# Project 4: XGBoost and SVM Model Exploration with Breast Cancer Recurrence Dataset

By: Anna Victoria Lavelle and Antonio Jimenez

## Introduction:

For our final project we decided to explore the advanced classical algorithms XGBoost and Support Vector Machines for tabular data. Specifically, we explored how they compared to Random Forest classifier, K-Nearest Neighbor (KNN), and Multinomial Naive Bayes models from Project 2 when applied to the breast cancer recurrence dataset. This dataset was chosen due to the challenge of trying to balance accuracy versus recall as both model properties have large implications on breast cancer patients. While both models can be applied to regression problems, we only explore their capabilities in classifying whether or not a patient will experience recurrence. Additionally, we will be creating an inference server to allow easy deployment of the advanced classical algorithm models, which will be containerized using Docker.

## Technologies Used:

XGBoost, also known as Extreme Gradient Boosting, functions by combining several weak decision tree models to generate an improved model using gradient descent to optimize the final model. In other words, it creates a model, makes predictions, calculates loss, trains a model to reduce calculated errors, and then appends a new model until reaching an optimized final model. Due to the amount of support surrounding XGBoost and its popularity, the open-source library surrounding the advanced classical algorithm has numerous features such as smart handling of missing data, in-built cross-validation, parallel training, and more.

Support Vector Machines (SVM), similarly to XGBoost, can be used for both classification and regression problems. In its simplest form it finds a line between points to categorize the data, similarly to K-nearest neighbor. It differs from K-nearest neighbor in the fact that the algorithm attempts to find the optimal decision boundary that results in the largest margin between points of either class. When the data is not easily linearly separable the kernel trick is applied in order to transform the data into a higher-dimensional space where it is separable. Different kernel methods like linear, polynomial, and gaussian radial basis perform better based on the nature of the data.

## Data Exploration:

We will be using the dataset from Project 2 that was provided through the class GitHub. The data can be found here, as well as other websites like Kaggle and OpenML. Given that the data was previously explored in Project 2 we will reiterate our findings here. When analyzing the shape of the dataset we see that it contains 286 rows and 10 columns. Using the .head function we see that the column titles are class, age, menopause, tumor-size, inv-nodes, node-caps, deg-malig, breast, breast-quad, and irradiat. The function .info was used to evaluate properties of the data such as data type and number of non-null values. Through this we found that the data contained no null values. Upon further investigation we found that while the data contained "?" for missing data.

## Data Preparation:

Data preparation for both XGBoost and SVM models was the same as with the models from Project 2 and will be reiterated in this section. We began by removing rows with missing data. This was done for the node-caps and breast-quad columns. We then proceeded to prepare the data for one-hot encoding. This was done by converting the type of the class, menopause, node-caps, breast, breast-quad, and irradiat to type category. We then verified the type cast using the .info function. Lastly, the data contained integer ranges or binned values for the age, tumor-size, and inv-nodes which were transformed into floats by taking the average of the upper and lower limit of the bin range. Lastly we deleted duplicate rows.

While we did not need to make any changes to how we prepared the data, it should be noted that XGBoost does allow you to store data in a DMatrix which is optimized for memory and speed. Additionally, XGBoost has a built-in feature that takes care of columns that require one-hot encoding.

## Uni and Multivariate Plots:

When analyzing the data we saw that there were no values that seemed improbable and were worth eliminating. From the univariate plots generated we see that there were numerous large outliers in the "inv-nodes" column. We also see that most tumors range from sizes of 20 to 35mm with some outliers. From creating a histogram of the patients age, we see that it follows a standard distribution with a mean around the age of 50. We found that most of the cases contain a degree of malignancy that is equal to two. Finally, using a heat map to display correlation between the different columns. From this plot we found that recurrence is slightly positively correlated to invasive nodes, tumor

size, degree of malignancy, node capsule, and irradiation. These plots can be found in Project2 or SVM.ipynb

## Results:

Having explored and prepared the data we performed a 70/30 train-test split with stratify applied to the dependent variable. As in Project 2 we selected “class-reccurence-events" as our dependent variable that we are trying to predict. The rest of the columns were selected to be the dependent variable. We then proceeded to create the SVM and XGB models. For the XBG model a parameter grid was used with the following parameters: {'learning_rate': [0.01, 0.1, 0.2], 'max_depth': [3, 5, 7], 'min_child_weight': [1, 3, 5], 'subsample': [0.8, 0.9, 1.0], 'colsample_bytree': [0.8, 0.9, 1.0], 'gamma': [0, 0.1, 0.2], 'reg_alpha': [0, 0.1, 0.5], 'reg_lambda': [1, 1.5, 2]}. As XGB functions through trees, we see similar parameters to the random forest model. SVM’s grid search contained the following hyper-parameters:

{ "C" : np.arange(start=1, stop=3), "kernel" : ["poly","rbf","sigmoid"], "gamma" : ["scale","auto"] }.

For both SVM and XGB we trained one model optimized for recall and one for accuracy. The SVG model optimized for recall achieved a recall of 0.26 and an accuracy of 0.63 on the test data. We see that the model was somewhat overfit as the train accuracy was 0.83. Regularizing the data in the pre-processing could have been done to reduce overfitting and more extensive tuning of the hyper-parameters. For this model the hyper-parameters chosen by the grid search were:

{'C': 2, 'gamma': 'auto', 'kernel': 'poly'}.

The SVM model optimized for accuracy had a recall of 0.09 and an accuracy of 0.67 for the test dataset. The model contained a recall of 0.22 and an accuracy of 0.75 for training data. While the model optimized for accuracy experienced less overfitting, it achieved a recall that is significantly lower than that of the recall SVM model. This model is characterized by the hyper-parameters:

{'C': 2, 'gamma': 'scale', 'kernel': 'poly'}.

