Vishal Puppala ID: 448501
May 5, 2021 CSE 514A

Data Mining: Comparison of Methods

ReadMe

There is one script which implements all of the data mining methods that is submitted to Gradescope called. The file is titled 'comparisonofarchitectures.ipynb' and it is a notebook file that needs to be run on Jupyter Notebook. The easiest way to do this is by downloading the Anaconda Navigator app on a computer from which one can run Jupyter Notebooks. Additionally, the data file must be in the same directory as 'comparisonofarchitectures.ipynb' in order for the input dataset to consist of the data within this data file.

Furthermore, in order to successfully run the script without receiving any error messages, you will need to make sure you have installed the proper packages on your Anaconda Environment. You can utilize the "pip install _package_" command. For instance, to install Keras, you would need to enter pip install keras in your environment where you are running 'comparisonofarchitectures.ipynb'. Other packages needed are numpy, pandas, matplotlib, tensorflow, sklearn, and datetime.

Also, 'comparisonofarchitectures.ipynb' was adapted from this script on Github:
https://github.com/subhadeep-123/Breast-Cancer-Detection/blob/master/Breast_Cancer_Detection_Using_Various_Architectures.ipynb

Intro

In this report, we investigate the breast cancer Wisconsin dataset on the UCI repository. We have also conveniently attached a copy of the dataset on the back of this report.

Our goal is to find the DM architecture that will give us the best performance measures for predicting malignant vs benign tumors. Our dataset is very large with several features that factor into whether a tumor is malignant or benign with over 500 data points and over 30 features.

While each DM method implemented in this report is considered robust and reliable. Each one has its own aspects that might make it suitable for different forms for data. Moreover, they also have different run-times which may make one more preferable than another.

The following methods will be tested:

- Support Vector Machine (SVM)
- Artificial Neural Network(ANN)
- K-Nearest Neighbors (KNN)
- Random Forest
- Decision Tree

The model training method employed was 10-fold cross-validation in which the data set is shuffled (randomly) and then partitioned into 10 groups. Each of the 10 groups is then subdivided into 10 sections. One of the 10 sections in each group is reserved for the test set and the remainder is reserved for the training set. This process of 10-fold cross-validation helps ensure that our model is able to generalize to unseen data from the same data set instead of overfitting to our training data.

Performance Measures

Since this is a binary classification problem, we can use the metrics obtained from a 2x2 confusion matrix to evaluate the reliability of our model. These metrics found are the number of false-positives (FP), true-positives (TP), true-negatives (TN), and false-negatives (FN). From these basic metrics, we can formulate these more informative performance measures.

Accuracy = (TP + TN)/(P +N)
Sensitivity=TP/(TP+FN)
Precision=TP/(TP+FP)

Accuracy provides a measure of overall reliability. The others can provide more granular information regarding the robustness of specific predictions

Notes about Implementation:

All of the methods were implemented using Keras. For each method, we would use the Keras classifier function for that specific method to fit our training data to the classifier and then use the Predict functionalities to predict the test data labels by running the test data in the model fitted to the training data. Then afterwards, Keras functions were once against used to compare the estimated labels with the true labels of the test data which were used to tabulate performance measures.

Methods

K-Nearest Neighbors

- **Algorithm Summary:** The algorithm arranges the data (D) into a feature vector of size (nxd) where n is the number of data points and d is the dimensionality or number of features associated with each individual data point. For each point di in D (input space), the algorithm calculates a distance measure between di and all other points. A point is assigned to a class based on the average label of the k closest neighbors (where closeness is measured by the distance metric).
    - Theoretical Runtime: Let n be size of training set. Let d be the dimensionality or number of features associated with a data point
        - Finding the distance from a point in the test set to all point in the training set is O(n*d) since to find the distance between two points means to find the distance between the two points once per feature.
        - O(n*k) is the time needed to find the distance between each point and its k neighbors for all n points
        - Total Time= O(n*d + n*k)
- **Implementation and Results**
- Used Keras KNeighborsClassifier() function
- 10-fold cross validation was used (explained in intro.)

