BINF610 Final Report: Breast Cancer Classification

Introduction:

For my final project, I chose to focus on a dataset procured by the University of Wisconsin faculty. Specifically, the creators of this dataset include Dr. William H. Wolberg, W. Nick Street, and Olvi L. Mangasaria. Within this dataset, I have chosen to work with a breast cancer-related dataset to solve the problem of classifying breast cancer as either benign or malignant. The dataset I'm using is the Breast Cancer Wisconsin (Diagnostic) dataset, which contains 569 observations and 32 features derived from fine needle aspirate (FNA) samples of breast masses. The goal of this project is to develop an accurate machine learning model that can classify breast cancer cases based on the provided features.

Introduction to Methodology and Tools

To address this problem, I employed various machine learning techniques using Python as my primary programming language. The tools and libraries I used included NumPy, Pandas, Scikit-Learn, Matplotlib, and others. Within my project, I performed the following analyses:

- Random Forest Classifier: I created a random forest classifier model and trained and tested it on the labeled data from the dataset. I first started off by applying this model with default values. After this, I then tuned this model in an attempt to further optimize its performance.

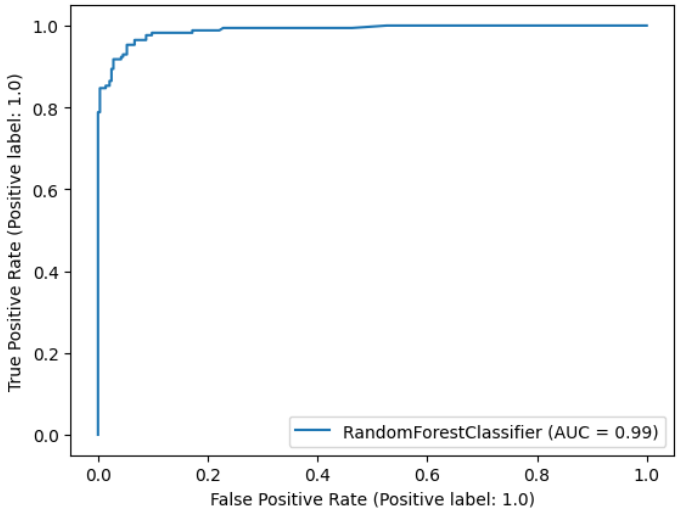
- Hyperparameter Tuning: Throughout my final project, I tuned each model's hyperparameters by leveraging the grid search function within the Scikit-Learn package. This allowed me to find the best combination of hyperparameter values that led to the creation of the most optimal model.

- Boosting: Within my final project, I implemented various boosting models to further enhance classification performance. Specifically, I used a Gradient Boosting Classifier Model, XGBoost Model, and an AdaBoosted Model.

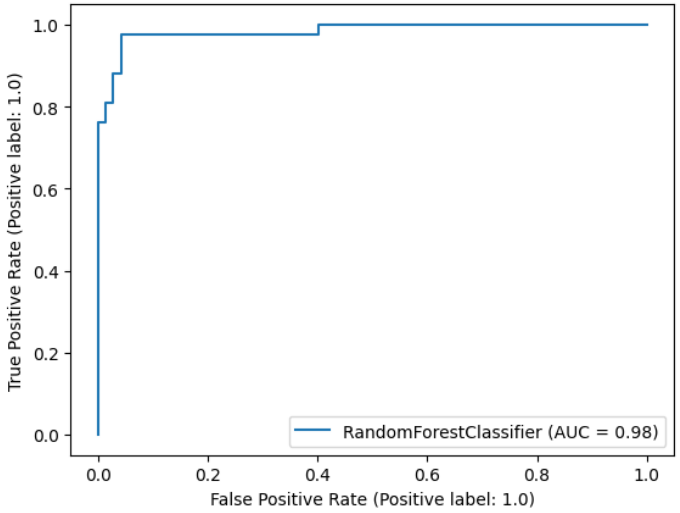
Random Forest Classifier:

To start, I first built a Random Forest Model. In this initial application of the algorithm, I chose to use Sklearn's default model architecture. By doing this, I was able to achieve a training accuracy score of 94.5% and a test accuracy of 93.9%.

