Project #2

COMP 3710

Artificial Intelligence Concepts

Comparison of Classifiers in a Classification Problem

Authors:

George Kaceli,

Aleksa Vignjevic,

Jayk Dow,

Yasir Saeed

Submitted to:

Dr. Robin Gras

March 29th, 2023

Abstract

The main purpose of this report is to present the effectiveness of different classifiers when applied to the Waveform Database Generator dataset [1][2], and to discover which classifiers produce the best performing models given certain hyper-parameters. Using GridSearchCV we implement a novel hyperparameter tuning and optimization technique to quickly tune each classifier. The optimized models are then tested on a 20/20/40 train validation test split for comparison.

The statistics for each model's performance during each stage were then organized in a CSV file, where statistics such as f1, recall, precision, and accuracy score which we used to compare against one another. Our findings showed that the random forest classifier gave the most promising results despite taking the longest compile since this method had the best accuracy, precision, recall, and f1 score. The other classification methods had scores remarkably close to the random forest classifier and consistent with those presented in UCI Irvine dataset page [1].

Introduction

Classification algorithms are used in machine learning to categorize data into distinct classes. However, the parameters and classification techniques which will produce the best performing models are not immediately clear from any dataset. This report focuses on the performance of multiple classification algorithms when applied to the Waveform Database Generator dataset, which contains 5000 instances with 40 attributes representing three distinct wave types. The main objective is to determine which of our chosen classifiers creates the best performing models with certain hyperparameters.

We specifically chose a few classification algorithms to apply to the dataset, including random forests, decision trees, k-nearest neighbors, support vector machines and neural networks (specifically MLP). GridSearchCV was used to automate and fine-tune the hyperparameters. The performance of each model was evaluated based on metrics such as accuracy, precision, f1 score and recall.

This paper will go over relevant literature, experimental setup and procedures, analysis of results, and potential future works and improvements to the classification process.

Relevant Literature Review

To review all relevant literature, let us first consult the book the data comes from and their findings. The book titled *Classification and Regression Trees (1984)* [3], written by Breiman, Friedman, Olshen and Stone introduces this problem in chapter 2 of their book when discussing tree classifiers. The data as introduced above was used to test their various models with varying degrees of accuracy. We found that different approaches yielded better results that decision trees which falls in line with the authors findings.

Furthermore, pointing to the discussion in class, if we are going to classify the strategies incorporated, we can see that there is an obvious link between random forest classification and decision trees. Decision trees accept an input which is the description of an object or situation through a set of features and then they output the predicted value or in this case the class label.

Random forest classification can be looked at as a set of decision trees which divide the features and train them on an ensemble of decision trees and then aggregate the results to predict in this case the appropriate class label.

Moreover, in the slides KNN (K-Nearest-Neighbors) is described as a simple tool for classification which takes the values of the k (k being a positive integer) nearest neighbors and used that to make predictions. SVMS (Support Vector Machine) on the other hand are an efficient algorithm for classification and referencing this paper we were able to determine that a radial based function or RBF SVM was able to produce optimal results for classification [4]. Multi-Layer Perceptron or MLPs are a feedforward artificial neural network that can be used for classification tasks and MLPs can handle both linear and nonlinear relationships between the output and the input. In fact, under some assumptions, it can be shown that MLPs are in fact SVMs which are maximizing the margin in the hidden layer space and where all hidden units are also SVMs in the input space [5].

Experimental Setup for Classification

Experimental Procedure

Our Dataset, a brief description

As part of the project requirements, we decided to work with the 'Waveform Database Generator' dataset [1][2]. Created in 1988[3], the database has exactly 5000 entries and 40 attributes/features (41 if you include the classification attribute). Each row of the data represents a 40-dimensional 'waveform' which corresponds to one of 3 wave types. The final column of the dataset specifies the 'type' of the wave (0,1,2); thus, this data represents a multi-class classification problem instead of a binary one. Formally....

