Tuning Classification Models

Doran Wu

December 01, 2021

A. ABSTRACT

This report explores various classification models for a specific dataset, Breast Cancer Wisconsin dataset from the UCI repository (https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/). By taking into account two primary factors: computational complexity and performance after hyperparameter tuning from K-Fold cross-validation, the report will attempt to justify the usage of the best model for this given dataset and classification problem.

B. INTRODUCTION

Problem and Goal

The dataset, Breast Cancer Wisconsin, contains 9 feature variables that are attributed to tumors growing on the breast tissue of a patient.

- Clump Thickness: 1 - 10
- Uniformity of Cell Size: 1 - 10
- Uniformity of Cell Shape: 1 - 10
- Marginal Adhesion: 1 - 10
- Single Epithelial Cell Size: 1 - 10
- Bare Nuclei: 1 - 10
- Bland Chromatin: 1 - 10
- Normal Nucleoli: 1 - 10
- Mitoses: 1 - 10

These features are used to predict whether the tumor growth is either benign, 2, or malignant, 4 for 699 instances. The overarching motivation for determining the best classifier pertaining to this dataset is to aid in the delivery of accurate prognoses for patients.

Factors to Consider

When choosing a model, the two primary factors that are considered in this report are computational complexity and validation accuracy on the final validation set. These are highly dependent on the dataset itself. In a dataset with low dimensionality and few samples like this one, some models will have a better validation accuracy as opposed to another. While not necessarily a factor that will be heavily considered this report, model interpretability is also an factor to consider. Take for example a k-nearest neighbors classifier and a deep neural network. The computational complexity of a KNN classifier will be less and its model interpretability better compared to a deep neural network given a small dataset. This is primarily due to the fundamental characteristics of the respective classifiers with a deep neural network containing potentially many hidden layers which obscures understanding.

For this report the average and standard deviation for both accuracy and runtime will be collected for all the models across 1000 training and prediction trials. These factors will be the principal motivators for what model I deem to be the best for this problem.

Classification Models in Analysis

- K-nearest neighbors
- Decision Tree
- Random Forest
- SVM with polynomial kernel
- SVM with RBF kernel
- Deep neural network with sigmoid activation layer
- Deep neural network with ReLU activation layer

C. METHODS

Pre-processing Procedure

The original dataset has a '.data' ext., therefore additional processing was necessary in order to utilize it to train and then predict with classifiers. The methodology of pre-processing can be broken up into these points which will be expounded upon...

- Evaluate original dataset and read into a list

Dataset file was read in as a string with each line separated from one another, with each line as an individual object that can be iterated over.

- Convert list to dataframe

To query the data easier, the list of strings was a converted into a pandas dataframe. While these method can cause a program significant overhead because the package will iterate through the entire dataset to interpret/assume datatypes for a column, for a small dataset like this one it is not a significant issue.

- Impute missing values

There are a few missing values in the current dataset however they are not able to be recognized as a nan value because in the dataset they are denoted as '?'. Therefore, the next step was to iterate through the dataset and convert them into nan values that the dataframe could recognize, using the numpy external package. All these steps serve as the foundation for imputing the values, however they are not significantly necessary and could be skipped if not using an external package to impute values. After iterating through the dataset and converting missing values with '?' to a np.nan value, we use an external package (sklearn) to impute the missing values. In other words, we replace the missing values with some constant number. Replacement possibilities could include the mean of the respective column that the missing value is in, the median of the column, or mode. For this report and its given results, replacement values utilized are the mean of their respective columns.

- Split dataset into independent and dependent set

While the dataframe has imputed missing values, it still does not know what the target will be. Consequently, we divide the dataset into its independent and dependent sets. The dependent variable for this case is the Class. Therefore, we select all variables excluding the ID (an identification variable however serves no other purpose) and target variable and form it as our independent set. Then, the dependent set solely selects the Class.

- Split dataset into training and test set

With the independent and dependent set, we can then divide the dataset into a training set with 90% of the data and a test set with 10% of the data. In order to keep factors constant/preserve data across multiple classifiers we use an external package from the sklearn library that would enable the dataset to be split into a training and test set for a given random state (random state is set to 1 for this report).

External Libraries and Packages used

- sklearn
  - models
    - The project uses most of the available classifiers offered by the sklearn library in order to do the initial cross validation and the final training and testing with classifier models.
    - The library has a few modules of interest, (neighbors, tree, ensemble, svm, and neural_network) which contain the necessary classifiers in this report
  - metrics
    - Once a model has been trained, in order to compare predictions and final validation set we import another module of sklearn called metrics which enable us to compare two lists and return the accuracy.
  - cross-validation
    - To conduct cross-validation we import the module from sklearn: model_selection. This will allow us to split a dataset into an arbitrary number of folds consistently provided a specified random state. In addition, we use a function called GridSearchCV. This function enables one to test a suite of parameters and essentially tune them over a series of fits. The number of fits is dependent on how many parameters a user wants to tune (all models in this report use 150 fits = (5 possibilities for one parameter) * (3 possibilities for second parameter) * (10 folds))
  - imputation
    - While the imputation step could have been done without external resources, the sklearn impute model does the work of imputation when it can recognize what values are nan or null.
- pandas
  - this package is helpful for processing he raw .data file into something resembling a .csv in which specific can be queried easily and rows being grouped together more efficiently
- matplotlib
  - To visualize the cross validation results we use the pyplot module from matplotlib which supports graphing.
- numpy
  - The imputation function from sklearn will only impute values that it recognizes, one of those values being of the type np.nan. Therefore, we import this package to iterate through the dataset and convert all '?' to the type np.nan.
  - The package is also used to calculate the mean and standard deviation of a list which is useful when getting information for a model after running it over a series of n trials.
- time
  - In order to get a reading for the time it takes to train the model with the dataset and the time it takes to form predictions we use a built in package of python called time. Then we can set specific checkpoints and calculate the differences between those checkpoints to get elapsed time.

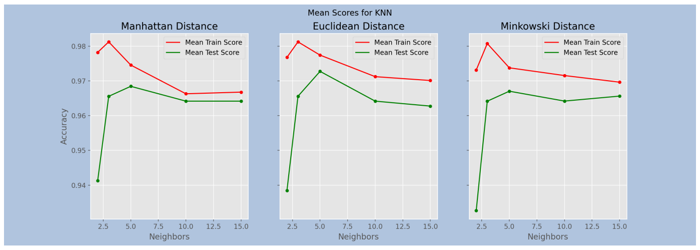
D. RESULTS

k-nearest neighbors

General Overview

KNN or k-nearest neighbors is a supervised machine learning classifier which stratifies points into classes based on how close each point is to one another. This distance can be calculated with various methods like the Manhattan Distance (L1 Norm) or the Euclidean Distance (L2 norm) between points. In addition, the class of points is determined by a majority voting principle dependent on the hyperparameter, k. For example, if k is set to 3, the classes of the 3 closest points of a point you want to classify will be checked. Whichever class appears more than the others across closest points will be the class of the new point.

Cross Validation Results



Details Needed to Replicate Results

Hyperparameters Tuned

- The number of neighbors or k: 5 possible values (2,3,5,10,15)
- The distance metric: 3 possible metrics (Manhattan, Euclidean, Minkowski)

Other comments

- Utilized GridSearchCV with 10 folds (150 fits) to test all parameter possibilities and find the best parameters (decided by the set of parameters which achieves the best accuracy on left out data)
- Hyperparameters not tuned were set at default per the sklearn library

Advantages and Disadvantages

Advantages

- Easy to understand simply because taking distance between points
- Requires no training before making predictions, reduces computational complexity and allows for the addition of new data to be efficient and does not impact accuracy
- Few parameters (the number of neighbors and what distance metric to use: Manhattan, Euclidean, Minkowski)