Random Forest Classifier Model AUC training curve:



Random Forest Classifier Model AUC test curve:



Based on the learning curves above, we can infer that the model is performing well. The Random Forest Classifier Model exhibits a high area under the curve value, indicating its effectiveness. Additionally, the model demonstrates a high true positive rate, further indicating that it's performance is well.

After creating a model with the default architecture, I aimed to further enhance its performance. To achieve this, I implemented a grid search method to fine-tune the model's parameters. For this particular model, I tested parameters such as the number of estimators, max depth, minimum samples split, and class weighting. Including regularization methods like max depth and minimum samples split was crucial to ensure the tuned model's ability to generalize well to unseen testing data. As a result, I obtained an optimal model with a training accuracy of 99.1% and a test accuracy of 96.5%.

Given the high accuracy values, I had concerns about overfitting. To investigate this, I created a classification report and performed cross-validation to evaluate the model's train and test precision macro scores, as well as recall macro scores. I chose these scoring methods because, in the context of this dataset, assessing the model's performance based on precision and recall scores is more valuable than accuracy. The following results were obtained:

Tuned Random Forest Classifier Model Test Classification Report:

	precision	recall	f1-score	support
0.0	0.99	0.96	0.97	72
1.0	0.93	0.98	0.95	42
accuracy			0.96	114
macro avg	0.96	0.97	0.96	114
weighted avg	0.97	0.96	0.97	114

Tuned Random Forest Classifier Model Cross Validation Scores:

```
#run a cross validation of 5 for the tuned rfc model on training scores:
scoring = ['precision_macro', 'recall_macro']
cv = 5

scores = cross_validate(rfc, X_train, y_train, scoring=scoring, cv=cv)

scores['test_recall_macro'].mean(), scores['test_precision_macro'].mean()
```

```
(0.9430856553147574, 0.9408329815786847)
```

```
#run a cross validation of 5 for the tuned rfc model on test scores:
scoring = ['precision_macro', 'recall_macro']
cv = 5

scores = cross_validate(rfc, X_test, y_test, scoring=scoring, cv=cv)

scores['test_recall_macro'].mean(), scores['test_precision_macro'].mean()
```

```
(0.930515873015873, 0.9242521367521366)
```

As we can observe from the screenshots provided above, the tuned Random Forest Classifier Model demonstrates good generalization to unseen data. I base this conclusion on the similarity between the model's train and test cross-validated precision and recall scores.

Implementing Boosting Methods:

After constructing the Random Forest Classification model and performing tuning, my objective was to develop a more accurate model for classifying benign and malignant cases. To accomplish this, I created a Gradient Boosted Classifier model, an XGBoost Model, and an AdaBoost model.

Starting with the Gradient Boosted Classifier model, similar to my previous implementation of the Random Forest Model, I began with the model's default architecture. By doing so, I attained a training accuracy of 100% and a test accuracy of 96.5%.

In an effort to further enhance the Gradient Boosted Classifier model, I incorporated hyperparameter tuning. For this model, I focused on tuning the learning rate and minimum samples split. Once again, just like the previous Random Forest Classifier model, I included a regularization parameter to mitigate the risk of overfitting the tuned model. After completing the tuning process, the model achieved a training accuracy of 100% and a test accuracy of 98.2%.

Gradient Boosted Classifier Model Test Classification Report:

	precision	recall	f1-score	support
0.0	0.99	0.99	0.99	72
1.0	0.98	0.98	0.98	42
accuracy			0.98	114
macro avg	0.98	0.98	0.98	114
weighted avg	0.98	0.98	0.98	114

Gradient Boosted Classifier Model Cross Validated Scores:

```
#run a cross validation of 5 for the tuned gradient boosting classifier model on training scores:
scoring = ['precision_macro', 'recall_macro']
cv = 5

scores = cross_validate(clf, X_train, y_train, scoring=scoring, cv=cv)

scores['test_recall_macro'].mean(), scores['test_precision_macro'].mean()
```

```
(0.951857585139319, 0.955751510465506)
```

```
#run a cross validation of 5 for the tuned gradient boosting classifier model on test scores:
scoring = ['precision_macro', 'recall_macro']
cv = 5

scores = cross_validate(clf, X_test, y_test, scoring=scoring, cv=cv)

scores['test_recall_macro'].mean(), scores['test_precision_macro'].mean()
```

```
(0.9549603174603174, 0.9518315018315018)
```

To validate the high accuracy of the model, I created a test classification report and implemented cross-validation testing. The provided screenshots above demonstrate that the tuned Gradient Boosted Classifier Model performs well and exhibits good generalization to unseen data. The model's cross-validated precision and recall scores, both for the train and test sets, remain

consistently high. These results further reinforces my belief that the tuned model effectively generalizes to unseen data without being overfit.

