Ross Brancati
CS 589 – Homework 3 Report
October 15, 2021

1. Support Vector Machines

1. Matching scenarios
   a. This matches with *figure 2.* The decision function for this kernel function, k, is $g\big(\phi(x_i)\big) = \sum_{j=1}^{N} \alpha_j (u \cdot v + (u \cdot v)^2) + b$, which is a polynomial kernel of order 2. This kernel is a hyperbolic function, so the only option that would match is *figure 2*.
   b. This matches with *figure 3.* The decision function for this kernel, k, is $\mathrm{g}\big(\phi(x_i)\big) = \sum_{j=1}^{N} \alpha_j (\exp\,(-5\|u - v\|^2)) + b$, and when e is raised to a larger negative number in the decision function, the kernel is small. When the kernel is small, the classification of the algorithm will not generalize as well, and we will see a tighter decision boundary, which is shown in *figure 3*.
   c. This matches with *figure 1*. For the same reason mentioned in question 1b, except this time we have a small γ, which would generate a looser decision boundary, which is shown in *figure 1*.

2. Expanding the ellipse equation yields the following:

$$c(x_1 - a)^2 + d(x_2 - b)^2 = 1$$
$$cx_1^2 - 2cx_1a + ca^2 + \quad dx_2^2 - 2dx_1b + db^2 = 1$$

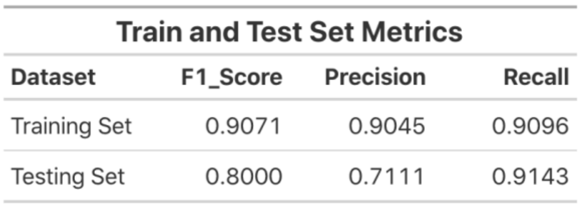
For the feature space mentioned in the question, the coefficients on the expanded equation are simply the weights. So, we could rewrite the equation as

$$0 = -2acx_1 - 2bdx_1 + cx_1^2 + \; dx_2^2 + ca^2 + db^2 - 1$$

The weights are then $w = (-2ac, -2bd, c, d, 0)$, and there is also an equation of a circle $a^2 + b^2 - 1$, which can be rewritten as $a^2 + b^2 = 1$, where 1 is the radius of the circle. The previous expansion shows that SVMs with this kernel can separate any elliptic region from the rest of the plane, because the portion that represents the equation of a circle can be identified as the intercept, which allows for linear separability. Related to this concept, a linear decision boundary in the feature space $(1, x_1, x_2, x_1^2, x_2^2, x_1x_2)$ would only depend on the weights associated with the terms (x_1, x_2, x_1^2, x_2^2), and the remaining features $(1, x_1x_2)$ would yield an elliptical decision boundary in the two dimensional plane (x_1, x_2).

3.

Table 1: Training and testing set F1 Score, Precision, and Recall for all features in the Breast Cancer Wisconsin (Diagnostic) Dataset using generated SVM model.

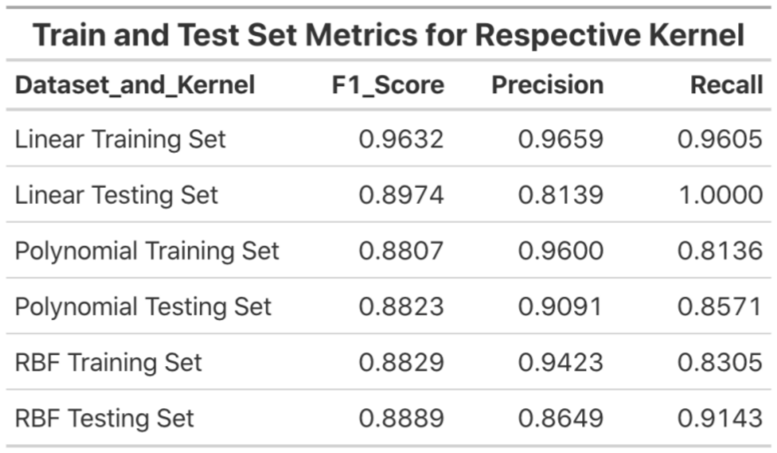


Train and Test Set Metrics

Dataset	F1_Score	Precision	Recall
Training Set	0.9071	0.9045	0.9096
Testing Set	0.8000	0.7111	0.9143

4.

