Group 20
Felipe Flores - 1640618

Data Set - https://www.kaggle.com/datasets/alexteboul/diabetes-health-indicators-dataset

Introduction

The idea behind using this data set has to deal with personal matters. As several people within my family are diabetic and I personally feel that the medical field needs all the assistance possible to help those patients that are in need. So it was more for personal desire to create models and learn what I can.

Data Description

The data set that is being used for this project deals with diabetic patients and their current health status. There are 22 features in the data set. Of the 22 features there are 3 that are not categorical and are quantitative, while the rest of the 19 features are categorical by nature. From the 22 features I went and reduced the number down to 9 features when selecting features.

The following are all the features in the data set and their nature, range, and description.

Diabetes: [0, 1]	Binary labeling regarding whether a patient has diabetes. 1 = yes, 0 = no. Qualitative	HvyAlcoholConsump: [0, 1]	Heavy drinker having more than 14 drinks per week and 7 for female. 1 = yes, 0 = no Qualitative
HighBP: [0, 1]	Binary Labeling for having High Blood Pressure. 1 = yes, 0 = no. Qualitative	AnyHealthCare: [0, 1]	Any form of health care such as: health insurance or prepaid plans. 1= yes, 0 = no Qualitative
HighChol: [0, 1]	Binary Labeling for having High Cholesterol. 1 = yes, 0 = no. Qualitative	NoDocbcCost: [0, 1]	In the past year the patient needed to see a doctor but couldn't because of cost. 1 = yes, 0 = no Qualitative
CholCheck: [0, 1]	Whether the patient has had a cholesterol check in the last 5 years. 1 = yes, 0 = no Qualitative	GenHelth: [1, 5]	A scale measuring the general health of a patient. 1 = excellent health 2 = very good health 3 = good health 4 = fair health 5 = poor health Qualitative
BMI: [12, 98]	Body Mass Index measurement Quantitative	MentHlth: [0, 30]	Number of days of poor mental health scale in the past 30 days. Quantitative

Smoker: [0, 1]	Whether the patient has smoked at least 100 cigarettes in their life. 1 = yes, 0 = no Qualitative	PhysHlth: [0, 30]	Number of days with physical illness or injury in the past 30 days. Quantitative
Stroke: [0, 1]	Has the patient ever had a stroke? 1 = yes, 0 = no Qualitative	DiffWalk: [0, 1]	Have serious difficulty walking or climbing stairs 1 = yes, 0 = no Qualitative
HeartDiseaseorAttack: [0, 1]	Coronary heart disease or myocardial infarction 1 = yes, 0 = no Qualitative	Sex: [0, 1]	Gender of patient 1 = male 0 = female Qualitative
PhysActivity: [0, 1]	Done physical activity in the past 30 days. 1 = yes, 0 = no Qualitative	Age: [1, 13]	Age level category 1 = 18-24 9 = 60-64 13 = 80+ The values in between represent ages that are not listed. Qualitative
Fruits: [0, 1]	Consume 1 or more fruits per day. 1 = yes, 0 = no Qualitative	Education: [1, 6]	Education level 1 = never attended school 6 = Graduated college The numbers in between are levels of education until college Qualitative
Veggies: [0, 1]	Consume 1 or more veggies per day 1 = yes, 0 = no Qualitative	Income: [1, 8]	Income Scale 1 = less than $10,000 8 = $75,000+ Qualitative

Preprocessing

Initially the dataset was unbalanced as there were more non-diabetic patients by more than 100,000 patients. So it was necessary to balance this data set before making any models as this would lead to a bias towards the non-diabetic patients. To achieve a balanced dataset I used undersampling. This was then able to balance the data set but the caveat to that is the number of samples was reduced from 254,000 to roughly 70,000.

In addition to this there were no missing values that needed to be corrected outside of the imbalance set. So the only form of change that was needed was the undersampling.

Feature Selection

Through the project there were three methods of selecting features that were used. The methods were Correlation, LassoCV, and Wrapper method (sequential selection).

Firstly, was correlation and lasso as I wanted to see how hand selected features compared to a selection method that was automated. Through my tests lasso proved more effective in selecting features with greater importance. As lasso yielded better results in all models.

Secondly, was the wrapper method which was the best of the feature selection methods. There was still some testing that needed to occur with this method as there were various options that gave different subsets of the data set. In the end I settled with forward direction using Random Forest Classifier as the estimator. As this pair had the highest accuracy when used on models.

