Cogs 189 Project

David Aminifard
A15451805

Introduction

Given publicly available, EEG, eye-state data, this report will experiment with and evaluate multiple binary classification algorithms to predict whether someone's eyes are open or closed. EEG eye state data are records of someone's brain activity when their eyes are open and closed. Not only is such data useful for becoming familiar with classification algorithms, but it can provide insight into the predictability of brain activity based on eye state and, naturally, reveal the effect eye state has on brain activity. In addition, there may be endless applications for algorithms that make use of brain signals and eye state, such as medical systems and advanced car systems for detecting fatigue.

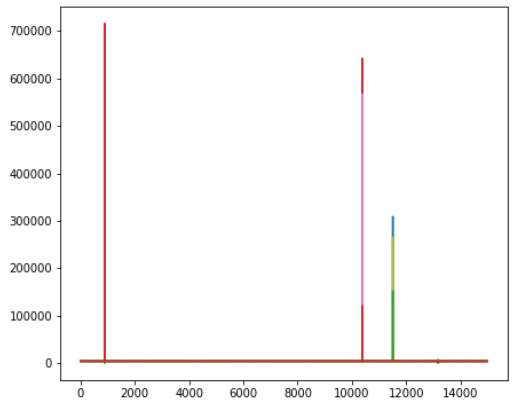
Related Work

Interestingly, a study titled, "Ensemble Classifier for Eye State Classification using EEG Signals", which was written by, "Ali Al-Taei" utilizes the same dataset I plan to use. In his paper, he combined two algorithms, the K Star algorithm and Random Forest Classifier in order to predict whether someone's eyes are opened or closed. In addition to combining two classification algorithms, Al-Taei tests the performance of a Support Vector Machine algorithm (SVM), Hidden Markov Map classifier (HMM), and a Radial Basis Function classifier (RBF). Al-Taei reveals that the Support Vector Machine, Hidden Markov Map, and Radial Basis Function classification algorithms yield a much lower accuracy than his proposed method of combining the Random Forest and K Star classifiers. At the very end of his paper, Ali Al-Tae not only mentions the importance of using a variety of technologies and datasets, but the optimization of algorithms in terms of performance and "processing time" [1]. In this report, I will look into improving classifiers not only in terms of accuracy but also in terms of time as well, because the practicality of a classification system is likely largely impacted by its ability to produce results in a timely manner.

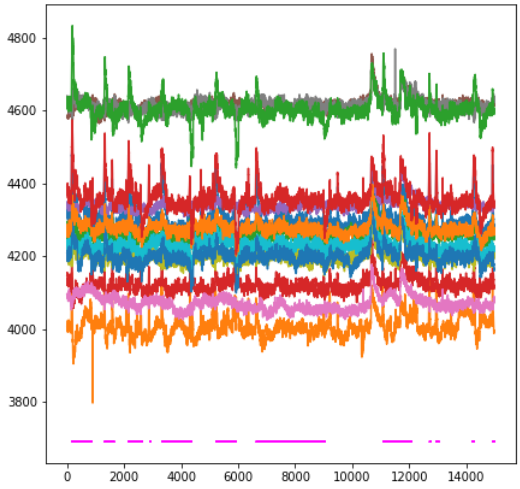
Methods

The dataset comes from an Emotiv EEG Neuroheadset and represents one continuous measurement that lasted one hundred and seventeen seconds. For cleaning the data, I did preprocessing in order to remove outliers. Doing so required that I iterate through every column with EEG data, and apply a function that changes an entry to the average of its column if it exceeds the average by 300. I chose 300 as the threshold because it seemed to remove outliers and yet take EEG signal spikes into account as well, which may be important for making predictions later on. In addition, in order to gain some intuition of the dataset, I located where in the visualization (Fig. 2) the individual's eyes are opened and closed. This way, it is clear that there is a vague connection between spikes and dips in the

data and the person's eyes opening and closing.



(*Figure 1.* The data before preprocessing/cleaning. Huge outliers present)



(*Figure 2.* The data after preprocessing/cleaning. Huge outliers are removed and signal spikes are preserved. The magenta lines at the bottom indicate when the individual's eyes are opened.)

After the data was preprocessed/cleaned, I created a system for evaluating multiple classification algorithms using the data. From the original dataset, I shuffled the data and I created train and test datasets where 33% was allocated to the test set. From there I trained and tested four classification algorithms, i.e. Linear Discriminant Analysis (LDA), Ridge Regression, Random Forest Regression, and K Nearest Neighbor. While doing so, for the algorithms, I changed parameters to optimize them in terms of performance and processing time.

As an overview of the methods used for getting the following results, I'll summarize them in the below paragraph.

For LDA, I changed the "solver" and "shrinkage" parameters and used all possible combinations between each to see what worked best with the given data. In addition, for the Ridge Regression Classifier, I optimized the regularization strength, the "alpha" variable, via iterating through many powers of 10 and setting alpha to the corresponding value. Similarly, for the Random Forest Classifier, I iterated through many values, between 50 and 2000 where step equals 50, to optimize the number of decision trees to use in the forest. Not only does this improve the accuracy, but it also improves the processing time. Lastly, for the K nearest neighbor classifier, I iterated through some values that specify how many neighbors are analyzed, 1 through 30 where step equals 1, to optimize the algorithm and possibly improve the processing time.