Table 2: Training and testing set F1 Score, Precision, and Recall for all features in the Breast Cancer Wisconsin (Diagnostic) Dataset with different kernels generated with the sklearn.svm.SVC model. The polynomial kernel was of degree = 3.



Train and Test Set Metrics for Respective Kernel

Dataset_and_Kernel	F1_Score	Precision	Recall
Linear Training Set	0.9632	0.9659	0.9605
Linear Testing Set	0.8974	0.8139	1.0000
Polynomial Training Set	0.8807	0.9600	0.8136
Polynomial Testing Set	0.8823	0.9091	0.8571
RBF Training Set	0.8829	0.9423	0.8305
RBF Testing Set	0.8889	0.8649	0.9143

5.

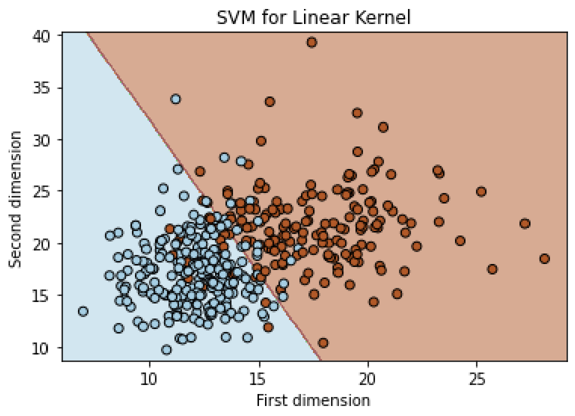


Figure 1: Decision boundary for the first two features in the Breast Cancer Wisconsin (Diagnostic) Dataset using a linear kernel SVM.

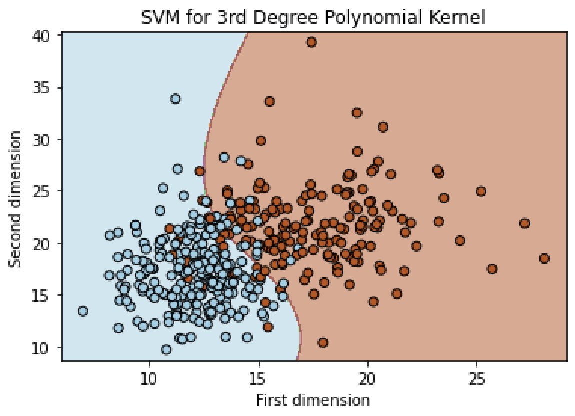


Figure 2: Decision boundary for the first two features in the Breast Cancer Wisconsin (Diagnostic) Dataset using a 3rd degree polynomial kernel SVM.

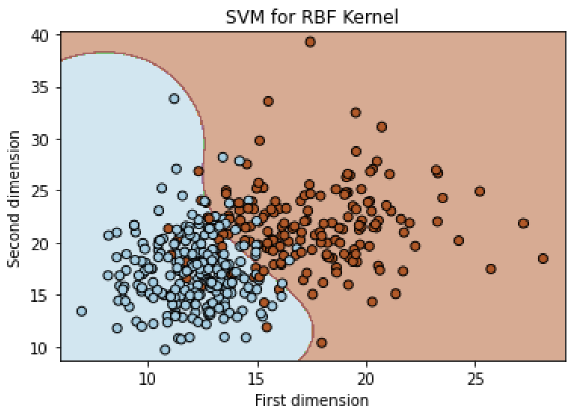


Figure 3: Decision boundary for the first two features in the Breast Cancer Wisconsin (Diagnostic) Dataset using an RBF kernel SVM.