Of the listed features the best selected features were: BMI, stroke, HeartDiseaseorAttack, PhysActivity, Veggies, GenHlth, MentHlth, PhysHlth, Income. This subset again was produced from the wrapper method.

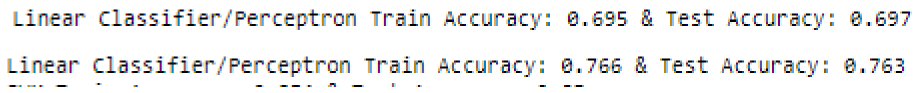
Models

There were 5 models in total that were used throughout the project which were: Perceptron, KNN, Logistic Regression, Random Forest Classifier, and SVM. Since there were two different feature selection methods being used, I had also hypertuned each model for the 2 different feature selections. For example, KNN had its parameters hypertuned for both the correlation feature selection as well as for the lasso feature selection. This was done to make it fair for all the models as they would have their best possible scores for each possibility.

Of these models the best performer overall was Random ForestClassifier. SVM and Logistic Regression were close seconds as they were significantly better than the other remaining models.

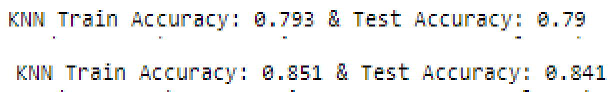
Percepton

Perceptron was the weakest model of the group for both feature selection methods. The accuracy for the model was roughly 69% for correlation and 76% for lasso selection. Compared to the other more complex models it's understandable as to why this is the case. I think the data that is present in the data set is not linear and too complex to attempt at separating through a straight line. The following images are the accuracy scores for the correlation set and the lasso set respectively.


```
Linear Classifier/Perceptron Train Accuracy: 0.695 & Test Accuracy: 0.697

Linear Classifier/Perceptron Train Accuracy: 0.766 & Test Accuracy: 0.763
```

KNN (K Nearest Neighbor)

KNN is the 4th best model out of the group but this does not mean it is worse than the top 4. As it is only marginally worse than the 3rd model being Logistic Regression. The accuracy for the model was roughly 79% for correlation selection and 84% for lasso. Compared to Perceptron the performance is drastically improved. I attribute this to the more complex nature of the model. The following images are the accuracy scores for the correlation set and the lasso set respectively.

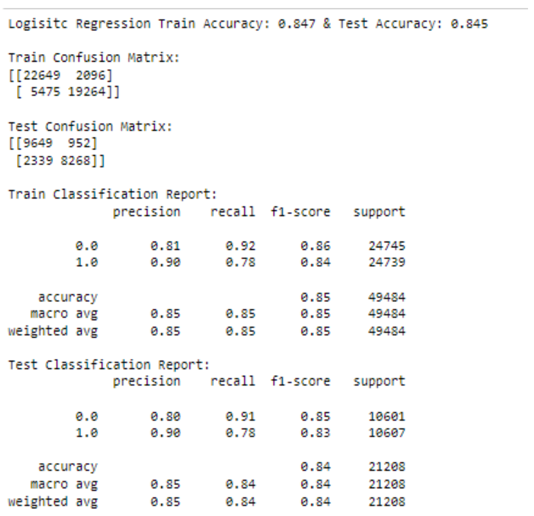

```
KNN Train Accuracy: 0.793 & Test Accuracy: 0.79

KNN Train Accuracy: 0.851 & Test Accuracy: 0.841
```

Logistic Regression

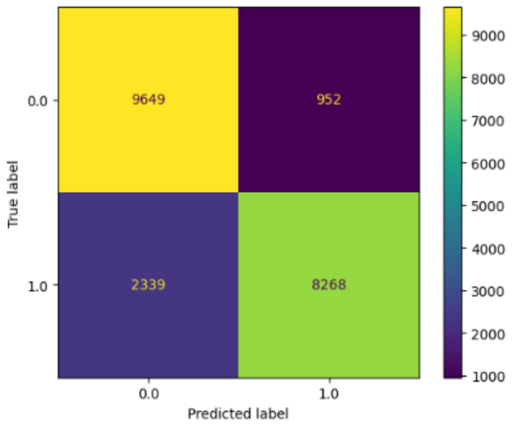
Logistic Regression comes in 3rd place for best model. Although that does not mean that the other models are significantly better. Looking at the numbers SVM is only better by a slight margin. The accuracy for the model was roughly 81% for correlation and 84% with lasso selection. The following image is the classification report and matrix of the model on the correlation and lasso selection respectively.

Grid Search was used to hypertune the model. The parameters that were used to tune the model are as follows: C value, l1_ratio (this was for certain penalties that used this parameter), penalty, random_state, and solver.

