within one of the features. However, there are many unique, mathematically complex ways you can split your data.

Model 2: *Gaussian Naive Bayes*

The Gaussian Naive Bayes classifier is a simple probabilistic model based on Bayes' theorem and assumes that the features are independent of each other, given the class.The classifier calculates the probability of a data instance belonging to a particular class by combining the prior probability of the class and the likelihood of the features given that class. In the Gaussian Naive Bayes, it is assumed that the features follow a Gaussian (normal) distribution.

During the training phase, the classifier estimates the mean and standard deviation of each feature for each class. These parameters are then used to compute the likelihood of observing a particular feature value given a class. During prediction, the classifier calculates the probability of each class for a given data instance and assigns the instance to the class with the highest probability.

Gaussian Naive Bayes is computationally efficient and performs well on datasets with continuous or numerical features. It works particularly well when the feature independence assumption holds reasonably well. However, if the feature independence assumption is violated, the classifier may not provide accurate results.

A more complete mathematical description of the Gaussian Naive Bayes model can be found in the accompanying python notebook file. I had trouble writing the probability equations in this document, so I used latex in the python notebook.

Model 3: *K Nearest Neighbor (KNN)*

K Nearest Neighbor Algorithms classify a new object by looking at how neighbors are classified. For example, if the new data point's features place it close to more data points classified as banana, than data points classified as apple then it will be classified as a banana not an apple. K is the number of neighbors you want to look at when determining your unknown object's class.

- k = 9. Find the closest 9 neighbors, if it is 5 apples and 4 bananas, classify as apple.
- k = 1. Find the closest neighbor, if it is a banana, then the new data point is classified as a banana.
- k = n. Find n closest neighbors. The class that is the majority of n neighbors classifies the new data point.

Closeness is usually determined by distance, but can be determined using any similarity metric.
KNN is reliant on distance, so making sure your data is normalized before input it into the model is very important.

Model 4: *Neural Network*

Simple neural networks, also known as multi-layer perceptrons (MLPs), are composed of an input layer, one or more hidden layers, and an output layer. Each layer consists of multiple interconnected nodes, called neurons or units.In a neural network, information flows in a forward direction, from the input layer through the hidden layers to the output layer. Each neuron in a layer receives inputs from the previous layer, performs a weighted sum of the inputs, applies an activation function, and passes the result to the next layer. The activation function introduces non-linearity into the network, allowing it to model complex relationships between inputs and outputs.

During the training phase, the network adjusts the weights connecting the neurons to minimize the difference between the predicted outputs and the true outputs. This process is typically performed using optimization algorithms like gradient descent. The objective is to find the optimal set of weights that results in accurate classification predictions.

Neural networks are known for their capability to learn complex patterns and relationships in the data. They can handle both linearly separable and non-linearly separable datasets. Moreover, neural networks have the ability to automatically extract relevant features from the input data, reducing the need for manual feature engineering.

Model 5: *Random Forest*

Random Forest is an ensemble learning method for classification tasks. It combines multiple decision trees to make predictions. Each tree in the Random Forest is built on a random subset of the training data and features. During training, the trees are constructed using a process called bagging, where each tree is trained independently. The final prediction from the Random Forest is determined through a majority vote or averaging of the predictions made by the individual trees.

Random Forests are known for their robustness and ability to handle high-dimensional data. They are effective at handling noisy and irrelevant features, as well as mitigating overfitting. The ensemble nature of Random Forests helps to improve generalization and reduce variance in predictions. Additionally, Random Forests provide an estimate of feature importance, which can aid in understanding the relative contributions of different features to the classification task. They are widely used in various domains, including finance, healthcare, and image recognition, due to their versatility and solid performance.

The key advantages of Random Forest include its ability to handle large and complex datasets, feature selection capabilities, resilience to overfitting, and accurate predictions. However, they may be computationally expensive, especially with a large number of trees and features. Regularization techniques such as limiting tree depth or adjusting the number of features sampled at each node can help mitigate this. Overall, Random Forests are a powerful classification model that excels in handling diverse and challenging classification problems.

I was surprised how well my Naive Bayes models performed through the project, as NB models usually work a lot better with continuous data, and most of the dataset was categorical. Despite this, looking through the model building, particularly early on. For Feature Set 1, and Feature Set 2 my NB model did incredibly well, out performing all of my other models in classifying both classes. However, when all other models performance significantly jumped during Feature Set 5, the NB model did not, and was the worst performer when PCA was used as a feature generation tool.

My sequential neural network performed incredibly poor, despite hours of working to tweak hyperparameters. As you can see in almost every feature iteration, the sequential neural network really struggled to classify true negatives. Yield the least true negatives of any model is every feature set. Simple models like my decision tree out performed it every Feature Set. This goes into a larger theme of simple models performing admirably throughout the entire project. Reinforcing the idea that using the correct tool is better than the fanciest tool. Although, in the end the MLP neural network model was of the top two models, throughout the project the decision tree and KNN, rather simple models were out doing the more complex neural networks and random forest models. And even during Feature Set 5, they performed on par with the complex models.

A summary of what I learned from building the 25 models

1. **Model Selection:** Through building multiple models, I gained insights into the strengths and weaknesses of different algorithms. I learned which models are well-suited for specific types of data and classification tasks. This knowledge will guide future model selection and help me make more informed choices.
2. **Feature Importance:** By analyzing the performance of different models, I gained a deeper understanding of the most important features that contribute to accurate classification. I identified which features consistently had the greatest impact on model performance and which attribute selection methods worked and did not work toward optimizing model performance.
3. **Hyperparameter Tuning:** Experimenting with various hyperparameter settings for each model allowed me to optimize my models performance. I learned how different parameter values influenced the model's accuracy and efficiency. This knowledge will be used to fine-tune models in future projects.
4. **Data Preprocessing:** Handling 25 models required significant data preprocessing steps. I gained experience in data cleaning, handling missing values, feature scaling, and other data preprocessing techniques.
5. **Model Evaluation Metrics:** By evaluating and comparing the performance of multiple models, I deepened my understanding of different evaluation metrics. I learned how to assess model performance beyond just accuracy, considering metrics like precision, recall, F1-score, or area under the ROC curve (AUC-ROC).
6. **Project Management:** Building 25 models required effective project management skills. I learned to organize my work, maintain proper documentation, and streamline the model-building process, automating a lot of the preprocessing, attribute selection, model build, and model evaluation steps with python functions. This experience enhanced my ability to handle large-scale data science projects in both academia and work efficiently.

Top Performing Model

When dealing with an imbalanced dataset, where one class is significantly more prevalent than the other, the evaluation criteria for choosing the best model should go beyond simple accuracy. Accuracy alone can be misleading in imbalanced scenarios, as a model that predicts only the majority class (non-arthritis) will have high accuracy but provide limited value.

Here are several evaluation metrics I used to assess the performance of my binary classification models on the imbalanced CDC dataset:

1. **Precision:** Precision measures the proportion of true positive predictions among all positive predictions. In the context of our classification problem, precision represents the ability of the model to correctly identify individuals with arthritis (For Class=1). A higher precision indicates fewer false positives (65 total FP). Seen in the confusion matrix in **Figure 1.**
2. **Recall (Sensitivity or True Positive Rate):** Recall calculates the proportion of true positive predictions among all actual positive instances. In arthritis classification, recall represents the ability of the model to capture all individuals without arthritis. Higher recall indicates fewer false negatives (71 total FN). Seen in the confusion matrix in **Figure 1.**
3. **F1 Score:** The F1 score is the harmonic mean of precision and recall. It provides a single metric that balances both precision and recall. It is particularly useful when you want to find a balance between identifying true positives and minimizing false positives.
4. **Area Under the ROC Curve (AUC):** AUC summarizes the overall performance of the model by calculating the area under the ROC curve. It provides a single value that indicates the model's ability to distinguish between positive and negative instances. A higher AUC suggests better discrimination. Seen in the ROC Curve in the middle of **Figure 1**, showing a close to optimal ROC curve.
5. **Precision-Recall Curve:** This curve plots precision against recall, allowing you to analyze the model's performance across various thresholds. It is especially useful when the positive class is rare, as it focuses on the precision-recall trade-off. Seen in the PRC plot on the right of **Figure 1**, showing a close to optimal PRC plot.