Disadvantages

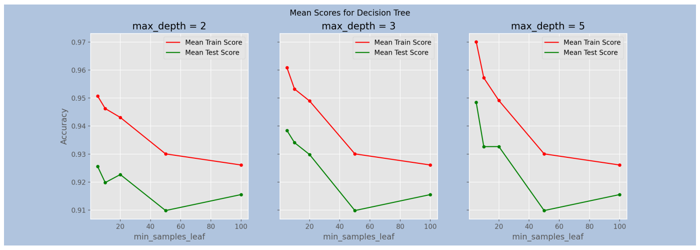
- Good for a small dataset however runtime increases significantly with a dataset with many instances or features
- Need to normalize or standardize data first before applying KNN because distances between values can be skewed

Decision tree

General Overview

As its name suggests, the decision tree classifier is one of the most interpretable models because from a high level outlook, the model looks like a flowchart. From the root of the tree we have the entire population of the dataset. In order to build the tree/traverse down, we divide split the dataset into various nodes from top down by initially calculating the entropy for each class and then the entropy after each split of an attribute or using the gini impurity to calculate splits instead.

Cross Validation Results



Details Needed to Replicate Results

Hyperparameters Tuned

- Maximum depth of tree: 3 possible values (2, 3, 5)
- Minimum samples per leaf: 5 possible values (5, 10, 20, 50, 100)

Other comments

- Utilized GridSearchCV with 10 folds (150 fits) to test all parameter possibilities and find the best parameters (decided by the set of parameters which achieves the best accuracy on left out data)
- Hyperparameters not tuned were set at default per the sklearn library

Advantages and Disadvantages

Advantages

- Intuitive/easy to understand and visualize
- Pre-processing is not very necessary, can handle missing values, outliers, categorical variables without issue and has no need for normalization or standardization
- Few parameters (the number of neighbors and what distance metric to use: Manhattan, Euclidean, Minkowski)

Disadvantages

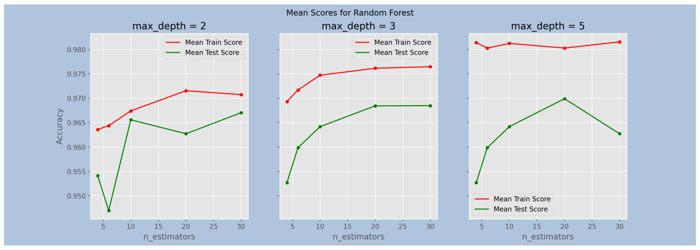
- Due to the nature of decision trees, the trees tend to overfit when given new data because the tree will become more complex as specificity increases
- For small datasets it works well, however similar to KNN, decision trees will become too complex if given a larger dataset which can lower accuracy for new points

Random Forest

General Overview

A random forest is essentially an extension of the decision tree. Rather than one decision tree that makes a prediction, a random forest utilizes constructs an arbitrary number of decision trees with bootstrap aggregating and feature randomness. For a classification problem, random forest takes the majority vote across all decision trees and utilizes that as its output.

Cross Validation Results



Details Needed to Replicate Results

Hyperparameters Tuned

- Maximum depth of tree: 3 possible values (2, 3, 5)
- Number of Trees in Forest: 5 possible values (4, 6, 10, 20, 30)

Other comments

- Utilized GridSearchCV with 10 folds (150 fits) to test all parameter possibilities and find the best parameters (decided by the set of parameters which achieves the best accuracy on left out data)
- Hyperparameters not tuned were set at default per the sklearn library

Advantages and Disadvantages

Advantages

- Lowers the tendency to overfit of a decision tree by using a majority voting rule across multiple decision trees
- As opposed to decision trees, random forests can handle new data that is fed into it well because only one tree of the group would be affected (less sensitive to noisy data)