Of the params the best overall model is as follows.
C values: 8, l1_ratio: 0, penalty: “l1”, random_state: 42, solver: “saga”



```
Logisitc Regression Train Accuracy: 0.847 & Test Accuracy: 0.845

Train Confusion Matrix:
[[22649  2096]
 [ 5475 19264]]

Test Confusion Matrix:
[[9649  952]
 [2339 8268]]

Train Classification Report:
              precision    recall  f1-score   support

         0.0       0.81      0.92      0.86     24745
         1.0       0.90      0.78      0.84     24739

    accuracy                           0.85     49484
   macro avg       0.85      0.85      0.85     49484
weighted avg       0.85      0.85      0.85     49484

Test Classification Report:
              precision    recall  f1-score   support

         0.0       0.80      0.91      0.85     10601
         1.0       0.90      0.78      0.83     10607

    accuracy                           0.84     21208
   macro avg       0.85      0.84      0.84     21208
weighted avg       0.85      0.84      0.84     21208
```

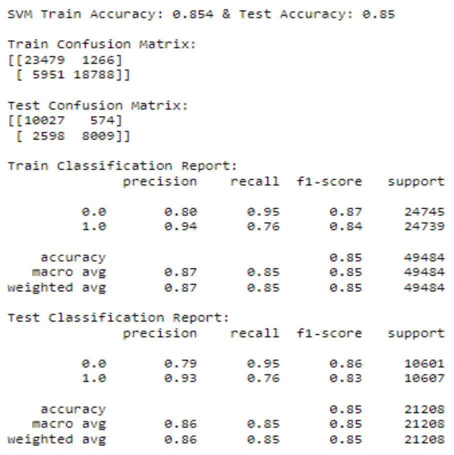


SVM

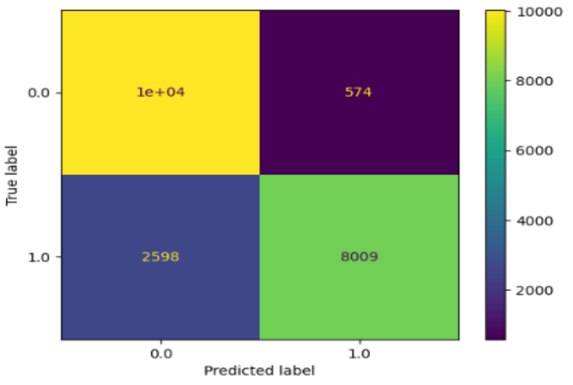
SVM comes in 2nd beating out Logistic Regression by a small margin. The accuracy for the model is roughly 81% for correlation and 85% for lasso selection. I personally do not think that the model is one that should be used unless there is a significant increase in accuracy and not the marginal increase that we see. This is due to the amount of time and computation resources needed to run this model. However, the accuracy score cannot be ignored. The following image is the classification report as well as confusion matrix for the SVM model using the lasso feature selection.

As for hypertuning, grid search was used and the parameters that were tuned were: C value, kernel, degree, and gamma. For the ranges and other information regarding the hyper tuning please look at the Group_20_newmodels file.

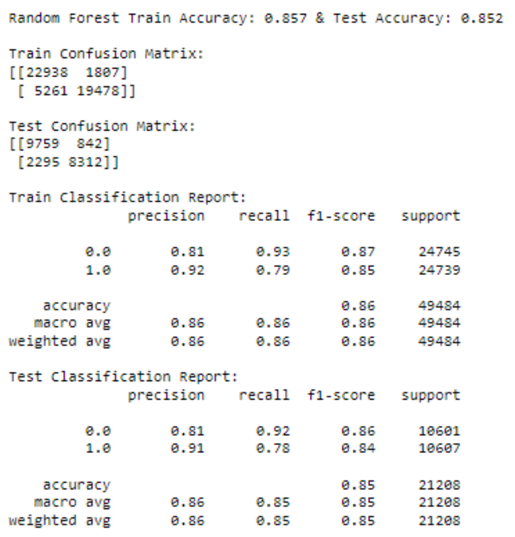
The best overall parameters for the model that yielded the highest scores was.
C:5, degree:3, gamma: “auto”, kernel: “rbf”



