Christopher Bussen
CPS 349
3/29/2023

Assignment 4 Report

Descriptions of classification algorithms:

1. K-Nearest Neighbors: The K-Nearest Neighbors classification algorithm works by looking at the distance between points in latent space. It starts by storing the training data and finding the k-nearest training points to the new data point in question. The class of this new data point is then determined by what the class of the majority of its k-nearest neighbors is. More specifically, it draws decision boundaries and groups several data points together based on the center of clusters. In this algorithm, the value of k is defined by the user and is typically determined through a series of tests that give some indication as to what value will provide the highest level of accuracy in classification. This value typically depends on the size and structure of the dataset. This algorithm can be implemented using KNeighborsClassifier from the sklearn library.
2. Decision Tree: The Decision Tree classification algorithm works in a similar way to a flow chart in the sense that you start from the top and follow a series of true or false statements until you reach the end of the tree. Once you reach the end of the tree, you are given a prediction as to what the class of the data point is. At each node, the algorithm splits the dataset using the feature that will give the most information about the potential class of the data point. Each node also provides a Gini value (measure of purity/quality of each split – closer to 0 means better), number of samples, values, and predicted class. One important factor of decision trees is the maximum depth that you decide to give the tree (this is once again defined by the user when the tree is created). If you give too low of a depth, the tree will not be able to give you very much information. However, if you give too high of a depth, this can cause overfitting which will lead to worse accuracy. This algorithm can be implemented using DecisionTreeClassifier from the sklearn library, and also has several functions for cleanly displaying the tree.
3. Random Forest: The Random Forest classification algorithm utilizes the framework of decision trees, but instead of using just one it actually uses a collection of trees. More specifically, random forests work by creating a specific number of decision trees, each of which are trained differently. These trees often use different subsets of the training data during this process. The class of a data point is then determined by combining the predictions of all of the trees and taking the class that is most commonly predicted. For random forests, the number of trees is provided by the user. Once again, you do not want to make this number too small or too big, but random forests are noted to be good at limiting overfitting, making them an especially useful classification algorithm. This algorithm can be implemented using RandomForestClassifier from the sklearn library.

Table of Parameters

Christopher Bussen
CPS 349
3/29/2023

Parameters used across models:

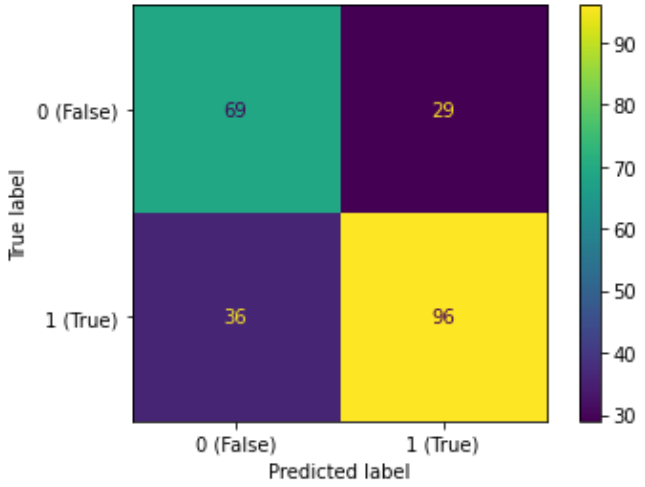
	Dataset size	Features	Train/test split	x_train size	x_test size
0	(918,13)	11	75%/25%	(688,11)	(230,11)

	y_train size	y_test size	k for KNN	Tree depth	Trees in forest
0	(688,)	(230,)	13	4	50

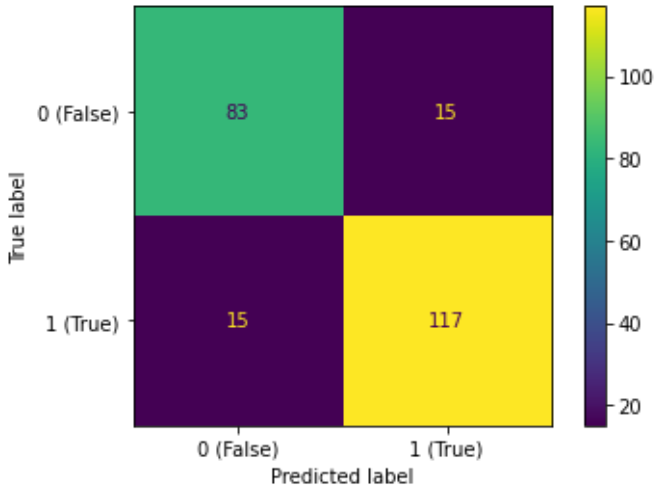
The table above displays the parameters that were used throughout this assignment across each model. As seen above, it shows things like the sizes of different datasets, how many features this dataset has, and the important user-defined parameters for each classification algorithm.