    - 90% training set which was fed into KNeighbors function and then fitted into a KNN classifier that was used to predict the test set labels
- We find that k=10 (number of neighbors=10) produced was associated with a strong performance evaluation than k=1 through k=9 and we notice that when k>10, the performance worsened
- We varied the distance measure and found that Manhattan Distance and Euclidean distance were equally good (produced exact same statistics)
- **Comments:** Very good results. Makes sense since KNN is strong with classifying categorical data (like malignant vs benign)

- `Time taken to complete random search:  0:00:00.002468`
- `Accuracy Score: 0.9642857142857143`
- `Precision Score: 0.9767441860465116`
- `Recall Score: 0.9767441860465116`
- `F1 Score: 0.9767441860465116`
- `Classification Report:`
- `              precision    recall  f1-score   support`
-
- `           0       0.92      0.92      0.92        13`
- `           1       0.98      0.98      0.98        43`
-
- `    accuracy                           0.96        56`
- `   macro avg       0.95      0.95      0.95        56`
- `weighted avg       0.96      0.96      0.96        56`

Support Vector Machine:

- **Algorithm Summary:** Support vector machines are classifiers that address the Curse of Dimensionality by projecting the input data into a higher-dimensional space and then configuring a hyperplane boundary to separate the data into classifications. One major source of variation among support vector machines are the choice of kernel functions which apply data to a higher dimensional space. Kernel functions vary because each one typically tunes to a different similarity measure.
    - Theoretical Runtime: According to internet resources, the runtime of support vector machines depends on the dimensionality of the data, number of data points, kernel function choice, and number of support vectors.
- **Implementation and Results:**

- Kernel: I tried with the polynomial and Gaussian kernels
- I was able to implement softmax by adjusting the regularization term of the Keras SVC function. However, increasing the degree of regularization did not improve performance. Neither did lowering the regularization penalty. So we concluded that softmax was subpar.
- 10-fold cross validation was used (explained in intro.)
    - 90% training set which was fed into Keras SVC function and then fitted into a SVM classifier that was used to predict the test set labels
- **Comments:** Very good results. The Gaussian and polynomial kernels produce identical results but the SVM using the gaussian kernel takes roughly twice as long. I don't know why there is a difference in speed but I surmise it is because Gaussian kernels are designed to detect local structures instead of global patterns so it might be less efficient on larger problems.

Gaussian Kernel Results

- Time taken to complete random search: 0:00:00.008909
- Accuracy Score: 0.9821428571428571
- Precision Score: 0.9772727272727273
- Recall Score: 1.0
- F1 Score: 0.9885057471264368
- Classification Report:
-
	precision	recall	f1-score	support
0	1.00	0.92	0.96	13
1	0.98	1.00	0.99	43
accuracy			0.98	56
macro avg	0.99	0.96	0.97	56
weighted avg	0.98	0.98	0.98	56

Polynomial Kernel Results

- Time taken to complete random search: 0:00:00.004683
- Accuracy Score: 0.9821428571428571
- Precision Score: 0.9772727272727273
- Recall Score: 1.0
- F1 Score: 0.9885057471264368
- Classification Report:
-
	precision	recall	f1-score	support
0	1.00	0.92	0.96	13

- 1 0.98 1.00 0.99 43
-
- accuracy 0.98 56
- macro avg 0.99 0.96 0.97 56
- weighted avg 0.98 0.98 0.98 56

Decision Trees:

- **Algorithm Summary:** Decision tree classifiers are algorithms which assign data points to classes based on whether certain criteria are met at incremental steps. Decision trees are constructed by sequentially partitioning data based upon whether a certain quality or measure aspect is met. Data partitions are continuously sub-partitioned until enough information has been collected to assign a label—this point where a label assignment is possible is analogous to the last level of a tree where the leaves are. The root is then analogous to the first partition.
    - Theoretical Runtime: The runtime to construct a single decision tree is typically O(m * n log(n)) where n is the number of items in the training set and m is the number of features per node.
- **Implementation and Results**:
    - The Sklearn DecisionTreeClassifier function has a random_state parameter which controls the degree to which an item selected in bootstrapping is randomly done so. Since randomness eschews deterministic fitting and adds greater uncertainty to our classification process, the *random_state* estimator was set to 0.
    - 10-fold cross validation was used (explained in intro.)
        - 90% training set which was fed into DecisionTreeClassifiefunction and then fitted into a decision tree classifier that was used to predict the test set labels
    - Criterion: Minimize Gini impurity
        - The Gini impurity measures the likelihood of a random data point being labeled/classified incorrectly. Less computationally expensive than entropy criterion.
    - Criterion: Minimize entropy
        - Entropy measures the level of classification uncertainty. We want to minimize this uncertainty to maximize the information we have regarding

classification outcomes. More computationally expensive than Gini impurity.

- **Comments:** Looking at the performance metrics for the decision tree with gini score and the decision tree with entropy as split criteria, it is difficult to say which exactly is better. One valid indicator of overall quality is accuracy and entropy is more accurate to a small degree. Entropy also has higher precision, recall, and F1 score. However, the decision tree with entropy as a split-criteria is also slower but only by a single decimal point so this isn't too significant. Overall, the decision tree with entropy seems to be the more optimal decision tree.

Gini Score

```
Time taken to complete random search:  0:00:00.008707
Accuracy Score: 0.9107142857142857
Precision Score: 0.9523809523809523
Recall Score: 0.9302325581395349
F1 Score: 0.9411764705882352
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.85      0.81        13
           1       0.95      0.93      0.94        43

    accuracy                           0.91        56
   macro avg       0.87      0.89      0.88        56
weighted avg       0.91      0.91      0.91        56
```

Entropy

```
Time taken to complete random search:  0:00:00.010193
Accuracy Score: 0.9464285714285714
Precision Score: 0.9761904761904762
Recall Score: 0.9534883720930233
F1 Score: 0.9647058823529412
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.92      0.89        13
           1       0.98      0.95      0.96        43

    accuracy                           0.95        56
   macro avg       0.92      0.94      0.93        56
weighted avg       0.95      0.95      0.95        56
```

Random Forests:

- **Algorithm Summary:** Random forests can be thought of as more accurate alternative to decision trees. The random forest method constructs multiple decision trees for classification and then uses the entire ensemble of constructed random forests to label data. Random forests are ideal as they exploit the simplicity of decision trees but also use a more diverse data sample to produce classifications.
    - Theoretical Runtime: The runtime to construct a single decision tree is typically O(m * n log(n)) where n is the number of items in the training set and m is the number of features per node. Creating k decision trees, where k is a positive integer, would imply a runtime of O(k*m * n log(n)). The last step is to return the mode or mean of all the k decision trees which should take constant time. Thus, the runtime to construct a random forest is conjectured to be O(m * n log(n))
- **Implementation and Results**:
- Since we need to construct several decision trees, we can create different versions of a random forest algorithm depending on the splitting criteria we choose to employ.
- The Sklearn RandomForestClassifier function has a random_state parameter which controls the degree to which an item selected in bootstrapping is randomly done so. Since randomness eschews deterministic fitting and adds greater uncertainty to our classification process, the *random_state* estimator was set to 0.
    - Criterion: Minimize Gini impurity
        - The Gini impurity measures the likelihood of a random data point being labeled/classified incorrectly. Less computationally expensive than entropy criterion.
    - Criterion: Minimize entropy
- **Comments:**
        - Entropy measures the level of classification uncertainty. We want to minimize this uncertainty to maximize the information we have regarding

classification outcomes. More computationally expensive than Gini impurity.

- **Comments:** The random forest is more accurate when using the Gini impurity criterion. Surprisingly, however, it is slower than the random forest using the entropy criterion. However, it is only slightly slower so the random forest with the entropy criterion is the better algorithm.