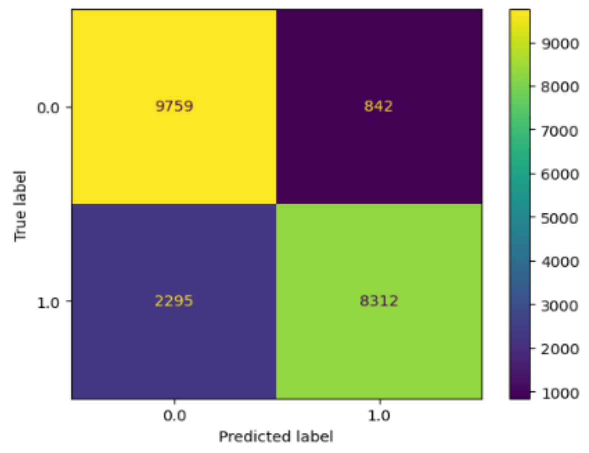
```
SVM Train Accuracy: 0.854 & Test Accuracy: 0.85

Train Confusion Matrix:
[[23479  1266]
 [ 5951 18788]]

Test Confusion Matrix:
[[10027   574]
 [ 2598  8009]]

Train Classification Report:
              precision    recall  f1-score   support

         0.0       0.80      0.95      0.87     24745
         1.0       0.94      0.76      0.84     24739

    accuracy                           0.85     49484
   macro avg       0.87      0.85      0.85     49484
weighted avg       0.87      0.85      0.85     49484

Test Classification Report:
              precision    recall  f1-score   support

         0.0       0.79      0.95      0.86     10601
         1.0       0.93      0.76      0.83     10607

    accuracy                           0.85     21208
   macro avg       0.86      0.85      0.85     21208
weighted avg       0.86      0.85      0.85     21208
```



Random Forest Classifier



```
Random Forest Train Accuracy: 0.857 & Test Accuracy: 0.852

Train Confusion Matrix:
[[22938  1807]
 [ 5261 19478]]

Test Confusion Matrix:
[[9759  842]
 [2295 8312]]

Train Classification Report:
              precision    recall  f1-score   support

         0.0       0.81      0.93      0.87     24745
         1.0       0.92      0.79      0.85     24739

    accuracy                           0.86     49484
   macro avg       0.86      0.86      0.86     49484
weighted avg       0.86      0.86      0.86     49484

Test Classification Report:
              precision    recall  f1-score   support

         0.0       0.81      0.92      0.86     10601
         1.0       0.91      0.78      0.84     10607

    accuracy                           0.85     21208
   macro avg       0.86      0.85      0.85     21208
weighted avg       0.86      0.85      0.85     21208
```



The overall best model that was created from step 1 was Random Forest Tree. This classifier model produced the best results for both feature selection methods. In addition to that the speed at which the model ran was much faster than KNN and SVM. The accuracy of the model was 81% with correlation and 85% with lasso features.

As for hypertuning, grid search was used and the parameters that were measured and paired were: criterion, min_samples_split, min_samples_leaf, max_features, random_state, and max_depth. For this model it was important that random_state is consistent across all tests, this is why it was included in the params. The best parameters for the model are as follows.

Criterion: “entropy”, max_depth: 15, max_features: “sqrt”, min_samples_lead: 17, min_samples_split: “2”, random_state: 42

HyperTuning

As discussed in the section related to the models from step 1, grid search was used for hypertuning. The function that was used was GridSearchCV from sklearn. The reason behind using a function like this is to find the best possible model. While the default parameters for the models are fine it doesn't yield the best possible results. There are even times where the difference in the parameters can make the difference anywhere from 1%-15% accuracy. Depending on the application the model is being used for, 1% can be something large. So, it is important to get the most out of the model possible.

In addition there is also bias and variance that are crucial parts of determining the usefulness of a model. As bias is the error given by real world problems that are too complex for a simple model to predict. Having a high bias model means it is too simple and will yield errors and poor performance on training and testing data. While variance is the opposite of bias and deals with the model being too complex for a simple problem. In short the model with high variance has overfitted and is too sensitive to data, so when exposed to testing data it will perform poorly. These values are something we want to minimize as this will help lead to producing the best overall model. These are some of the values that are looked at by GridSearchCV when determining the best model. As it wants the highest accuracy but also the lowest error (bias and variance).

One key part of GridSearchCV is the cross validation aspect of the function. Cross validation is a method of resampling that to measure how well it will do to independent samples of data. Essentially the data that is being fed to the model is partitioned into a specified number of subsets. In the case of GridSearchCV it uses a method called k-folds. Where the model is validated k times, each time using a different fold as the validation set and the rest are training sets. Overall, cross validation is meant to serve as a reliable way of determining a model's performance.