Confusion Matrices

k-Nearest Neighbors confusion matrix:

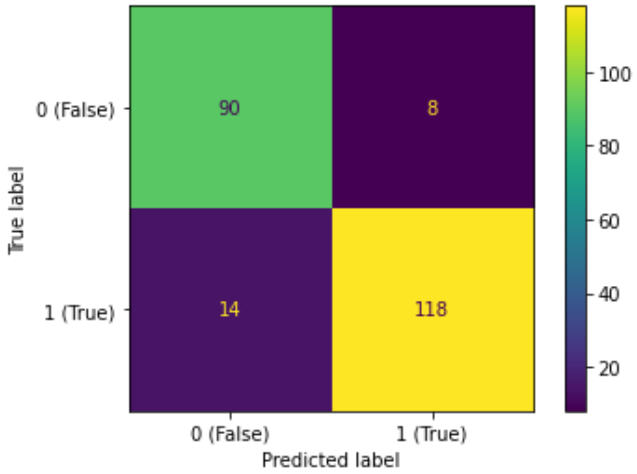


Decision Tree confusion matrix:



Christopher Bussen
CPS 349
3/29/2023

Random Forest confusion matrix:



All of these confusion matrices were created using confusion_matrix from sklearn and comparing. In doing so, I created a set of predictions using the x_test data and compared it to the y_test data. Additionally, I used ConfusionMatrixDisplay from sklearn's metrics library in combination with matplotlib.pyplot to display this more visually pleasing matrix (instead of just printing the original confusion matrix which simply shows numbers without labels).

<u>Performance Table</u>

KNN table:

```
Metrics for KNN Classifier:
    Accuracy  Precision    Recall  F1 score
0   0.717391      0.768  0.727273  0.747082
```

Decision Tree table:

```
Metrics for Decision Tree Classifier:
    Accuracy  Precision    Recall  F1 score
0   0.869565   0.886364  0.886364  0.886364
```

Random Forest table:

```
Metrics for Random Forest Classifier:
    Accuracy  Precision    Recall  F1 score
0   0.882609   0.913386  0.878788  0.895753
```

Overall table:

```
Metrics for all classifiers:
                    Accuracy  Precision    Recall  F1 score
k-Nearest Neighbors  0.717391   0.768000  0.727273  0.747082
Decision Tree        0.869565   0.886364  0.886364  0.886364
Random Forest        0.882609   0.913386  0.878788  0.895753
```

Christopher Bussen
CPS 349
3/29/2023

These tables were created using the methods from sklearn's metrics for accuracy score, precision score, recall score, and f1 score. I calculated all of these values and then created a pandas DataFrame which I was easily able to print. For the overall table, I simply combined the values for each classification algorithm into one column for each metric.

Description of Parameters

For each model, it was very easy to notice how big of an impact the user-defined parameters made on their ability to properly classify different data points. If you did not set them high enough, the models would struggle to gain enough depth on the available information and would not be super accurate in its predictions. On the other hand, if you set the parameters too high, the models would be overfitted and would once again struggle to properly classify data.

For k-Nearest Neighbors, the main parameter that is defined by the user is the value of k, which is the number of nearest training points the model finds to the given data point when trying to classify it. I started by creating a for loop to test k values from 1 to 20, creating a new KNN model each time, fitting the model with training data, and making predictions with the testing data. Finally, each time through the for loop I printed the accuracy score for each new KNN model. For the first few values of k, the accuracy tended to be decently lower than the rest. As k increased, the differences in accuracy began to decrease. However, once k began to get notably large, accuracy started to decrease due to overfitting. As a result, I found that a k value of 13 would normally give the highest accuracy, thus would be the best value for classifying the testing data.

For the Decision Tree model, the main parameter defined by the user is the max depth of the tree, or the number of levels it will go into (how many true or false statements a data point will go through before classification). In order to decide what value would be best for this model, I simply went through and changed the value for max depth and then fit the model, made predictions using the test values, and then calculated the performance metrics for the model. I started with a max depth of 1, which gave somewhat low metrics. From there, I kept increasing the max depth by 1 and saw notable improvements in each metric. Eventually, I reached metrics close to .90 with a max depth of 4 and thought that this might be the optimal depth. To make sure, I tested depths of both 5 and 6 and noticed quick drop-offs in the performance of the model for these values. As a result, I decided that a max depth of 4 would be the best value for a decision tree to classify the data.

Finally, for the Random Forest model, the user-defined parameter is the number of trees that will be used in the forest. I figured from the start that it would likely be best to have a somewhat large number of trees because a classification is made based on the predicted class of the majority of the trees, therefore more trees would give a better sample size. To test different values, I followed the same process I used for k-Nearest Neighbors, where I used a for loop with a new random forest model each time through and fitted the model then predicted

values with the test data. I made this loop go from 0 to 100 and printed the accuracy of each model. From this loop, I saw that a low number of trees typically gave worse accuracy as I originally expected. However, I also noticed that accuracy did not really take a very large hit as the number of trees got quite large. From this I concluded that random forests are not as prone to overfitting as other models are. Despite this, I still did not want to set my number of trees for my model too high as to noticeably increase runtime and slow down the process. After running this loop a few times, I found that the best metrics were usually obtained by using about 50 trees in the random forest.

Conclusion

In conclusion, I believe that the Random Forest model performed the best based on all of the outcomes that I saw. I figured this would be the case as soon as I saw all of the confusion matrices, primarily because the random forest was the best at predicting the what the true labels were. This model correctly predicted 90 data points to be 0's for heart disease (not present) as well as correctly predicted 118 data points to be 1's for heart disease (present). These were both more than either of the other models. Additionally, the claim that the random forest was the best for classifying this data is also supported by almost all of the metrics. From the table above, we can see that the random forest has the best accuracy, precision, and f1 score. The only metric it is not first in is recall, but it is only slightly below the decision tree. I figured that this would be the case because I originally thought that k-Nearest Neighbors would struggle due to the high number of features in the dataset. As a result, I thought that one of the models with a tree structure would likely do better. Furthermore, I knew that random forests were basically collections of decision trees trained in different ways. Because the random forest takes the classification of whatever the majority of the decision trees are, I knew this would likely add another layer of accuracy to the model. Due to all of these factors, I believe that the random forest model performed the best when classifying the given data.

Here's a screenshot of my decision tree by the way: