Eric Sirinian
Phillip Kerger
CISC5800: Machine Learning
Prof. Daniel Leeds
December 9, 2018

Classification on the 'Adult' Data Set: Neural Network and Principal Component Analysis Approaches

Fordham University, Fall 2018, CISC5800

Abstract:

We work with the "Adult" Data Set from the University of California Irvine Machine Learning Repository with the goal of comparing the classification methods of Neural Networks and applying PCA to logistic classification. The goal is to successfully classify whether a person earns a salary of $50,000 USD or not based on certain features about the individual. We preprocess the dataset to amend its shortcomings and make it more usable for our machine learning purposes. The issue of having categorical and numerical features is resolved through impact encoding. We design a neural network and a logistic classifier. Improvements are made by changing the activation function from the sigmoid function to the hyperbolic tangent function for the neural network and incorporating PCA with the support vector machine. The obtained correct classifications on our testing data were 74.61% for a basic neural network, 77.08% for the improved neural network, 76.58% for simple logistic classification, and 84.32% for logistic classification with PCA improvements. A discussion follows on the successes and shortcomings of each method implemented.

Introduction:

Classification tasks based on large data sets have steadily become more and more important in our society. Natural language processing, targeted advertising, image recognition, and recommendation algorithms are all of crucial interests to major industry leaders like Amazon, Google, Facebook, Microsoft, or Apple who rely heavily on machine learning and classification methods.

In order to become familiar with and analyze different machine learning techniques, we apply a neural network and a logistic classifier using PCA to the "Adult" data set from the UC Irvine Machine Learning Repository, to be found here: https://archive.ics.uci.edu/ml/datasets/adult . We work with the "adult.data" file only. The raw data set consists of 32,561 entries classified by whether or not the given person earns more than $50,000 USD yearly or not. Each data point consists of the following features: age (continuous), workclass (categorical), fnlwgt (continuous), education (categorical), education (categorical), education-num (continuous), marital status (categorical), occupation (categorical), relationship (categorical), race (categorical), sex (categorical), capital-gain (continuous), capital-loss (continuous), hours per week (continuous), native country (categorical). This data set is particularly interesting because one expects to have correlations between the features and the class of each data point, but no single feature seems like it would necessarily overpower other ones. Furthermore, given every feature above, we still expect the income of an individual to have significant variance which makes classification more challenging. However, as we will continue to show, our classification methods manage to achieve strong results despite the challenges the data set poses.

Pre-processing:

Before starting classification, we turn to improvements to be made on the dataset to ensure the machine learning techniques we apply will work well on the features and values we have. The most obvious issue in the raw data is data points that have a "?" as their value. There are about 2,400 points exhibiting this issue. Since the data set is sufficiently large, we simply exclude all points with such a value for any feature. We regard losing these data points as worth not having to infer anything about feature values in our data.

Another problem we encounter with the data is the number of countries possible in the "native country" feature, which makes this feature difficult to work with as categorical. However, the underlying figure of importance in this feature is actually the average wealth of the native country of each person, since we are concerned with the income of the individual. We thus conclude that is best to replace the categorical value of each country into a continuous value of the GDP per capita of the country, using data from The World Bank from 1996. However, data for Laos, Taiwan, and Outlying US territories for 1996 is not available from the World Bank, so we dropped the few data points with these country values.

Furthermore, looking into the feature "fnlwgt", we found that this is a figure from SIPP to reflect an individual's general well-being in society. A major factor in determining this number is the person's income, so we felt it would inappropriate to use this figure to classify a person's income: Since income is a direct input into this figure, we would in a sense be using income to classify income. We thus remove this feature entirely from the dataset.

We also decide do consolidate a few of the categorical features into less categories by combining similar categories into one. We group the "workclass" feature values into not-working, self-employed, government, and private, the "education" feature values into high-school dropout, high

school diploma, professional school, associate's degree, bachelor's degree, master's degree, doctorate, and the "marital status" into married, single, divorced, separated, and widowed. Grouping the categories into these new ones will help us avoid overfitting. Since we have the information on what level of education each individual has, we also drop the "education num" feature since it is less informative than the "education" feature.

After making these changes, our new dataset consists of 30,018 data points, each of which has 12 features and its class value. Of those features, 7 are categorical and 5 are continuous. We split the data set into a training set consisting of the first 20,000 points of the set and a test set consisting of the last 10,018 data points. After making these changes, we are ready to apply our machine learning techniques to the data.