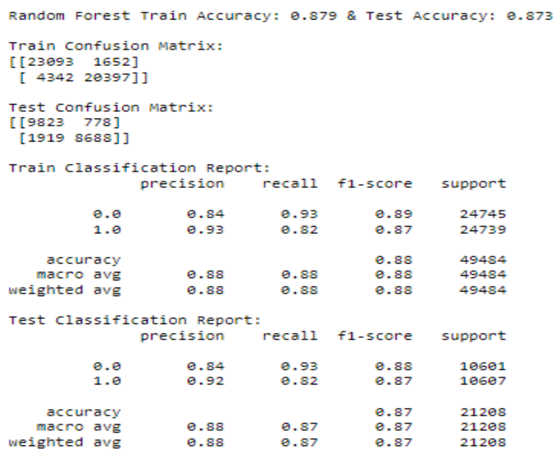
It is important to note that each model was hypertuned twice for the two different feature selection methods that I used. As seen in the “newmodels” file, the changing of features can lead to drastically different models. In order to have the features and models be on an even playing field it was important to create the best possible models for the two feature sets. In turn this was able to produce greater results for the models and the features.

Wrapper Method

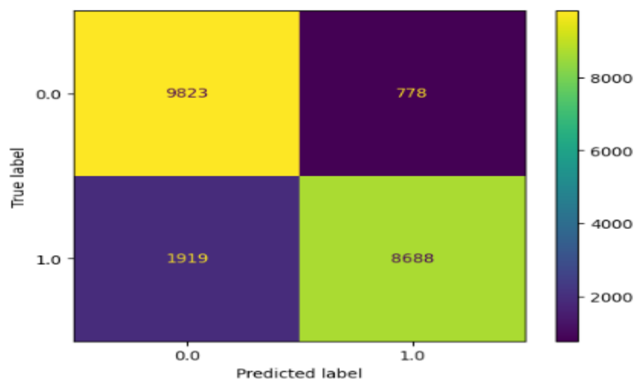
Through some research and the link that was provided I settled on using SequentialFeatureSelector which is a function from sklearn. This function allows you to choose an estimator and the direction for selection. For this I thought it was best to use KNN for the purpose of its ability to identify clusters and grouped trends, and Random Forest Classifier due to its overall performance up to this point. These two models were used for the estimator and there were only two options for direction. Which were backwards and forwards. This meant that there were in total 4 different variable selections that were possible as they all did not yield the same results of what features were selected. (analyticvidhya)

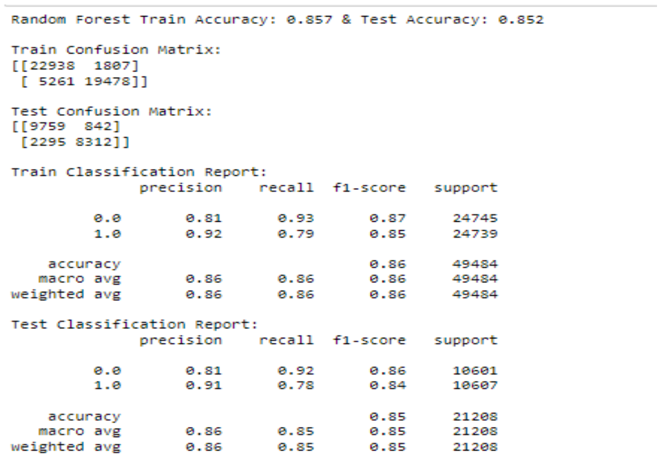
All 4 possible subsets of features were tested using the Random Forest Classifier. Of all 4 test results the best subset was the one produced by the following parameters: Random Forest estimator and forward direction.

From the results it was determined that the features to select were the following 9: BMI, Stroke, HeartDiseaseorAttack, PhysActivity, Veggies, GenHlth, MentHlth, PhysHlth, Income. The following are the results made using these features with the best model from step 1.