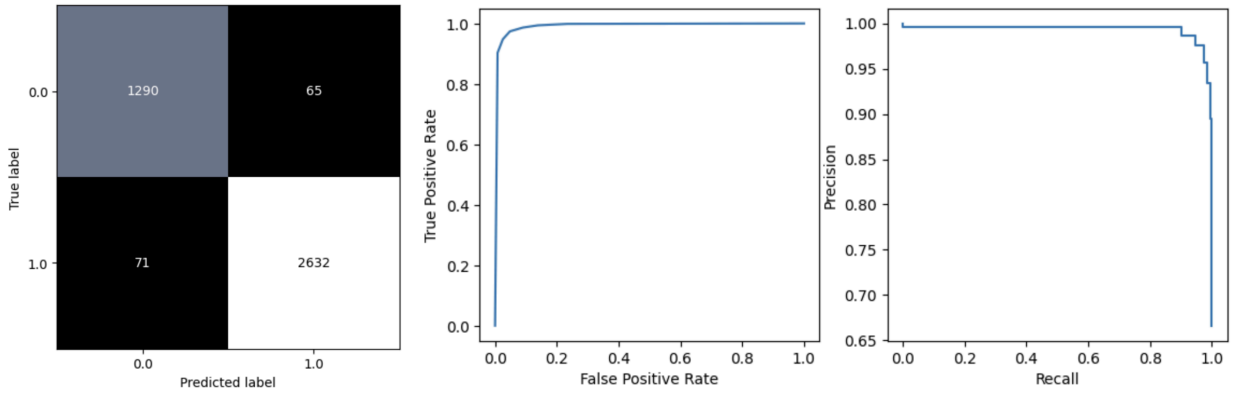
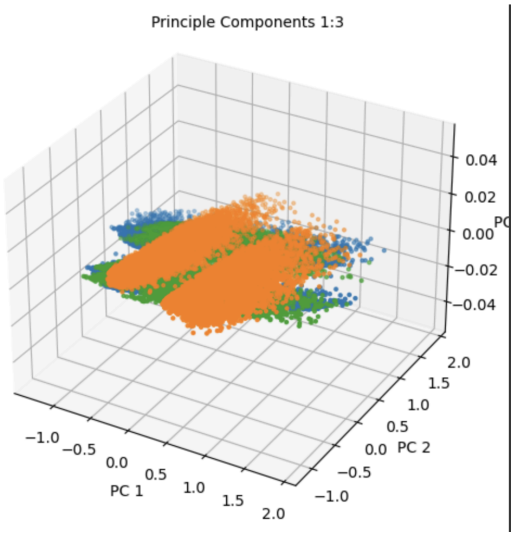


Figure 1.0: *Confusion Matrix and ROC and PRC for KNN classification model using Feature Set 5.*

The **KNN model** using **Feature Set 5** was the model and attribute selection method that gave me the best performance. Feature Set 5 used PCA feature generation using the 31 features in Feature Set 3 as the input features in which to build its four principal components. This PCA feature generation method

was the clear best attribute selection method out of the five for all 30 models. Because I used PCA, there are no “reduced features” ; instead there are 4 principle components built from the features in Feature Set 3, so therefore I did not list them here because they are literally simply named “0, 1, 2, 3”. But they can be seen in the attached excel file named “thayer-best-train.csv”. However, a visualization of three of these components/features can be seen in the figure below.



The Multi-layer Perceptron (MLP) Classifier, a simple neural network, using the same attribute selection method was a very close second, and could be argued for just as easily. However, due to the nature of the classification task and the dataset's imbalance, I assigned greater importance to the classification of true negatives compared to other evaluation metrics. The KNN model identified 1290 true negatives, the highest among all models. Additionally, the KNN model has the highest true negative rate (true positive rate for Class = 0) among all 30 models of 0.952. The KNN model also had very high weighted F1, ROC Area, MMC measures, showing that the model was not just outstanding at identifying true negatives, but was also great at correctly identifying true positives. Furthermore, the precision and recall of Class=0 are extremely high at 0.976 and 0.974 respectively, the second highest amongst all models seen in **Table 4**. These metrics show the KNN model's ability to correct. Specifically, this higher precision indicates our KNN had very few false negatives.

Model	Features	Class	TPR	FPR	Precision	Recall	F1	ROC Area	MCC
KNN	F5	0	0.974	0.048	0.948	0.952	0.95	0.963	0.925
		1	0.952	0.026	0.976	0.974	0.975	0.963	0.925
		weighted	0.968	0.041	0.967	0.966	0.967	0.963	0.925

Table 4. *Performance metric summary for KNN classification model using Feature Set 5.*

The objective of this project was to develop a classification model capable of identifying individuals in the population who are likely to have arthritis or similar conditions. Since the negative class represents the disease, it is naturally rare in the population and, consequently, also rare in our sample. This rarity poses

a challenge for classification, as there is less data available for training compared to the more prevalent positive class. Hence, why I highly valued models with high true negative rates.

Furthermore, given that arthritis is a treatable disease, it is more valuable to correctly diagnose individuals with arthritis (or related conditions) rather than correctly identifying those without arthritis. In practical terms, the ability to accurately classify individuals with a disease holds more significance in most real-world scenarios.Considering the rarity of the negative class and the potential benefits of correctly identifying individuals with the disease, placing greater emphasis on the classification of true negatives is justified in this context.

That being said, MLP Classifier was marginally better in almost all other metrics than the KNN model. Most notably the MLP Classifier was outstanding at false negatives (low FPR for Class=0). Which is important, in the context of our classifier, as you do not want to falsely diagnose individuals with arthritis who do not have arthritis.

In conclusion, because I placed heavy emphasis on the classification of true negatives, the KNN model was the best model I observed. However, there is a clear argument to be made for the MLP model, especially in cases where reducing false negatives is the goal.

Conclusion

Top Five Attributes
For each feature selection method I used my Random Forest model to calculate the gini importance of each feature in that feature section method. These importance scores are all listed in the accompanying excel file in the 'gini importance' tab. Below are the five features that were consistently in the top 5-7 for all feature selection methods.

Feature	Rename	Description	Importance
x.llcpwt	design_weight	Final weight: Land-line and cell-phone data	0.12694
x.bmi5	bmi	Computed body mass index	0.12578
employ1	employ	Employment Status	0.11492
x.ageg5yr	age	Reported age in five-year age categories calculated variable	0.10765
x.state	state	State FIPS Code	0.08976

Table 3.0: *Top 5 features in the 2018 BRFSS Survey Dataset with corresponding gini importance metric.*

All 5 of these features have the consistently highest importance scores compared to the 35 total features used throughout the project. Some of the features like *design_weight, age*, and *bmi* I can understand how they are heavily correlated to our class having arthritis. However, *employ* and particular *state* surprised me. I initially thought that state would be an irrelevant feature and wasn't going to include it, but it proved throughout the entire project to be one of the most important features for my classification models.

Key Learnings and Takeaways from Building 30 Classification Models

1. **Thorough Data Exploration**: Investing time in understanding and exploring the dataset proved to be valuable in the model-building process. A comprehensive understanding of the data upfront can save time and lead to better model performance.
2. **Persistence in Model Improvement:** When faced with a lack of accuracy improvements in the models, perseverance is key. Trying out different approaches and techniques is essential, as many models can behave like black boxes. Continuously experimenting until finding a successful approach can lead to better results.
3. **Potential for Ensemble Methods:** Given more time, utilizing an ensemble method that combines all 30 models or incorporates a selection of the best models could be explored for the final analysis. Ensemble methods can often provide improved performance and robustness.
4. **Sampling Methods and Subset Analysis:** Although the assignment did not cover sampling methods in detail, focusing on specific subsets of the data, such as looking specifically at older individuals, should be considered. This approach could address the imbalances in the dataset. Looking at older people would most likely increase the occurrence of people having arthritis, our rare class. Furthermore, removing young people would reduce the number of individuals without any health conditions. A lot of the features in the dataset had to do with various ailments, so having a bunch of young people without ailments does not give our models much data to work with. Sampling for individuals 65+ would remedy this.
5. **Handling Missing Data:** Using median as a missing value replacement worked well, but I believe utilizing non-null values to estimate and replace missing data would have proved more effective. Or using K-nearest neighbors (KNN) imputation may also be another viable solution for dealing with missing values.
6. **Suitability of Neural Networks:** While neural networks (NN) can be powerful tools, they may not always be the optimal choice for binary classification tasks out of the box, particularly with a high number of binary variables because of their black box nature, and the about of parameter tuning required for optimal performance compared to other simpler options that may perform better.
7. **Importance of Model Parameters:** Understanding the parameters specific to each model type and their impact on performance is crucial. For example, I found that tweaking parameters, such as max_depth, can have significant ramifications on my tree model performance. Gaining knowledge of all relevant parameters for each model type is essential for effective modeling.

Building 30 classification models helped me build a much deeper understanding of the modeling process and what I can do to build more accurate and robust classification models.