Methods and Results:

M.1: Impact Encoding

A clear challenge the data set poses is the mixture of categorical and numerical features. Having both feature types present makes it difficult to directly implement classical classification methods like typical Bayes or logistic classifiers. We solve this issue by applying impact encoding, also known as likelihood encoding. The idea behind it is the following: For each categorical feature value, we care about the conditional likelihood of the class of a data point being 0 or 1 given that feature. Therefore, the categorical feature values can be replaced by the conditional likelihoods we observe in the training data. It is important to notice that these are learned probabilities that we infer from our training data. For each categorical feature with n possible values f^1, f^2, …, f^n, we calculate the following probability:

$P(\text{class} = 1 \mid f^i) = (\#\text{points with } f^i \text{ and class } 1) / (\#\text{points with } f^i)$

We then replace the feature value f^i with P(class = 1| f^i). For example, in our training data we observe P(class = 1 | Education = "Master's") = 0.56, meaning we replace "Master's" with the numerical value 0.56. This turns all of our features into learned numerical probability values, allowing us to work more effectively with neural networks and support vector machines.

M.2: Neural Network

The first method we implement is a neural network with an activation function improvement. We construct the neural network as one with two hidden layers. There are 12 feature inputs fed in to the neural network, in addition to the offset. The first layer consists of two nodes, the second layer consists of five nodes, the third layer consists of three nodes, and there is a single node in the output layer. The weights for each layer were initialized as values between 0 and 1. The feed forward functions computes the initial r values and output values by feeding the dot product of the weights and input values (or r values) in to the activation function. The back propagation function then proceeds to find the change in each layer's weight vectors by using gradient ascent. This varies based on the different activation functions in the basic and improved network.

For the sigmoid function, we implement the standard formulas explained in lecture, utilizing different equations at the top layer and lower layers. For the hyperbolic tangent function, we define the activation function as $tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$. Taking the derivative with respect to x yields $tanh(x)' = (1 - tanh(x)^2).$ This leads us to a different update rule for each weight vector than for the Sigmoid function, namely $\Delta w = \varepsilon(1 - r^{top^2})(y - r^{top})r^{top-1}$, for the top layer. This also changes the update formulas accordingly (both delta and error signal formulas) for the lower layer, following suit of the hyperbolic tangent derivative provided above. Both models, basic and

improved, were trained on the train split data by looping the feedforward and back propagation functions successively 1500 times. This allowed the network to properly update and achieve optimal layer weights to predict the outcome of the test split data.

Below is a code section of the neural network utilizing the hyperbolic tangent activation function instead of the standard Sigmoid function. We show the feed forward and the top layer portion of the back propagation function:

```
def feedforward(self, test = None):
        if test is None:
            self.layer1R = myTanH(np.dot(self.input, self.layer1W))
        else:
            self.layer1R = myTanH(np.dot(test.values, self.layer1W))

        self.layer2R = myTanH(np.dot(self.layer1R, self.layer2W))
        self.layer3R = myTanH(np.dot(self.layer2R, self.layer3W))
        self.output = myTanH(np.dot(self.layer3R, self.layer4W))

    def backprop(self):
        epsilon = 0.3

        delta_W41 = epsilon * (1 - np.square(self.output)) * (self.y - self.output) *
self.layer3R[:, 0].reshape(-1,1)
        delta_W42 = epsilon * (1 - np.square(self.output)) * (self.y - self.output) *
self.layer3R[:, 1].reshape(-1,1)
        delta_W43 = epsilon * (1 - np.square(self.output)) * (self.y - self.output) *
self.layer3R[:, 2].reshape(-1,1)

        for delta1, delta2, delta3 in zip(delta_W41,delta_W42,delta_W43):
            self.layer4W[0] = self.layer4W[0] + delta1
            self.layer4W[1] = self.layer4W[1] + delta2
            self.layer4W[2] = self.layer4W[2] + delta3

        top_error = (1 - np.square(self.output)) * (self.y - self.output)

        ec31 = np.sum(np.array([self.layer4W[0] * e for e in top_error]))
        ec32 = np.sum(np.array([self.layer4W[1] * e for e in top_error]))
        ec33 = np.sum(np.array([self.layer4W[2] * e for e in top_error]))
```

[Runtimes: Training the model - $O(n^2)$, Running with test data: $O(n)$]

Using the basic implementation of the neural network, we achieved an accuracy of 71.61%. In order to judge the performance of this model we implemented a neural network from the MLPClassifier in the sklearn package in Python. To ensure that the third party classifier outputted accurate results we utilized the cross validation score method from the sklearn package. We

ran the third party classifier with five cross folds to receive scores of 26.12 %, 78.83% , 80.95%, 79.10%, and 79.19%, leading to an average accuracy of 68.83%. As evident by the accuracy scores, our basic implementation of our model performed better than the third party model. For the improved neural network model, utilizing the hyperbolic tangent function as the activation function, we achieved an accuracy of 77.08%, a good amount better than the basic implementation.