2. Ensemble Methods

1. If one of the features is a strong predictor of the class label, not all the trees in a random forest will have that particular feature as a root node. When a random forest model chooses the feature to split the data, not all features are typically selected by choosing the best split via the Gini index. Unless the max features setting is set to include all features at all trees, the model will randomly select some number of selected features to generate the best split. Since this choice is random, we cannot say that every single tree will always select the best feature. What can be said is that out of those randomly selected features, the best split will be chosen based on the best Gini index out of those features. As previously stated, however, it is not common practice to select the max feature value as all the features. In fact, the sklearn.ensemble.RandomForestClassifier defaults to choosing the number of features at each node as the square root of the number of features. So, unless we change this parameter to be an integer value of the number of features, we will not always get that feature that strongly predicts the class label.

2. Bagging-based ensemble models select several observations from the training and testing set and these observations can overlap. In other words, training set 1 and training set 2 can have the same observations in their bags because it samples the training set with replacement. The random resampling procedure provides enough diversity to decrease variance and overall improve performance. Due to this sampling with replacement procedure, it is possible to train the different learners in parallel. The different learners in bagging-based ensembles are fit independently, and the end state classification accounts for all of these learners. To conclude, bagging-based ensemble models can be trained in parallel for the reasons mentioned above.

3. Boosting-based ensembles, on the other hand, cannot be trained in parallel. This is because boosting-based learners follow a sequential order. In other words, one split on the training data is made, and based on the results of that iteration, a new learner is trained. The training of the current learner is based on the results of the previous learner. As we progress through the individual learners in the boosting-based model, each model in the sequence is fit in a way to give more importance to poorly fit observations in the previous iteration. Ultimately, it puts more weight on the observations that classified poorly in the previous iteration. Thus, we cannot train these in parallel because each iteration is dependent on the previous one.

4.

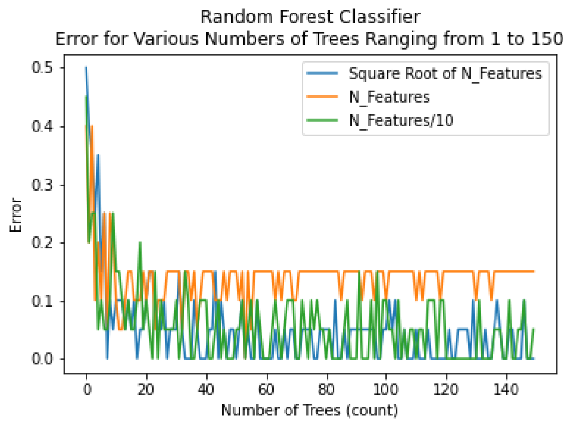


Figure 4: Error of test set data for increasing number of trees for three different max feature values for the Random Forest Classifier Model using the Gene Expression Dataset.

5. In the previous question, I tested three different values for the maximum number of features (*max_features*) to consider for the best split at each tree in the random forest. Given that there are 300 total features in this dataset, the blue, orange, and green lines used 17, 300, and 30 features to determine the best split. I would think that the error would be lowest for the split utilizing more features, but that is not the case in this problem. It looks like using the square root of the number of features or the number of features divided by 10 yielded the lowest overall error. My intuition would tell me that this is because including all of the features at each tree overfits the data on the training set. In every case, increasing the number of trees in the forest (*n_estimators*) decreased the overall error. Random forests use bagging methods, so if the number of trees in the forest is too small, we may not account for some of the data because it may be missed when the random bags are generated. Some of the samples in the training set could only be predicted at only one tree, or possibly not even at all. In this dataset, we only have 63 samples, which is why the error decreases so quickly at the beginning and plateaus after just about 10-15 trees. The weak learners fail to contribute to improving the classification after about 10-15 trees. In any case with random forests, it is important to tune both the number of trees and the number of features included in the split to minimize the error while still ensuring that the model is generalizable to new observations.