Moving on from the Gradient Boosted Classifier model, I proceeded to develop an XGBoost Model. Similarly to the previous models, I initially utilized the XGBoost model with its default architecture. With this default model, I achieved a training accuracy score of 100% and a test accuracy score of 98.2%.

Following the creation of the default model, I performed tuning by employing a parameter grid search. The parameters I selected for further optimizing the model included the learning rate and maximum depth. Once again, I incorporated a regularization attribute like maximum depth to mitigate the risk of overfitting in the tuned model. Consequently, I attained a training accuracy score of 100% and a test accuracy score of 98.2% with the tuned model.

Tuned XGBoost Model Test Classification Report:

	precision	recall	f1-score	support
0.0	0.99	0.99	0.99	72
1.0	0.98	0.98	0.98	42
accuracy			0.98	114
macro avg	0.98	0.98	0.98	114
weighted avg	0.98	0.98	0.98	114

Tuned XGBoost Model Cross Validated Scores:

```
#run a cross validation of 5 for the tuned XGBoost model on training scores:
scoring = ['precision_macro', 'recall_macro']
cv = 5

scores = cross_validate(xg_cl, X_train, y_train, scoring=scoring, cv=cv)

scores['test_recall_macro'].mean(), scores['test_precision_macro'].mean()
```

```
(0.9618679050567595, 0.9588237751240847)
```

```
#run a cross validation of 5 for the tuned XGBoost model on test scores:
scoring = ['precision_macro', 'recall_macro']
cv = 5

scores = cross_validate(xg_cl, X_test, y_test, scoring=scoring, cv=cv)

scores['test_recall_macro'].mean(), scores['test_precision_macro'].mean()
```

```
(0.944325396825397, 0.9463888888888888)
```

Based on the above observations, it is evident that the tuned XGBoost model performs well on unseen data. Once again, I reason this from the relative stability of the cross-validated precision and recall scores. The absence of a significant difference between the training and test precision and recall scores reinforces the model's reliability.

Following this, I proceeded to implement an AdaBoosted Model. Initially, I chose to employ the XGBoost Model as the estimator within the AdaBoosted model. However, this configuration resulted in a training accuracy score of 62.6% and a test accuracy score of 63.2%. Because of this, I opted to utilize the next most accurate model, which was the tuned Gradient Boosted Classifier model, as the estimator within the AdaBoosted Model. By making this adjustment, I achieved a training accuracy of 100% and a test accuracy of 97.4%.

After constructing the AdaBoosted Model, I conducted additional hyperparameter tuning. The parameters I focused on in the parameter grid search were the learning rate and the minimum samples leaf for the estimator. Once again, I included a regularization parameter such as minimum samples leaf to mitigate the risk of overfitting in the tuned AdaBoosted Model. Following the grid search, the optimal AdaBoosted Model achieved a training accuracy of 100% and a test accuracy of 97.4%.

Tuned AdaBoosted Model Test Classification Report:

	precision	recall	f1-score	support
0.0	0.97	0.99	0.98	72
1.0	0.98	0.95	0.96	42
accuracy			0.97	114
macro avg	0.97	0.97	0.97	114
weighted avg	0.97	0.97	0.97	114

Tuned AdaBoosted Model Cross Validated Scores:

```
#run a cross validation of 5 for the tuned Adaboost model on training scores:
scoring = ['precision_macro', 'recall_macro']
cv = 5

scores = cross_validate(ada_model, X_train, y_train, scoring=scoring, cv=cv)

scores['test_recall_macro'].mean(), scores['test_precision_macro'].mean()
```

```
(0.9495356037151703, 0.9480094117567551)
```

```
#run a cross validation of 5 for the tuned Adaboost model on test scores:
scoring = ['precision_macro', 'recall_macro']
cv = 5

scores = cross_validate(ada_model, X_test, y_test, scoring=scoring, cv=cv)

scores['test_recall_macro'].mean(), scores['test_precision_macro'].mean()
```

```
(0.8752380952380951, 0.8880203293438587)
```

Unlike the other models, which showed smaller discrepancies between the training and test precision and recall scores, this tuned model displayed a larger difference in scores. As a result of this, the larger disparity could indicate that the model struggles to generalize effectively to unseen data and may be overfit.

