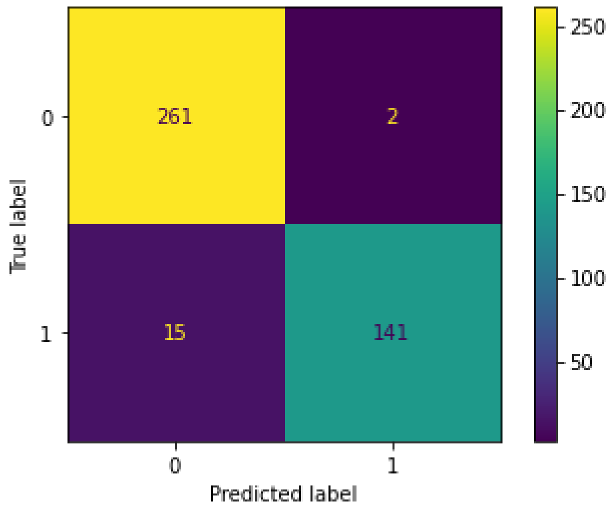
Vibhav Jha
BE700 HW4

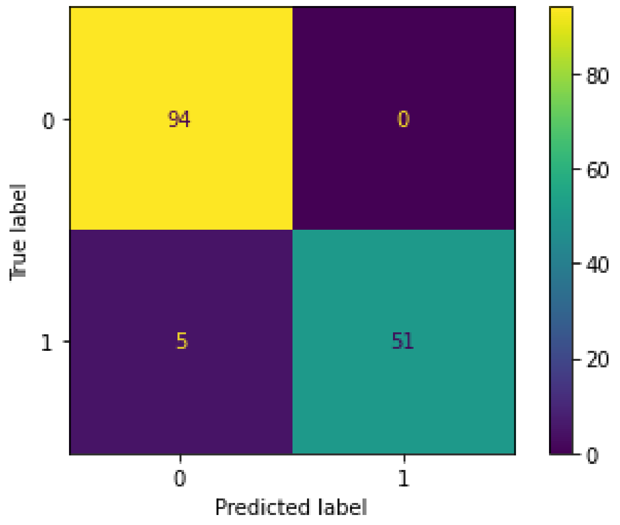
1. Imported Modules
    a. Sklearn
        i. Svm.svc
        ii. Preprocessing
        iii. Metrics.confustion_matrix, ConfusionMatrixDisplay
    b. Pandas
    c. Joblib
        i. Dump, load
2. Description of Classifier (next page)
3. Achieved Training and Testing Sensitivities and Specificities

