CS 6375.003

Project Report

Sayana Sabu

Spring 2023

Introduction:

My project focused on an exploratory analysis on the Pima Indian Diabetes Dataset. This dataset is a binary classifier that tells if a test subject has diabetes (1) or does not have diabetes (0) based on several factors (feature variables). In what follows, I will explain how I used different Machine Learning models, tuned hyperparameters, and calculated numerous probabilities to produce the optimum values for various metrics.

Logistic Regression:

In a binary classification dataset, one of the first models we can use is Logistic Regression. We want to be able to split the dataset to training and testing and see how accurately we can predict the outcome based on the data provided. I used scikit-learn’s Logistic Regression and split the data accordingly: 80% for training, 20% for testing. One thing to be noted is that the features are not all on the same scale. For example, there is a feature variable named ‘Age’ which ranged from [21, 81] and another feature variable named ‘DiabetesPedigreeFunction’ which ranged from [0.078, 2.42]. We cannot omit any of these variables, as they are all considered medical predictors in determining if the test subject has diabetes or not. However, when applying it to the models, I normalized the feature vectors with preprocessing from scikit-learn. This way the model can use them all on the same scale when using the training data to fit the dataset.

After splitting the data and normalizing the features, I used the predict function to find y\_hat – the values of the model from X\_test based on the fit function used with (X\_train, y\_train). I then compared y\_hat with y\_test with the accuracy score function, which resulted in 82.5% accuracy. I thought this percentage was rather good. However, the metric that suffered the most was recall which had a score of 61.7%. A closer look at the data and generating a confusion matrix showed that there was a decent amount of data points misclassified as negative (False Negative values). 18 points to be precise, which was nearly 12% of all testing data points. Part of the reason for this may be an imbalance of data points in the overall dataset (the ratio of 0 to 1 is roughly 2:1). This led me to wonder how we can improve this recall score. I created a threshold array (values between [0, 1] incrementing every 0.05) to test the probability a point will be classified as a 0 or 1 when compared against a threshold value from the array. If the probability was greater than or equal to the threshold, it will be classified as 1, else as a 0. This helped with calculating the values necessary for precision, recall, and F1 such as true positives, true negatives, false positives, and false negatives. This further allowed me to plot an ROC-AUC curve and a PR curve as well. ROC-AUC curve showed a value of 0.679, and the PR curve showed that the best threshold was when t = 0.65 because that resulted in the highest F1 score.

K Nearest\_Neighbors:

The next model I used was the KNearest\_Neighbors (kNN) model to determine how accurately the model classifies a new data point. To keep up with the general rule of thumb of keeping K as an odd number, I ran the algorithm from scikit-learn on the following values:

|  |  |  |  |
| --- | --- | --- | --- |
|  | K = 3 | K = 5 | K = 7 |
| Accuracy | 0.740 | 0.799 | 0.773 |
| Precision | 0.578 | 0.682 | 0.636 |
| Recall | 0.553 | 0.638 | 0.596 |

As we can see, the accuracy goes up from K = 3 to K = 5 but goes back down at K = 7. This leads to us to probably the most fundamental question in the kNN algorithm: what is the optimum K value? To test this, I created an array for K values ranging [1, 25] so we could loop through them and perform a cross validation. I then appended the mean of these values to another array called accuracies and plotted the two against each other. This resulted in the following graph:

Chart, line chart

Description automatically generated

For this dataset, the highest accuracy for kNN occurred when K = 17. This indeed produced the highest accuracy of 81.2%. However, the recall score was 61.7% which was actually less than the recall score for when K = 5. So, although we found a competent method for finding the optimum K value, there still exists a bit of a trade off with the other metrics. This could be due to underfitting (more datapoints are taken into consideration to classify a new one).

Decision Trees:

The last model I tested on the diabetes dataset was Decision Trees. Given the number of classifiers, I wanted to see how the Decision Tree Classifier from scikit-learn would split the nodes. As we know, Decision Trees are a model that have several parameters that we can change. I would like to focus on the two I altered for comparison purposes: Criterion and Max Depth.

The first tree I made had all default parameters. To be more specific, criterion = ‘gini’ and max\_depth = None. The default splitting criterion was the Gini Index and Max Depth (which indicates how deep we want the tree to be) had no boundaries. Now, we know a decision tree can always overfit, and giving no numerical capacity on Max Depth, that is what I expected the algorithm to do. However, with these parameters, it performed rather well with an accuracy of 78%. To condense my findings from the Decision Tree Classifier, here are all the trees produced, along with the parameter settings and accuracy scores:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Decision Tree 1 | Decision Tree 2 | Decision Tree 3 | Decision Tree 4 |
| Criterion | Gini | Gini | Entropy | Entropy |
| Max Depth | None | 7 | None | 7 |
| Accuracy | 0.779 | 0.792 | 0.734 | 0.682 |

We can see from the table that the first two trees performed better despite changing the max\_depth parameter. In the latter two trees, accuracy was significantly lower. This led me to wonder how the data affected the splitting criterions. Based on the website *towards data science*, entropy is “a measure of disorder or impurity in a node.” The more fluctuating the node is, the higher the entropy. The same website says that the gini index “measures the probability for a random instance being misclassified when chosen randomly.” We want both of these numbers to ideally be low. We can establish from these definitions that the classifier performed weaker for when criterion = entropy, because it had a difficult time calculating the maximum Information Gain (the lower the IG, the higher the entropy). However, it performed better in predicting the randomness of a variable, hence why accuracies were higher when criterion = gini.

Conclusion:

In conclusion, all the models we tested on the diabetes dataset provided sufficient accuracies. Fine-tuning some of the parameters in these models showed how the classifier behaves differently when fitting the training data. Logistic Regression showed that a threshold of 0.5 doesn’t necessarily produce the highest values for metrics such as precision, recall, and F1. kNN showed that despite finding an optimum K value to get the best accuracy, if it is a rather large number, it could lead to underfitting which will bring down some of the scores for the other metrics. And lastly, Decision Trees showed that information gain is not necessarily the best splitting criterion for a node; sometimes it is best to split it based on the randomness for misclassification (gini index). This particular fact was probably my biggest takeaway from this project, as I was under the false assumption that maximum information gain always led to the most ideal tree. Overall, binary classification datasets may seem like simple models, but based on the variability of the features, number of datapoints, balance between the two classes, various machine learning algorithms will produce intriguing metric scores that will allow you to grasp the characteristics of the particular dataset and help you understand the behavior of the machine learning algorithms.

Citation:

<https://towardsdatascience.com/decision-trees-explained-entropy-information-gain-gini-index-ccp-pruning-4d78070db36c>