Author: Nurzhan Kanatzhanov
Instructor: Weixiong Zhang
Course: CSE 514 - Data Mining
Date: April 17, 2021

1 Introduction

As the saying goes, there's also no "free lunch" in machine learning and data mining. Determining how to select the "best" DM method or algorithm is important in practice and it depends on a multitude of factors, including the type of problem at hand (dataset), learning methods with their respective hyper-parameters, and the type of output that we are looking for.

Some of the more important factors to consider are the size of the training data, number of features, and if we are striving for high accuracy or high interpretability of our classification model. For instance, when there are not that many observations for training data but a decent amount of features to be considered, we might want to prioritize algorithms that have a high bias and a low variance (e.g. linear SVM). On the other hand, if our data is sufficiently large, we can go for low bias/high variance algorithms (e.g. k-Nearest Neighbors, kernel SVMs, Random Forest, Decision Trees, etc.). Similarly, the common accuracy-interpretability trade-off that these DM algorithms encounter is also important, because if inference is our final goal, then we might go for algorithms that are more restrictive, but at the cost of lower accuracy. If we want to be as accurate as possible, then we might go for highly flexible algorithms (like SVMs, random forest) that have a low interpretability.

Ultimately, it all comes down to the objectives of our problem. Of course, with each individual classification model it is also important for us to distinguish between certain similarity/distance measures as well as different performance evaluation methods and techniques to evaluate predictive models (e.g. we are using 10-fold cross-validation in this project).

2 Methods and Programs Used

In this project, I'm using Jupyter Notebooks and Python (version 3), along with Python's famous `scikit-learn` and `matplotlib` libraries to use 10-fold cross validation and all the classification methods. Specifically for the 10-fold cross-validation technique, I am using `sklearn`'s `cross_val_score` function all throughout my project. The `cross_val_score` function, by default, uses k-fold cross-validation. This works by splitting the data set into k equal folds. Thus, if we have 10 folds (fold1, fold2, fold3, ...), then the algorithm works as follows:
(1) Use folds 1-9 as our training set for any classification model and test performance on fold10.
(2) Use fold 1 and folds 3-10 as our training and test performance on fold2.
So each fold is used for both training and testing. You got the idea.
The `cross_val_score` function estimates the accuracy on the test set for every fold, and then I average out the results of the 10 folds. However, because we are using 10-fold cross validation, I pass 10 to the `cv` parameter, which specifies the number of folds for a **stratified** k-fold, which is a variation of default k-fold that returns stratified folds. The folds are made by preserving the percentage of samples for each class, which is good for us, because there are 458 benign samples and 241 malignant samples in our cancer dataset (different proportions).

Below is the list of candidate methods with their respective parameters that I am describing in this report:

- k-Nearest Neighbors (kNN) with Eucledian, Manhattan, and Minkowski distances
- Decision Tree with gini impurity and entropy/information gain splitting criteria
- Random Forest with different number of trees in the forest, splitting criteria, and bagging
- Polynomial kernel SVM
- Gaussian kernel SVM
- Deep neural network with Sigmoid activation function
- Deep neural network with ReLu activation function

README: If you are trying to run a Jupyter Notebook (.ipynb file), there are a lot of guidelines online of how to do it, but some of the best I found are here: How to Use Jupyter Notebooks.
To run a piece of code, click on the cell to select it, then press SHIFT+ENTER. Alternatively, you could press on "Cell" and then "Run All" to run all of the code cells at once. The code can also be seen in the PDF along with all the descriptions, comments, tables, and graphs I left there for you to look at.

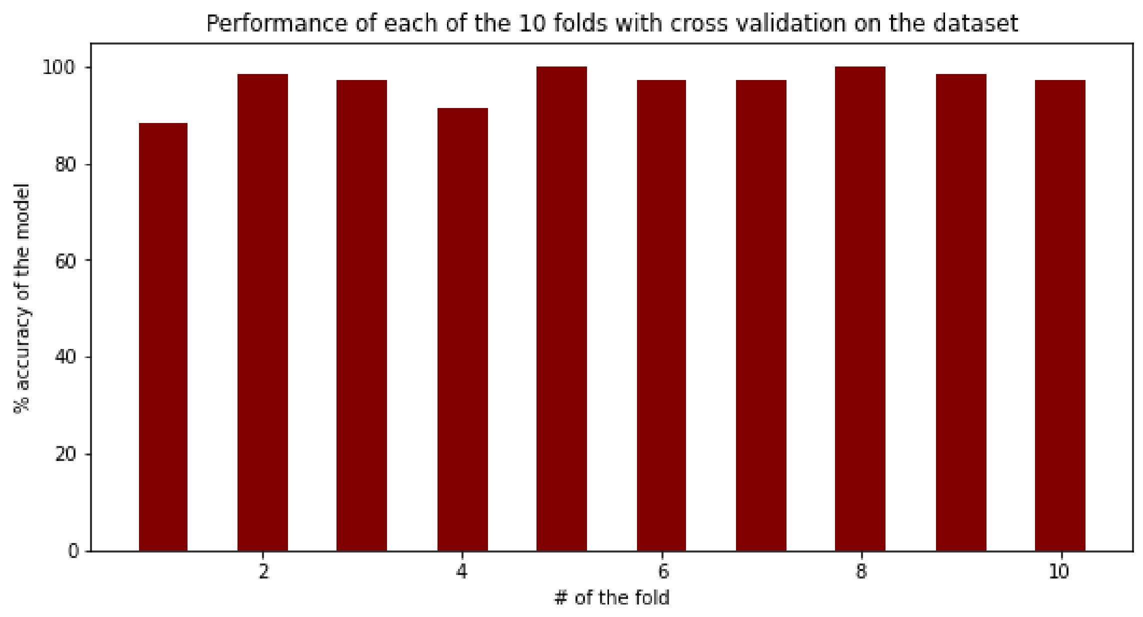
If you need some more explanation, don't hesitate to contact me at: nurzhan.kanatzhanov@wustl.edu.

3 Pre-processing of the Data

After loading the data breast cancer data from UCI breast cancer dataset, I noticed that there are 16 instances of missing values in the "Bare Nuclei" column, so let's fix them first. Because those empty values appear as NaN in my dataframe, I quickly fix the NaN values by randomly choosing a value depending on the range of values in the specified column (Bare Nuclei). I also drop the "Sample code number" column from the dataset as it is not needed for the learning of the algorithm for each candidate method. I divide the input features (X) and the target values (y) - the target values are the "Class" column that have either of the two values {2,4}, where 2 stands for benign and 4 stands for malignant samples of breast cancer. We are now ready to train and analyze each candidate.

