Physiological Data Modeling

Benjamin Ferrell and Mathew Gray

University of Texas at Dallas
bjf091000@utdallas.edu mxg127530@utdallas.edu

I. Introduction

Physiological Data Modeling, the topic chosen for this project, contains several obstacles that must be overcome before accurate predictions can be made. These obstacles are derived from the acquisition of data through sensors. Sensors are inherently noisy, can be combined to provide a more accurate representation than each sensor taken individually, and represent continuous values rather than discrete. The goal of our project was to obtain the highest accuracy possible when predicting annotations for each instance of the data. Given for each data instance was the age, handedness, gender, nine sensors each pertaining to a different reading, and the session time that had elapsed since the beginning. In this paper, annotations refer to the classification given to one instance of the data. Classifications for this data set are multivariate.

Given the previous information, we determined that the most important attributes for accurately predicting an annotation would be the sensor data. Because the data at hand is continuous we developed five unique classifiers. These classifiers are Gaussian Naive Bayes, Decision Trees using K-splits, K-Nearest Neighbors (kNN), Multi-class Perceptron, and a Multilayer Perceptron. Each of these classifiers will be discussed in detail in the implementation section of the report.

II. Referenced Work

In the development of several of our classifiers scholarly articles, powerpoints, and wikis were referenced. Following are each of the documents referenced and how they influenced our project.

First is the powerpoint developed by Tom Mitchell which describes in detail the process for computing the mean, variance, and predictions for Gaussian Naive Bayes. We followed the slides precisely and did not assume any independence on either the data or annotations.

Microsoft Research hosts a paper titled "Efficient Determination of Dynamic Split Points in a Decision Tree." This paper influenced our decision tree classifier by informing us on how to make splits based on the data. Most notably from the paper, we used the algorithm for Uniform Approximation when determining where to make our splits.

The Multi-class Perceptron is an extension of the binary classification perceptron and the algorithm is given in the slides "Linear Classifiers: Perceptron" by Roberto Paredes, Alfons Juan, Enrique Vidal.

The Multilayer Perceptron was modeled after the paper from Leonardo Noriega entitled "Multilayer Perceptron Tutorial."

III. Implementation

In this section we'll describe each of the classifiers developed for this data set and their adaptations.

I. Gaussian Naive Bayes

For the implementation of Gaussian Naive Bayes we assumed that the most important attributes were the sensor data. Each of these are continuous variables rather than discrete and thus must all be given gaussians. For each gaussian there is a mean and variance which is computed for each annotation in the data set. The mean is computed by averaging over all the data for each annotation individually. Variance is computed by squaring the difference

between each data point and the mean. A prior is computed by dividing each type of annotation by the total number of annotations in the data. Using these three values it is possible to predict an annotation given the sensor data.

Table 1: *Time Complexities and Accuracy*

Classifier	Training Time (seconds)	Testing time (seconds)	% Accuracy
Gaussian Naive Bayes	18 s	421 s	71.95 %
Decision Trees	20 s	9 s	71.90 %
kNN	N/A	18910 s	100 %
Perceptron	22 s	6 s	70.4 %
Neural Network	15 s	1 s	73.5 %

II. Decision Trees

Decision trees were implemented using the k-splits method for continuous variables. Our algorithm for determining where to make the cuts is Uniform Approximation referenced from the Microsoft Research paper. This involves giving a uniform range to each cut based on the minimum and maximum values of the sensor. The formula given in the paper for determining the splits is as follows where $min(j)$ and $max(j)$ are the minimum and maximum for sensor j.

$$c_i = min(j) + i * \frac{max(j) - min(j)}{k+1} \quad (1)$$

Gain was chosen as the heuristic for determining which attribute to split on next. For the accuracy measure computed to gain we chose Entropy. Entropy was computed using this formula:

$$\sum_{i=1}^{k} log_2 \frac{X_i}{|X|} * \frac{X_i}{|X|} \quad (2)$$

Where Xi is the number of annotations that fall under the th split and |X| is the total number of annotations in all three splits.