Most Accurate Model

After creating multiple models, the tuned Gradient Boosting Model proved to be the most accurate, achieving a training accuracy of 100% and a test accuracy of 97.4%

Tuned Gradient Boosted Classifier Model Architecture:

```
clf = GradientBoostingClassifier(learning_rate=0.701,
                                 min_samples_split=70, n_estimators=96,
                                 random_state=1234)
```

```
clf.fit(X_train, y_train)
clf.score(X_train, y_train), clf.score(X_test, y_test)
```

```
(1.0, 0.9736842105263158)
```

Despite achieving high accuracy scores with the tuned Gradient Boosted Classifier Model, I felt it was important to further explore the reliability of these scores. To accomplish this, I decided to conduct a 10-fold cross-validation test on the model's train and test accuracy scores.

Tuned Gradient Boosted Classifier Model Cross Validated Accuracy Scores:

```
scores = cross_val_score(clf, X_train, y_train, cv=10)
scores
```

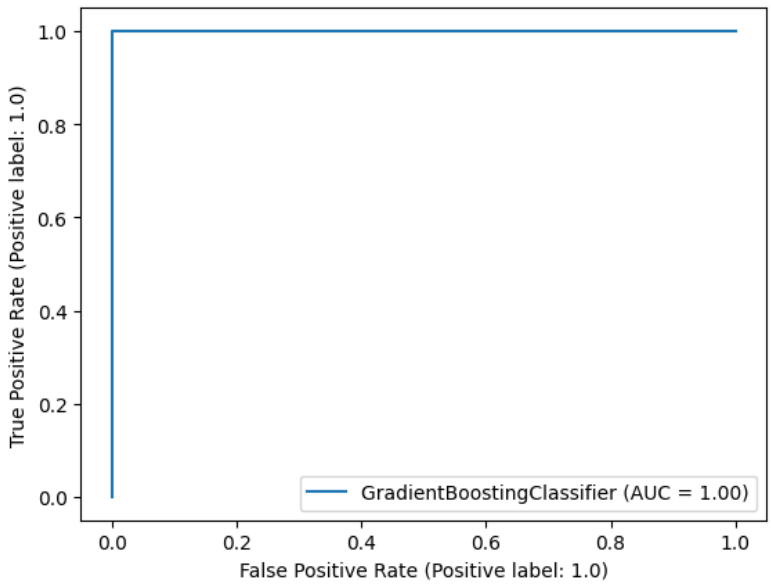
```
array([0.93478261, 0.95652174, 0.95652174, 0.97826087, 0.97826087,
       0.97777778, 0.93333333, 0.95555556, 0.91111111, 0.88888889])
```

```
scores = cross_val_score(clf, X_test, y_test, cv=10)
scores
```

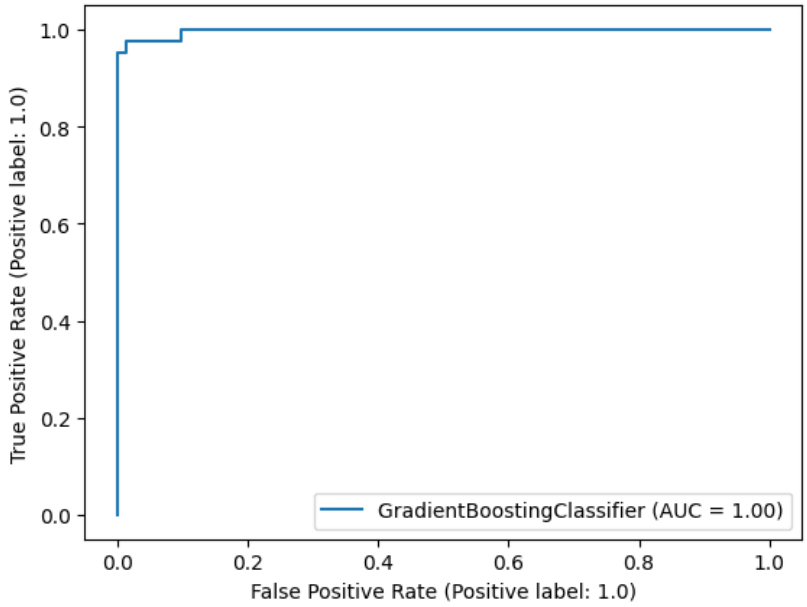
```
array([1.        , 1.        , 1.        , 1.        , 0.90909091,
       1.        , 0.81818182, 0.81818182, 0.81818182, 0.90909091])
```