Appendix

Performance Metrics for all 30 Models

Model	Features	Class	TPR	FPR	Precision	Recall	F1	ROC Area	MCC
Decision Tree	F1	1	0.834	0.455	0.622	0.545	0.581	0.689	0.392
		0	0.545	0.166	0.785	0.834	0.809	0.689	0.392
		weighted	0.719	0.356	0.731	0.737	0.733	0.689	0.392
Gaussian NB	F1	1	0.791	0.41	0.586	0.59	0.588	0.691	0.381
		0	0.59	0.209	0.794	0.791	0.792	0.691	0.381
		weighted	0.725	0.343	0.725	0.724	0.724	0.691	0.381
KNN	F1	1	0.868	0.566	0.623	0.434	0.512	0.651	0.337
		0	0.434	0.132	0.754	0.868	0.807	0.651	0.337
		weighted	0.697	0.422	0.71	0.723	0.708	0.651	0.337
Random Forest	F1	1	0.82	0.438	0.61	0.562	0.585	0.691	0.39
		0	0.562	0.18	0.789	0.82	0.804	0.691	0.39
		weighted	0.721	0.35	0.729	0.734	0.731	0.691	0.39
Sequential NN	F1	1	0.982	0.91	0.718	0.09	0.16	0.536	0.17
		0	0.09	0.018	0.683	0.982	0.806	0.536	0.17
		weighted	0.731	0.675	0.694	0.684	0.59	0.536	0.17
MLP NN	F1	1	0.844	0.456	0.636	0.544	0.586	0.694	0.405
		0	0.544	0.156	0.787	0.844	0.814	0.694	0.405
		weighted	0.723	0.353	0.736	0.744	0.738	0.694	0.405
Model	**Features**	**Class**	**TPR**	**FPR**	**Precision**	**Recall**	**F1**	**ROC Area**	**MCC**
Decision Tree	F2	1	0.844	0.477	0.627	0.523	0.57	0.683	0.385
		0	0.523	0.156	0.779	0.844	0.81	0.683	0.385
		Weighted	0.715	0.368	0.728	0.737	0.73	0.683	0.385
Gaussian NB	F2	1	0.77	0.387	0.572	0.613	0.592	0.691	0.377
		0	0.613	0.23	0.799	0.77	0.784	0.691	0.377
		Weighted	0.731	0.339	0.723	0.718	0.72	0.691	0.377
KNN	F2	1	0.87	0.565	0.626	0.435	0.514	0.653	0.341
		0	0.435	0.13	0.754	0.87	0.808	0.653	0.341
		weighted	0.698	0.421	0.712	0.725	0.71	0.653	0.341

Sequential NN	F2	1	0.989	0.967	0.6	0.033	0.063	0.511	0.077
		0	0.033	0.011	0.671	0.989	0.8	0.511	0.077
		weighted	0.738	0.722	0.647	0.67	0.554	0.511	0.077
MLP NN	F2	1	0.861	0.47	0.656	0.53	0.586	0.695	0.415
		0	0.53	0.139	0.785	0.861	0.821	0.695	0.415
		weighted	0.725	0.358	0.742	0.75	0.743	0.695	0.415
Random Forest	F2	1	0.826	0.465	0.607	0.535	0.569	0.681	0.374
		0	0.535	0.174	0.78	0.826	0.803	0.681	0.374
		weighted	0.712	0.365	0.722	0.729	0.725	0.681	0.374

Model	Features	Class	TPR	FPR	Precision	Recall	F1	ROC Area	MCC
Decision Tree	F3	1	0.871	0.529	0.646	0.471	0.545	0.671	0.375
		0	0.471	0.129	0.766	0.871	0.815	0.671	0.375
		weighted	0.709	0.396	0.726	0.737	0.725	0.671	0.375
Gaussian NB	F3	1	0.803	0.449	0.584	0.551	0.567	0.677	0.36
		0	0.551	0.197	0.781	0.803	0.792	0.677	0.36
		weighted	0.71	0.363	0.715	0.719	0.717	0.677	0.36
KNN	F3	1	0.844	0.487	0.622	0.513	0.562	0.678	0.376
		0	0.513	0.156	0.776	0.844	0.808	0.678	0.376
		weighted	0.711	0.374	0.724	0.733	0.726	0.678	0.376
Sequential NN	F3	1	0.979	0.898	0.704	0.102	0.178	0.54	0.177
		0	0.102	0.021	0.685	0.979	0.806	0.54	0.177
		weighted	0.728	0.666	0.691	0.686	0.596	0.54	0.177
MLP NN	F3	1	0.849	0.466	0.64	0.534	0.582	0.692	0.403
		0	0.534	0.151	0.784	0.849	0.815	0.692	0.403
		weighted	0.721	0.359	0.736	0.744	0.737	0.692	0.403
Random Forest	F3	1	0.828	0.452	0.616	0.548	0.58	0.688	0.389
		0	0.548	0.172	0.785	0.828	0.806	0.688	0.389
		weighted	0.719	0.355	0.729	0.735	0.731	0.688	0.389

Model	Features	Class	TPR	FPR	Precision	Recall	F1	ROC Area	MCC
Decision Tree	F4	1	0.741	0.49	0.496	0.51	0.503	0.625	0.249
		0	0.51	0.259	0.751	0.741	0.746	0.625	0.249

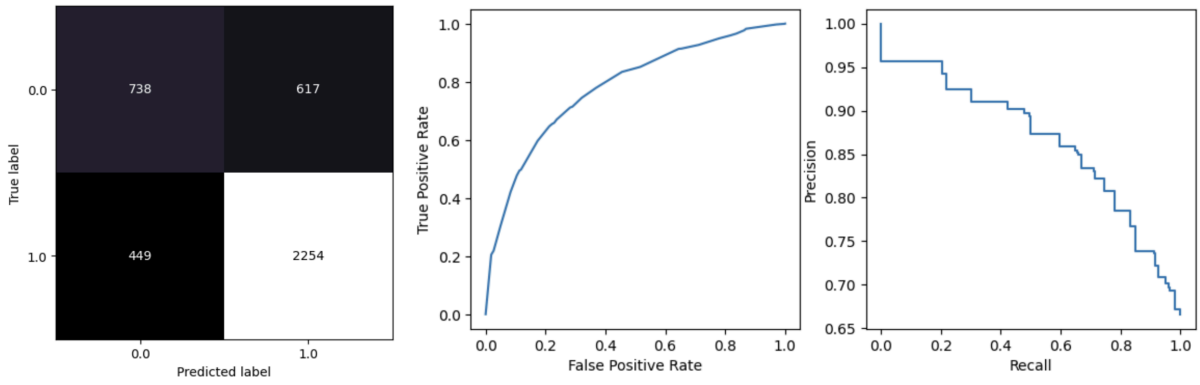
		weighted	0.668	0.415	0.666	0.664	0.665	0.625	0.249
Gaussian NB	F4	1	0.849	0.519	0.616	0.481	0.54	0.665	0.355
		0	0.481	0.151	0.766	0.849	0.805	0.665	0.355
		weighted	0.702	0.394	0.716	0.726	0.717	0.665	0.355
KNN	F4	1	0.85	0.514	0.618	0.486	0.544	0.668	0.36
		0	0.486	0.15	0.767	0.85	0.806	0.668	0.36
		weighted	0.704	0.391	0.718	0.728	0.719	0.668	0.36
Sequential NN	F4	1	0.999	0.99	0.765	0.01	0.019	0.504	0.059
		0	0.01	0.001	0.668	0.999	0.8	0.504	0.059
		weighted	0.748	0.742	0.7	0.668	0.539	0.504	0.059
MLP NN	F5	1	0.845	0.466	0.634	0.534	0.58	0.69	0.398
		0	0.534	0.155	0.784	0.845	0.813	0.69	0.398
		weighted	0.72	0.359	0.734	0.741	0.735	0.69	0.398
Random Forest	F5	1	0.85	0.514	0.618	0.486	0.544	0.668	0.36
		0	0.486	0.15	0.767	0.85	0.806	0.668	0.36
		weighted	0.704	0.391	0.718	0.728	0.719	0.668	0.36

Model	**Features**	**Class**	**TPR**	**FPR**	**Precision**	**Recall**	**F1**	**ROC Area**	**MCC**
Decision Tree	F5	1	0.96	0.089	0.92	0.911	0.915	0.935	0.873
		0	0.911	0.04	0.955	0.96	0.958	0.935	0.873
		weighted	0.941	0.073	0.943	0.944	0.943	0.935	0.873
Gaussian NB	F5	1	0.923	0.226	0.835	0.774	0.804	0.849	0.712
		0	0.774	0.077	0.891	0.923	0.907	0.849	0.712
		weighted	0.856	0.175	0.872	0.874	0.872	0.849	0.712
KNN	F5	1	0.974	0.048	0.948	0.952	0.95	0.963	0.925
		0	0.952	0.026	0.976	0.974	0.975	0.963	0.925
		weighted	0.968	0.041	0.967	0.966	0.967	0.963	0.925
Sequential NN	F5	1	0.943	0.296	0.862	0.704	0.775	0.824	0.686
		0	0.704	0.057	0.864	0.943	0.902	0.824	0.686
		weighted	0.828	0.217	0.863	0.863	0.86	0.824	0.686
MLP NN	F5	1	0.982	0.055	0.964	0.945	0.955	0.964	0.932
		0	0.945	0.018	0.973	0.982	0.978	0.964	0.932
		weighted	0.964	0.042	0.97	0.97	0.97	0.964	0.932