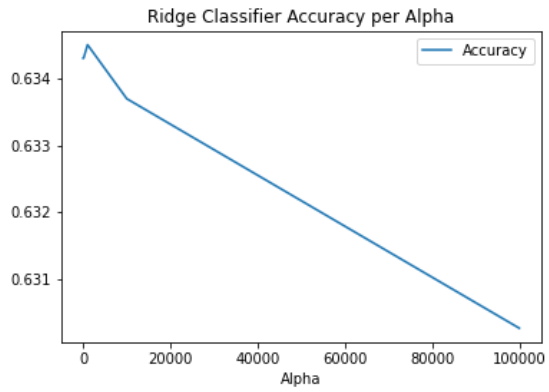
For each classification algorithm, I recorded corresponding accuracy metrics. In addition, for the Random Forest and K nearest neighbor classifiers, I recorded the processing time since it seemed appropriate for optimization purposes given the parameters I experimented with.

The following section will detail the results of the various methods used, since for each method there is a corresponding result.

Results

The first algorithm I experimented with was Linear Discriminant Analysis, where the accuracy was initially 63.39%. This was a surprisingly low result, and it seemed to indicate that the data may not be linearly separable. Interestingly, the accuracy of the algorithm remained exactly the same despite whether I changed the "solver" to "svd", "lsqr" or "eigen". However, setting the "shrinkage" variable to "auto" improved accuracy slightly by raising it to 63.51%. For the purposes of pure speculation, I suspect improving the estimation of covariance matrices, which is what the shrinkage variable does when set to auto, had this much impact on accuracy because the training data is made up of 10,036 entries that each have 14 dimensions, which may possibly constitute a dataset that is not suitable for calculating a high quality covariance matrix.

The second algorithm I experimented with was Ridge Regression, which is a multiple-regression model where an "alpha" variable can be changed in order to manipulate the regularization strength of the algorithm. The ideal alpha tends to be different from task to task, so it is ideal to optimize the Ridge Regression algorithm with this in mind. In this case, I found that the best alpha was 1000, which resulted in an accuracy of 63.45%, which was slightly worse than the Linear Discriminant Algorithm.



(*Figure 3.* The accuracy of the Ridge regression algorithm decreases after peaking at 63.45% when alpha equals 1000.)

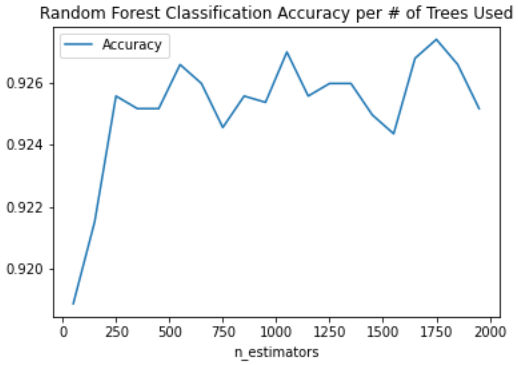
	Alpha	Accuracy
0	1000.00000	0.634506
1	0.00001	0.634304
2	0.00010	0.634304

(*Table 1.* The top three accuracy scores are ranked descending with corresponding alpha values.)

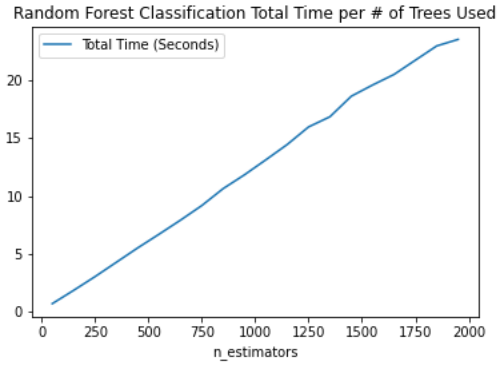
The accuracy of the algorithm peaks early on as the alpha is incremented. Once it hits 1000, the accuracy starts to decrease linearly as the alpha increases exponentially. Thankfully, incrementing alpha does not seem to increase processing time for the

algorithm. So in this case, time was not a concern.

The third algorithm I tested was a Random Forest Classifier where I edited a variable called “n_estimators”, which controls the number of decision trees used to make a classification. While increasing the number of trees tends to improve accuracy, it also increases the processing time of the algorithm. As a result, I decided to find the optimal point where processing time does not take too long, i.e. under 1 minute, and is reasonably accurate.
Fortunately, I found that the best number of decision trees to use is 1,150 and the corresponding processing time was roughly 25 seconds.



(*Figure 4.* The accuracy of the Random Forest Classifier peaking at about 1,150 trees.)