Given the non-linear nature of the data, we were not surprised that the kernel “poly” performed the best for both models. The XGB model trained for recall achieved a recall of 0.26 and an accuracy of 0.72 for test data, which performed better than both of the SVM models. This model, however, was largely overtrained as it achieved 0.67 recall

and 0.9 accuracy on the training data. The hyper-parameters chosen by the grid search were:

{'colsample_bytree': 0.9, 'gamma': 0, 'learning_rate': 0.2, 'max_depth': 3, 'min_child_weight': 3, 'reg_alpha': 0.1, 'reg_lambda': 2, 'subsample': 0.9}

The XGB model trained for accuracy achieved a recall of 0.22 and an accuracy of 0.7 for test data, and a recall of 0.48 and accuracy 0.84 for training data. The hyper-parameters chosen were:

{'colsample_bytree': 0.8, 'gamma': 0, 'learning_rate': 0.2, 'max_depth': 4, 'min_child_weight': 3, 'reg_alpha': 0, 'reg_lambda': 2, 'subsample': 1.0}.

**Table 1: Models and their recall/accuracy.**

| Model | Recall (test data) | Accuracy (test data) |
|---|---|---|
| SVM - Accuracy | 0.09 | 0.67 |
| SVM - Recall | 0.26 | 0.63 |
| XGB - Accuracy | 0.22 | 0.7 |
| XGB - Recall | 0.26 | 0.72 |
| Random Forest - Recall | 1.00 | 0.3 |
| KNN - Recall | 0.22 | 0.7 |
| Multi-Naive Bayes - Recall | 0.26 | 0.66 |

As seen from Table 1 a comparable model was achieved with both SVM and XGB. It is worth noting that excluding the Random Forest model due to its unique property of having a recall of 1, an improved well rounded model was generated.The XGB-recall model has a higher or equal recall while achieving higher accuracy. We were surprised to see the XGB-recall experienced higher accuracy than the XGB-accuracy model. To ensure fair comparison, the same preprocessing was done on all of the models and the same random_state was used when creating the test train split.

## MLOps:

An inference server was created to allow users to easily submit data to the 4 models generated using the advanced classical algorithms. The models were saved and loaded into the flask app using the pickle library. The user is then able to get information about the different models through GET request which provides the name, model type, accuracy on test data, recall on test data, and if it is optimized for accuracy or recall. Before making a POST request to one of the models the user must post the data by passing in a csv file that has been preprocessed to fit the model. Once the data has been loaded in the user is able to create a POST request which will return the predictions of the model. A list of all the possible routes can be found in Table 2. The inference server was then containerized using Docker allowing the user to easily use the inference server from their machine.

**Table 2: Inference server routes.**

| Route | Method | Description |
|---|---|---|
| /models | POST | Loads in the data from the provided csv file |
| /models/acc_xgb | GET | Returns information of the acc_xgb model |
| /models/acc_xgb | POST | Returns classifications of the data using acc_xgb model |
| /models/recall_xgb | GET | Returns information of the recall_xgb model |
| /models/recall_xgb | POST | Returns classifications of the data using recall_xgb model |
| /models/acc_svm | GET | Returns information of the acc_svm model |
| /models/acc_svm | POST | Returns classifications of the data using acc_svm model |
| /models/recall_svm | GET | Returns information of the recall_svm model |
| /models/recall_svm | POST | Returns classifications of the data using recall_svm model |

## Conclusion:

The main goal of this project was to explore the advanced classical algorithms Support Vector Machines and XGBoost and implement models to tackle the breast cancer dataset from Project 2. Two models were created per algorithm, one optimized for recall and the other for accuracy. The results were compared to a Random Forest, KNN, and Multi-Naive Bayes model, where we saw that we were able to create an improved model using XGB optimized for recall.

Through this process we were able to learn about the numerous features built around these advanced classical algorithms. While we did not explore them here, it would be interesting to evaluate the training efficiency of these models, parallelism, DMatrix utilization, and other features.

When completing this project we found that there were many ways to tune and enhance the performance of such models. XGB, for example, can be optimized to reduce overfitting. Furthermore, we had originally trained the advanced classical models with data that was pre-processed by replacing missing data with the most common value instead of deleting the row. By replacing the missing data we are able to retain more data, which we believe might be responsible for the enhanced performance. The XGB model trained for accuracy achieved a recall of 0.33 and accuracy of 0.74. The XGB model trained for recall experienced a recall of 0.42 and accuracy of 0.76. While smaller, an improvement was also seen in the SVM models. The large improvement seen within the XGB model displays that there is a lot more to be explored in terms of maximizing the performance of these algorithms.

Lastly, given that Random Forest and XGB are based on decision trees, it would be interesting to further investigate the differences in the prediction. The same can be said for KNN and SVM as they both work to create decision boundaries. Combining these different models would be interesting to see as well. Specifically, a model such as Random Forest which has low accuracy but perfect recall paired up with a well rounded and accurate model like XGB-recall. We believe that by combining these two models we might be able to increase the recall of XGB-recall while minimizing the impact it would have on its accuracy.

ChatGPT was used to debug the inference server and figure out how to feed the batch of data to the server. Additionally, it was used to gain an understanding of how the advanced classical algorithms functioned. This included deciding the hyper-parameters to feed the grid search.

## Resources:

SVM:
https://scikit-learn.org/stable/modules/svm.html
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

XGB:
https://xgboost.readthedocs.io/en/stable/
https://www.datacamp.com/tutorial/xgboost-in-python
https://www.geeksforgeeks.org/xgboost/

Data:
https://raw.githubusercontent.com/joestubbs/coe379L-sp24/master/datasets/unit02/project2.data