Disadvantages

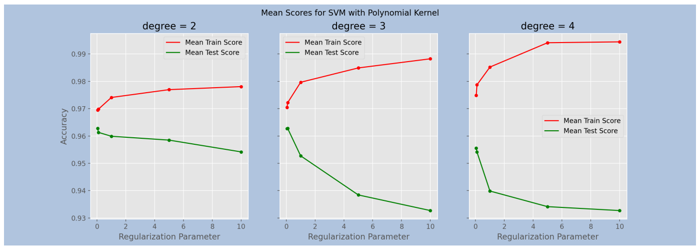
- Significantly more computationally expensive as opposed to Decision Tree simply because it is creating multiple decision trees
- Increased runtime because it is an ensemble learning algorithm

SVM using the polynomial kernel

General Overview

An SVM or Support Vector machine is a supervised ML algorithm that can be applied for both regression and classification problems. The main idea behind SVM or method behind it are hyperplane. Hyperplanes help to partition the dataset into classes which aid in classification. While visualizing the way an SVM constructs a hyperplane with linear data is not difficult, for non-linear data, projection onto a higher plane is necessary. Therefore we utilize the 'kernel trick' method to project data onto a higher dimension. The kernel trick calculates the dot product of two vectors. A polynomial kernel type will increase the degree in order to get more dimensions to calculate the support vector classifiers.

Cross Validation Results



Details Needed to Replicate Results

Hyperparameters Tuned

- Degree of the polynomial kernel: 3 possible values (2, 3, 4)
- L2 Norm: 5 possible values (0.05,0.1,1,5,10)

Other comments

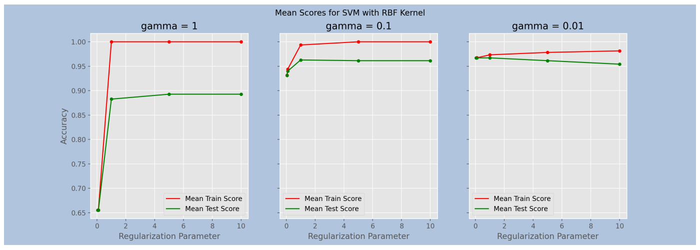
- Utilized GridSearchCV with 10 folds (150 fits) to test all parameter possibilities and find the best parameters (decided by the set of parameters which achieves the best accuracy on left out data)
- Hyperparameters not tuned were set at default per the sklearn library

SVM using the RBF kernel

General Overview

Instead of using a polynomial kernel type, a SVM utilizes a Radial Basis Function (RBF) to project data onto a higher dimension. Similarly to nearest neighbors, the RBF is dependent on the euclidean distance between points in order to move forward with classification.

Cross Validation Results



Details Needed to Replicate Results

Hyperparameters Tuned

- Gamma Coefficient for Sensitivity: 3 possible values (1, 0.1, 0.01)
- L2 Norm: 5 possible values (0.05, 0.1, 1,5, 10)

Other comments

- Utilized GridSearchCV with 10 folds (150 fits) to test all parameter possibilities and find the best parameters (decided by the set of parameters which achieves the best accuracy on left out data)
- Hyperparameters not tuned were set at default per the sklearn library

Advantages and Disadvantages of SVM (generalized)

Advantages

- Can handle non-linear data with kernel tricks like adjusting the degree (polynomial) or utilizing a radial basis function
- Is highly effective as opposed to other classifiers for datasets with high-dimensionality
- Can apply the L2 norm which helps to prevent overfitting on data

Disadvantages

- An important aspect of SVM are the support vectors, these can necessitate significant memory
- Will need to spend significant time on cross-validation to find the best kernel for a given problem
- Low model interpretability due to projection onto higher dimensions

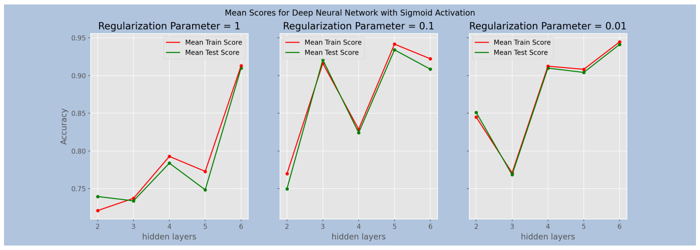
Deep neural network with sigmoid activation