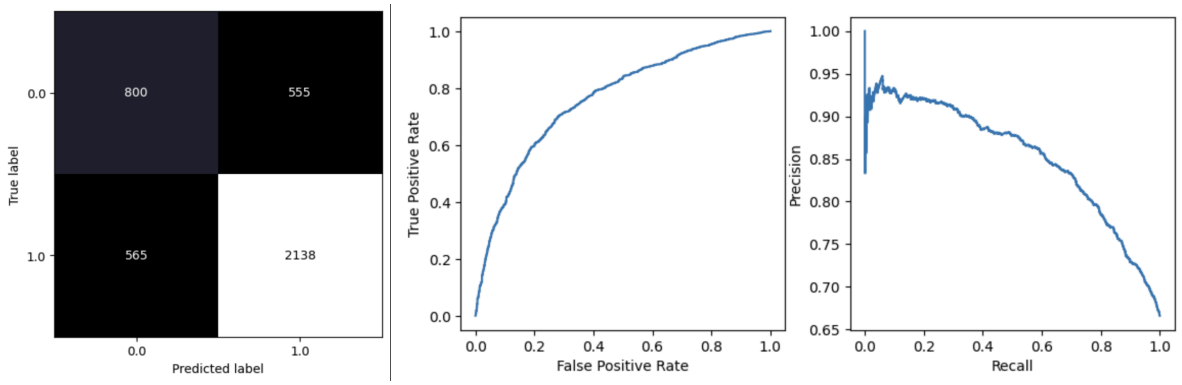
Random Forest	F5	1	0.979	0.071	0.957	0.929	0.943	0.954	0.915
		0	0.929	0.021	0.965	0.979	0.972	0.954	0.915
		weighted	0.954	0.054	0.962	0.963	0.962	0.954	0.915