4 K-Nearest Neighbors (K-NN)

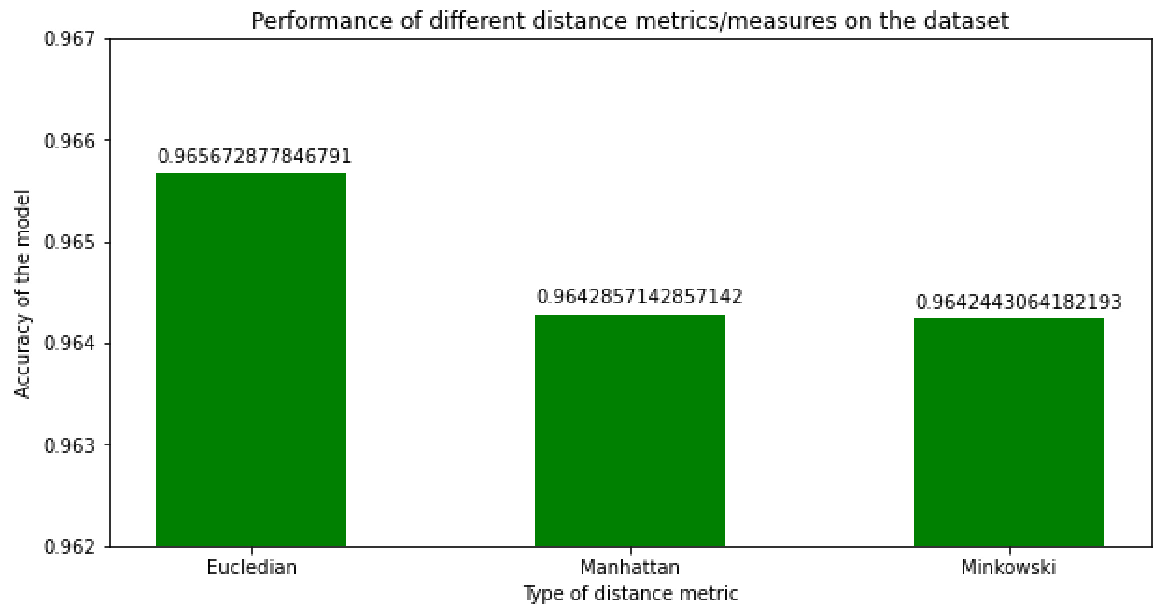
First, I run through the y-values and count up how many samples are benign and how many are malignant. There are 458 benign samples and 241 malignant samples in the dataset, so I might want to consider using a stratified k-fold method, as it is important to preserve the same distribution of samples in the train and test set. Using Python's extensive `sklearn` library, I first use the default `KNeighborsClassifier` model to train with a 10-fold cross validation technique. In order to train and test our k-NN model using cross-validation, we will use the `cross_val_score` function with a cross-validation value of 10. `cross_val_score` takes in our k-NN model and our data as parameters. Then it splits our data into 10 groups and fits and scores our data 10 separate times, recording the accuracy score in an array each time. Here is the output:



Using 10-fold cross-validation, our mean score is about **96.57%** (accuracy). This is a more accurate representation of how our model will perform on unseen data than if we were not using cross-validation.

Now, onto the metrics:
Recall: 0.9543 = 95.43%
Precision: 0.9490 = 94.90%
F1-score: 0.9504 = 95.04%
Running time elapsed: 0.08382 seconds

In terms of the used **distance measure**, I went with the default Euclidean distance to classify data points using a K-NN classifier. Let's see if the accuracy of our algorithm gets better if we use other distance measures:

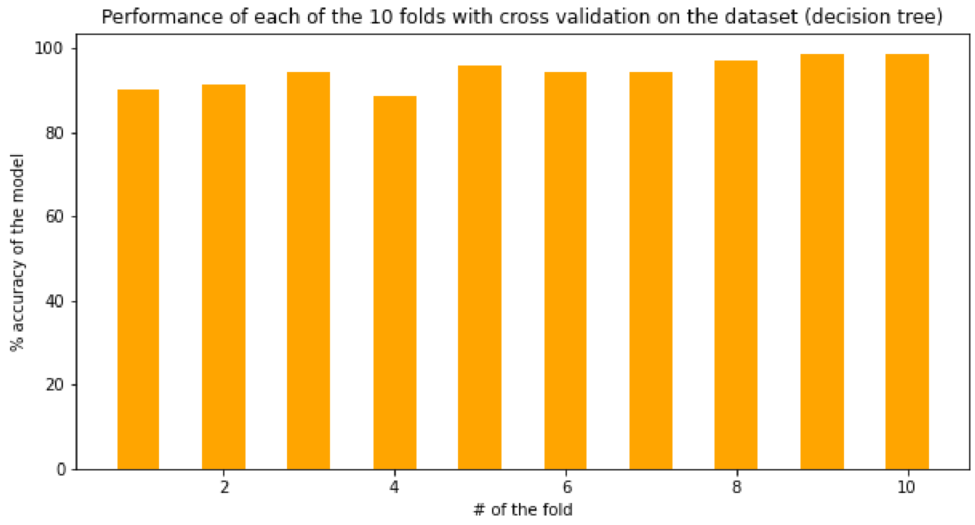


In fact, it does not. So, I do not see a reason to use a different distance measure than the default

Eucledian distance for k-nearest neighbors.

5 Decision Tree

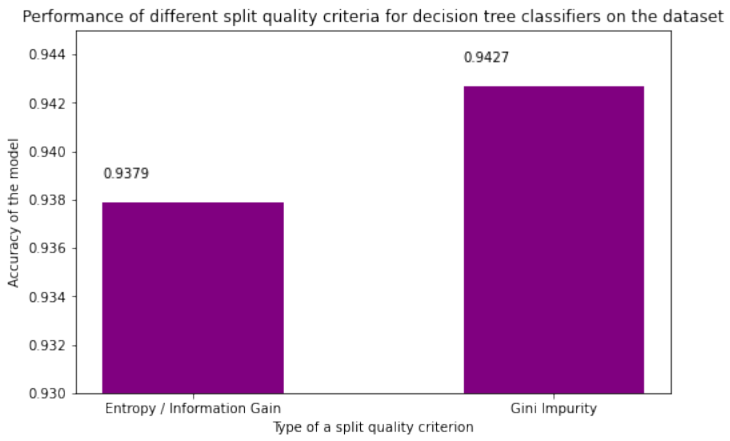
Similarly, we will use the `cross_val_score` function with a cross-validation value of 10. `cross_val_score` takes in our `DecisionTreeClassifier` and our data as parameters. Then it splits our data into 10 groups and fits and scores our data 10 separate times, recording the accuracy score in an array each time. Here is the output:



Using 10-fold cross-validation, our mean score is about **94.28%** (accuracy).

The cross-validation performance metrics are below:
Recall: 0.9252 = 92.52%
Precision: 0.9255 = 92.55%
F1-score: 0.90828 = 90.83%
Running time elapsed: 0.2180 seconds

Since there are no actual distance measures for decision tree classifier per se, we will see if using different splitting techniques would improve our algorithm. We will use different attribute selection measures then. Attribute selection measure is a heuristic for selecting the splitting criterion that partition data into the best possible manner. By default, the `sklearn` library's decision tree classifier uses the Gini impurity function to measure the quality of the split. Now, using the entropy criterion as the primary measure of a quality of a split of a decision tree, I ran classification tests 1000 times and averaged out the accuracies of both methods. Here are the results:



In fact, the Gini impurity criterion is more accurate for our breast cancer dataset. The gini impurity measures the frequency at which any element of the dataset will be mislabelled when it is randomly labeled, whereas entropy is a measure of information that indicates the disorder of the features with the target.

$GiniIndex = 1 - \sum_j p_j^2$ $Entropy = -\sum_j p_j \cdot log_2 \cdot p_j$ (where p_j is the probability of class j)

Because entropy uses logarithms, its calculation is a bit slower than the calculation of a gini index, so it is good that using a gini criterion for our dataset, we are both faster and (a bit) more accurate.

