Project 1 Report

Neer Patel

***Abstract*—Provided data about many snake-like robots in a given environment, classifiers were used to categorize a given snake's position relative to the obstacles in the area. The classes were 0, 1, 2, and 3 – where 0 represented a collision. The classifiers used to make the predictions were KNearestNeighborsClassifier, AdaBoostClassifier, DecisionTreeClassifier, and RandomForestClassifier. Out of these, it was found that the AdaBoostClassifier with a RandomForestClassifier as its estimator had the highest accuracy of classifications.**

***Index Terms*—KNearestNeighbors, AdaBoost, DecisionTree, RandomForest**

I. Introduction

The purpose of this project was to classify the clearance distance of a snake-like robot given its links' points and clearance distances. The clearance distances for each row, which represented a snake-like robot through its links' points, was classified into four categories: 0, 1, 2, and 3. A clearance distance of 0 represented the snake being in collision with an obstacle, 1 meant that the clearance was $0 < c \leq 1$, 2 meant that the clearance was $1 < c \leq 2$, and 3 meant that the clearance was more than 2.

II. Results and Discussion

There are four classifiers that were used to categorize the data: KNearestNeighbors, DecisionTree, RandomForest, and AdaBoost Ensemble.

A. K-Nearest Neighbors Classifier

The KNearestNeighbors classifier categorizes data based on the distance between its surrounding points and their given categories. The main parameters that influence its accuracy are the number of neighbors to look at (k) and the distance measurement being used. I decided which k-value to use through trial and error; I plotted the data and determined which k-value had the highest accuracy. As can be seen from Figure 1, the highest accuracy occurs at $k = 16$. Then, I determined the distance measure to use based on the the given scenario. Since we are using links to represent the snake robot, and our data is given in (x, y) points – it makes the most sense to use Euclidean Distance as our distance measure. Note that setting $p = 2$ in Sci-Kit's implementation changes the distance measure to Euclidean. The performance, even when optimized, was only accurate for about 74.6%.

B. Decision Tree Classifider

Next, I experimented with the Decision Tree Classifier, but I found that it did not perform as well as the K-Nearest Neighbors classifier. There are many parameters that can be altered, and after experimenting, I found that the default parameters worked the best. I first tested the effect of the max depth parameter on the accuracy, as shown in Figure 2. The trials in Figure 2 hint that $180,000$ would be a good number for max depth, but that was misleading. I tried 125000, 150000, 170000, 180000, and 185000 and observed that there was no noticeable correlation to the accuracy. I further noticed hat having no max depth produced the highest accuracy.

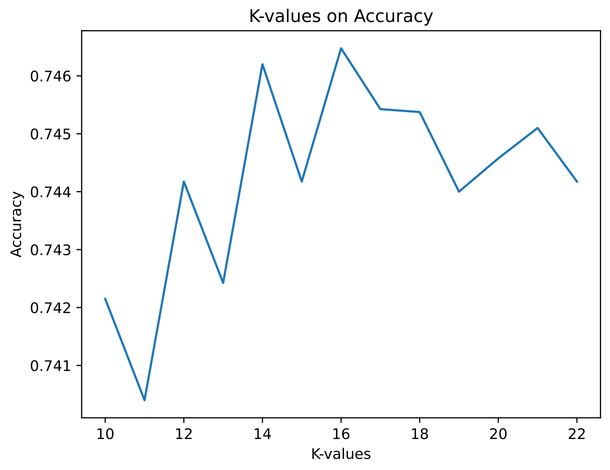

Fig. 1.

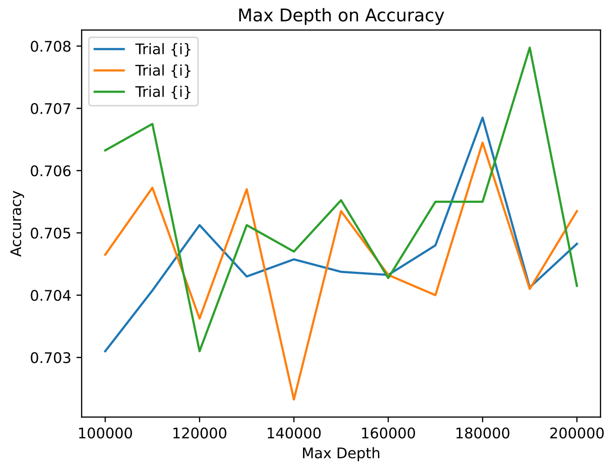

Fig. 2.