6.

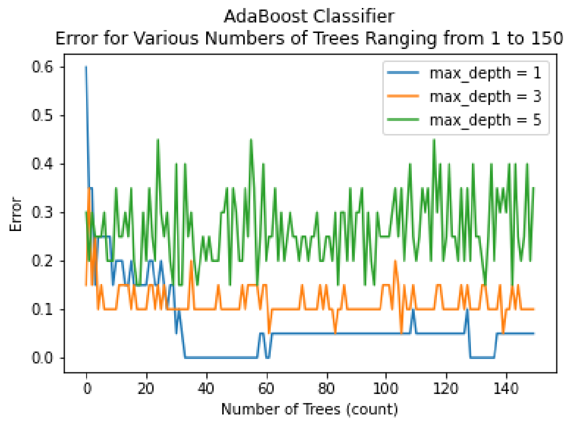


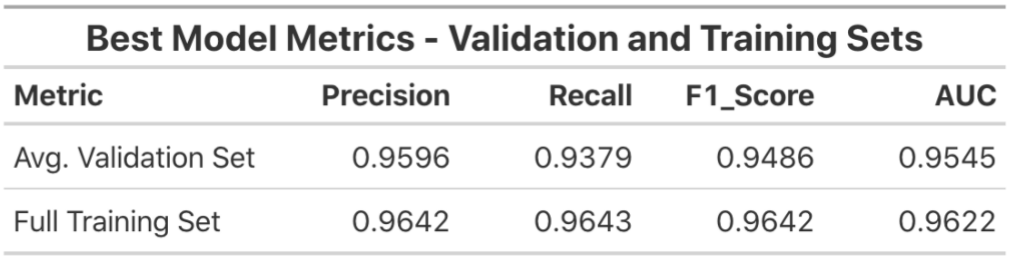
Figure 5: Error of test set data for increasing number of trees for three different max depths (1, 3, and 5) of an AdaBoost Classifier with Decision Tree base estimator using the Gene Expression Dataset.

7. In the previous question, I tested three different depths of the decision tree base estimator and plotted the error associated with each decision tree depth. I was surprised to see that setting the max depth to 5 resulted in the highest error for most of the number of trees in the ensemble model. This is most likely due to overfitting on the training set. When the tree number was over approximately 30, the error for the max depth of the base estimator was the lowest. However, when the number of trees was very small (ie 1-5), the error at a max depth of 1 was higher or about the same as the other maximum depths. This is probably due to underfitting of the training data when the number of trees was low. The max depth of 3 yielded error values between the max depths of 1 and 5 when the number of trees was greater than 30. Overall, once the error plateaus, we can infer that the weak learners are not contributing to improving the classification accuracy. To conclude, this stresses the importance of tuning hyperparameters correctly when building these ensemble models. It is crucial to test multiple values of both number of trees and max depths of the base estimator to generate the best results on the testing dataset, which is going to differ depending on the dataset that you build the models from. One must look at the number of features and the number of training examples in the training and testing data, and test various values of max depth and number of trees to get the best results on the testing dataset.

3. Train Your Best Model

1.

Table 3: Average validation and training set evaluation metrics for the Logistic Regression classifier model trained and tested on the Breast Cancer Wisconsin (Diagnostic) dataset. The hyperparameters for this model included an L2 regularization function, 10000 max iterations, and C set to 0.8.