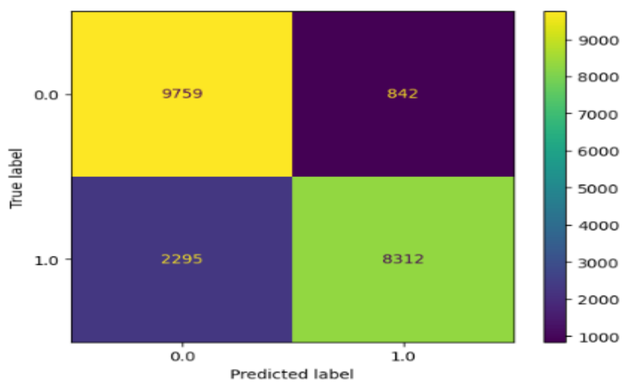


```
Random Forest Train Accuracy: 0.879 & Test Accuracy: 0.873

Train Confusion Matrix:
[[23093  1652]
 [ 4342 20397]]

Test Confusion Matrix:
[[9823  778]
 [1919 8688]]

Train Classification Report:
              precision    recall  f1-score   support

         0.0       0.84      0.93      0.89     24745
         1.0       0.93      0.82      0.87     24739

    accuracy                           0.88     49484
   macro avg       0.88      0.88      0.88     49484
weighted avg       0.88      0.88      0.88     49484

Test Classification Report:
              precision    recall  f1-score   support

         0.0       0.84      0.93      0.88     10601
         1.0       0.92      0.82      0.87     10607

    accuracy                           0.87     21208
   macro avg       0.88      0.87      0.87     21208
weighted avg       0.88      0.87      0.87     21208
```





```
Random Forest Train Accuracy: 0.857 & Test Accuracy: 0.852

Train Confusion Matrix:
[[22938  1807]
 [ 5261 19478]]

Test Confusion Matrix:
[[9759  842]
 [2295 8312]]

Train Classification Report:
              precision    recall  f1-score   support

         0.0       0.81      0.93      0.87     24745
         1.0       0.92      0.79      0.85     24739

    accuracy                           0.86     49484
   macro avg       0.86      0.86      0.86     49484
weighted avg       0.86      0.86      0.86     49484

Test Classification Report:
              precision    recall  f1-score   support

         0.0       0.81      0.92      0.86     10601
         1.0       0.91      0.78      0.84     10607

    accuracy                           0.85     21208
   macro avg       0.86      0.85      0.85     21208
weighted avg       0.86      0.85      0.85     21208
```



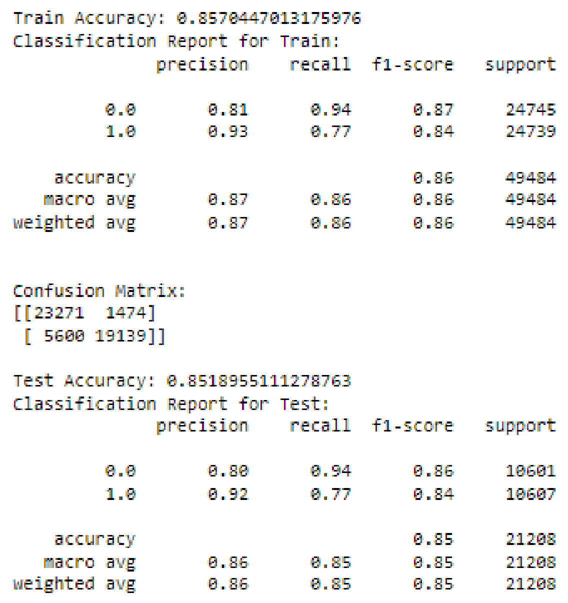
As can be seen the model on the left performs far better than the model on the right. The model on the left is formed through the wrapper method that was used. For the remainder of the report I will be referring to this model as the best model as it will be the best comparison.

Additional Models

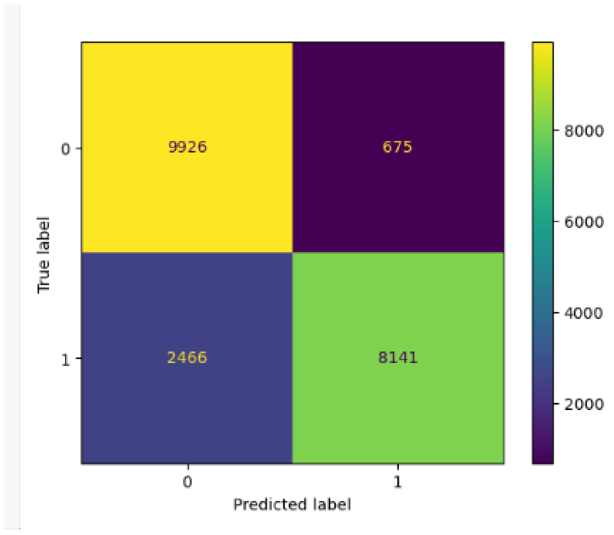
There were 4 additional models that were created which are: XGBoost, Deep Learning Model, Neural Network with 2 hidden layers, and an ensemble model.