M.3: PCA Classification

The second method we implemented is a support vector machine with the improvement of principal component analysis. After applying the impact encoding as described, we initialize a starting vector with random entries and apply a standard gradient descent learning algorithm to find the optimal weights for each entry of the vector. We use a step size of 0.01 as our hyper-parameter in the learning function. A relatively small step size is desirable due to the likelihood encoding resulting in having feature values between 0 and 1 for all features that were originally categorical. Using the basic implementation of the logistical classifier, we obtain a correct classification on our testing dataset of 76.58% (runtime: 0.50052 seconds). Note that changing hyper-parameters such as the step size or initial vector barely have any effect on this likelihood.

To improve the classification, we use principal component analysis. In order to do so, we first normalize our dataset. This is especially important in this context because our feature columns take very different values; the capital gains feature, for example, varies from 0 to 99,999, whereas many of our other values are between 0 and 1. Our goal in PCA is to find the axes amongst which our data varies the most, and PCA on the raw data set would simply conclude the capital gains feature to be the axis of highest variance

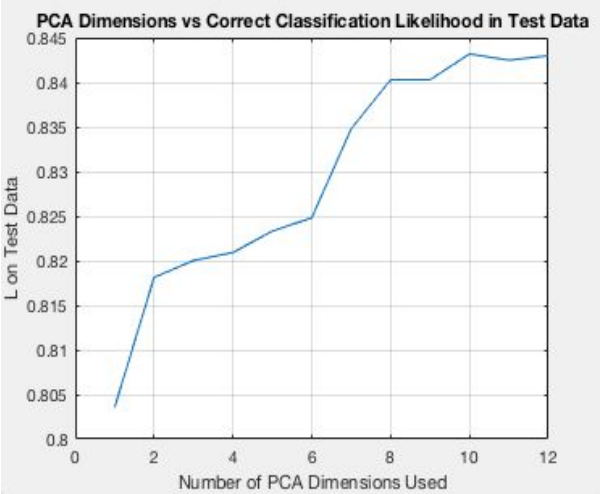
as a result of the nature of its entries. We thus first normalize the dataset, expressing each feature value in terms its standard deviation away from the mean of that feature. To do this, we use normalized_data = normalize(data) in Matlab. We can then apply PCA to find the axes along which our data truly exhibits the greatest variation. We obtain the PCA vectors through pca_vectors = pca(normalized_train_data). After obtaining these, we are primarily concerned with how many principal components we need to reliably classify our data. We thus reconstruct our train and test data using 1, 2, 3, …, 12 of the PCA vectors and construct a new classier for each. To achieve this, we take the dot product of each data point with each PCA vector used. These values are recorded and used as new train and test data sets. We construct the classifier in the same manner as the original one and record the correct classification likelihood on the test data for each number of PCA dimensions used. This process being significantly complex, we include the loop to accomplish this (Matlab):

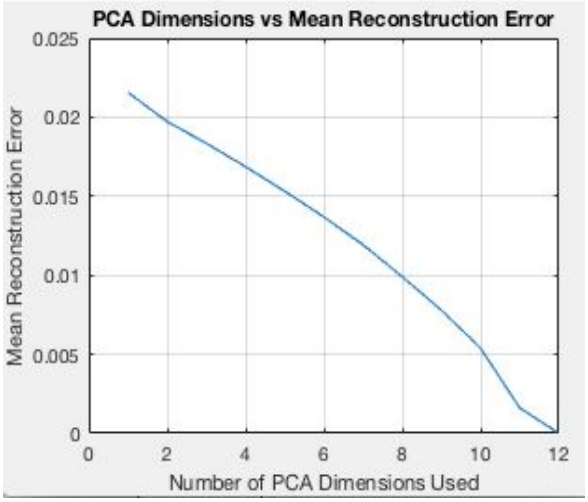
```
for NUM_PCA_DIMS = 1:MAX_PCA_DIMS
    %apply PCA to train and test data, reconstructing with NUM_PCA_DIMS pca vectors
    %svmfeed(:,13) is the columns of 1's for the offset
    pca_train_data = horzcat(zeros(TRAIN_DATA_SIZE, NUM_PCA_DIMS), svmfeed(:,13));
    pca_test_data = horzcat(zeros(length(test_classes),
NUM_PCA_DIMS),newtestdata(:,13));
    for i = 1:TRAIN_DATA_SIZE
        for j = 1:NUM_PCA_DIMS
            pca_train_data (i, j) = dot(norm_train_data(i,1:12), pca_vects(:,j));
        end
    end

    for i = 1:length(test_classes)
        for j = 1:NUM_PCA_DIMS
            pca_test_data (i, j) = dot(norm_test_data(i,1:12), pca_vects(:,j));
        end
    end

    %% SVM Part
    %learn w vector:
    w0 = ones(1,NUM_PCA_DIMS+1);
    w_vect_pca = learn_w_vector(w0, pca_train_data, train_classes);

    pca_test_data_accuracy(NUM_PCA_DIMS) = svm_test(pca_test_data, w_vect_pca,...
    test_classes);

    %% Record Reconstruction error
    %reconstruct train data
```

```
    train_reconstr = zeros(size(norm_train_data));
    for i = 1:TRAIN_DATA_SIZE
        temp_x = zeros(12,1);
        for j = 1:num_pca_dims
            temp_x = temp_x + pca_vects(:,j)*pca_train_data(i,j);
        end
        train_reconstr(i,:) = temp_x;
    end
    %record error
    temp_err = norm_train_data - train_reconstr;
    reconstr_errors(num_pca_dims) = sqrt(sum(sum(((temp_err.^2)))))/TRAIN_DATA_SIZE;

end
%pca_test_data_accuracy records correct test classification likelihoods L
%reconstr_errors records data reconstruction errors
```