Results with Gini Impurity

- Time taken to complete random search: 0:00:00.225426
- Accuracy Score: 1.0
- Precision Score: 1.0
- Recall Score: 1.0
- F1 Score: 1.0
- Classification Report:
- `              precision    recall  f1-score   support`
- `           0       1.00      1.00      1.00        13`
- `           1       1.00      1.00      1.00        43`
- `    accuracy                           1.00        56`
- `   macro avg       1.00      1.00      1.00        56`
- `weighted avg       1.00      1.00      1.00        56`

Results with Entropy

```
Time taken to complete random search:  0:00:00.209325
Accuracy Score: 0.9642857142857143
Precision Score: 0.9767441860465116
Recall Score: 0.9767441860465116
F1 Score: 0.9767441860465116
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.92      0.92        13
           1       0.98      0.98      0.98        43

    accuracy                           0.96        56
   macro avg       0.95      0.95      0.95        56
weighted avg       0.96      0.96      0.96        56
```

Artificial Neural Network (Deep):

Summary: The advent of neural networks allow for the computational power of a single perceptron to be amplified by allowing for perceptrons to connect to one another and allow for computation to be built up as input travels along the network and also be error-corrected through

backpropagation. However, deep neural networks have multiple layers and and have considerably more perceptron-connections which allows for much stronger computation but also results in longer forward runtime and longer backpropagation and also requires greater spatial complexity. Moreover the runtime of the a deep neural network that is densely conencted increase exponentially with each layer so obviously a deep neural network with multiple layers will require more time than the aforementioned methods. Moreover, we need to remember that the input does not just travel along the network from the input layer to output layer just once completes the journey multiple times, thanks to backpropagation which allows for the readjustment of weights each journey until convergence is achieved. This can take over 100 or even 1000 iterations resulting in a very computationally burdensome, though powerful, engine.

- **Implementation:**
- Units per Layer: 20
- Number of Hidden Layers: 3
- Kernel Initializer: Uniform
- Optimizer: Stochastic Gradient Descent and Adam
    - Stochastic Gradient Descent optimizes the objective function by only taking the gradient at a smaller subset of points instead of all data points. This results in an increase in runtime, since there is a reduced computational burden. However, there is also a loss in accuracy
    - Adam is a new-fangled optimizer which is an example of something that uses adaptive learning rates, where the learning rate is dependent on the performance at a given moment during learning. Thus, this flexibility can allow for greater computational efficiency and also faster convergence. Adam also leverages a property from the RMSprop where the learning rate is adapted to account for each parameter, allowing for even more versatility.
- We tried two artificial neural networks, each with the same network architecture but with different activation functions: ReLu and sigmoid activation functions. The sigmoid activation function is the most commonly used choice but its use can lead to the vanishing gradient phenomenon where the gradient gets really small very quickly before gradient descent can converge upon a solution. ReLU solves this by having a gradient

that isn't as steep, thus bypassing the vanishing gradient phenomenon and allowing for more iterations.

- For both implementations we tried for n=100 iterations as increasing the number of iterations didn't appear to change our results meaningfully. Since this is a deep neural network, we wanted to add more than just 1 hidden layer so we added 3 to increase informational gain. Moreover, each hidden layer is a dense layer, meaning that each perceptron is deeply connected to other perceptrons in other layer--a hallmark of deep neural networks to increase informational gain. Since deep neural networks exponentially increase the informational gain, there is an increased danger of overfitting which is why we applied a dropout to each layer.

ANN Results (ReLU):

ReLU

We initially experimented with a dropout rate of 0.1 for all layers. However, when we tried this with ReLU with (optimizer=sgd, initializer=uniform), we got suboptimal results (mediocre accuracy and precision score).

```
Accuracy Score: 0.7678571428571429
Precision Score: 0.7678571428571429
Recall Score: 1.0
F1 Score: 0.8686868686868687
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        13
           1       0.77      1.00      0.87        43

    accuracy                           0.77        56
   macro avg       0.38      0.50      0.43        56
weighted avg       0.59      0.77      0.67        56
```

When we tried a dropout rate of 0.2 for all layers, which was equally as bad as 0.1 for all layers.

We expected to get an improvement when we had a dropout rate of 0.1 for 2 layers and 0.2 for the third. This decision was justified on the basis that there is more overfitting in the last layer than in the earlier layers so there is also more of a need to inactivate redundant perceptrons in the last layer. However, this did not change our results from 0.2 dropout for all layers.