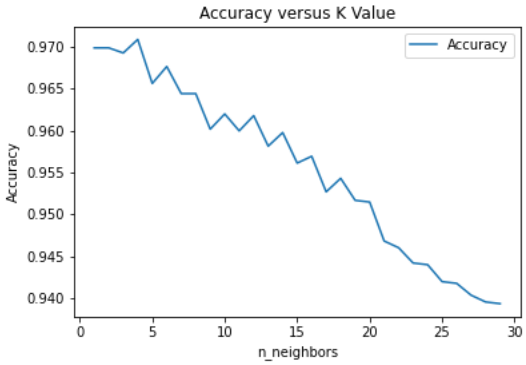


(*Figure 5.* The processing time per tree used. The time increases linearly as more trees are utilized.)

Surprisingly, at a certain point, the accuracy of the Random Forest classification algorithm seems to maxout and stop improving. It’s important to take this into account when efficiency is paramount, because we want the algorithm that is the most efficient and accurate.
Lastly, I experimented with the K nearest neighbors classification algorithm. I optimized the algorithm through incrementing neighbor counts, by editing the variable ‘n_neighbors’, similarly to how I optimized the Random Forest classifier through incrementing the number of trees being used. Before experimenting with the K nearest neighbors algorithm, I suspected that the algorithm would become worse as “n_neighbors” increased because the scope of neighboring labels analyzed would increasingly utilize unrelated data points that should not be associated with the current entry. In addition, I suspected that the algorithm will perform much better when an odd number of neighbors are used for classification since the classification is made by finding the labels of the n nearest neighbors and labeling the current entry based off of the most represented label among the neighbors. It seems that analyzing an even number of neighbors would make finding a majority harder than an odd number of neighbors because an even number of neighbors could result in a tie in label representation more often. A tie would result in necessitating a random guess

of the current entry's label, which would decrease the accuracy of the classifier.

I was very surprised to see that some of my assumptions were incorrect because the algorithm's performance was best when 4 neighbors were analyzed, which is an even number and contradicts my initial assumptions that an odd number would be best. However, my assumption that accuracy would decrease as the k value is incremented was mostly correct once k was incremented past 4.



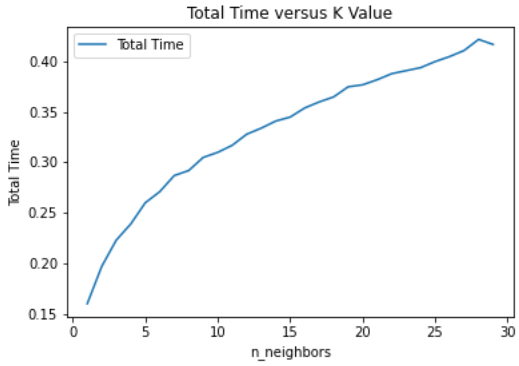
(*Figure 6.* Accuracy of the K Nearest Neighbors classifier peaks early on as the K value is incremented.)

	n_neighbors	Accuracy
0	4	0.970874
1	1	0.969862
2	2	0.969862
3	3	0.969256
4	6	0.967638

(*Table 2.* The K Nearest Neighbors classifier is most accurate when an odd number of neighbors are utilized per entry that is classified.)

It's unclear how the accuracy of the classifier is so high compared to the others used. As mentioned earlier, the data is shuffled and ⅓ is reserved for the test dataset while ⅔ is used for training, so the high accuracy does not seem to be a result of overfitting.

Interestingly, the processing time of the algorithm does increase as the k value increases. However, unlike the linear increase we saw with the processing time for the Random Forest classifier, increasing the k value increases the processing time logarithmically. The seemingly logarithmic increase can be seen in the visualization below.



(*Figure 7.* Processing time of the K Nearest Neighbors classifier seems to increase in a logarithmic manner.)

As a result, it seems that not only is the K nearest neighbor algorithm more accurate than the Random Forest classifier, but it is also more time efficient as well.

Discussion

This project revealed that small nuances among algorithms can make huge differences in terms of applicability. I suspect that the data was not linearly

separable, which was why the Ridge Regression Classifier and Linear Discriminant Analysis yielded relatively low accuracy compared to the Random Forest and K nearest neighbors classifiers. Since K nearest neighbors and the Random Forest classifiers are non-linear classifiers, it makes sense that they performed better than the linear classifiers I tested.

I was very surprised to see that the K nearest neighbors classifier performed better when an even K value was used. Further investigation into this may prove to be very interesting, because it could be related to the quirks of the data itself, or may reveal faults in my own methods that were undetected.

If I had more time, I'd investigate the linear separability of the dataset to uncover whether that is the underlying issue affecting the linear classifiers. In addition, I would do more analysis of the K nearest neighbors algorithm to better understand why it was extremely accurate and time efficient when compared to the Random Forest classifier. Lastly, I would utilize more performance metrics, such as Balanced Error Rate, so that I can account for potential false positives and false negatives that would otherwise be undetected by my current methods.

Related Content

The Github repository used for this project can be found here.

References

[1] Al-Taei, Ali. (2017). Ensemble Classifier for Eye State Classification using EEG Signals.