General Overview

Before delving into what a 'deep' neural network is, a neural network is multilayered node chart which resembles a human brain from input to output. The three primary layers of a neural network or NN is its input layer which is connected by edge to a hidden layer and then finally connected by edge to an output layer. A deep neural network is simply a neural network with multiple hidden layers. As you traverse through the NN, input will be transformed into a weighted sum of itself. How that input is transformed into an output is dependent on the activation function. In this case with the sigmoid or logistic function, input is centralized between the range of (0,1) no matter the sign of that number. The result are probabilities which are utilized to train the DNN.
The sigmoid function is...

$$S(x) = \frac{1}{1+e^{-x}}$$

Cross Validation Results



Details Needed to Replicate Results

Hyperparameters Tuned

- Hidden Layer Sizes (Nodes, Hidden Layers): 5 possible values ((6,2), (6,3), (6,4), (6,5), (6,6))
- L2 Norm: 3 possible values (1, 0.1, 0.01)

Other comments

- Utilized GridSearchCV with 10 folds (150 fits) to test all parameter possibilities and find the best parameters (decided by the set of parameters which achieves the best accuracy on left out data)
- Hyperparameters not tuned were set at default per the sklearn library

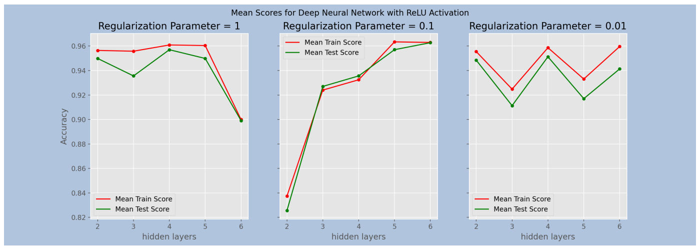
Deep neural network with ReLU activation

General Overview

This is similar to a DNN with a sigmoid activation layer, however instead of using a sigmoid function, the activation layer utilizes the Rectified Linear Unit Function. This function will output two possibilities, the original input if the input passed is positive or 0 if the input is ≤ 0.
The ReLU function can be represented by this formula...

$$R(x) = max\{0, x\}$$

Cross Validation Results



Details Needed to Replicate Results

Hyperparameters Tuned

- Hidden Layer Sizes (Nodes, Hidden Layers): 5 possible values ((6,2), (6,3), (6,4), (6,5), (6,6))
- L2 Norm: 3 possible values (1, 0.1, 0.01)

Other comments

- Utilized GridSearchCV with 10 folds (150 fits) to test all parameter possibilities and find the best parameters (decided by the set of parameters which achieves the best accuracy on left out data)
- Hyperparameters not tuned were set at default per the sklearn library

Advantages and Disadvantages of Deep Neural Networks (generalized)

Advantages

- Similar to SVM, DNN can work well for non-linear data
- Can conduct feature engineering automatically without any user pre-processing techniques needed
- Once trained, can perform efficiently on new data

Disadvantages

- Computational Complexity increases significantly the more hidden layers inserted between the input and output layer
- Performs poorly on smaller datasets compared to other more simpler models like decision trees or knn (models that also have a shorter runtime for smaller datasets)
- Model interpretability is low because as its name suggests hidden layers cannot be viewed

E. DISCUSSION

Data - Tuned Classifiers: Training Runtime across 1000 Trials		
Model	Avg Accuracy	std of Accuracy
KNN	0.0021	0.0004
Decision Tree	0.0020	0.0002
Random Forest	0.0255	0.0030
SVM (polynomial kernel)	0.0032	0.0006
SVM (rbf kernel)	0.0064	0.0007
Deep neural network (sigmoid)	0.3895	0.0422
Deep neural network (ReLU)	0.3691	0.0231

Data - Tuned Classifiers: Prediction Runtime across 1000 Trials		
Model	Avg Accuracy	std of Accuracy
KNN	0.0037	0.0016
Decision Tree	0.0011	0.0002
Random Forest	0.0034	0.0004
SVM (polynomial kernel)	0.0013	0.0002
SVM (rbf kernel)	0.0024	0.0003
Deep neural network (sigmoid)	0.0020	0.0011
Deep neural network (ReLU)	0.0019	0.0005