III. kNN

Implementing kNN was straightforward and no extra features were added. Euclidean distance is used to determine the distance one test example is from each of the training examples.

$$\sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \quad (3)$$

No pruning is used after reading in the training data therefore each test example compares against over five hundred thousand training examples. This complexity results in longer testing times but overall higher accuracy which will be discussed in the results section of this report.

IV. Multi-class Perceptron

The perceptron algorithm learned in class was for binary classification. Our data has over 50 classifications so a multi-classification perceptron had to be implemented. It is essentially the same algorithm with a few tweaks. The main change is instead of having one weight vector, there are C weight vectors with each corresponding to a specific class. The positive dot product indicates the appropriate class while the non-positive dot product indicates it is not appropriate. For training, weights are iterated over for each data vector per class and error correction is implemented when a misclassification is encountered. Then come classification time, all but one of the dot products should be negative and this will indicate the appropriate class.

V. Neural Network

A simple (single hidden layer) neural network was implemented to attempt to classify the test data better. The program takes in the max iterations, number of hidden nodes, and activation function as parameters. During training, a sample vector's output is computed and the error is back propagated on misclassification. Two activation functions were implemented:

$$y = \frac{1}{1 + e^{(}- x)} \tag{4}$$

and

$$y = tanh(x) \tag{5}$$

with derivates

$$y' = x * (1 - x) \tag{6}$$

and

$$y' = 1 - tanh^2(x) \tag{7}$$

respectively.

IV. Results

Time complexities and accuracy are reported in *Table*1. Each value was obtained by computing the average value over ten runs. In most cases this was not necessary as training and testing didn't depend on initialization of values; however, the amount of work being done on the CPU adversely affected the run time of kNN. The training set for physiological data modeling used contains 580,276 instances and the test set contains 720,813 instances.

One issue with this data set is the annotation set; when no activity is being performed the annotation is set as 0. A large portion of the data is labeled with this annotation (71%) which causes the data to be skewed in its favor. One approach to dealing with this is simply ignoring the data instance when it is 0 and only considering annotations with a non-zero annotation. This would likely drastically improve both Gaussian Naive Bayes and Decision Trees; however, this assumption was not outlined in the data set description and therefore we decided against this approach. Our approach involves including the annotation 0 as a classification. Although this skews classifiers such as Gaussian Naive Bayes and decision trees, which will be discussed later, it gives us a good baseline for improvement. We know that always selecting the annotation 0 will result in a classification accuracy of 71% and therefore we want to improve upon that.

I. Hardware

This section is included as a reference for the time complexities given by *Table*1. Improving the hardware on which each classifier is run should drastically improve training and testing times.

- CPU: AMD Phenom II X6 1055T Black Edition @ 2.80 GHz
- RAM: 8 GB Corsair DDR3 @ 1333 MHz
- HDD: WD5000AAKS 500 GB @ 100 MB/s

II. Gaussian Naive Bayes

Our implementation of Gaussian Naive Bayes was 71.95 % accurate. This is due to the overwhelming prior given to the annotation 0 as it makes up 71 % of the data. One possible improvement to this implementation would be to ignore the 0 annotation in the data completely for both testing and training examples. This would likely improve the accuracy for two reasons. The first is that we no longer have the large amount of noise generated by the 0 annotations which inflates its Gaussian. Secondly, the large prior that dwarfs the other priors would then be gone.

III. Decision Trees

Decision trees using k-splits achieved a similar accuracy to Gaussian Naive Bayes. Again this is due to much of the data set containing the 0 annotation. When determining which annotation a leaf node should contain we analyze each data point that matches the splits we've made. Since the majority of the data is annotated as 0 it is often selected as the annotation for multiple leaves even under the same set of splits.