	Sensitivity	Specificity
Training	0.9038	0.9924
Testing	0.9107	1.0

4. Confusion Matrices
    a. Training:

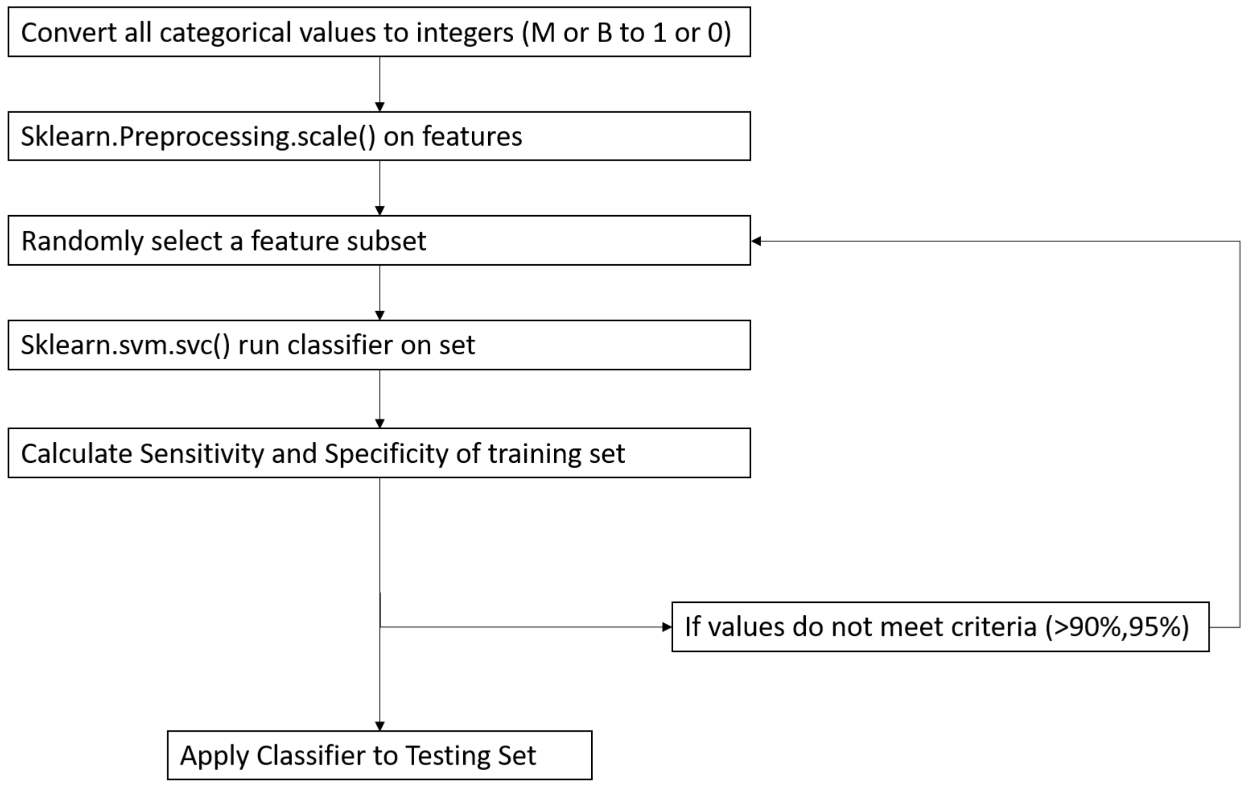


    b. Testing:



Description of Classifier

I specifically chose sklearn.svm.svc() while allowing the for the defaults of the function to be used. I intentionally did not choose a linear kernel, as I wanted a more “natural” or smooth classification. I also wanted to emphasize one versus one approach for classification or essentially binary choice for each class versus class. There were only two classes to classify, making it suitable for these purposes.
In my case I simply scaled the data using preprocessing.scale() [I attempted a few different scalers, MinMaxScaler() and StandardScaler() namely, however unless I significantly decreased the amount of features, I was achieving 100% sensitivity and specificity values]. Preprocessing.scale() is a “leaky” method, but I felt as if centering each feature around the mean and then basing the scaled values from variance a more robust method to view the togetherness of the data. This scaling performed well. After all the features have been scaled, I chose a small subset of them to be used in classification. Considering the small size of this dataset, I pseudo randomly chose some features, and evaluated the sensitivity and specificity values for what was best. [If this data was more complex or had more features, I would be more selective about features and use methods such as correlation analysis, k folds etc.] After picking a subset, the data is fed into the SVM classifier for training and subsequent classification.


Convert all categorical values to integers (M or B to 1 or 0)
Sklearn.Preprocessing.scale() on features
Randomly select a feature subset
Sklearn.svm.svc() run classifier on set
Calculate Sensitivity and Specificity of training set
If values do not meet criteria (>90%,95%)
Apply Classifier to Testing Set

Some Extra Notes: Initially when I attempted this problem, I wrote a neural network with multiple layers, and a total of 6 neurons, however I was consistently getting accuracies of 1 each time. Initially I attempted to correct this by manipulating the training data (removing features) however it would not change. When I plotted the corresponding confusion matrix I would just get a single value of 419, the size of the data set. Eventually, I settled on a simpler approach of using SVM. Though I still struggled with some similar issues and I still got a 100% specificity on the testing set.