Data - Tuned Classifiers: Performance across 1000 Trials		
Model	Avg Accuracy	std of Accuracy
KNN	0.9571	1.1102
Decision Tree	0.9429	0.0
Random Forest	0.9658	0.0149
SVM (polynomial kernel)	0.9286	$2.2204 * 10^{-16}$
SVM (rbf kernel)	0.9571	$1.1102 * 10^{-16}$
Deep neural network (sigmoid)	0.8616	0.0550
Deep neural network (ReLU)	0.8675	0.1173

Performance Comparison

The model that achieved the highest average accuracy across 1000 trials for all selected models in this report was the Random Forest Classifier. It achieved an average accuracy of 0.9658 on the final validation set. Comparing across general classification model groups (disregarding KNN), performance-wise the models that performed the best were derived from a decision tree (decision tree and random forest). The next group of models that performed slightly worse were the Support Vector Machines. Finally, the group of models that performed significantly worse than the others were the deep neural networks.

Runtime Comparison

The decision tree had the lowest average runtimes for both training and prediction, with 0.0020 and 0.0011. Then, the KNN model which was slightly slower came in second with times of 0.0021 and 0.004 for training and testing. Then grouping the two together, the SVM models came in slightly slower than the KNN model. Then the random forest came in second-to-last which is understandable because it is an aggregation of decision trees. Finally the worst group run-time wise were the deep neural networks, with average training times more than 100x other models.

Conclusion

Model to Choose

Meshing the metrics from the discussion table, the model that I believe is best suited for this specific problem is the SVM with an rbf-type kernel. While it did not achieve the highest average accuracy across 1000 trials, it ranked 2nd and 3rd for accuracy. In addition, the standard deviation is really low, therefore we can expect the accuracy to be highly clustered around the mean. In addition, considering run-times the rbf kernel SVM has a slower runtime than both the decision tree and knn, however meshing both accuracy and standard deviation of the accuracy the SVM performs better than both. Finally, while the SVM with the rbf kernel had an average training and prediction run-time slightly slower than the polynomial-kernel, the difference in accuracy is why I believe the rbf kernel surpasses the polynomial kernel for this problem.

Further Considerations

If given a dataset with significantly more features and instances then I will definitely spend more time in the cross-validation phase. For this problem, each model was tuned with 15 potential parameter combinations in mind totalling 150 fits because of 10 folds. Instead of just 15 parameter combinations, I will experiment with simply more parameter combinations with a larger range between the specific values for a parameter. For example, using a larger range of decision trees that can be implemented in the random forest or trying more (smaller) L2 regularization values. While this process will be undoubtedly more computationally expensive than if less parameters were tuned, understanding what hyperparameters are more influential to a model as opposed to others (tuning certain hyperparameters will not influence the data significantly as tuning others). In addition, I would invest more time into the pre-processing phase. Essentially, I would apply unsupervised learning methods like reviewing a correlation table between features or PCA to reduce dimensionality (taking away features that are highly correlated with one another or those that are barely correlated with the target/class/dependent variable).

While this is not necessarily the most important point, it would be insightful to examine the impact cache size has on the runtime of the SVM models. From what I understand, the impact of a kernel's cache size is negligible for a small dataset with few feature variables, however given a dataset with high dimensionality and many instances, choosing the right cache size based on hardware or cloud computing power will heavily impact the runtime of the model.

It is important to note the caveats of this report. One of those being the cross-validation phase. The selection of what hyper-parameters to tune and the range of values for that specific parameter to tune (if numerical parameter like the L2 norm) can impact whether or not a given classifier performs or runs better than what was presented in this report. Another caveat is the hardware that I am utilizing which could impact the results that anyone may receive when trying to replicate the values that I got. While accuracy might stay consistent, runtime which could vary significantly.