In the above screenshot, we can see that the cross-validated accuracy scores exhibit similarity between the train and test results. While this model is not perfect, the cross validated scores demonstrate the ability of the model to generalize well to unseen data.

Tuned Gradient Boosted Classifier AUC Training Curve:



Tuned Gradient Boosted Classifier AUC Test Curve:



Tuned Gradient Boosted Classifier Test Classification Report:

	precision	recall	f1-score	support
0.0	0.99	0.97	0.98	72
1.0	0.95	0.98	0.96	42
accuracy			0.97	114
macro avg	0.97	0.97	0.97	114
weighted avg	0.97	0.97	0.97	114

Tuned Gradient Boosted Classifier Confusion Matrix:

- [[71 2]
- [1 41]]

Given the above, this means that the the model has:

- True Positive (TP) 71
- True Negative (TN) 41
- False Positive (FP) 2 --> predicted that the person has cancer, but they actually do NOT
- False Negative (FN) 1 --> predicted that the person does not have cancer, but they actually do

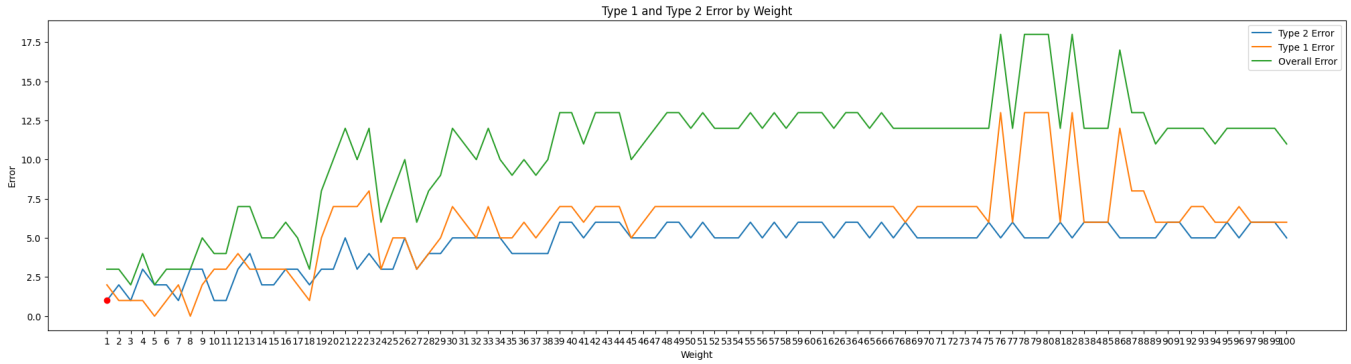
From the above screenshots of the AUC curves, test classification report, and the confusion matrix, we can conclude that the tuned Gradient Boosted Classifier Model is performing well. Within the context of the confusion matrix, we can see that with the tuned model, we only

misclassified 3 observations out of the entire test set. Diving even deeper into these results, only 1 observation was a false negative, which in the context of this specific dataset would be the worst-case scenario - I would rather treat someone for cancer and for them to not actually have it (Type 1 error - False Positive) rather than turn someone away telling them that they don't have cancer and not treating them when in reality they actually do (Type 2 error - False Negative). Because of this model's results, I was motivated to further implement sample weighting in an attempt to reduce the tuned model's output of type 2 errors as much as possible with the goal of eliminating them.