$a\,wave,\ w = \{x_i, 1 \leq i \leq 40\}$ (Is a set of points, 40 for each column respectively)$where\ (value(x_i),\ x_i)\ is\ a\ point\ in\ that\ wave$ (each row has 40 points describing a wave, + 1 wave type column at the end).

UCI provides two datasets for this problem; we chose version 2[2] that has 40 attributes and 1 target attribute (40+1). Version 2[2] is an extension of an earlier version 1[1] which only provides 20+1 attributes for the classification.

Our Pipeline

To compare how different classification methods perform on the dataset we decided to use Scikit-learn. The huge community behind the library and plethora of helpful features made it an obvious choice in terms of ease of use, capability, and availability of helpful guides. However, this posed the question...***how should we tune hyperparameters for the best result on the dataset?***

Manually comparing each combination of hyperparameters would have taken too long, instead we incorporated hyperparameter optimization into our pipeline. This was accomplished using GridSearchCV to automate testing **all** combinations out of a set of hyperparameters, then choosing the best performing combination.

As mentioned above there is no way in advance to know the best hyperparameters, instead we pass GridSearchCV a dictionary of hyperparameter options for a given classification technique as well as a metric to score the models with. We decided to rank models based on accuracy, recall, and precision choosing the highest performing accuracy models parameters. GridSearchCV works by running every combination of hyperparameters we wish to optimize through 5-fold cross validation on the training set. This in turn means GridSearchCV will train **n** combinations $\cdot$ 5 times each where...

$$n = \prod_1^{\#\ parameters} unique\ \ values\ of\ parameter$$

For example, on random forest, GridSearchCV will run 5-fold CV on $3 \cdot 2 \cdot 3 \cdot 3 = 54$ combinations. Thus training, $\mathbf{n} \cdot 5 = 270$ times for our random forest classifier! We iteratively repeat this for the presented parameter dictionaries for decision tree, KNN, SVC, and neural network (MLP) classifiers as well.

After finding optimized parameters for classifiers on this dataset we then need to compare the performance of each fitted model. However, that poses the question of ***how do we compare our different models efficiently***? This second part of the pipeline works off the optimized models generated in the first phase. We take these models and then iteratively run each of them on a 20/20/40 (train/validation/test) split on the data. Of course, different splits for test/train may produce more favorable results, however since we already ran 5-fold cross validation to select the models we only compared a few splits (The split influenced so little after cross-validation), with 20/20/40 providing the bests results. To process our results more easily, effort was put into creating logging and CSV writing functionality to make our results and development more efficient. For each model metrics about its hyperparameter-set, accuracy, precision, recall, f1 score, and execution times are all output to a combined CSV where each row represents a different model. Although this is not a software development course the addition of these quality-of-life improvements to the pipeline made analysis and debugging much easier. To establish a baseline, we also created a separate pipeline without optimization running the same models with some common hyperparameters set to establish a baseline for the hyperparameter tuning. We ran each pipeline separately and each produced their own CSV files prepended with ‘regular’ (for un-optimized) and ‘optimized’ respectively.

Chosen classifiers and their best performing hyperparameters

Decision Trees

Decision Trees partition their input into smaller subsections based on input features. Each internal node corresponds to a choice on which of the input features to split on, leading to a label with the answer (class label). The goal of this model is to create a simple set of decision rules to run predictions with. decision trees are a very beginner friendly classifier to understand so we decided to include it in our comparison. Our unoptimized model ran with 100 estimators and a random state of 42. After running hyperparameter tuning we found that a max depth of 15, max feature consideration to 3, a minimal sample to split a node to 5 and 300 estimators was optimal. Comparisons of these two models are in Figure 1.1 and Figure 1.2, respectively.

Random Forest

Random Forest incorporates multiple decision trees to produce better results. Each decision tree is trained on a randomized subset of the data and features. The final prediction is made by averaging the predictions of all the trees. We chose to use random forest as it is a common classifier and is built off decision trees. Since we included one, we decided to include the other to see how DT (Decision Tree) models would compare to RF (Random Forest) models. We ran the default parameters as a baseline, our optimized parameters were a max depth of 5, max feature consideration to 20, and a splitter policy of splitting on the best. Refer to Figure 1.1 and Figure 1.2 for statistics.