XGBoost Model

First model that was created was the XGBoost model. Which is similar to ensemble in the sense that it uses a series of models although this model combines the models into one model to make decisions. It was an interesting experience using this model for the first time as the documentation is quite sparse, although the model is fairly easy to use. Even though it's not necessary I still performed grid search to hyper tune the model. In the end I was given the best possible params which are:



```
Train Accuracy: 0.8570447013175976
Classification Report for Train:
              precision    recall  f1-score   support

         0.0       0.81      0.94      0.87     24745
         1.0       0.93      0.77      0.84     24739

    accuracy                           0.86     49484
   macro avg       0.87      0.86      0.86     49484
weighted avg       0.87      0.86      0.86     49484

Confusion Matrix:
[[23271  1474]
 [ 5600 19139]]

Test Accuracy: 0.8518955111278763
Classification Report for Test:
              precision    recall  f1-score   support

         0.0       0.80      0.94      0.86     10601
         1.0       0.92      0.77      0.84     10607

    accuracy                           0.85     21208
   macro avg       0.86      0.85      0.85     21208
weighted avg       0.86      0.85      0.85     21208
```



```
{'colsample_bytree': 0.5, 'learning_rate': 0.3, 'max_depth': 6, 'min_child_weight': 1, 'objective':
'binary:logistic', 'seed': 42, 'subsamples': 0.5}
```

The parameters above are the best possible parameters for XGBoost model.

Extreme Deep Learning Model

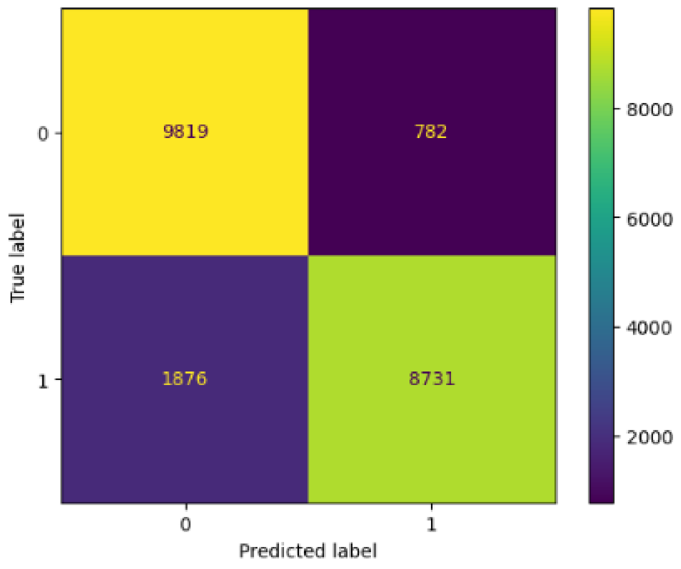
The second model that was created was the Extreme Deep Learning Model. Unlike other deep learning models it does the following: does not utilize back propagation, weights from hidden layers are randomly generated, and the output is linear rather than sigmoidal. At the heart of it the model is still a feed forward neural network. (kaggle)

The results that it yielded were quite impressive as it was 100% accurate. Whether this is due to an error of creation on my end or the model's true performance is unknown. Despite that the model still reaches 100% accuracy on both the Sequential selected data and the lasso selected subset.

Neural Network

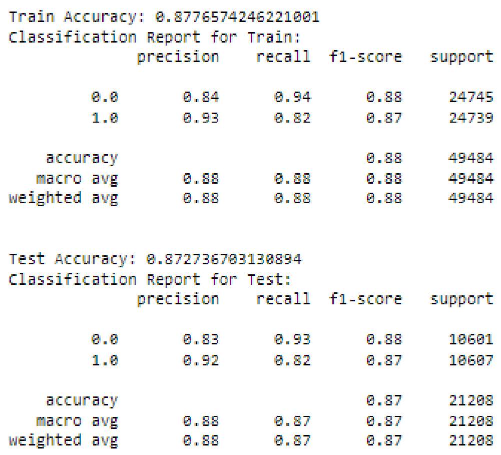
The Third Model that was created was a Neural Network with 2 hidden layers. This model was a basic feed forward neural network that utilized the library tensorflow. In which there is an input layer that feeds into a determined number of hidden layers, then outputs into an output layer.

Using the wrapper selected features the model was able to generate a fairly accurate model of roughly 87%. The following confusion matrix is produced from the model.

