Machine Learning Final Project

Adam Noack
Eugene, OR
anoack2@uoregon.edu

Abstract

The task this final project focused on involved predicting whether a student earned a passing grade for a class given attributes for the student. To do this, I used a support vector machine (SVM) and a deep neural network (DNN). After performing a hyperparameter gridsearch for each algorithm using sklearn and its MLPClassifier and SVC, I implemented both of these algorithms with the optimal hyperparameters from scratch using numpy.

1 Introduction

I have used both DNNs and SVMs in the past to solve prediction tasks, but despite having studied the math behind both algorithms, before completing this project, how these algorithms were learning was still somewhat of a mystery to me. To gain a better understanding of how these algorithms were operating, I decided I would code both algorithms from scratch[1].

I needed a dataset on which I could test the performance of the algorithms I built and prove they were working. I decided on the student performance dataset made available by UCI (Cortez and Silva, 2008). I chose it for its fairly small size and intuitive features. After preprocessing this data, I trained sklearn's built in MLPClassifier (a deep neural network) and SVC (a support vector classifier) algorithms on the data to determine baseline accuracy levels for each algorithm and decide which hyperparameters led to the best validation data results. After doing this, I implemented both algorithms from scratch using numpy and python's math library.

The results I obtained using the algorithms that I coded from scratch did just as well as the corresponding sklearn baselines, and I now understand the backpropagation and sequential minimal optimization algorithms much better.

[1]All of the code written for this project can be found at https://github.com/a1noack/nn_and_svm_from_scratch

2 Background

2.1 The SVM

2.1.1 Brief history and usage

The SVM was invented by Vladimir Vapnik in 1963, but was not actually used until the 1990s when the "kernel trick" was discovered. The kernel trick allows SVMs to learn nonlinear boundaries. Since this discovery, the SVM has been proven to be a highly reliable and dynamic learning algorithm, especially when the number of training instances is relatively small - less than 100,000 - or if the number of attributes for each instance is large.

2.1.2 How it learns

The objective function for the soft margin SVM is as follows.

$$\text{maximize} \quad L(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \mathbf{K}(\mathbf{x}_j \mathbf{x}_i)$$

$$\text{subject to} \quad 0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{m} \alpha_i y_i = 0$$

Interestingly, the solution involves only dot products between instances and alphas. The dot product

can be substituted by a kernel function (This was already done in the equation above. **K** is the kernel function). As stated previously, this kernel function allows for the SVM to find nonlinear boundaries between classes. Two of the most popular kernel functions are:

- Gaussian Kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}} \quad (1)$$

- Polynomial

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \mathbf{x}_j)^p \quad (2)$$

To optimize this function, we use Platt's Sequential Minimal Optimization (SMO) algorithm, a coordinate ascent approach. Pairs of alphas, (α_i, α_j) where $i \neq j$, are optimized jointly until convergence.

2.2 The DNN

2.2.1 History and usage

The DNN, or multilayer perceptron, was first used in the 1960s, and the backpropagation algorithm, the standard optimization algorithm for DNNs, was known under a different name as early as the 1970s. Despite the fact that the ideas underlying the DNN are fairly old, DNNs have only just recently become practically useful. This is because modern day computers have finally acquired the computing power needed to train DNNs in the last ten years or so, and access to the large datasets that DNNs need to learn has been steadily increasing. DNNs are particularly useful when the number of training is very high and the number of features relative to the number of samples is somewhat low.

2.2.2 How it learns

The DNN is essentially a number of single layer, feedforward neural networks linked together. The activation values for the $i+1$th layer can be found as follows:

$$a_{i+1} = \sigma(\mathbf{w_{i+1}} a_i + b_{i+1}) \quad (3)$$

Here a_i are the activation values from the previous layer, and $\mathbf{w_{i+1}}$ is the weight matrix connecting the nodes in the ith layer to the nodes in the $i+1$th layer. b_{i+1} is the bias vector for the $i+1$th layer. σ is the activation function. In a DNN these activation vectors are passed through to the next layer via another weight matrix. This is done for the desired depth of the network, and culminates in an output layer of a desired size.

To train the network, the network's output for a given input is compared to that input's correct label. The degree to which the predicted value differs from the actual value constitutes the error for a given sample. These errors are usually aggregated in some way across training samples using a loss function such as mean squared error. The derivative of this loss function with respect to each weight in the network is then computed. Derivatives of the loss function w.r.t. weights connecting early layers are calculated by propagating the error computed at the output layer back through the network.