K-Nearest Neighbours (KNN)

KNN finds the nearest K points to a new point in the feature space and the distance between each data point is measured using a distance algorithm. The parameter we play around with are the K value and the weights of each data point. Which can either be uniform or distance which means the closer the data point the more it is weighed. We chose to use KNN as it is a common fundamental algorithm for classification problems. Our unoptimized model set the K value to 5 and the weights as uniform. After conducting our grid search, we found that the K value of 15 and setting the weights as uniform produces the most optimal results for our classification problem. Refer to Figure 1.1 and Figure 1.2 for statistics.

Support Vector Machine (SVM)

Support Vector Machines find a hyper plane that best separates the input space. SVM particularly seemed like a desirable choice as they can handle multiclass classification like our dataset. Our unoptimized model ran with kernel set to linear. After hyperparameter tuning we instead found that setting the kernel to 'Radial Basis Function' produces the best results. Refer to Figure 1.1 and Figure 1.2 for statistics.

Neural Network (MLP)

Neural Networks function on layers of nodes to learn complex patterns from a dataset adjusting the weights respectively to make later predictions. Since we have non-linear feature data, we chose to use a Multilayer Perceptron as they are useful in non-linear relationships between features and classes. For our unoptimized model we set the number of neurons in the i^{th} hidden layer to 100 and set the maximum iterations to 1000. After running hyperparameter tuning we discovered that setting activation function to 'identity', the batch size for stochastic optimizers to 4 and the learning rate to 'adaptive' produced the best results. Refer to Figure 1.1 and Figure 1.2 for statistics.

Overall Findings

To conclude, the various methods that were employed in our classification problem yielded remarkably interesting results in terms of what methods performed best on our dataset. For classifying waveforms into three distinct classes, we incorporated a variety of classifiers that each had their own positives and negatives. From this we were able to produce 5 machine learning models that were able to make predictions on our dataset, and to evaluate these 5

models we determined that, using metrics such as accuracy, precision, recall and F1 score would enable us to make fair and accurate judgments on performance.

During our testing phase, a vast amount of effort had to be put into determining the best hyper-parameters for our respective models. As mentioned in the experiment setup we determined it would best to automate the process using the GridSearchCV class for hyper-parameter optimization. From this we were able to determine the best hyper-parameters for each classifier.

From the metrics listed in the first paragraph, it was clear to see that when evaluating each of these models, the decision tree classifier model was unable to keep up with the other models as it pertained to all the metrics which were, accuracy, precision, recall and f1 score, falling well below the 80% mark. To note, in the book that the dataset is taken from, using CART decision tree algorithms the authors were able to produce a model that was 72% accurate, in our case we managed to be 75% accurate with our decision tree model.

Moreover, as it pertains to the rest of the models, we see a bit more parity being achieved amongst them. In terms of accuracy, precision and recall the random forest classifier model was able to edge out SVMs (Support Vector Machine) and MLPs (Multi-Layer Perceptron) by a razor thin margin most of the time. Meanwhile, our KNN (K- Nearest-Neighbor) model fared the worst out of the remaining 4, with accuracy scores around 82-83% on most runs, While the other models managed accuracy scores around 85-86%.

Our findings highlighted that on more complex problems, in this case waveform classification it was easy to see how more simple algorithms like decision tree underperformed, this was due to decision trees overfitting on the training data which is highlighted by the gap between the training and validation accuracy and thus producing underwhelming results. As it pertained to the other algorithms, they produced more consistent results, however random forest classifiers also displayed signs of overfitting which were highlighted by the large gaps between the training and validation accuracies.

To conclude, SVMs and MLPs were able to produce the most consistent results and did not show signs to overfitting on the dataset, our decision tree and KNN models were not able to keep up with the other models while others lagged behind, decision trees most notably. However, even with showing signs of overfitting, the random forest model edged out SVMs and MLPs by a razor-thin margin in test accuracy. This could be because our dataset contained lots of noise which hindered the performance of the SVMs and MLPs and allowed for our random forest model to outperform them, since its robustness and randomness allows each ensemble of a random decision tree to train on a subset of data.