Best Model Metrics - Validation and Training Sets

Metric	Precision	Recall	F1_Score	AUC
Avg. Validation Set	0.9596	0.9379	0.9486	0.9545
Full Training Set	0.9642	0.9643	0.9642	0.9622

I selected four different models to test with the Breast Cancer dataset – Support Vector Machine, Logistic Regression, AdaBoost, and Random Forest. For each of the models, I tuned the hyperparameters by selecting a certain set of hyperparameters, adjusting those hyperparameters, and selecting the best set of hyperparameters that produced the best evaluation metrics. For each model, the hyperparameter tuning procedure was as follows:

Support Vector Machine: initially, I tried 10,000 max iterations, but python threw a 'Solver Terminated Early' warning and suggested that I leave this parameter set to -1, which indicates that there is no limit on the number of iterations. I tried the linear, rbf, and polynomial kernels (degrees = 3, 4, and 5). The linear kernel performed the best based on the evaluation metrics, so I decided to go with that one.

Logistic Regression: started with a low number of iterations, but python threw a warning that the loss function did not converge, so I increased the number of iterations until it did converge, which ended up being 10000 iterations. I then altered the C parameter, which is the inverse of the regularization strength. I started with the default value of 1, and tried some different values such as 0.01, 0.1, 0.5, 0.8, and 10. Setting C=0.8 yielded the highest evaluation metrics for both the cross-validation and training set.

AdaBoost: I employed a decision tree base estimator as we did in the second problem and tried a few different max depth values (1, 3, and 5) with a learning rate of 0.1. The max depth value of 1 produced the highest evaluation metrics. I then tried changing the learning rate and got each of the training set evaluation metrics to be 1.0, but I believe that it was overfitting the data because my cross-validation metrics decreased from when the learning rate was set to 0.1. This stresses the importance of performing cross-validation to ensure that the model is generalizable to unseen samples. I believe that this would not perform well on predicting labels of new data.

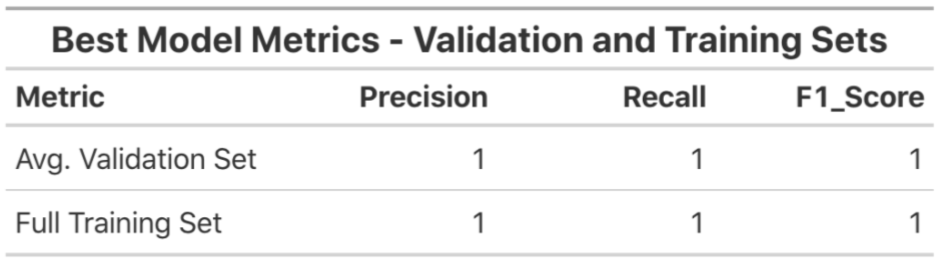
Random Forest: initially I left the hyperparameters as default which sets the max features to the number of features. I also left the number of estimators (which represents the number of trees in the forest) as default (default value = 100). With these hyperparameters, all metrics on the training set ended up being 1.0, which again was indicative of possibly overfitting on the training set. Next, I tried adjusting the max features to the square root of the number of features, but this did not improve performance and resulted in training set metrics of 1.0. Lastly, I tried lowering the number of estimators because of this perceived overfitting. When I lowered it to 20, the training set evaluation metrics dropped below 1, but the validation set metrics also declined. I decided that this may not be the best model to classify these data because of concerns of overfitting. Also, other models ended up with better evaluation metrics, so ultimately, I went with a different model.

After the procedures followed above, I found that the Logistic Regression model with an L2 regularization function, 10000 max iterations, and C set to 0.8 yielded the best evaluation metrics using cross-validation, so I chose this model as my best model. I believe the logistic regression model did well on this data set and matched my expectations because the average F1 score for the cross-validation procedure was very close to 0.95, which is close to the ideal value of 1.0.

Note: I wasn't sure what we were able to use in terms of generating the folds for the cross-validation procedure, so I included two scripts in my submission. One is titled cross_validation_q3_1_random.py which implements random sampling without replacement for selecting the folds (this is more of a manual procedure for cross-validation). The other is cross_validation_q3_1_stratifiedkfolds.py which employs the StratifiedKFold method to generate the folds, but the cross-validation procedure was still manually implemented in this script. I got very similar results using both random and StratifiedKFolds methods, which makes sense because the random sampling procedure most likely includes both classes given that there are many samples in the dataset.