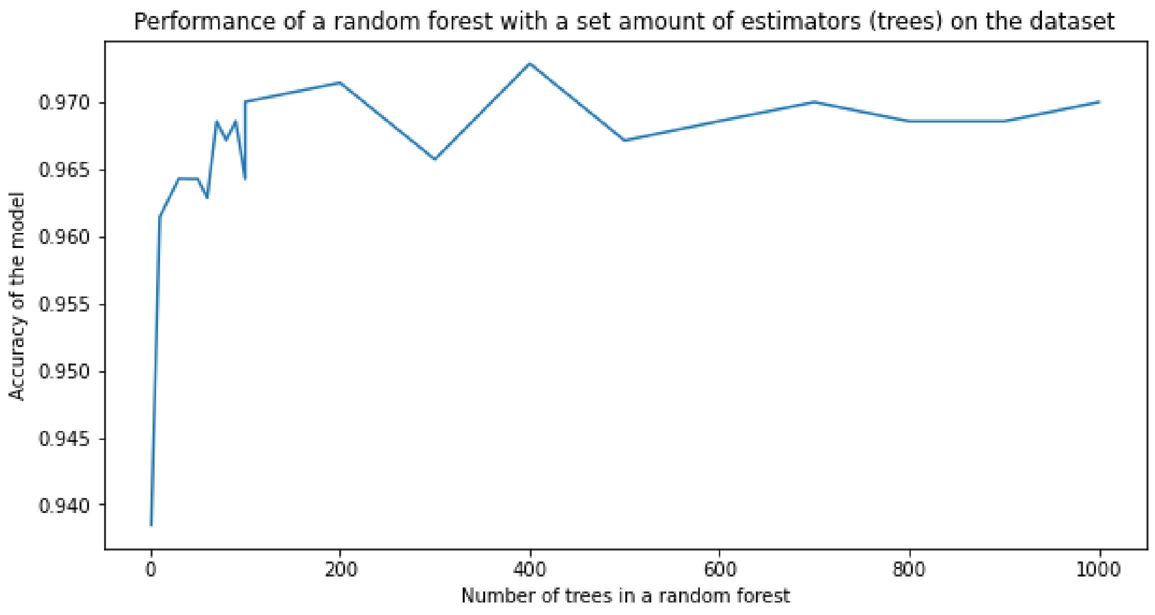
6 Random Forest

I won't do more graph comparisons on the performance of each fold with 10-fold cross validation on the dataset, but dive in more on the actual parameters for the random forest classifier. Since I used the `cross_val_score` function again with the `sklearn ensemble` library's `RandomForestClassifier`, the default parameters are as follows:
100 trees in a forest for training, all using the gini criterion for splitting, with bootstrapping (bagging), and not using out-of-bag (OOB) samples. We can tweak all these parameters and see if changing some of these similarity measures will provide any effect on the random forest classifier for this dataset.

But first, here's the performance of the default classifier:
Accuracy: 0.9657 = 96.57%
Recall: 0.9585 = 95.85%
Precision: 0.9538 = 95.38%
F1-score: 0.9550 = 95.50%
Running time elapsed: 1.0836 seconds

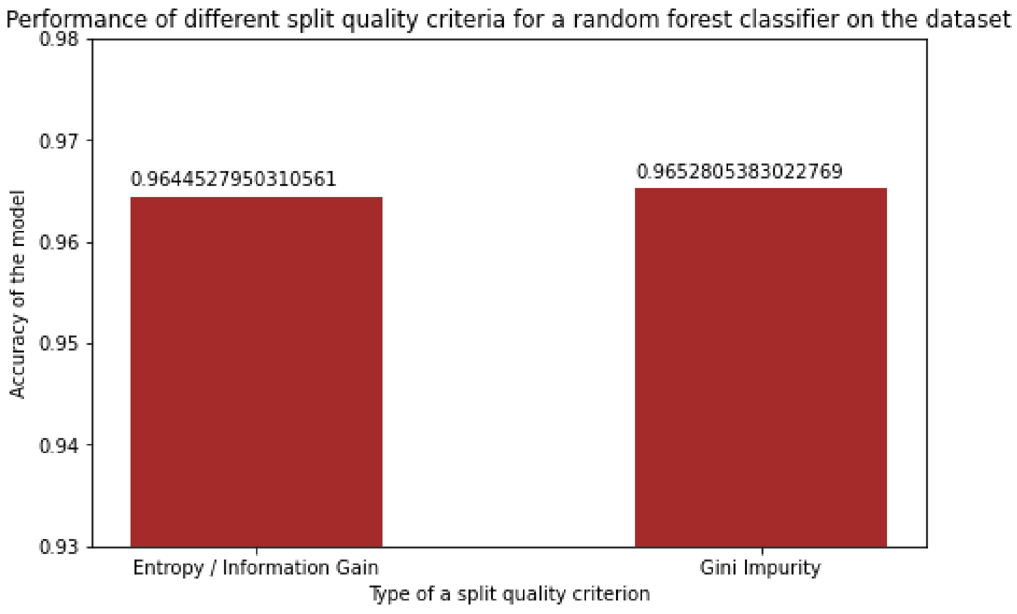
We can clearly see that when working with random forest, since there are 100 decision trees in the forest, the running time of the classifier is longer on average than the single decision tree classifier (1.08 seconds compared to 0.22 seconds).

With random forests, if we keep everything constant but change the number of estimators (trees) in the forest, here are the results (up to 1000 trees in the forest, with 100 being `sklearn`'s default):



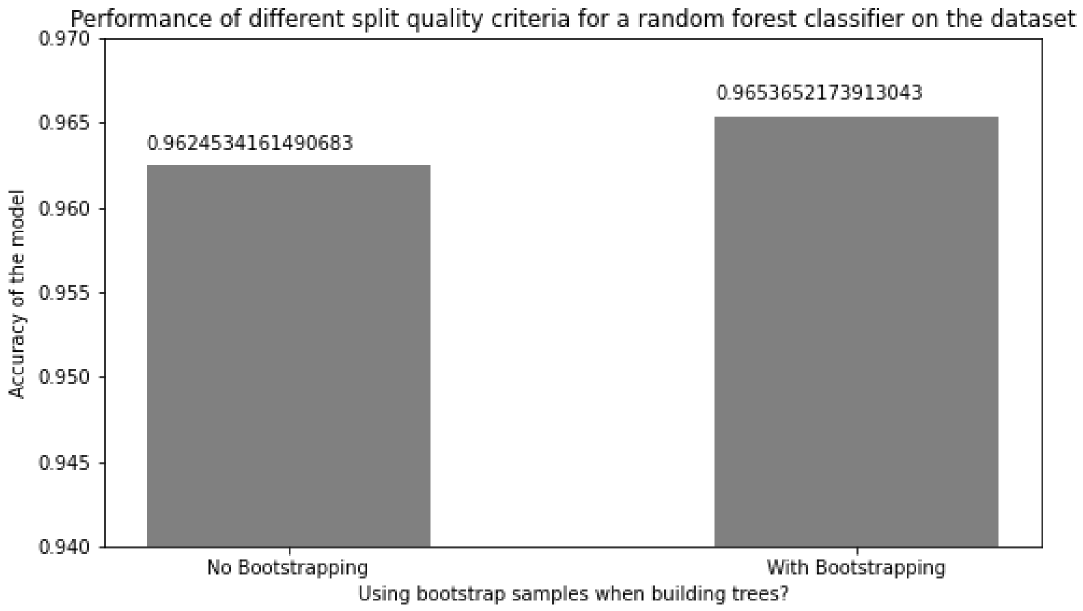
We can see that with our dataset, around 400 trees is our highest accuracy with 10-fold cross-validation, which is about $0.9728 = 97.28\%$.