Future Works

After doing multiple tests with classifiers on the Waveform Database Generator dataset, we noticed that we could improve the classification process to get better results. The first and most obvious way to improve the results and expand on the work already done is to try more classifiers supported inside of Scikit-Learn. In the experiments done, only a handful of classification methods were applied to the dataset, so trying other commonly used methods for classification could show that there are ways to improve accuracy and precision. Moreover, as it pertains to generating waveforms and then classifying them, more classification labels can

improve accuracy as more specific labels will be easier to categorize or it could become less accurate due to higher probability of choosing the wrong label. Furthermore, having more class labels would make this a more computationally expensive process, which may hinder inquiry down this path.

References

[1] UC Irvine machine learning repository. (n.d.). https://archive-beta.ics.uci.edu/dataset/108/waveform+database+generator+version+2

[2] UC Irvine machine learning repository. (n.d.). https://archive-beta.ics.uci.edu/dataset/107/waveform+database+generator+version+1

[3] Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (2017). Regression trees. Classification And Regression Trees, 216-265. https://doi.org/10.1201/9781315139470-8

[4] Ring, M., & Eskofier, B. M. (2016). An approximation of the gaussian RBF kernel for efficient classification with SVMs. *Pattern Recognition Letters*, *84*, 107-113. https://doi.org/10.1016/j.patrec.2016.08.013

[5] Collobert, R., & Bengio, S. (2004). Links between perceptrons, MLPs and SVMs. *Twenty-first international conference on Machine learning - ICML '04*. https://doi.org/10.1145/1015330.1015415

Appendix

Figure 1.1 (Below, Un-optimized or 'regular' model performance)