Confusion Matrix and ROC & PRC Plots for all 30 Models

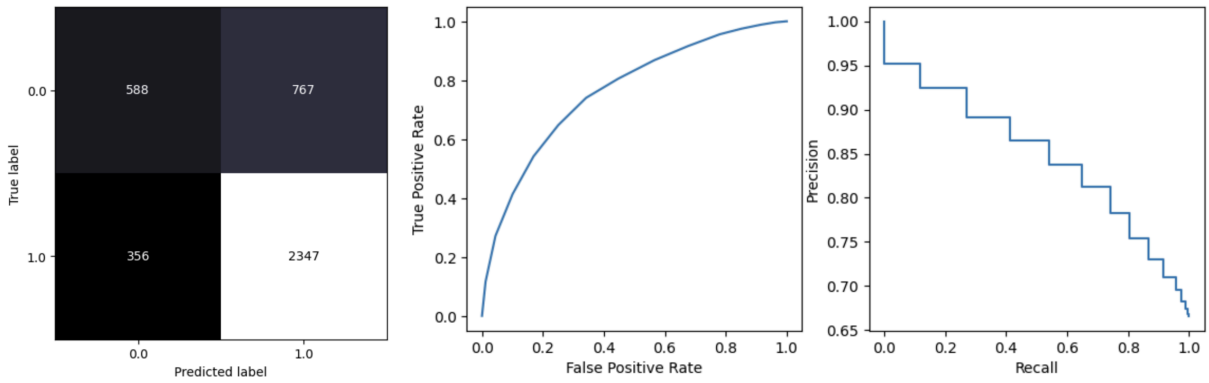
1. Decision Tree Feature F1



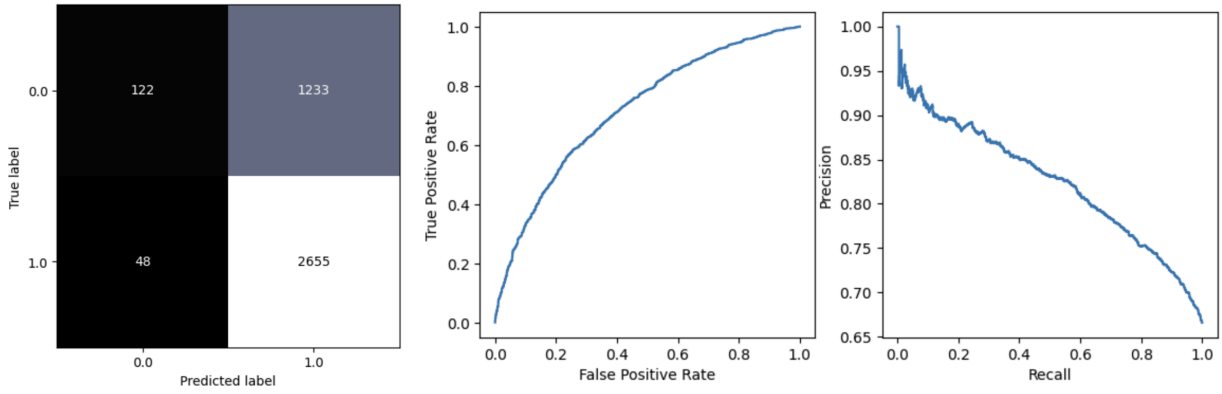
2. Gaussian Naive Bayes F1



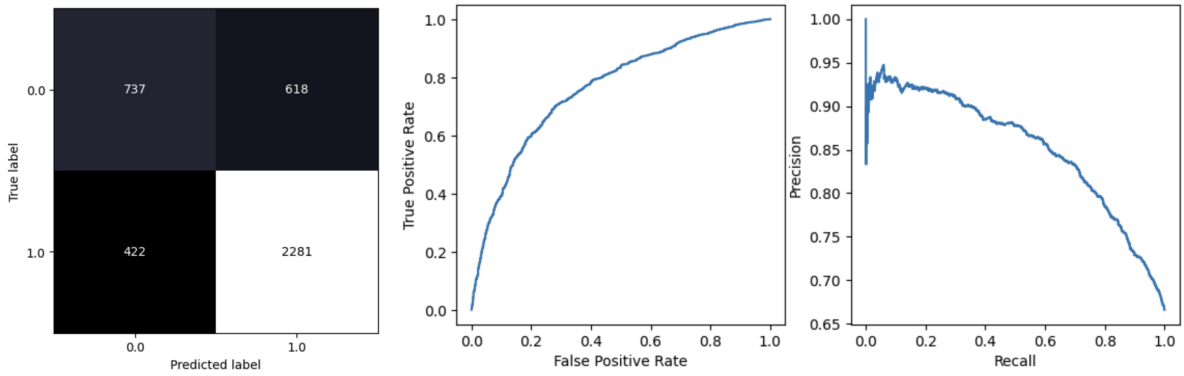
3. KNN F1



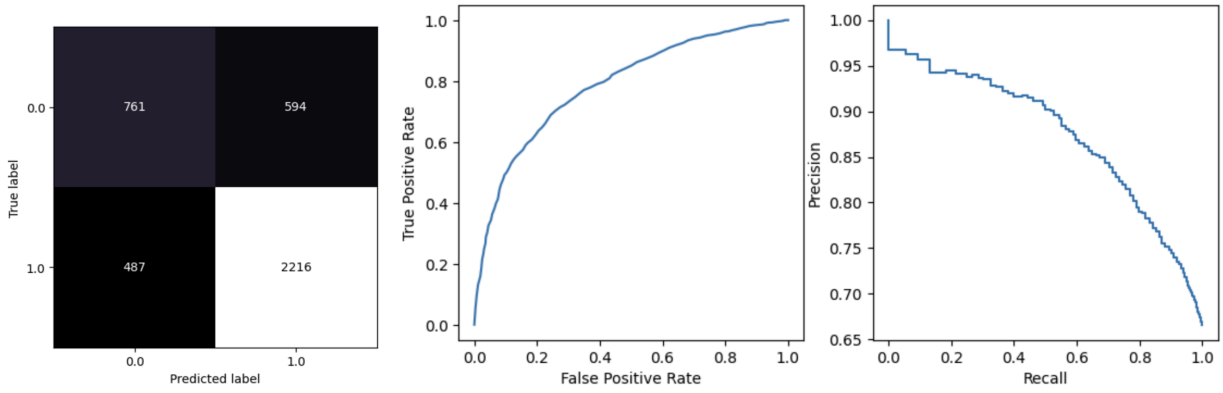
4. Sequential Neural Network F1



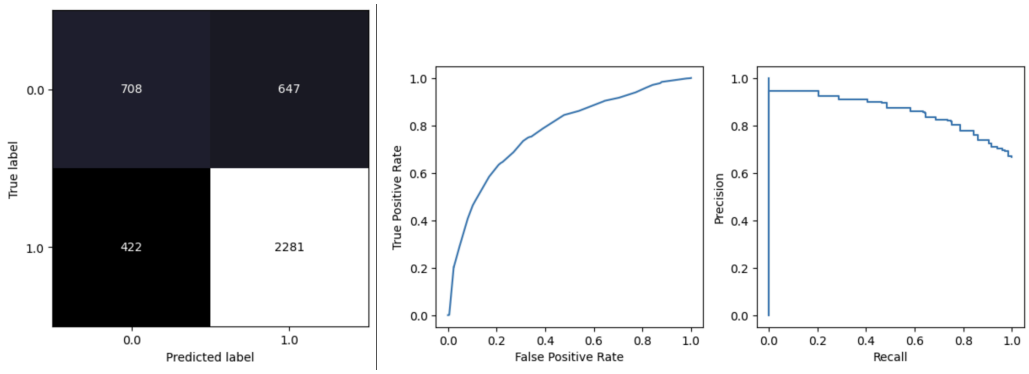
5. MLP Neural Network F1



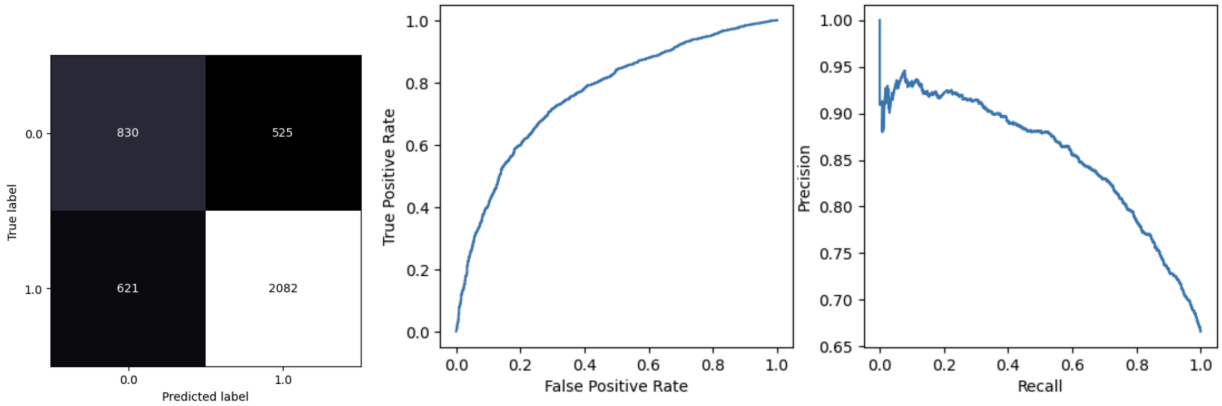
6. Random Forest F1



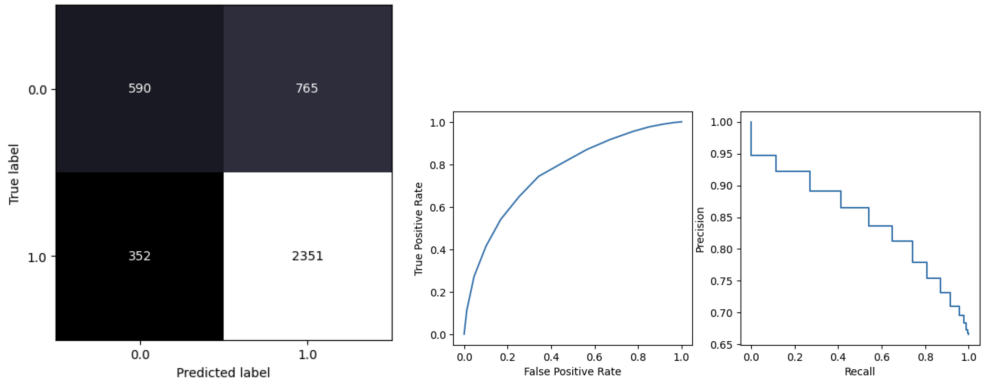
7. Decision Tree Feature F2



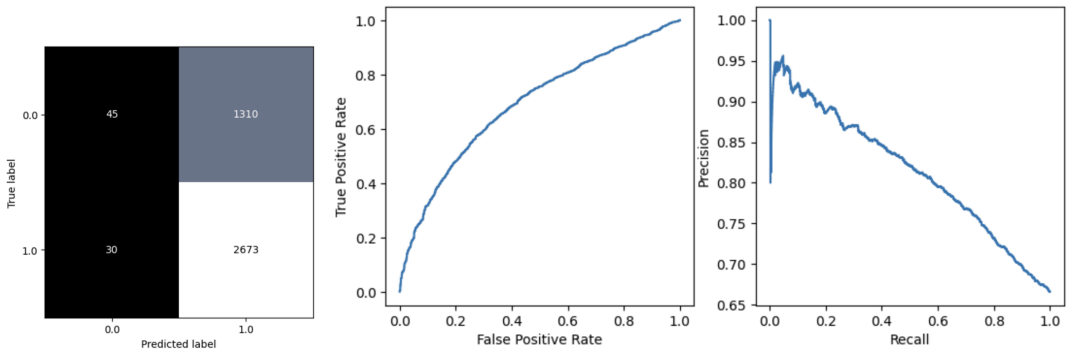
8. Gaussian Naive Bayes F2



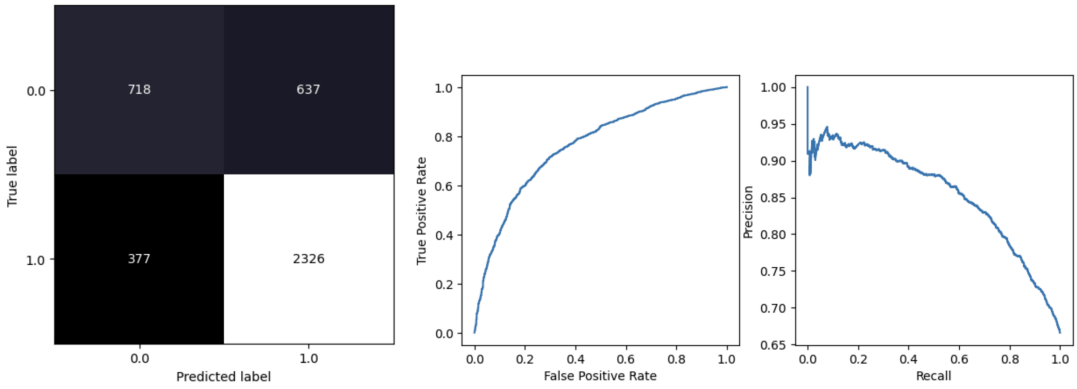
9. KNN F2