Next, let's see which splitting criterion is more accurate for random forest classifier:



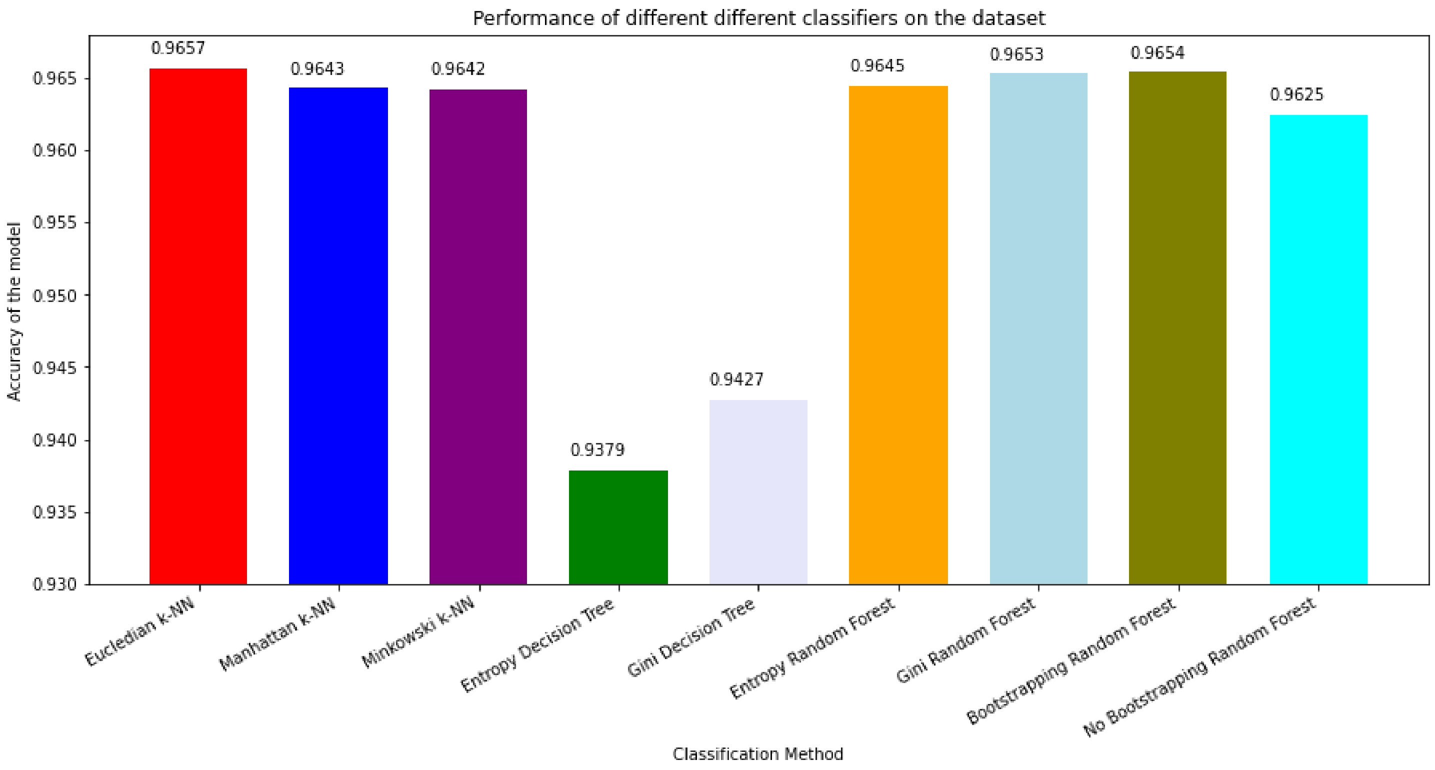
After running 100 tests with both the gini impurity and the entropy/info gain criteria on random forest classifiers (with a default of 100 trees in the forest and bootstrapping), the results are barely different, only about 0.08% different from each other.

Now, the final parameter/attribute that we can tweak is the use of bootstrap samples when building tree. The parameter is set to true by default, but let us see if anything changes if we turn bootstrapping off:



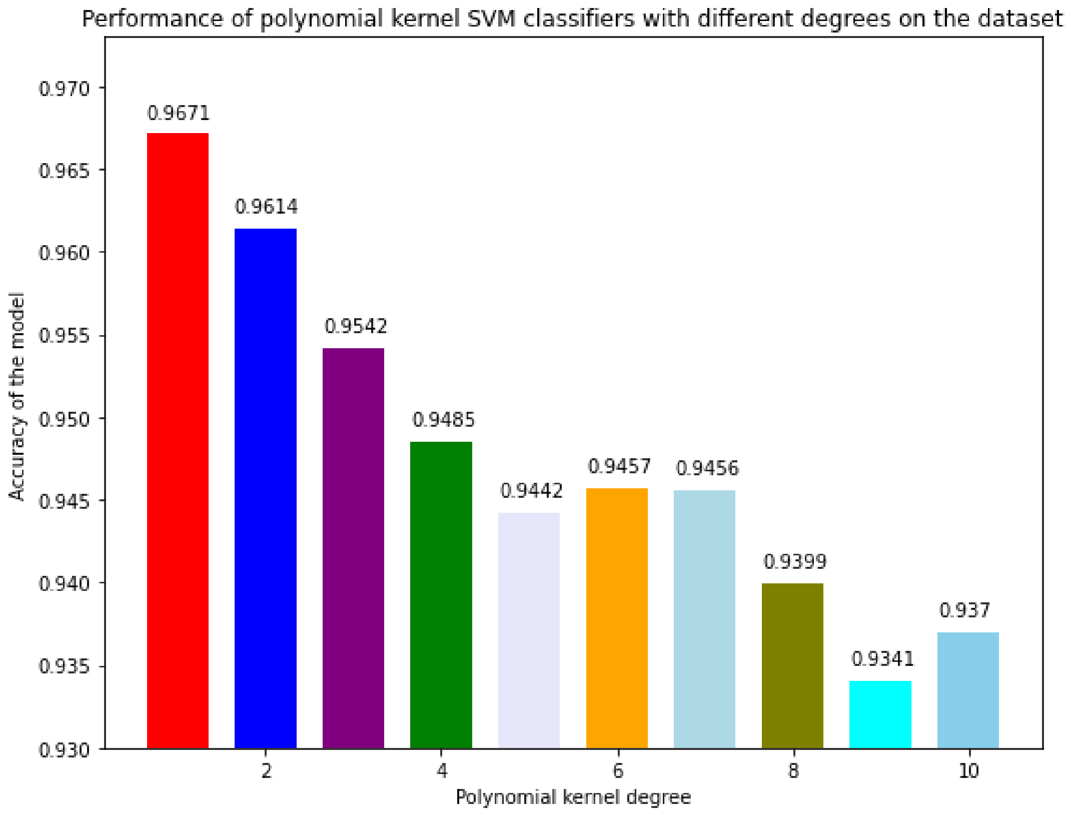
What is bootstrapping though? Bootstrapping is a statistical re-sampling technique that involves random sampling of a dataset with replacement. Bootstrapping is actually extremely useful since it allows generation of new samples from a population without having to go and collect additional ”training data”. So, it seems like when we use bootstrapping and 10-fold cross validation together with random forests, our accuracy is a bit better (I averaged it out after 50 runs).