Further optimizing tuned Gradient Boosted Classifier by implementing sample weighting

```
type2scores_list = []
type1scores_list = []

#declare weights to try for sample_weights:
weights_to_try = np.arange(1, 101, 1).tolist()

#create for loop to try each weight:
for weight in range(0, len(weights_to_try)):
    clf.fit(X_train, y_train, sample_weight=[1 if i == 0 else weights_to_try[weight] for i in y_train])
    y_pred = clf.predict(X_test)

    #print out weight and it's cm:
    print(f"Weight: {weights_to_try[weight]}")
    print(confusion_matrix(y_test,y_pred))

    #for blank space inbetween each cm:
    print("")

    type2score = confusion_matrix(y_test,y_pred)[1, 0]
    type2scores_list.append(type2score)

    type1score = confusion_matrix(y_test,y_pred)[0, 1]
    type1scores_list.append(type1score)
```

To achieve the reduction in type 2 errors, I created a function that tried 100 different sample weightings. These sample weights ranged from values of 1 to 100. After that, I then fit the model based on the sample weighting and trained and tested it. Within the screenshot below, you can see the results of how changing the sample weighting affected the model's errors.



From the visual above, we can see that our goal of prioritizing the reduction of type 2 errors while also considering the overall combined type 1 and type 2 errors is achieved by applying a sample weight of 3 for every predictor value that is malignant. After discovering this, I implemented the sample weighting within the tuned Gradient Boosted Classifier Model, as shown in the screenshots below.

Implementing optimal sample weight within tuned Gradient Boosted Classifier Model:

```
clf = GradientBoostingClassifier(learning_rate=0.701,
                                 min_samples_split=70,
                                 n_estimators=96,
                                 random_state=1234)
```

✓ 0.1s

```
clf.fit(X_train, y_train, sample_weight=[1 if i == 0 else 3 for i in y_train])
```

✓ 0.7s

```
GradientBoostingClassifier(learning_rate=0.701, min_samples_split=70,
                           n_estimators=96, random_state=1234)
```

```
clf.score(X_train, y_train, sample_weight=[1 if i == 0 else 3 for i in y_train])
```

✓ 0.1s

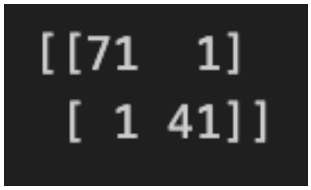
```
1.0
```

```
clf.score(X_test, y_test, sample_weight=[1 if i == 0 else 3 for i in y_test])
```

✓ 0.1s

```
0.9797979797979798
```

Tuned Gradient Boosted Classifier Model including the optimal sample weight to reduce errors:



The above represents the confusion matrix that was produced by implementing the optimal sample weight into the tuned Gradient Boosted Classifier Model. We can observe that we were able to further reduce the errors, prioritizing the reduction of type 2 (false negative) errors first.

Conclusion

In conclusion, in this project, I aimed to develop a breast cancer classification model using the Breast Cancer Wisconsin Diagnostic dataset. To accomplish this, I employed various machine learning techniques such as Random Forest Classifier, Boosting, and Hyperparameter Tuning. The models' performance was evaluated using ROC curves, cross-validation, classification reports, and confusion matrices. Ultimately, I successfully created a reliable and accurate model that minimizes type 2 errors, making it potentially applicable in clinical practice for breast cancer diagnosis.

References:

- Dataset: Breast Cancer Wisconsin (Diagnostic)
  Source: UCI Machine Learning Repository
  URL: https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(diagnostic)

- Documentation: scikit-learn - GradientBoostingClassifier
  Source: scikit-learn.org
  URL:
  https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

- Documentation: scikit-learn - RandomForestClassifier
  Source: scikit-learn.org
  URL:
  https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

- Documentation: scikit-learn - AdaBoostClassifier
  Source: scikit-learn.org

URL:
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

- Documentation: XGBoost - Setting Parameters
  Source: xgboost.readthedocs.io
  URL:
  https://xgboost.readthedocs.io/en/stable/python/python_intro.html#setting-parameters

- Documentation: scikit-learn - GridSearchCV
  Source: scikit-learn.org
  URL:
  https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html