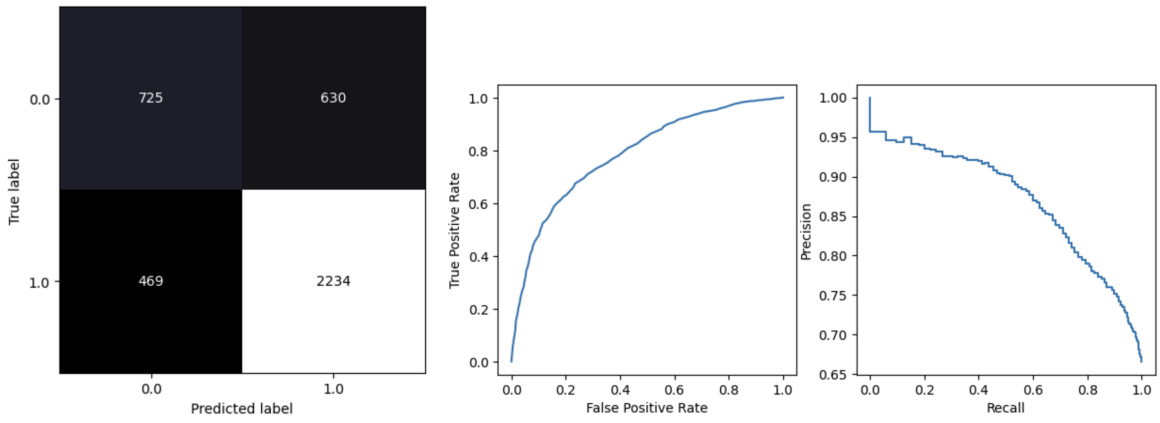
10. Sequential Neural Network F2



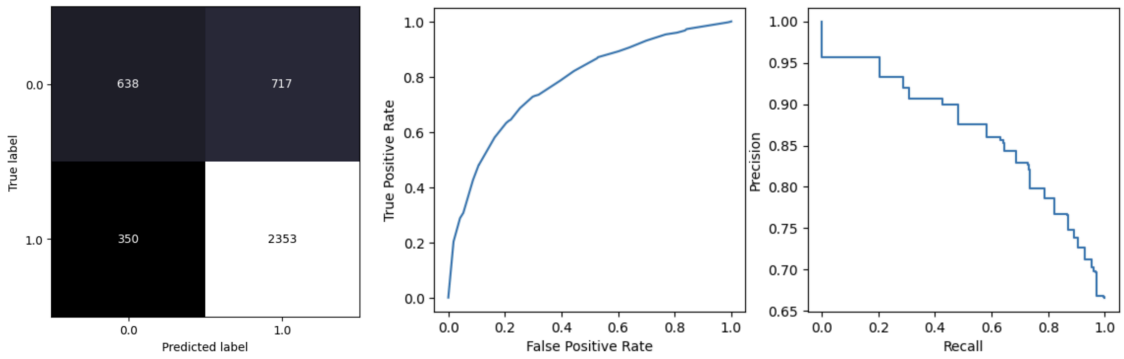
11. MLP Neural Network F2



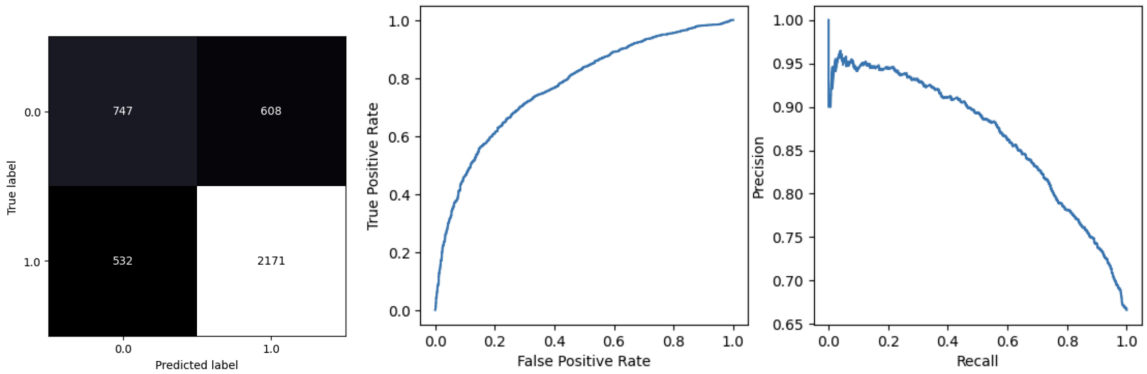
12. Random Forest F2



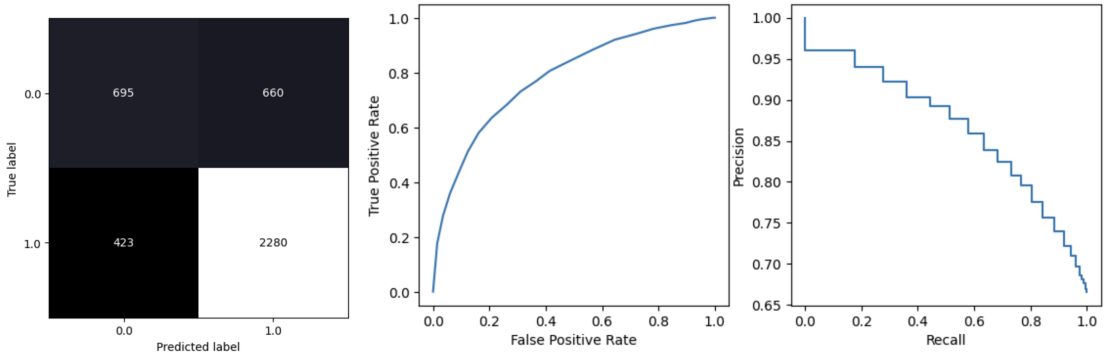
13. Decision Tree Feature F3



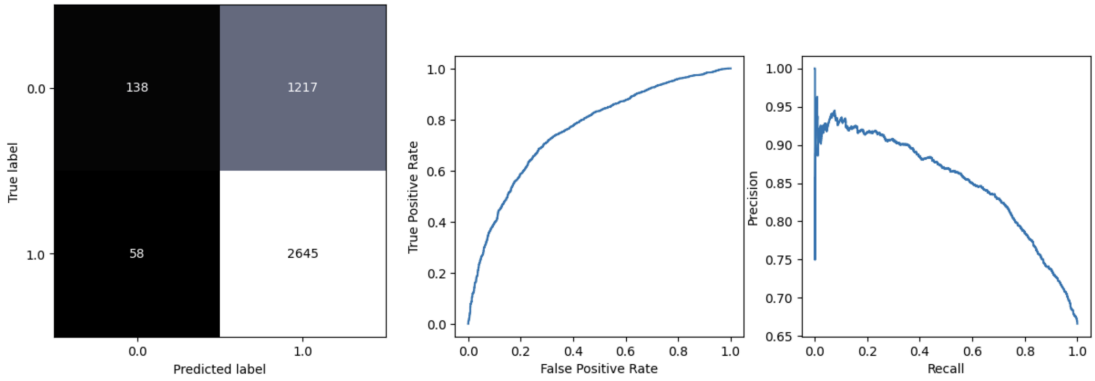
14. Gaussian Naive Bayes F3



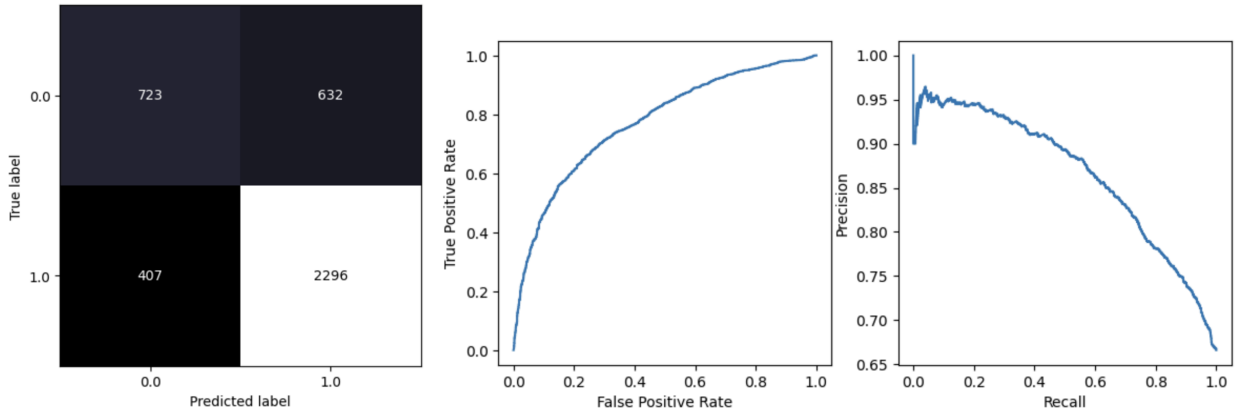
15. KNN F3



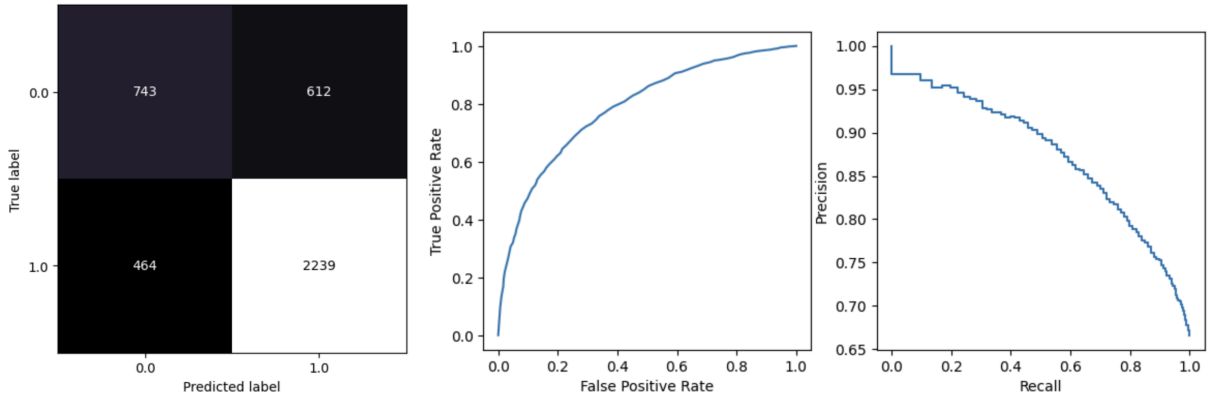
16. Sequential Neural Network F3



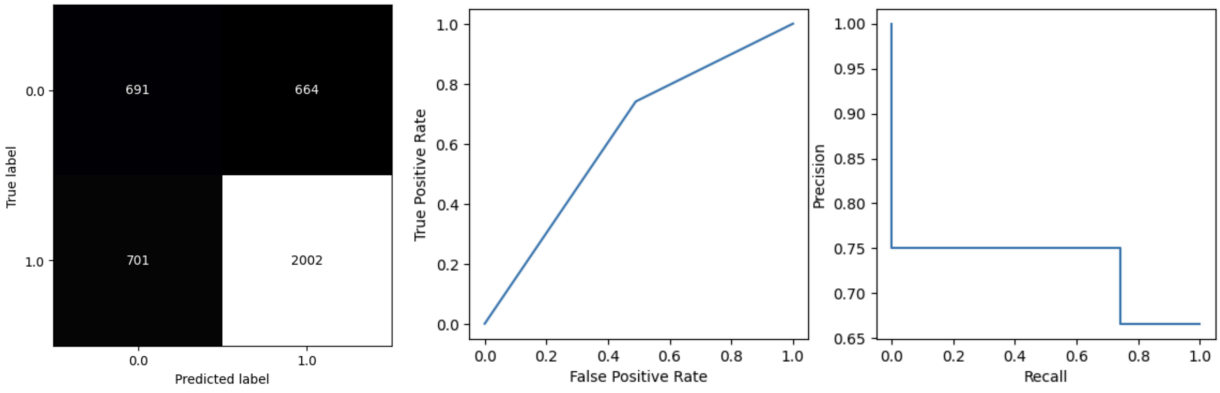
17. MLP Neural Network F3



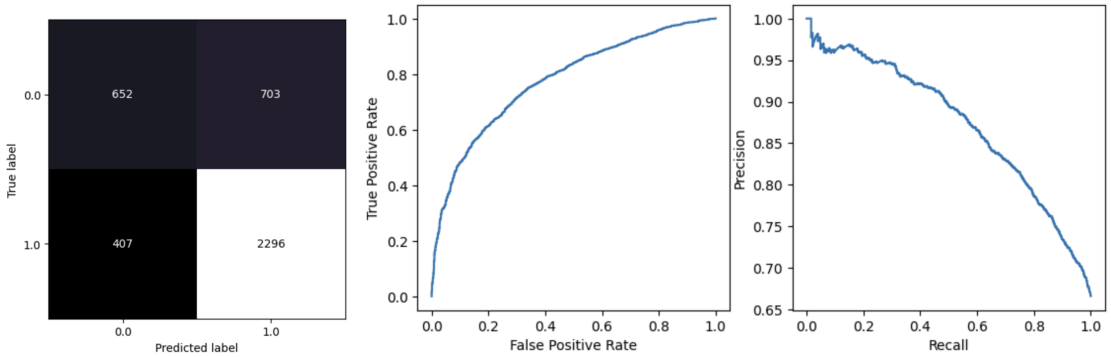
Random Forest F3



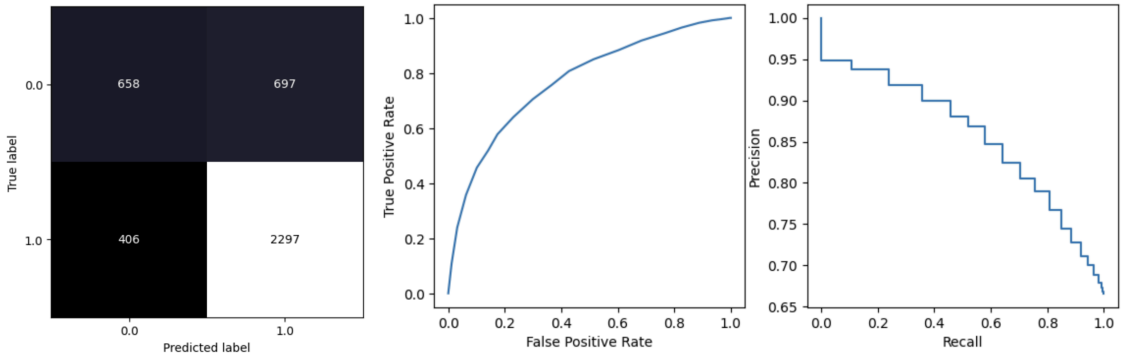
