Assignment 1

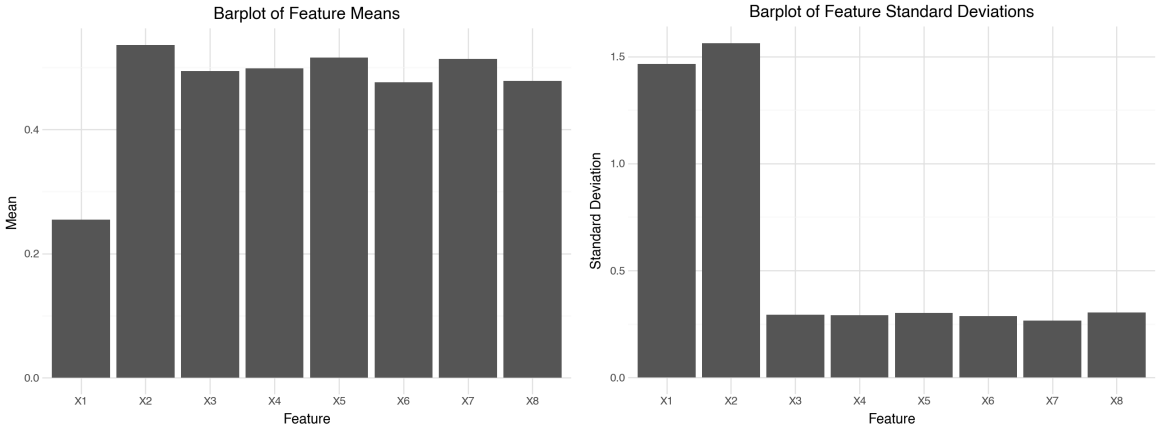
Kavin Ravi

Introduction

This assignment involves using several classification algorithms on a dataset in order to investigate which performs optimally. Methods include SVM (Support Vector Machines), Logistic Regression, and KNN (K-Nearest Neighbors), with Grid Search Cross-Validation for hyperparameter tuning. The dataset includes 1,000 entries of presumably randomly or formulaically generated data across 8 independent variables, X1-8 (also referred to as "features" or "predictors") and 1 dependent variable, Y (also referred to as the "target" variable). All predictor columns were continuous numeric variables, with the target being a binary column.

Analysis

Two statistics I will use for feature assessment is to compare the means and standard deviations (variances) of each feature, which could help determine the proper scaling method and any pre-preparation of data necessary before or during model construction. Below are the plots of the data means and standard deviations, respectively:



The above plots reveal some interesting details that weren't apparent with a simple summary statistic table or the seaborn pairplot: X1 has a significantly lower mean than the other 7 features, and both X1 and X2 have *very* significantly higher standard deviations than the rest of the features. In both cases, the "rest" of the features all have rather uniform means and standard deviations. Because both SVM (especially with the Radial Basis Function kernel) and Logistic Regression operate using Euclidean distance, this difference in means is important to deal with, otherwise X1 may be dominated by the other features. The difference in mean could indicate a benefit from some sort of range standardization, using something like MinMaxScaler which bounds the range from [0,1]. However, this doesn't address the wildly differing standard deviations as seen in the second plot, since differing standard deviations indicates a fundamental difference in data distributions. This might indicate benefit from StandardScaler, which

normalizes both mean and standard deviation, so the data would have mean = 0 and unit variance. There's a possibility that something like RobustScaler could also work, but for the sake of simplicity StandardScaler should suffice for this use case. Next, to examine the correlation matrix of the data:

	X1	X2	X3	X4	X5	X6	X7	X8
X1	1.000000	-0.162925	0.053404	0.048797	0.064429	-0.030689	-0.271790	-0.004266
X2	-0.162925	1.000000	0.063820	0.176044	0.037858	-0.059990	0.178142	-0.020490
X3	0.053404	0.063820	1.000000	-0.007330	0.028529	-0.074335	-0.107514	-0.013038
X4	0.048797	0.176044	-0.007330	1.000000	0.160035	0.028746	0.026834	0.083458
X5	0.064429	0.037858	0.028529	0.160035	1.000000	-0.054253	0.022389	-0.073225
X6	-0.030689	-0.059990	-0.074335	0.028746	-0.054253	1.000000	-0.129064	0.021826
X7	-0.271790	0.178142	-0.107514	0.026834	0.022389	-0.129064	1.000000	-0.016308
X8	-0.004266	-0.020490	-0.013038	0.083458	-0.073225	0.021826	-0.016308	1.000000

Granted, this output is fairly difficult to look through, but to summarize, the most 'extreme' value present is a value of ~-0.27 between X1 and X7, indicating a weak negative correlation, and nothing worth concern. Beyond that, no other features have any meaningful correlation.