[total runtime: 17.4s. Runtime using 10 PCA dims only: 2.61 ($O(n^2)$)]

The results we obtain are presented in the following figures:





As expected, we see the mean reconstruction error for each point is reduced as more PCA reconstruction vectors are used. The error is calculated across our training set as the sum of the distances from each of the reconstructed points to the original points, divided by the total number of points. Surprisingly, reconstructing the data with just one PCA vector results

in a correct classification test likelihood of L = 80.36%, which can be attributed to the relatively small reconstruction error obtained. Comparing this to the L=76.58% from the regular logistic classifier, the implementation of PCA proves to be very strong. The dimension of the data feature space can be reduced to just 1 and still be classified effectively! As more vectors are used to reconstruct the data, L steadily increases until it plateaus for 10 dimensions used. At this point, we achieve L = 84.32%. Again, this is a significant improvement to the L = 76.58% achieved by the original classifier.

The only significant hyper-parameter in this method is the step size used in the learning function for the weights. Letting ε run across the values of 10, 1, 0.1, 0.01, and 0.001, we obtain the following likelihood results:

ε	10	1	0.1	0.01	0.001
L from basic logistic classification	0.7658	0.7658	0.7659	0.7675	0.7674
L after PCA, using 10 dimensions	0.7907	0.7925	0.8351	0.8432	0.8362

We see the basic logistic classifier is very rigid with respect to changes to ε, while after applying PCA, the smaller values of ε tend to yield greater L values. Since the PCA data is normalized, having a smaller magnitude for values, it makes sense that a smaller ε value gives us better results.

Discussion / Conclusion:

Method	Basic NN	Activation function improved NN	Basic Logistic Classification	PCA improved Logistic Classification
L Value	74.61%	77.08%	76.58%	84.32%

Although each classifier has its pros and cons, in the context of the data set we see that the classifiers do exhibit different performance. Measuring by their effectiveness in correctly predicting the classes for the test data, the best classifier is the PCA improved logistic classification, followed by the activation function improved neural network, the basic logistic classification, and the basic neural network. We see that both of our improved methods outperform the standard methods, which should not be surprising. PCA restructures the data such that the axes of highest variance are considered, and hyperbolic tangent is generally a better activation function for the purpose of constructing neural networks than the Sigmoid function.[1]

However, the margin with which the PCA logistic classification outperforms the other methods is larger than we would have anticipated, at 84.32% using 10 PCA dimensions. This method being especially effective for this data set tells us that the principal axes of variation hold a lot of importance with regard to the class value of any given point. Furthermore, the PCA method is able to achieve very high L values with fewer feature dimensions than the other methods. In practice, if data storage was an issue, this would make PCA extremely effective since we could significantly reduce the size of our data set to only a set of PCA vectors and weights for each point. For example, using only 6 PCA dimensions would cut the size of

[1] Karlic & Olgac, *Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks*
(http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.740.9413&rep=rep1&type=pdf)

our data set in half but still give us a correct classification likelihood of L=82.5. We also see the improved neural network performed strongly at 77.1% correct classifications. However, given the increased complexity and runtime compared to the PCA classification, PCA comes out as the clear winner for this dataset.