18. Decision Tree Feature F4



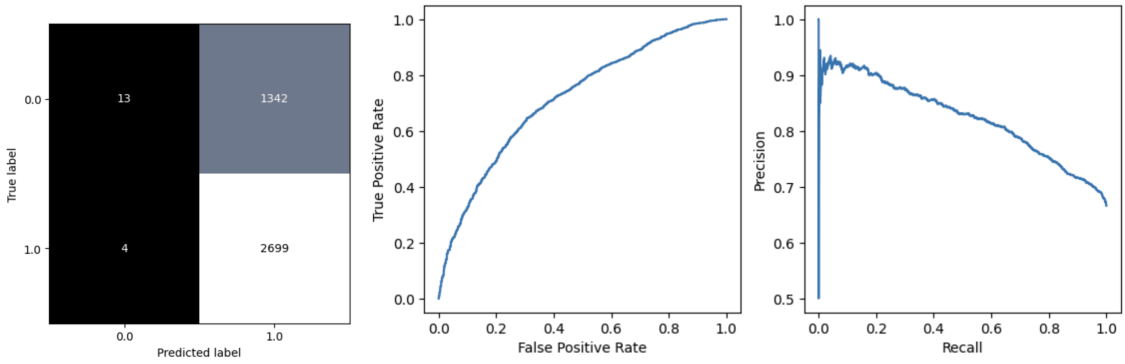
19. Gaussian Naive Bayes F4



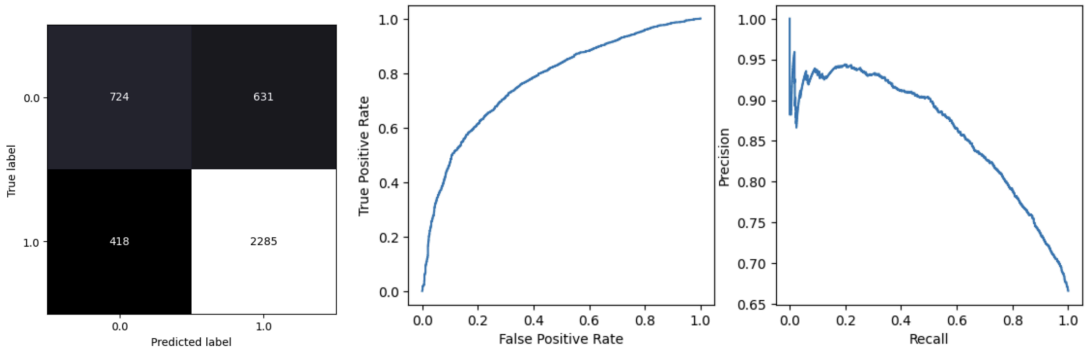
20. KNN F4



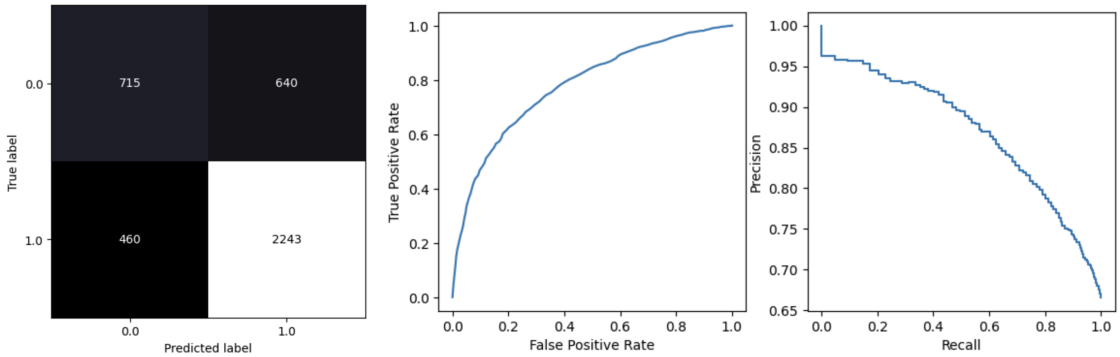
21. Sequential Neural Network F4



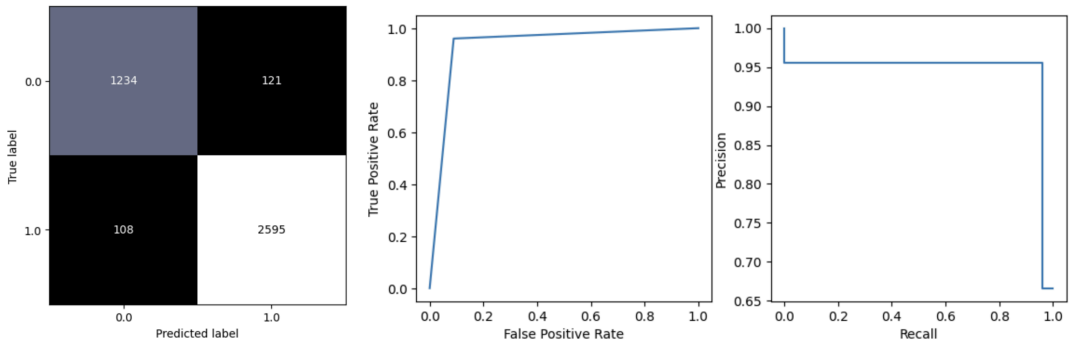
22. MLP Neural Network F4



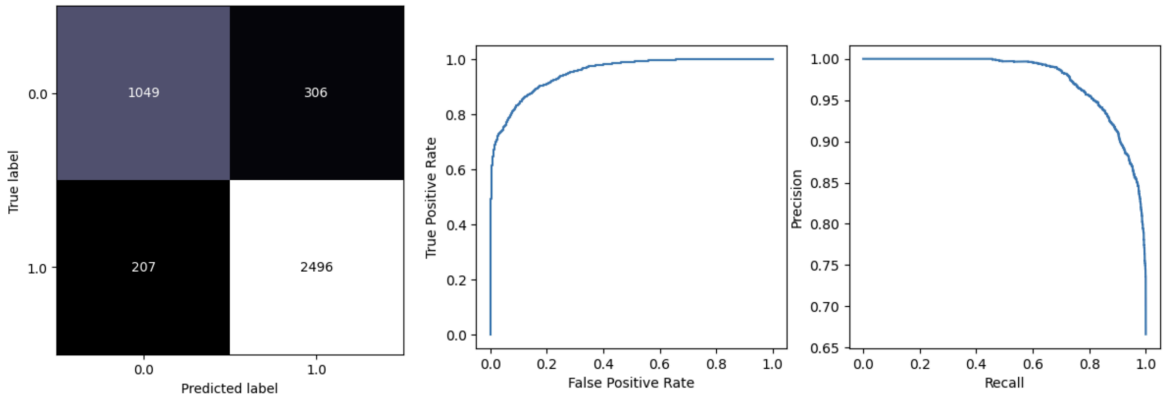
23. Random Forest F4



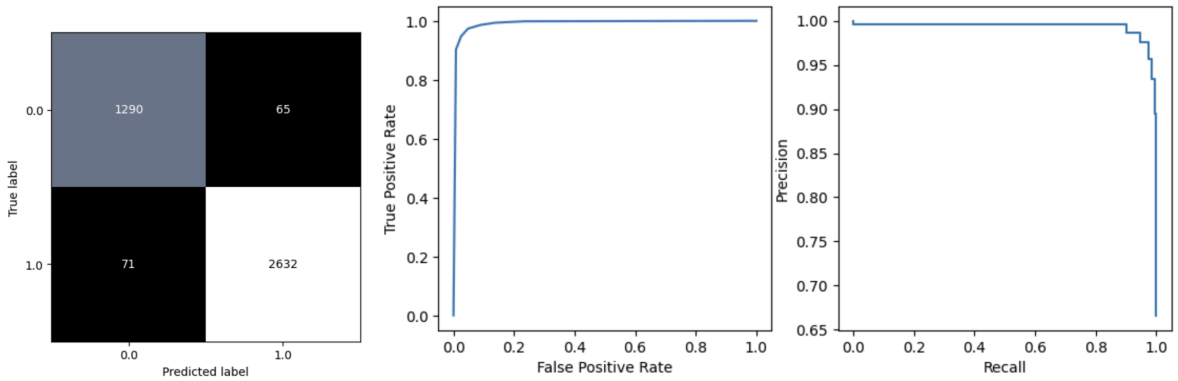
24. Decision Tree Feature F5



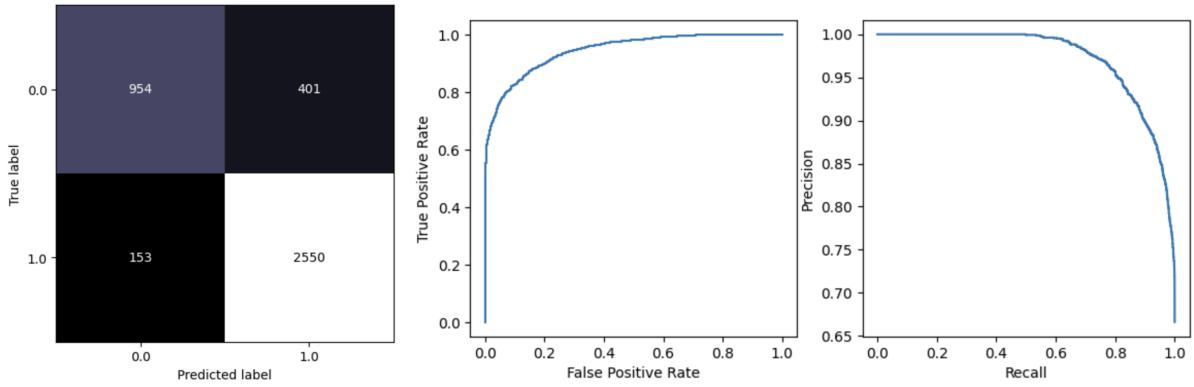
25. Gaussian Naive Bayes F5



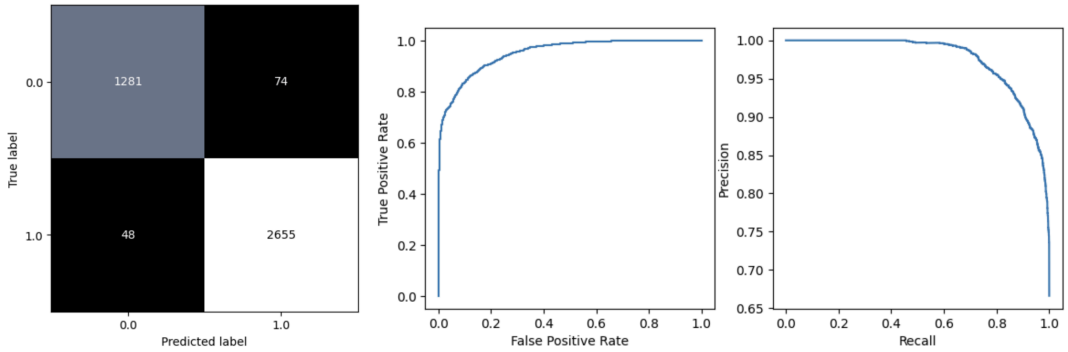
26. KNN F5



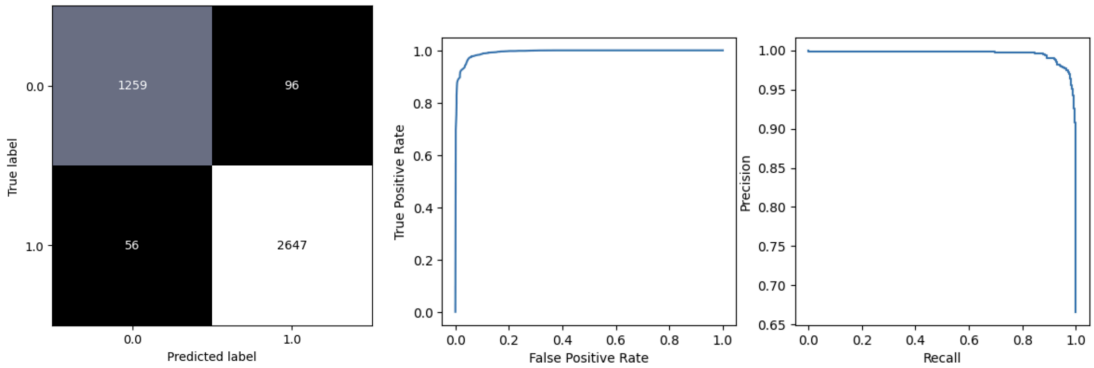
27. Sequential Neural Network F5



28. MLP Neural Network F5



29. Random Forest F5



Resource

https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn

https://github.com/jakevdp/PythonDataScienceHandbook/blob/master/notebooks_v1/05.05-Naive-Bayes.ipynb

https://www.kdnuggets.com/2019/04/naive-bayes-baseline-model-machine-learning-classification-performance.html/2#:~:text=Can%20handle%20both%20numeric%20and,perform%20better%20with%20categorical%20data.

https://scikit-learn.org/stable/modules/naive_bayes.html#gaussian-naive-bayes

https://www.statology.org/plot-multiple-roc-curves-python/

https://www.analyticsvidhya.com/blog/2015/12/improve-machine-learning-results/