Methodology

Before any models were implemented, the data was imported and splitted 80/20 with a seed of 3619 (arbitrary). As previously mentioned, there will be three classification models present. The first is a SVM model. After splitting the data, a pipeline is constructed including a GridSearchCV to optimize for the C, Gamma, and Kernel. Upon running the cell, it was determined that the best parameters were C = 1, Gamma = 2, and kernel = RBF. A C of 1 indicates a margin that is neither narrow nor wide. Intuitively, the "C" parameter determines how thin or wide the margin is. A smaller C results in a thinner margin that allows for far fewer violations that can allow for more accurate classification but risks overfitting. A larger C results in a wider margin that allows for more violations that can allow for more flexibility but risks overfitting. Across all possible provided values for C, 1 falls on the narrow side of the middle. Next, for the gamma value. Gamma determines how far of a reach any given training point has, where a small gamma results in wider influence, resulting in a smoother decision boundary, and a larger gamma results in shorter influence, resulting in a more "wiggly" decision boundary. The optimal gamma was 2, which was the second highest value provided, indicating the model prefers each training point to have very small influence, leading to a rougher decision boundary. Finally, the optimal kernel was RBF. Considering "polynomial" wasn't even one of the options, and the use cases for linear kernels are so limited, this was hardly surprising, as RBF kernels are far more flexible and are able to deal with messy, non-linearly separable data far more effectively.

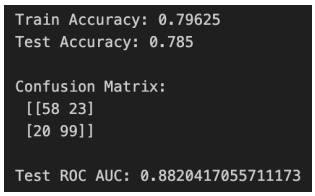
The second model was a classic Logistic Regression model, for which there were no hyperparameters. Since this was a simple logistic regression model, there were no specified hyperparameters, such as any regularization technique.

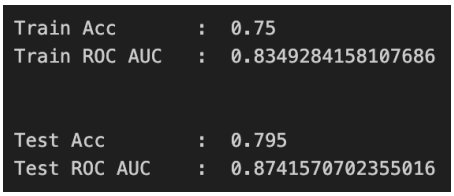
The final model was a KNN model, where hyperparameters to be optimized were the number of “k”’s, or nearest neighbors. Distance was left at the default (Euclidean), and the model’s chosen number for “k” was 30, which was the very highest value it was offered in the parameter grid. The high number of “k” indicates potential underfitting (high bias, low variance), but that isn’t proven.

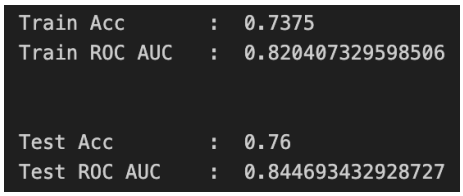
Per the Extra Credit assignment, I ran PCA on the dataset to reduce it to 2 features. Although the purpose of only retaining 2 PC’s is understandable, for visualization, I would have preferred to keep 2 for visualization and run a model with enough to retain 95% cumulative variance, because as will be discussed, model performance (SVM per results) was less than ideal, which is understandable since the dataset was probably overly simplified keeping only 2 principal components. The PCA pipeline led into an SVM model on the 2 principal components. Using the same Grid Search as during the standard SVM pipeline, the optimal hyperparameters resulted in the same as during the normal model (C = 1, Gamma = 2, kernel = RBF).

Results

Results for the models are shown below, in order of left to right, are: SVM-LR-KNN:







Because it’s not present for all 3, Train ROC/AUC will be ignored and the metrics being compared will be Training Accuracy, Testing Accuracy, and Testing ROC/AUC. Ranking by Training Accuracy, the ranking would be SVM followed by Logistic Regression followed by KNN. Ranking by Testing Accuracy, the ranking would be Logistic Regression followed by SVM followed by KNN. Ranking by Testing ROC/AUC, the ranking would be SVM followed by Logistic Regression followed by KNN. Seeing these rankings, KNN would be the clear first elimination because it ranked last in all training metrics. I would argue that SVM would be the ideal model to use in production because not only does it rank 1st in 2 of the 3 metrics, but its training and testing accuracies are also closer together than Logistic Regression, which raises potential concerns of slight underfitting.

Performance of the SVM on the PCA dataset is as follows, using the same hyperparameters as the initial SVM model. This model would hypothetically rank in 4th place for training accuracy, testing accuracy, and test ROC/AUC, making it the weakest performing across the board. I suspect this was due at least in part to the fact that it was limited to 2 principal components; if the cutoff wasn’t a number and instead a cumulative variance threshold cutoff (say, 95%), I believe performance may’ve been improved even more, perhaps surpassing all 3 of the above models. Of course, it is a handy trick for visualization, so it can’t be said that it was for nothing.

```
Train Accuracy: 0.7325
Test Accuracy: 0.73

Confusion Matrix:
 [[51 30]
 [24 95]]

Test ROC AUC: 0.7771553065670712
```

Reflection

During the course of this assignment, I ran into a number of issues, but none of them were particularly obstructive. The first issue was the syntax in passing in columns to the make_column_transformer object, which didn't appreciate the "c for c in df.columns" way of specifying features. It could very well be my lack of experience, but it took me an embarrassingly long amount of time to debug that error and do it the "right" (manual) way, which is why features are defined multiple times throughout the notebook. The second and main issue was to decrypt the wording of the extra credit question, since I was unclear on how I would go about plotting predicted vs actual when PCA doesn't inherently make predictions. That was until common sense kicked in and it became obvious that PCA was to be used to create the dataset that a model would be responsible for predicting (and plotting predictions after). Besides those aforementioned hiccups, the assignment was more or less smooth sailing. The grid search value for the nearest neighbors being "30" (the largest value offered) was definitely slightly concerning, but considering its lackluster performance across all categories I decided not to worry about it.