Let’s take a look at the comparison of all classification methods/algorithms with their respective parameters so far:



7 Polynomial Kernel SVM

Even though we are allowed to just use any polynomial kernel of our choice, I wanted to see how polynomial kernels with different degrees would affect the accuracy of our classification model. Starting from degree of one (1) (which is a bit different than just a linear kernel because of the additional parameter/coefficient called `coef0`) and up to degree ten (10), below is a bar graph of the accuracies of our models on the breast cancer dataset:



Very interestingly, a polynomial kernel with degree just one outperforms the other kernels with higher degrees. There is even a downward sloping trend (inverse relationship) between the average accuracy of the model and the degree of the polynomial kernel of the SVM model.

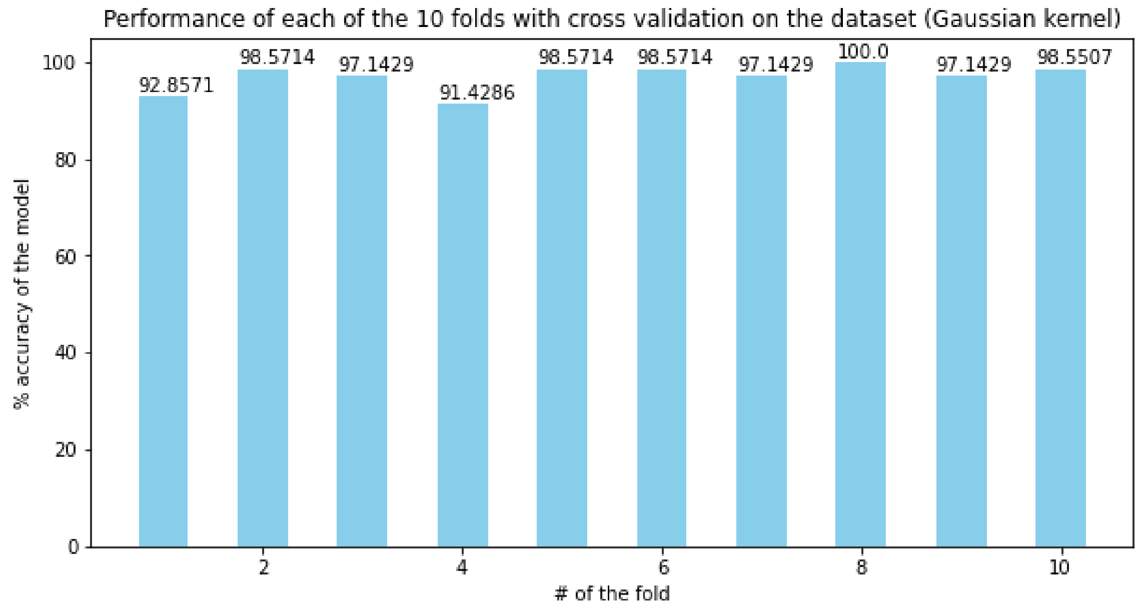
Here is the 10-fold cross-validation performance (averaged):
Accuracy: 0.9671 = 96.71%
Recall: 0.9587 = 95.87%
Precision: 0.9503 = 95.03%
F1-score: 0.9531 = 95.31%
Time elapsed: 0.06444 seconds

8 Gaussian Kernel SVM

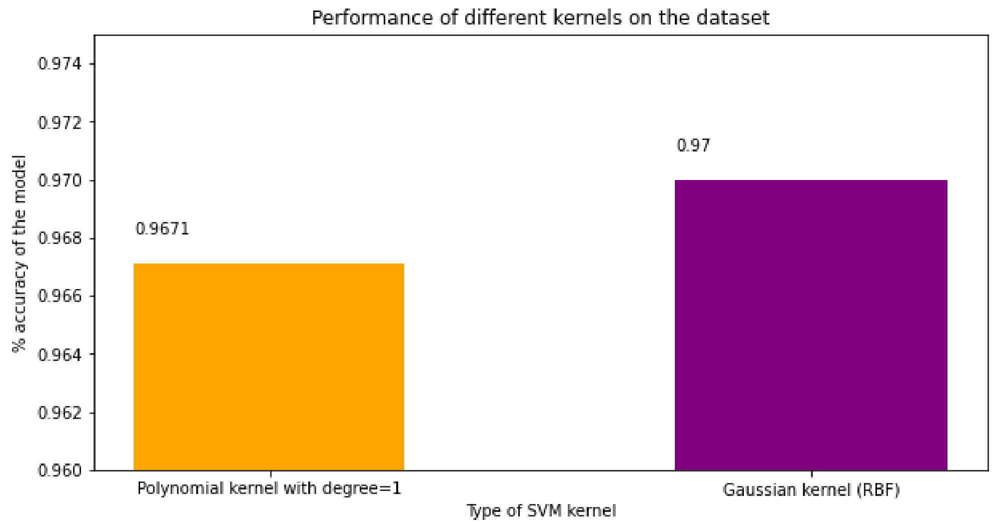
For the Gaussian kernel, we are using the `rbf` kernel, which stands for radial basis function (the two can be used interchangeably). The RBF kernel on two samples x and x', represented as feature vectors in some input space, is defined as:

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

Here's the performance of the Gaussian RBF kernel on each of the 10 folds of our breast cancer dataset:



We can see that even on one of the folds (#8), we reach 100% prediction accuracy for the model. Surprisingly, the polynomial kernel with degree of one and the Gaussian kernel SVMs are not that different in terms of average accuracy with 10-fold cross validation:



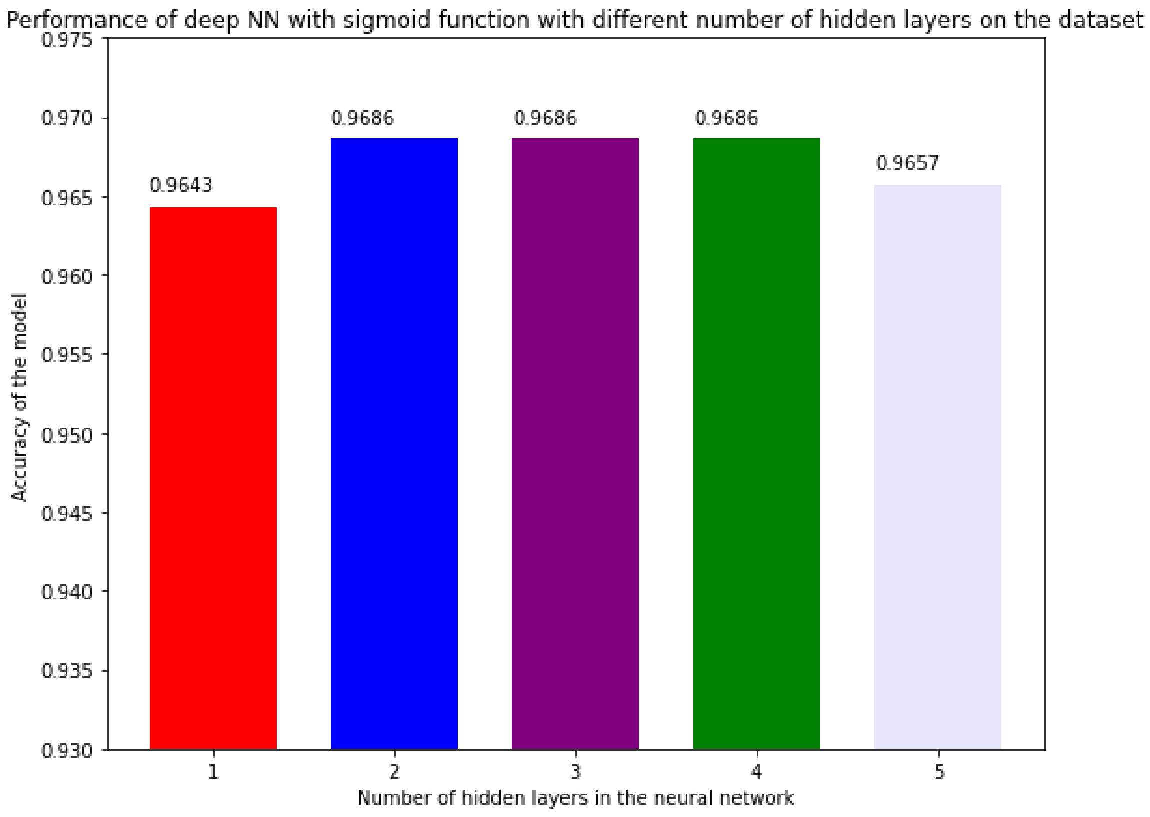
The two are only 0.29% away from each other in terms of accuracy, with the Gaussian kernel being a bit better on our dataset. Here are the other performance measures for the Gaussian kernel SVM:

Accuracy: 0.9700 = 97.00%
Recall: 0.9752 = 97.52%
Precision: 0.9423 = 94.23%

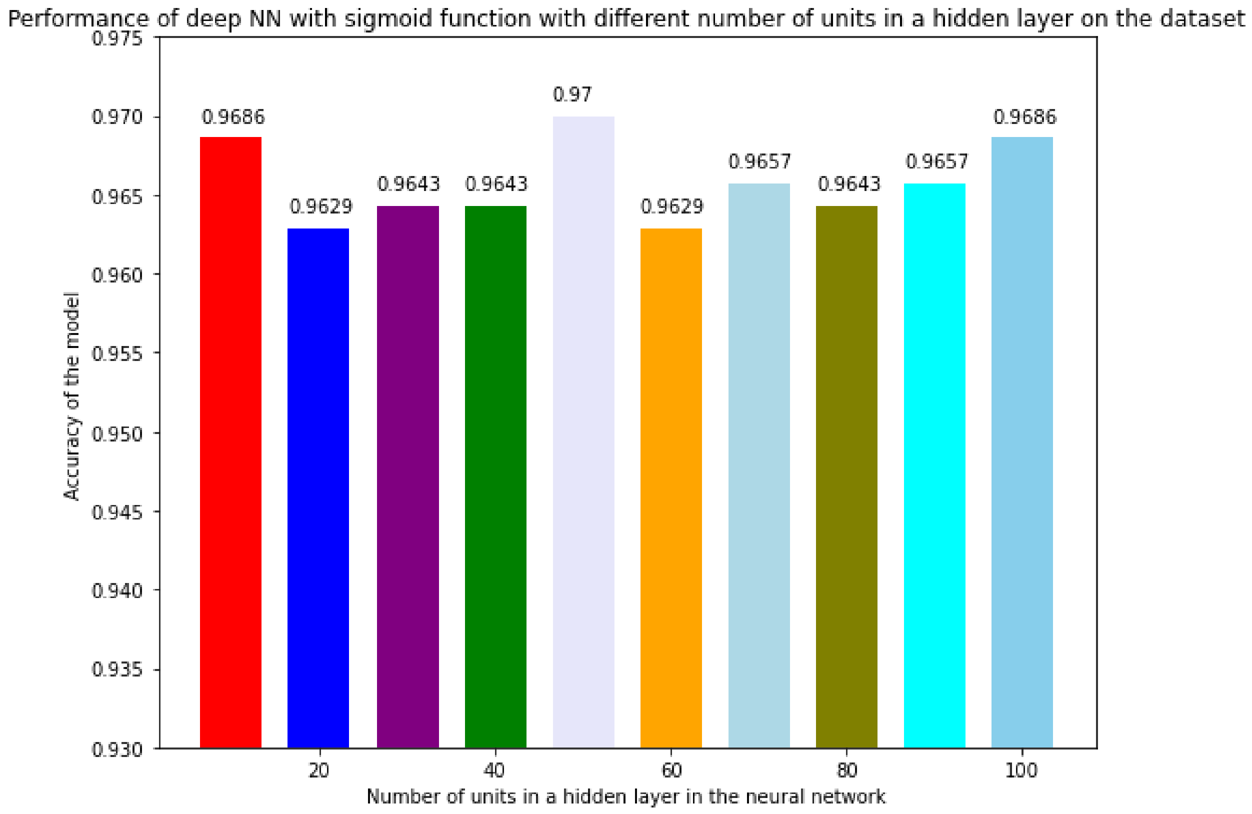
F1-score: 0.9578 = 95.78%
Time elapsed: 0.08167 seconds

9 Deep NN With Sigmoid Activation Function

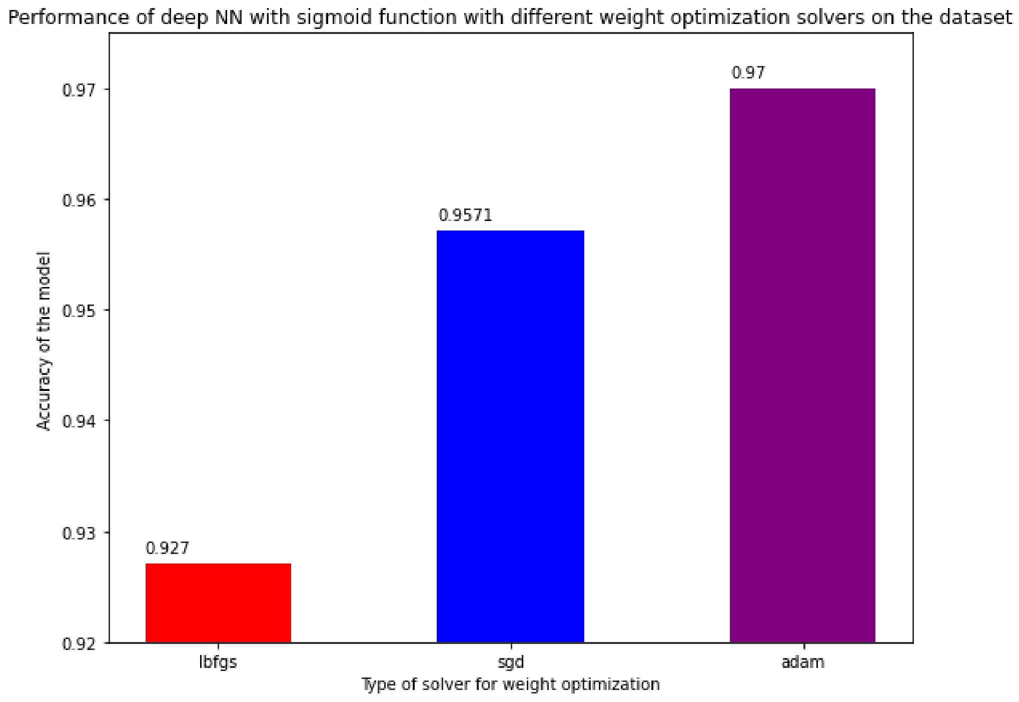
For the deep neural networks, I am going to use `sklearn`'s `MLPClassifier`, which a Multi-layer Perceptron classifier. By default, this neural network has 3 layers—one input layer, a single hidden layer with 100 units, and an output layer. We will play around with the number of hidden layers and units inside of them below. The classifier also uses a so-called `adam` solver for weight optimization, which refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba. By default, we are also using an alpha of 0.0001 (L2 penalty parameter - regularization term), as well as a constant learning rate of 0.001. We have options to switch to an adaptive or inverse scaling learning rate for stochastic gradient descent of our neural network. Let's first see if the number of hidden layers will have a large effect on the prediction accuracy of our breast cancer dataset:



It looks like more hidden layers does not mean more accuracy. With 2, 3, and 4 hidden layers in the neural network, it looks like the model caps at about 96.86 percent accuracy.
Now, let's see if we switch the number of units in a hidden layer of a neural network:



On our relatively small dataset, looks like a higher number of units in a hidden layer also does not help, as the maximum out of the numbers is around 50 units for a hidden layer, with an improved accuracy of 97.00%.
Now what if we compare the solvers for weight optimization of our deep neural network:



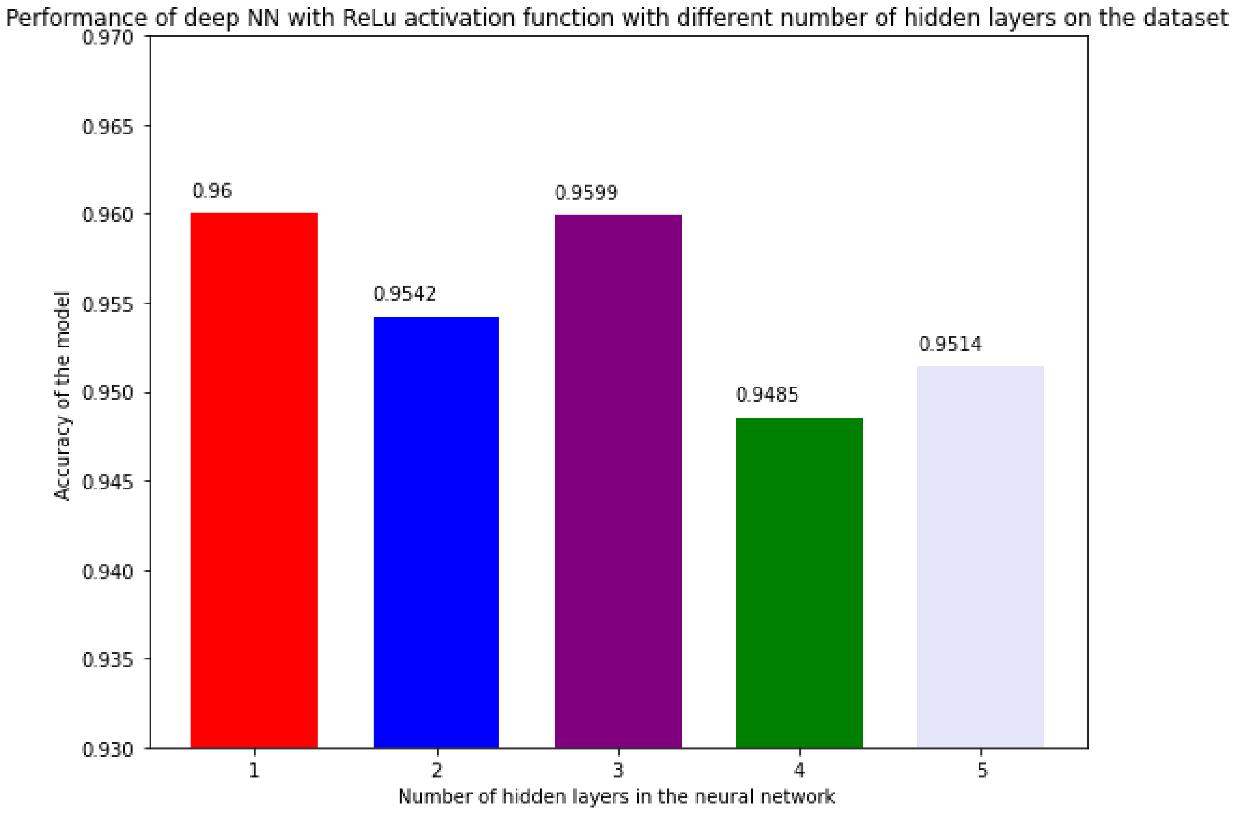
`lbfgs` stands for Limited Broyden–Fletcher–Goldfarb–Shanno algorithm, which is an optimizer in

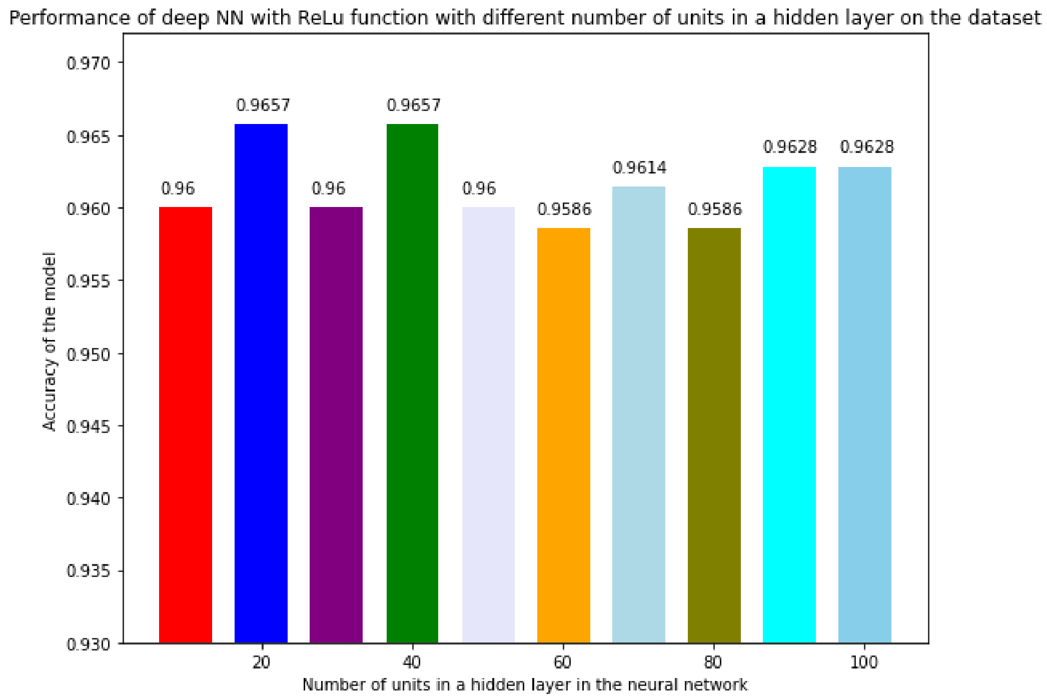
the family of quasi-Newton methods. L-BFGS determines the descent direction by preconditioning the gradient with curvature information. `sgd` stands for our normal Stochastic Gradient Descent, and I already mentioned above that `adam` is an SGD optimizer. It really looks like on our dataset, `adam` works noticeably best.
As usual, performance metrics (default MLP):