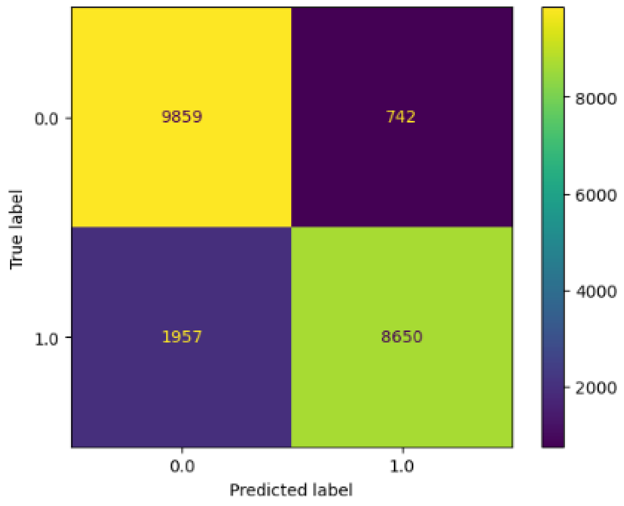


```
Test Accuracy: 0.8746699094772339
Classification Report for Test:
              precision    recall  f1-score   support

         0.0       0.84      0.93      0.88     10601
         1.0       0.92      0.82      0.87     10607

    accuracy                           0.87     21208
   macro avg       0.88      0.87      0.87     21208
weighted avg       0.88      0.87      0.87     21208
```

Ensemble Model



```
Train Accuracy: 0.8776574246221001
Classification Report for Train:
              precision    recall  f1-score   support

         0.0       0.84      0.94      0.88     24745
         1.0       0.93      0.82      0.87     24739

    accuracy                           0.88     49484
   macro avg       0.88      0.88      0.88     49484
weighted avg       0.88      0.88      0.88     49484

Test Accuracy: 0.872736703130894
Classification Report for Test:
              precision    recall  f1-score   support

         0.0       0.83      0.93      0.88     10601
         1.0       0.92      0.82      0.87     10607

    accuracy                           0.87     21208
   macro avg       0.88      0.87      0.87     21208
weighted avg       0.88      0.87      0.87     21208
```



The fourth and final model is an ensemble model. The exact ensemble model that was used was a VotingClassifier. In which a determined number of models would be run, and in this case it was the best performing 3 models from step 1. The ensemble model would then get the predictions from all 3 models and then take a majority vote for deciding the outcome.

On paper you would assume that the model would perform quite well but the performance is actually worse than that of the Random Forest Classifier by itself. As it would have been more effective to run the Random Forest Classifier instead of the ensemble model due to faster computational speed and better accuracy scores. So, it's a no brainer as to why that model would be leagues better.
The image to the side is the results from the ensemble model.

Results

Through all the testing and various models there are some noticeable results. Which are the overall best models and selection methods for this specific data set.

Firstly, it is the best feature selection method for this specific data set. In total there were 3 methods that were used to select the most important features. Those methods being Correlation, LassoCV, and a wrapper method (that being the SequentialFeatureSelector).
In terms of performance correlation was the worst, LassoCV was second best, and the overall best method was wrapper.

As for models, the best performing model is the Extreme Machine Learning Model. Again, it is important to note that I'm unsure of how truthful this model is. However, through the two different subsets that it has been given it performs the exact same for each. Aside from that the following models are also dependable: XGBoost, Ensemble, Random Forest Classifier, and Neural Network. All of these models have a fairly high accuracy and perform well.
If I had to choose one model it would be XGBoost simply due to its ease of use as well as high accuracy and fast computational speed. No the computational speed may differ from machine to machine but when multithreading on my personal machine the speed was quite fast.

Links

Github repository link:
https://github.com/Beyond-Image/EDS-Project

References

Extreme Learning Model References:
https://towardsdatascience.com/introduction-to-extreme-learning-machines-c020020ff82b

(kaggle)
https://www.kaggle.com/code/robertbm/extreme-learning-machine-example

Data Set Reference:
https://www.kaggle.com/datasets/alexteboul/diabetes-health-indicators-dataset?select=diabetes_binary_health_indicators_BRFSS2015.csv

XGBoost Reference:
https://www.simplilearn.com/what-is-xgboost-algorithm-in-machine-learning-article#:~:text=XGBoost%20is%20a%20robust%20machine,optimize%20their%20machine%2Dlearning%20models.

https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/

SequentualFeatureSelector Reference:
(analyticvidhya)
https://www.analyticsvidhya.com/blog/2020/10/a-comprehensive-guide-to-feature-selection-using-wrapper-methods-in-python/

https://scikit-learn.org/stable/auto_examples/feature_selection/plot_select_from_model_diabetes.html