3 Methods

3.1 Data

The data I used for this project was collected on 649 students from two Portuguese schools. Each student has 31 different attributes, e.g. which school the student attended, sex of student, age, number of absences, performance on midterms, etc.

Originally, the task was to predict the exact grade value for each student at the end of the term, but following the lead of (Cortez and Silva, 2008), I converted the regression data into classification data by changing each student's y value from a number between zero and 20 to either one or zero - one if they passed (had a score higher than nine), zero if they failed (less than or equal to nine).

No features were removed from the dataset during preprocessing.

All categorical attributes with more than two possible values were one-hot encoded.

The data was then normalized using both a mean-scaling approach and a min-max scaling approach.

After scaling the data, the data was split into training, validation, and test data with ratios .73/.12/.15, respectively.

3.2 Finding the best model architectures/hyperparameters

As stated previously, I first used sklearn's MLPClassifier and SVC classes to search for optimal hyperparameters and develop baselines for both the SVM and the DNN.

The DNN displayed the best performance with the following hyperparameters:

- two hidden layers, sizes 225 and 200
- learning rate of .01
- batch size of 32
- the relu activation function

The remaining hyperparameters were left untouched. I.e. the default hyperparameter values that sklearn provides for the MLPClassifier were used for all other hyperparameters.

The SVM displayed the best performance with the following hyperparameters:

- C = 5
- the radial basis function kernel

Again, the default hyperparameter values that sklearn provides for the SVC were used for all other hyperparameters.

With the knowledge from these grid searches, I set out to create both a DNN and an SVM from scratch using numpy with these optimal hyperparameters.

3.3 Implementing the DNN

Implementing the DNN using numpy was fairly straightforward. I initialized the weights of each layer by drawining values from a standard normal distribution, i.e. mean of zero and standard deviation of one. Then I multiplied each of these values by $\sqrt{\frac{2}{size_{l-1}}}$, where $size_{l-1}$ is the number of hidden units in layer $l-1$.

The only tricky part was the backpropagation algorithm. I used formulas from various textbooks I found online as models for my code. The code for the entire algorithm is completely vectorized. This means that my network can be trained very quickly.

3.4 Implementing the SVM

To optimize the SVM, the current best approach is the SMO algorithm. However, I found that implementing the full SMO algorithm would be too much of a challenge, so I opted to use a simplified version of the SMO algorithm. Unlike the full SMO algorithm, the simplified one is not guaranteed to converge, but it was much easier to code. More importantly, the coding process still allowed me to gain some intuition as to how the optimization is occurring. I used the psuedocode listed in a document from the Stanford CS Despartment to get a start on the coding [2].

To make the data for this prediction task work with the SVM objective function and SMO, I converted the labels for each sample from zero and ones to negative ones and positive ones, respectively.

4 Results

Compared with the sklearn model baselines, my hand-coded models did very well. The results can be seen in Figure 1.

When performing the grid search for the hyperparameters for each of the models, I treated the data normalization technique as an additional hyperparameter. It was determined that the mean-normalized data resulted in much higher validation accuracies across the board. Therefore, the results below were all obtained using this mean-normalized data.

My SVM performed only slightly worse on the test data than sklearn's SVC trained in a similar manner with the same hyperparameters.

My deep neural network performed slightly better than sklearn's model. Because both networks have essentially the exact same parameters, I am led to believe that this marginally better performance is due to the stochastic nature of the minibatching and because I only trained and tested the algorithm once. I think the performances would converge if I repeated this process and then averaged the results.

I was originally going to compare my results with those of Cortez et al., but it seems that they arranged the training and test data in a different way than I did. They split the 649 data points into two separate data repositories from the beginning. One was for

[2] http://cs229.stanford.edu/materials/smo.pdf

	SVM	DNN
sklearn	78.33	83.33
Hand-coded	76.67	85.00

Figure 1: Test accuracies of each model for each approach.

those students whose final grade (attribute G3) referred to their grade in a Portuguese language class, and the other repository held all of those samples in which the final grade represented the student's score in a math class. I lumped these two tasks into one, essentially making the prediction task more difficult. Therefore, I will not and cannot compare my results to Cortez et al.

5 Conclusion

Coding the SVM and DNN from scratch allowed me to gain an understanding of how these algorithms are working under their respective hoods. The results I obtained using my DNN and SVM are actually quite decent when compared with those of the models I trained using sklearn and those listed in (Cortez and Silva, 2008). All in all, this project was difficult but highly rewarding.

References

[Cortez and Silva2008] Paulo Cortez and Alice Silva. 2008. Using data mining to predict secondary school student performance. *EUROSIS*, 01.