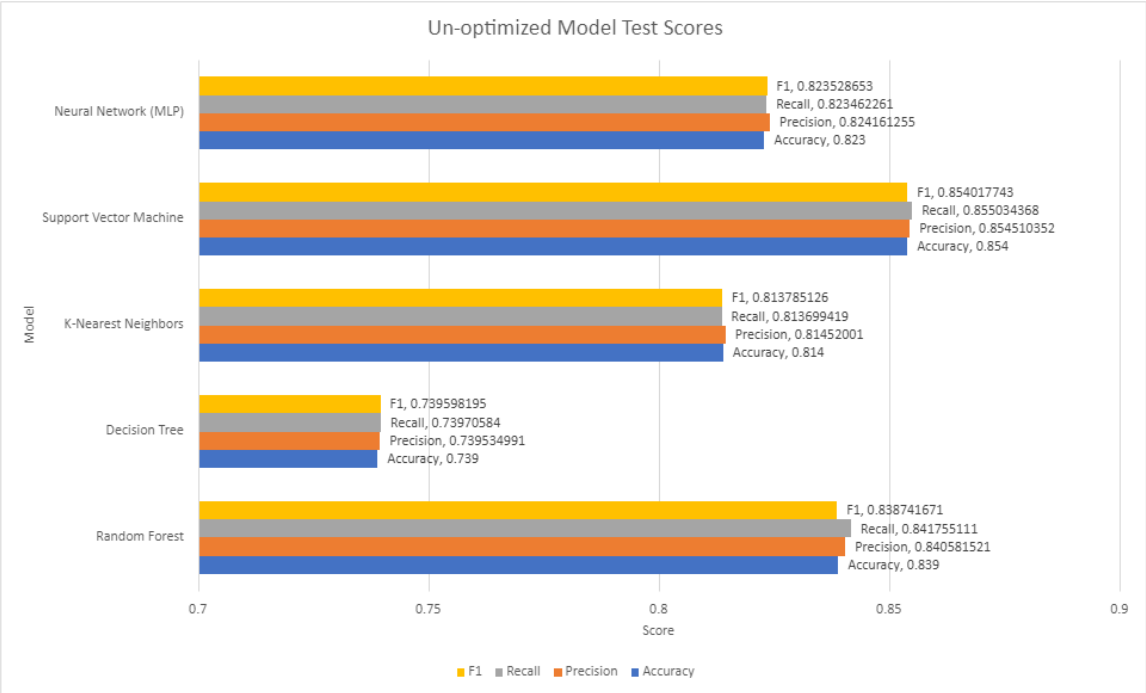
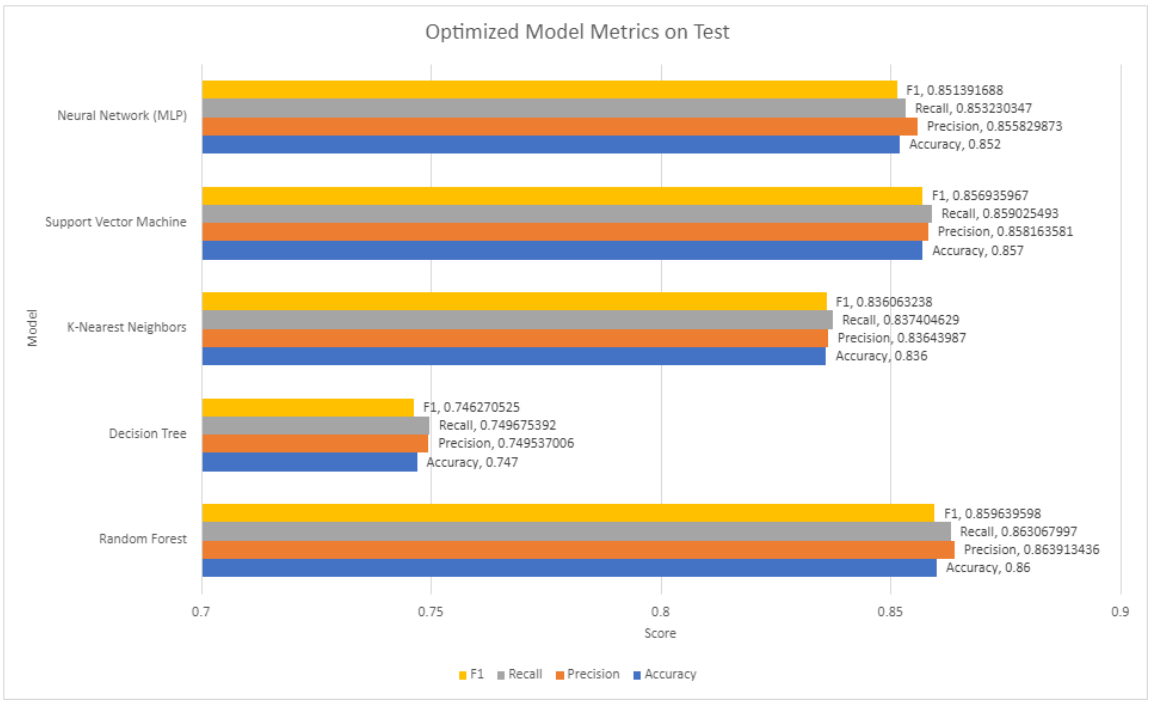


Figure 1.2 (Below, optimized models (decided from hyperparameter tuning) performance)


Optimized Model Metrics on Test
Neural Network (MLP)
F1, 0.851391688
Recall, 0.853230347
Precision, 0.855829873
Accuracy, 0.852
Support Vector Machine
F1, 0.856935967
Recall, 0.859025493
Precision, 0.858163581
Accuracy, 0.857
K-Nearest Neighbors
F1, 0.836063238
Recall, 0.837404629
Precision, 0.83643987
Accuracy, 0.836
Decision Tree
F1, 0.746270525
Recall, 0.749675392
Precision, 0.749537006
Accuracy, 0.747
Random Forest
F1, 0.859639598
Recall, 0.863067997
Precision, 0.863913436
Accuracy, 0.86
Model
0.7
0.75
0.8
0.85
0.9
Score
F1
Recall
Precision
Accuracy