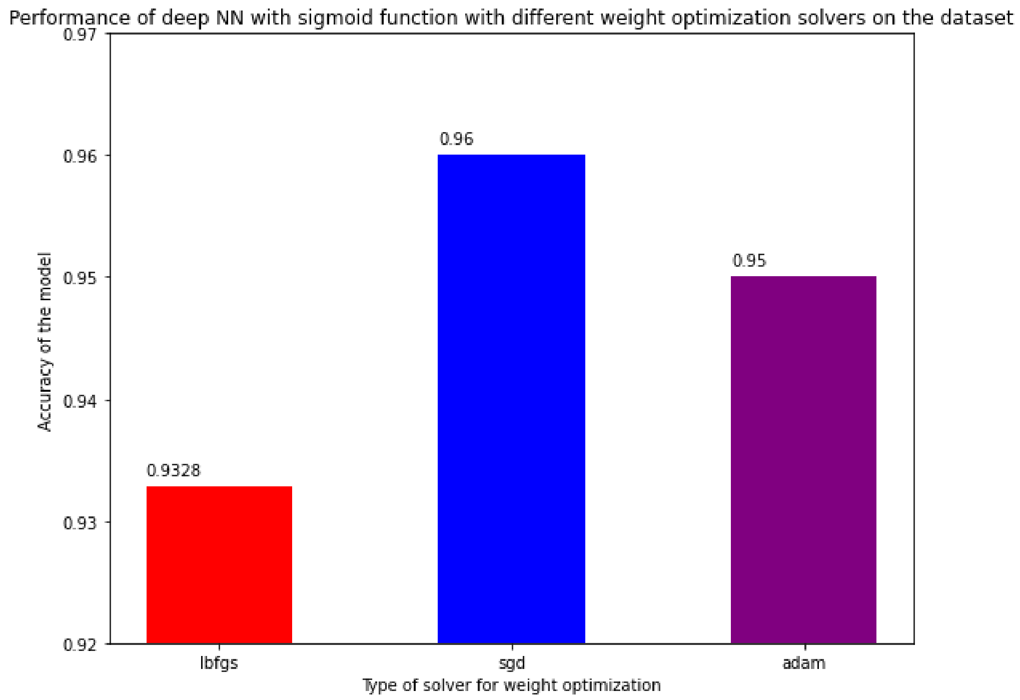
Accuracy: 0.9657 = 96.57%
Recall: 0.9502 = 95.02%
Precision: 0.9505 = 95.05%
F1-score: 0.9520 = 95.20%
Time elapsed: 7.1574 seconds

10 Deep NN With ReLu Activation Function

This one is exactly like the sigmoid function deep neural network, but I just change the activation function parameter to `relu`. Let's perform the same tests:



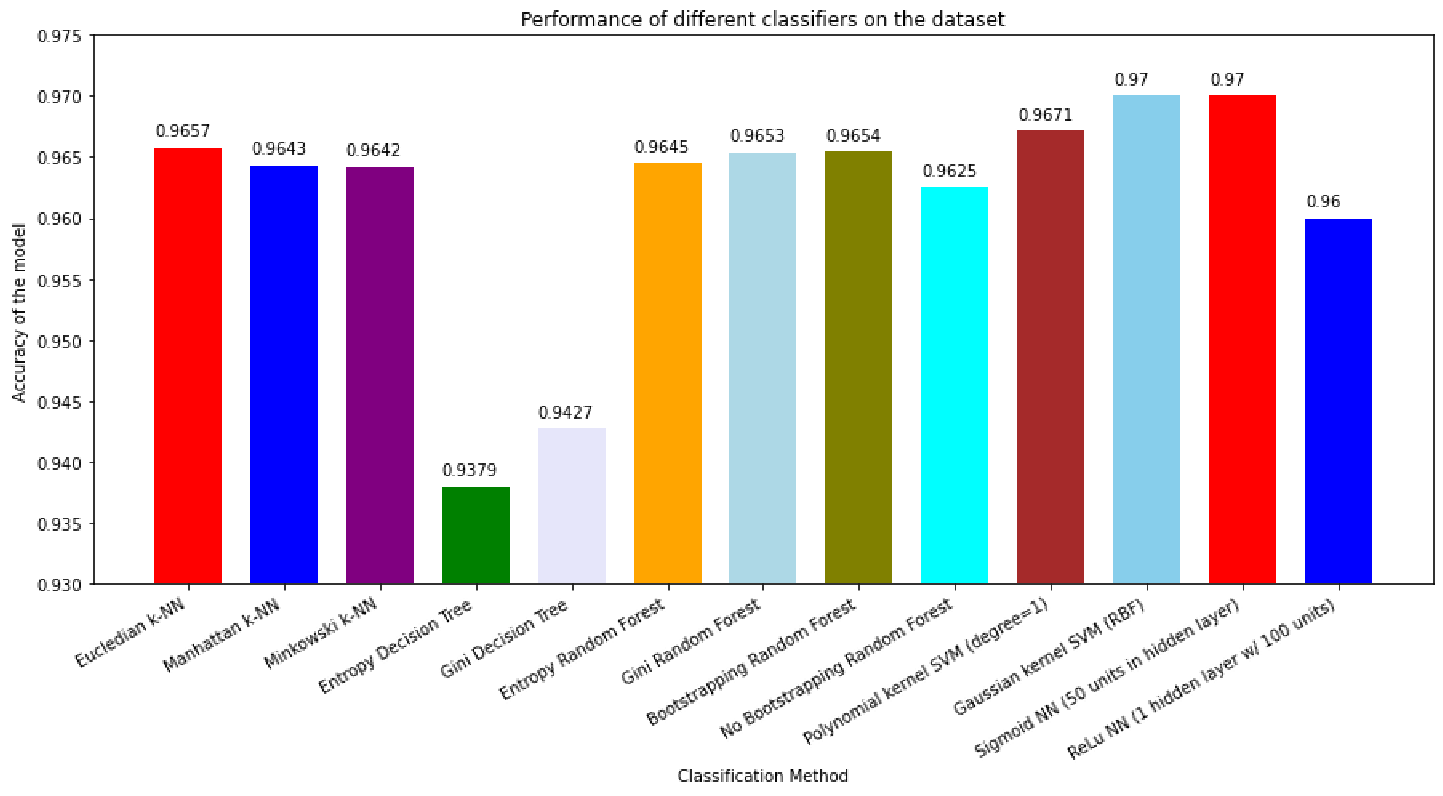




For the ReLu activation function, the results look pretty similar to the sigmoid results, except that the best solver for weight optimization is the original stochastic gradient descent rather than the `adam` parameter.

11 Final Comparison

This is the moment we have all been waiting for! Which of the classification methods performs the best on our breast cancer dataset? See the results below:



The top 5 prediction accuracies are as follows:
1 (tie). Gaussian kernel SVM (RBF) $\rightarrow$ **97%** accuracy
1 (tie). Deep NN with sigmoid activation function (50 units in hidden layer) $\rightarrow$ **97%** accuracy
3. Polynomial kernel SVM (degree = 1) $\rightarrow$ **96.71%** accuracy
4. Eucledian distance k-nearest neighbors (5 neighbors) $\rightarrow$ **96.57%** accuracy
5. Random forest with bootstrapping $\rightarrow$ **96.54%** accuracy

12 Conclusion

Throughout this report, I have really gained a lot of respect for data scientists (and overall people who work with data mining and machine learning), who day in and day out have to work with a lot of raw data and test out multiple procedures and hypotheses, as well as hypertune tons of parameters in order to increase prediction accuracy on some datasets just by a tiny margin, which is not at all easy to do. While the final results that I got are only applicable to this specific UCI breast cancer dataset, I would highly advise the next person who reads this report to be careful with choosing models to best predict values. Sometimes, it is good to first know what the objective of the problem with certain data is. I am glad that in this assignment, the classification methods were listed from more basic to more complex (like ensemble methods and neural networks), as it showed me how many different things there are to consider when building a good classification model.