IV. kNN

The accuracy of kNN is excellent due to the large amount of training data available. This amount of data, however, causes the classification time for each example to be longer than the other classifiers we've developed for this problem. One possible way to improve classification time is to use the GrowSet method on the training data. Starting from the empty set, data points would only be added to the training set if they cannot be correctly classified. This would substantially reduce the total amount of points compared against during testing which would vastly improve classification time.

V. Multi-class Perceptron

The perceptron algorithm is designed to linearly classify data. If the data is not linearly classifiable then convergence will never be reached during training. The data set we trained on is not linearly classifiable unfortunately and did not achieve great results. As usual, the annotation 0 was classified more so than any other annotation.

VI. Neural Network

The purpose of adding hidden units into a network of perceptrons is to create linear separation in the data when it is not linearly separable. The algorithm was expected to produce better results than a single perceptron, but it did not. Since the 0 annotation is so prominent in the data set, we were curious to see if better results could be obtained by ignoring them and focusing on the others. By doing this, the neural network achieved worse results indicating that this subset of data also has a lot of noise.

V. Future Improvements

Improvements could be made to each classifier to both improve time complexity and accuracy. For Gaussian Naive Bayes, Decision Trees, and Multi-class Perceptron that would be the exclusion of the training data that includes the 0 annotation. Any test example that is classified as such would be ignored for both classification and total number of examples. This would improve Gaussian Naive Bayes by removing the overwhelming prior on the 0 annotation as well as improve training and classification time by removing nine sets of means and variances. Decision Trees would also likely improve due to the fact that each other annotation would no longer have to compete with the large amount of 0 annotations when determining the leaf nodes annotation. Finally, the Multi-class Perceptron might converge when not including the 0 annotation.

Another improvement would be to the kNN algorithm. By using the GrowSet method on the training data the time complexity for classification could be vastly reduced at a minimal cost to accuracy. This trade off would likely cause an increase in training time but compared to the vast savings in classification time it would likely be worth it.

An improvement to the overall scheme of Physiological Data Modelling would be to include a first order Markov Assumption. This assumption would insist that the current annotation is dependent on the previous annotation. We came to this conclusion by viewing the training and testing data given in the data set. In most cases, many annotations in a row would be equivalent and did not rapidly change from one annotation to another. By including some bias given the output of the previous example, the accuracy of the current example could be improved.

VI. Conclusion

In this project we looked at Physiological Data Modelling and some machine learning techniques that might perform well on it. We explored five algorithms and identified what does and doesn't work in each. These algorithms are, Gaussian Naive Bayes, Decision Trees, k-Nearest Neighbors, Multi-class Perceptron, and the Multi-layered Perceptron. Several obstacles in Physiological Data Modelling made it difficult for the linear classifiers to perform well

such as noise in the data, sensor fusion, and multi-classification. After analyzing our results we determined several improvements to each of the classifiers and one improvement overall that could drastically improve the performance of each classifier. In conclusion, we developed five classifiers to train on the data and classify each example. These classifiers ranged in performance from adequate to excellent and had a vast range of time complexities for both training and testing. Physiological Data Modelling holds several obstacles to overcome and we now have a better understanding of what those are and how to deal with them.

References

[Roberto Paredes, Alfons Juan, and Enrique Vidal 2008] Paredes, Roberto, Juan, Alfons and Vidal, Enrique (2008). Theme 5 : Linear Classifiers: Perceptron *Linear Classifiers: Perceptron*, 21:22.

[Leonardo Noriega 2005] Noriega, Leonardo (November 17, 2005). Multilayer Perceptron Tutorial *The MLP Learning Algorithm*, 11.

[D. M. Chickering, Christopher Meek, and Robert Rounthwaite 2001] Chickering, D. M., Meek, Christopher, and Rounthwaite, Robert (December 2, 2001). Efficient Determination of Dynamic Split Points in a Decision Tree *Uniform Approximation*, 4.

[Tom Mitchell 2010] Mitchell, Tom (January 25, 2010). Gaussian Naive Bayes, and Logistic Regression *Gaussian Naive - Continuos X*, 8.