The next parameter and concept I considered was whether I would need to prune the tree or not. I chose to pre-prune the tree instead of post-pruning it because the runtime for post-pruning would be much higher. The parameter that enables pre-pruning is max leaf nodes. I experimented with a few options; however, the accuracy dropped considerably when I limited the number of leaf nodes, so I decided that there should not be a limit – which is the default parameter. In fact, I decided not to prune the tree at all. Overfitting on this data was not a concern, since the accuracy without pruning nor a max depth is about 70.4%.

Additionally, I looked at the split criterion and how that altered the accuracy. I experimented between the gini index and the entropy criterion, and I found that the gini index performed better. The gini index is the default split criterion that is used as well. Therefore, I decided to keep the parameters as they are for the DecisionTreeClassifier.

C. Random Forest Classifier

The Random Forest Classifier worked better than both of the previous classifiers, especially after tuning some parameters. A random forest classifier works very similarly to the DecisionTreeClassifier, except that it is a collection of many DecisionTrees on randomly sub-sampled data from the original dataset. After reading the default parameters, I discovered that bootstrap was the default, and I decided to keep it that way for performance purposes. I also noticed that a lot of the parameters overlapped with the DecisionTreeClassifier, so I used what I found to be true there for the RandomForestClassifier as well. However, there was a new parameter that was used to determine the number of estimators. I experimented extensively with this, and I found that it directly influenced the accuracy. I tried 40, 50, 75, 100, 125, 150, and 175. I found that the higher the estimators, the better the accuracy – but the runtime was greatly increased. Also, the difference in accuracy was negligible after 125, so I chose that as my value.

After more trial and error with other parameters, I discovered that applying max depth improved the accuracy for the RandomForestClassifier, even though it did not in the DecisionTreeClassifier. I tried 150000, 170000, 180000, and 185000. I chose 185000 because of the slight improvement it showed – this value is also supported from the trials on the DecisionTreeClassifier observed in figure 2. After tuning these parameters, the RandomForestClassifier obtained an accuracy of about 79.6%.

D. AdaBoost Ensemble Classifier

The classifier of my choice was the AdaBoost Ensemble Classifier. Prior to choosing this, I tried many others: VotingClassifier, HistGradientBoostingClassifier, BaggingClassifier, StackingClassifier, and MLPClassifier. The AdaBoostClassifier worked the best out of these. I tried it with and without an estimator, but it performed terribly without one. I then chose to use a RandomForestClassifier as my estimator, and I used more trial and error to determine what the best parameters would be for it within the AdaBoost Ensemble. I discovered that 150 estimators and a max depth of 170000 worked the best for a Random Forest Estimator. I then started messing around with the AdaBoostClassifier parameters – specifically the number of estimators and the learning rate. I tried many different combinations, and discovered that they were both inversely related; to elaborate, if one has a high number of estimators, then they would want a lower learning rate – and vice versa. The classifier had a long training time, but I was able to try estimators ranging from 40 to 200 and with learning rates ranging from 0.5 to 10. I noticed that the accuracy was higher when the learning rate was above 1, and that a high number of estimators did not always guarantee a high accuracy. I eventually reached a nice middle ground of 55 estimators and a 1.8 learning rate and obtained an accuracy of about 79.7%.

III. Conclusion

Overall, the AdaBoost Ensemble Classifier paired with a RandomForestClassifier as the estimator performed the best on my test set. I then ran the file on Miner2 to observe the accuracy with the true classification, and obtained an 80% accuracy (my Miner2 username is npatel49). I was very surprised at how difficult reaching 80% was. After testing many different classifiers and their parameters, the accuracies did not increase by much. Additionally, we were given an extremely clean data set – I assume noise and missing data would drastically decrease the accuracy of these classifiers. Although I was able to achieve an 80% accuracy, the runtime of my best classifier, AdaBoost, is quite long – this is an area of possible improvement.