Random Forests and Extreme Learning Machines on 1D Feature Vectors

Casey Bruno
Department of Electrical and Computer Engineering, Temple University
casey.bruno@temple.edu

Introduction: This final exam provided us with data from two similarly generated distributions but in two different dimensions (2D and 5D). Since the tow distributions were generated in similar ways, we were able to use the 2D data to test algorithms more quickly and be able to loosely predict how the algorithms would perform on the 5D data. In addition to wanting the best prediction results for both distributions, we also wanted to predict using two different algorithms: one from our traditional Machine Learning toolset and another implementing a type of neural network. After testing multiple different traditional Machine Learning algorithms and modifying the provided MLP Network to be unique, I ended up with two algorithms that were above the rest for two different reasons: Random Forests and Extreme Learning Machine.

Random Forests: Random Forests are a relatively straightforward traditional Machine Learning algorithm; create a certain number of Random Trees and decide based on the majority vote of those Random Trees. Random Trees themselves sequentially group the data by conditions until a certain pre-defined point or it is as "pure" as possible. For this lab, I used a gini impurity value to quantify this purity, which multiplies the proportion of data points within each class at every node to get a gini impurity score. Using this, each individual Random Tree was able to be optimized for the data it was trained on. Since this "optimal" tree would be the same for the same data, I used a technique called bootstrapping to make sure that each Random Tree in my Random Forest would be unique. Bootstrapping samples the training dataset with replacement, allowing each tree to be different and resulting in a "polling of experts" since many of the trees will be specified to subsets of the data and have never seen other data points in the training set.

I used Scikit-Learn to implement the Random Forests algorithm in Python 3.7.3, which provided me with functionality that would be difficult to implement if I wrote this algorithm myself. One such functionality is hyperparameter optimization, which can have a large effect on Random Forests. Scikit-Learn allows you to set a minimum number of splits, the maximum depth, and other such limitations on the Random Forests, but often these values are different for ever data set. I order to get the most out of my Random Forest, I used a few built in Scikit-Learn functions to test multiple combinations of parameters. Due to time constraints I settled with a random selection of parameters from the options that I provided, since an exhaustive comparison was extremely computationally and time expensive on the 5D data.

Extreme Learning Machine: Provided to us as an example was a simple Multilayer Perceptron (MLP) network which we were supposed to modify and improve upon for this final report. This MLP used Pytorch and I decided to use this as my base. Due to my inexperience with neural networks and with Pytorch, I wanted to make minimal changes to the model so I could be confident that it would work and wouldn't be over trained or excessive. During my research, I came across Extreme Learning Machines (ELM) which are essentially a subset of MLPs that have slightly different activation functions and almost exclusively have a single hidden layer while MLPs could have hundreds or thousands of hidden layers. One benefit of ELM is that theory support the power of ELMs in separating even non-linearly separable distributions using any activation function/combination of activation functions. I chose to stick relatively simple and use a modified rectified linear activation function multiplied by a parabolic rectified linear activation function, as I saw in a few examples in my research.

I used Pytorch 1.3 (CPU only) to implement the Extreme Learning Machine algorithm in Python 3.7.3. Using Pytorch was something completely new to me, as my limited work in deep learning had been done using tensorflow. One strength of Pytorch was that it had prebuilt loss and activation functions, which allowed me to experiment very easily. This simple experimentation was the basis for my algorithm, as much

of my time was spent making slight modifications to the algorithm and examining the resulting error rates to find improvements. Most of this experimentation was done on the 2D data, since I chose to work with larger hidden layers and the training for the 5D data was unreasonable on my CPU only for testing.

Results: My results for my traditional Machine Learning algorithm, Random Forests, were significantly better than I expected. With the baseline measurements being set with (while simple) neural networks, I expected comparable results to be quite difficult to obtain using a traditional Machine Learning approach, but I was proven wrong by my own results. While my neural network algorithm, Extreme Learning Machine, did not perform better than the baseline, I am satisfied with the results because I went through the effort to significantly change the provided example into something different, even if it did not pan out.

	2D Data			5D Data		
Algorithm	**Train**	**Dev**	**Eval**	**Train**	**Dev**	**Eval**
Scikit-Learn: Random Forests	07.22%	08.55%	08.35%	24.62%	36.96%	36.60%
Pytorch: Extreme Learning Machine	08.22%	08.50%	08.40%	36.47%	38.48%	38.32%

Table 1. Error rates from both algorithm on both sets of data. Algorithms were trained on the training data set and blind evaluated both the development testing set and evaluation set.

Conclusions: Although my personal neural network did not perform better than the baseline, this task give me an appreciation for the thought and care that goes into designing neural networks, since a lot of my experimentation in design led to worse results rather than better. It also gave me an understanding of the "luck" that goes into getting a good algorithm; sometimes the small changes amount to a significant difference, and it isn't exactly clear why or how. I also remember Dr. Picone being very clear in the beginning of the semester that even though neural network and deep learning are the popular subjects in Machine Learning at the moment, a well thought out and optimized traditional learning algorithm is often better suited for a given task, and I definitely saw how that was possible. By being able to understand how exactly my Random Forest was modeling the data I could make slight adjustments to the hyperparameters in order to get better performance. I saw the influence of this in my results: although others used the Random Forests algorithm and got significantly better results on the training data, their models were likely over trained and performed worse than mine on the development testing and evaluation data due to that. At the end of the day, although my algorithms were not the best performing in the class, I learned some valuable lessons about the process of designing and implementing an classification algorithm as well as the process in designing a neural network algorithm.