ANN Results (Sigmoid):

Sigmoid

The same results were obtained with sigmoid function being the candidate activation function for all layers.

```
Accuracy Score: 0.7678571428571429
Precision Score: 0.7678571428571429
Recall Score: 1.0
F1 Score: 0.8686868686868687
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        13
           1       0.77      1.00      0.87        43

    accuracy                           0.77        56
   macro avg       0.38      0.50      0.43        56

weighted avg       0.59      0.77      0.67        56
```

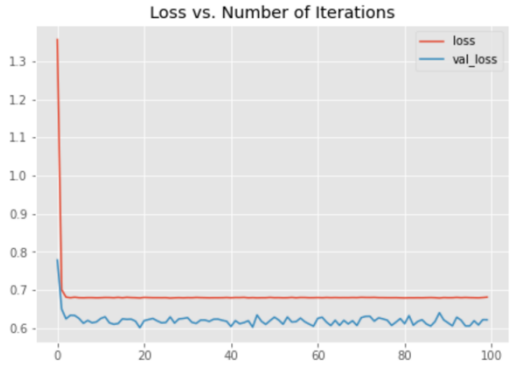
ANN Results (Mixed Network with Both ReLu and Sigmoid):

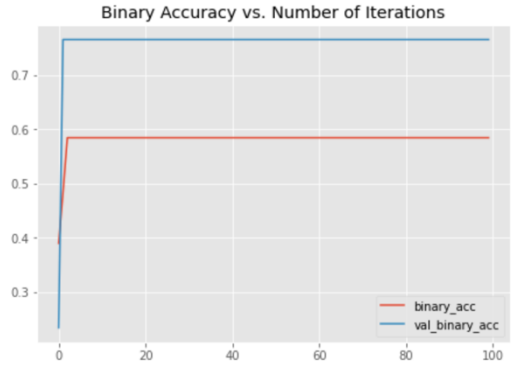
We were able to achieve more optimal functions with a combined network which incorporates sigmoid in the last layer and relu for the first 3.

```
Accuracy Score: 0.8571428571428571
Precision Score: 0.972972972972973
Recall Score: 0.8372093023255814
F1 Score: 0.9
Classification Report:
              precision    recall  f1-score   support
           0       1.00      0.85      0.92        13
           1       0.96      1.00      0.98        43
    accuracy                           0.96        56
   macro avg       0.98      0.92      0.95        56
weighted avg       0.97      0.96      0.96        56
```

Below are the plots showing how loss and binary accuracy change with the number of iterations. Notice that unusually, binary accuracy during training (binary_acc) is a lot lower than binary accuracy during validation (val_binary_acc). This is even more strange, when considering, that kfold cross-validation was implemented, so the test set should in theory be derived from the same distributions as the training partitions. One possible reason why this might be happening is

because of the utilization of dropout, which prevents overfitting to the training set, and in this case better generalizes to the validation set even more so than the set it was trained on. Yet, despite this wonderous miracle that the deep ANN generalizes very strongly to the test set, the binary accuracy is still relatively bad compared to the other networks.





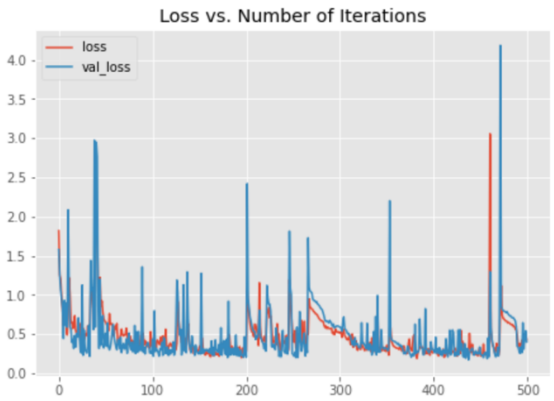
Adam Optimizer:

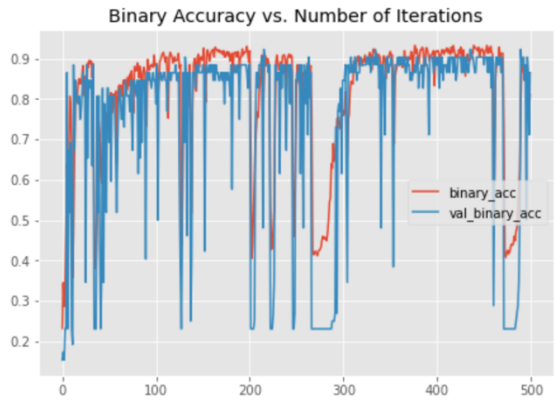
We got our best results using the Adam optimizer instead of using stochastic gradient descent.

```
Accuracy Score: 0.9464285714285714
Precision Score: 0.9761904761904762
Recall Score: 0.9534883720930233
F1 Score: 0.9647058823529412
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.92      0.89        13
           1       0.98      0.95      0.96        43

    accuracy                           0.95        56
   macro avg       0.92      0.94      0.93        56
weighted avg       0.95      0.95      0.95        56
```





Above are the plots for loss and binary accuracy plotted against the number of iterations. Note that there are more peaks and more jagged peaks and inconsistencies in the data in terms of sudden rises and drops of the peaks. However, the loss and binary accuracy values for both loss and validation remain very close to each other as the number of iterations increases, inspiring confidence.