2.

Table 4: Average validation and training set evaluation metrics for the Random Forest classifier model trained and tested on the Gene Expression Data dataset. The number of estimators was set to 100 and the max features was set to the square root of the maximum number of features.



Best Model Metrics - Validation and Training Sets

Metric	Precision	Recall	F1_Score
Avg. Validation Set	1	1	1
Full Training Set	1	1	1

I will keep this brief considering I am already over the recommended page limit of 5. In short, I followed the same procedure as outlined in question 3.1. I implemented SVM with linear, polynomial, and RBF kernels. I also tried the Logistic Regression with an L2 regularization function and multiple numbers of iterations and a learning rate of 0.1. Lastly, I implemented Random Forest and AdaBoost algorithms with various numbers of estimators and depths of the decision tree base estimator. I found that the Random Forest classifier with hyperparameters mentioned in the caption for table 4 yielded precision, recall, and F1 scores of 1.0 for the cross-validation set. Initially, I thought that this model might be overfitting the data, but given that cross-validation was implemented, I was confident that this model would predict well on the test dataset. I was having trouble calculating ROC_AUC score for the multiclass dataset, so I decided to just report these results as I worked through this extra credit problem. From what I read, one hot encoding must be utilized to calculate ROC_AUC, but I couldn't put the time towards figuring it out. I hope it is at least worth some extra credit points.

4. Model Selection and Hyperparameter Tuning

1. High bias machine learning classifiers are those that do not predict well on unseen data, which could be due to greater assumptions on the target labels. On the other hand, low bias assumes less about the labels and typically predict well on unseen data. Given this, KNN classifiers are typically low bias, but they can also be high bias depending on the dataset and how many neighbors are selected in the training process. If the number of neighbors is low, the model would be considered low bias because the new data points are only looking for the nearest neighbor in the training data. However, if the number of neighbors is high, and the data point is somewhere near the decision boundary, there is a chance that it could misclassify the new data point. Decision tree bias is dependent on the number of trees in the model. Shallow trees typically have high bias and low variance, whereas deeper trees tend to have low bias and high variance. This is because increasing the number of trees adds more separations of the data, so a new data points path down the decision tree would probably predict the correct label. On the other hand, a new data point in a shallow decision tree model may not predict the class label correctly. Logistic regression classifiers are considered to be high bias because they do not always predict unseen data correctly. However, they do tend to have lower variance, meaning that the risk of predicting an unseen data point incorrectly is lower. In machine learning, it is important to balance this bias and variance relationship to produce the best prediction results. Tuning parameters of any model can help to optimize the bias and variance tradeoff, so in reality any of these three models could be low or high bias, depending on the dataset and chosen hyperparameters.

2. To train one sample of data, it would take M units of time. If we have k cross-folds, each cross fold will have $\frac{M}{k}$ samples. Since one of these cross folds is the validation set, and that time is negligible, there would be $\frac{M(k-1)}{k}$ samples in the training set, which would take $\frac{M(k-1)}{k}(M)$ units of time, or $O(\frac{M^2(k-1)}{k})$. If $k = 5$, the time complexity would be $O(\frac{M^2(4)}{5})$. If we were to perform leave one out cross validation, the value of k would be equal to M because the model would be training on every sample in the dataset except for the one that it is testing on. That being said, the time complexity for leave one out cross validation is $O\left(\frac{M^2(M-1)}{M}\right) = O\big(M(M-1)\big) = O(M^2 - M)$. Leave one out cross validation is an ideal cross-validation method because you end up training on all data points, but the computational complexity increases. In general, increasing the number of cross-folds increases the computational complexity, but the quality of the estimate will also improve. Also, it is important to check the evaluation metrics when performing cross-validation to get the best hyperparameters of a model.