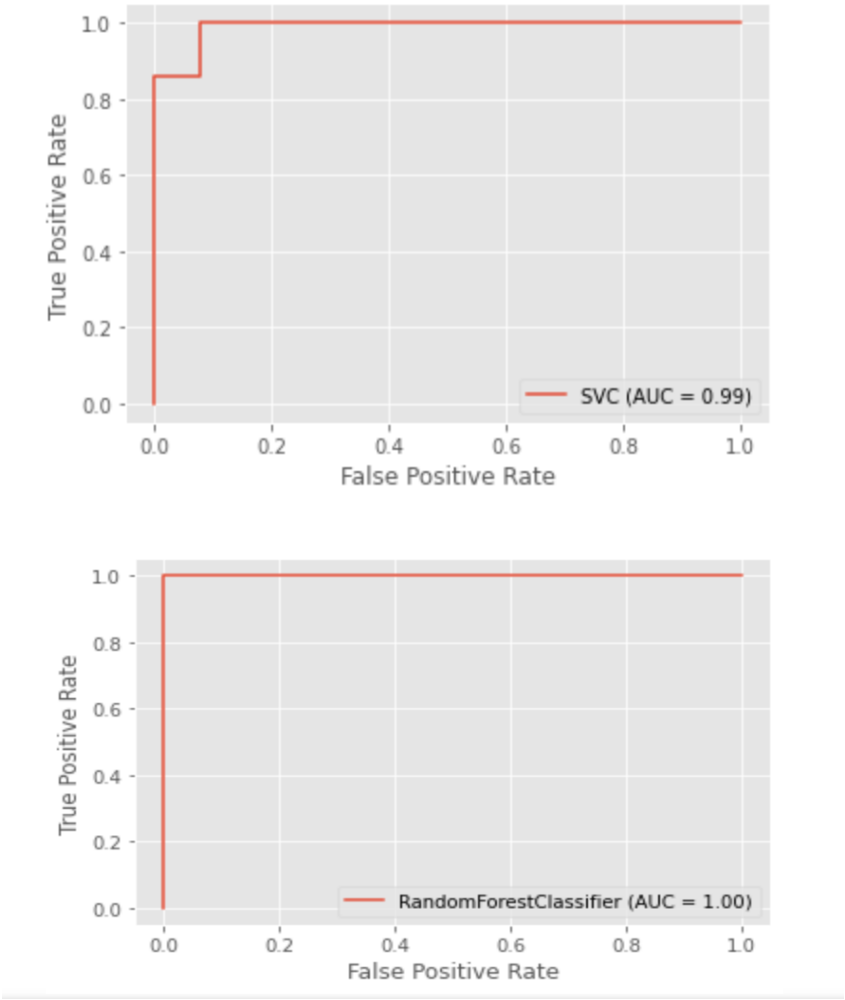
Determining Which Method is the Best

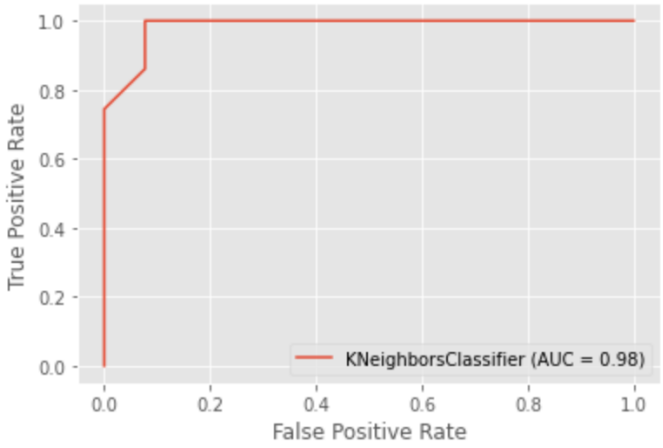
The best method is the one which has high scores in the important performance measures and also has a reasonable runtime. The artificial neural network with Adam has high performance measures that are close to those seen in decision tree with entropy, polynomial kernel SVM, and KNN. However, the artificial neural network is also very slow, as it requires 100 to 500 iterations. Also, the ANN is still worse in accuracy than both random forests, both SVMs, and KNN. The ANN also has subpar values for precision and f-score for classification 0. Because of all these factors, we can eliminate the ANN.

The Decision tree using entropy has an accuracy score extremely similar to that of the ANN with adam which is not that high. Moreover, the precision and f-score for classification 0 is not impressive (under 90%) so we can eliminate decision trees.

When comparing the last three methods, we can utilize ROC curves. It is no surprise that the ROC curve for the random forest classifier (with gini impurity) immediately shoots up straight to one at the start of the graph, as it has perfect accuracy which also implies perfect sensitivity and specificity which are measured by the ROC. Looking at the ROC curves for SVM with the polynomial kernel and KNN, these also are extremely strong, albeit worse than that for

the random forest with gini impurity. Moreover, the minor differences in the ROC curves between the SVM and KNN mirror how similar their performance statistics are. Moreover, the slight difference in slope between the two ROCs reflect very small differences in the area under the curve, so it is difficult to justify paying attention to such a minor difference, when considering that the random forest classifier is actually 100% accurate, sensitive, and specific.





Overall, I believe accuracy was the most important factor to consider which is why random forests with the gini impurity were proposed to be the best method. Accuracy is a measure of overall correctness, regardless of whether the individual has a malignant tumor or benign one. Thus, since receiving wrong diagnoses concerning malignant tumors and benign tumors are both very bad, accuracy should be the determining factor for which method is the best. Thus, I propose random forest with the Gini impurity to be our best classifier. Overall, even though run time is important to consider, runtime does not inspire confidence in patients and doctors. All of the non-neural network methods are quick and relatively easy to implement when compared to ANNs. However, if one valued time complexity more than performance metrics, then the random forest with the Gini impurity would not be ideal as it is very slow and slower than every other method except for the deep neural networks. If speed was more of a factor, then one would probably choose the SVM with polynomial kernel which has the fastest runtime and also has a high accuracy that is greater than 98%.

What I have learned:

This project made more familiar with how the methods we learned about in class are actually implemented. For instance, I did not know about that there were different optimizers data miners use when running deep artificial neural networks. Similarly, I also did not realize that there was

another split criteria in decision trees besides entropy. If I could redo this project, I would try to test each implementation on more parameters. For instance, I could have tested the artificial neural network on the RMSprop or momentum optimizers and I could have also tried to test the SVM on a different distance measure besides the dot product. One piece of advice I would give to others is to be willing to research other possible implementations of the methods than the ones you are used to---there are a lot of great differences that can be applied to an implementation just due to subtle changes in hyperparameter values like the type of activation function used.