

PHASE 2 REPORT

EEE 485 TERM PROJECT

Ahmad Zafar Khan

21403048

Hassan Ahmed

21500439

Problem and Dataset Description

The dataset we plan to use is a collection of images of fruits taken from different rotated angles. The total size of the dataset is 65429 images. It is divided into a training and test set of 48905 images and 16421 images respectively. There are 95 different fruits which means the classification algorithm we implement will have 95 classes to choose from. There is also a smaller dataset of 103 images with multiple fruits in the same image. This can be useful to test our algorithm on more complex training sets to see how they would perform in situations more similar to the real world. The dataset also does not combine different varieties of fruit so apples are separated into five different classes corresponding to five varieties.

For the dataset, we conducted some basic processing on it. This was done to standardize the data instances. To remove the variations between different types of data. This was done by subtracting the mean and dividing by the standard deviation. We can also perform PCA on the data to lower the dimensionality of the data. This will help increase the accuracy too and is planned for the next stage.

Review of Methods

# Neural Network

Convolutional Neural Network

For the third method, we created a Convolutional Neural Network to train the model to recognize the type of fruit from a given image. This network can be divided into two separate parts to clearly understand how it works and was implemented. The first part of the model works by passing the data into 2 convolution layers and then a maxpool layer. Then it is passed into a fully connected hidden layer that leads to the output layer.

The convolution layers work by applying filters to the image data to extract information from it. This works by multiplying a smaller matrix of numbers to the image dataset and striding through the image one by one both horizontally and vertically. The resulting matrix is a smaller one which reduces dimensionality and makes it faster and more feasible for the fully connected layer to operate. The convolution is done by all the initialized filters. The dimensions of the input to the first convolution layer is number of channels \* height \* width. The number of channels are usually three for the RGB values of an image but for our case it is the number of methods of normalization for the orientation gradients which are 4 in our case. When passed through the layer the dimensions become, number of filters \* height \* width. The feed forward layer equations are:

Where

For the back propagation we use the gradients of the functions.

And

## Training Phase

For updating our algorithm we used a version of the stochastic gradient descent known as adam gradient descent. This estimates the value of the mean and variance using parameters β1 and β2 and uses these to update the value of the weights/filters/biases. The formula used for updating these rules is:

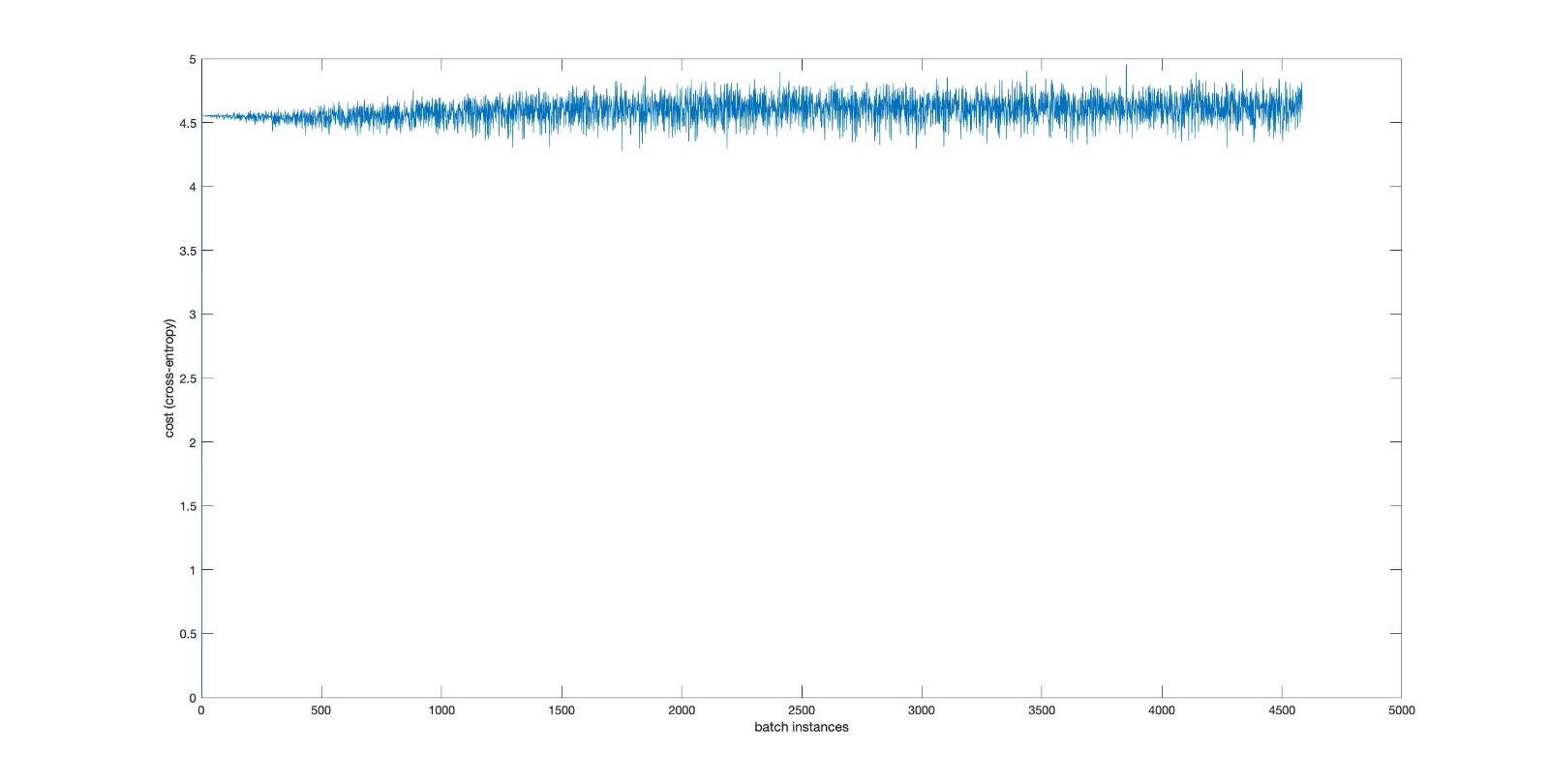
Where

The activation functions for the convolution, maxpooling and first hidden layers were ReLu while the output was the softmax function since we wanted the probabilities to sum to one for the output. The hidden layer had 100 nodes. They were all fully connected.

The time taken to train the dataset is 11.6 hours for 2 epochs. The batch size was taken as 32 datapoints from a training dataset of 48,905 images.

## Testing Phase and Results

When the accuracy of the model was tested it came to 2% which implied that the model could not train properly. It will require a lot more epochs than two to train completely. This is because of high number of classes. This will need a much more powerful computer to train but we can still show some results.



# Naïve Bayes Classifier

The Naïve Bayes classifier is based on the bayes rule given below:

The object of this classifier is to learn the probability for each class. Such a classifier can be extended to multiple classes therefore it was chosen.

Given that the feature set of the fruit data, which was a 20x20x4 matrix of HOG values for each label, was composed of continuous values we modeled the feature set on the gaussian distribution as follows:

## Training Phase

The mean and standard deviations were learned for each class. This was done by first isolating the each label of class k . The corresponding feature vectors in the train set were also isolated and the mean and standard deviation for each variable in the training feature set was learned from these isolated sets. This was continued until we had a mean and standard deviation of each of 1600 features for each class. So in total we had values for the mean and the standard deviation. These values were used to form the likelihood function for each class:

These observation are independent as per the naive assumption therefore we multiply these probabilities. And we do so for each class. The value for will be each feature vector of the test set in the testing phase.

We also had to learn the prior probabilities for each class . This was done by simply finding each instance of the class in the training label set and then dividing it by the total number of labels in the class. I.e.:

Where is the number of instances of label in the label set.

## Testing Phase

After the aforementioned parameters were learned, it was time to test. For this part We used the log likelihood function iteratively over each class of label. This was done for each feature vector. Then whichever value of for the likelihood function gave us the maximum value, the feature vector was attributed to belonging to that set. I.e.:

Where the probability of X given Y is the gaussian distribution given above, with each variable in the feature set occupying the variable in the gaussian distribution with its corresponding mean and standard deviation. The probability of y is the prior probability given above.

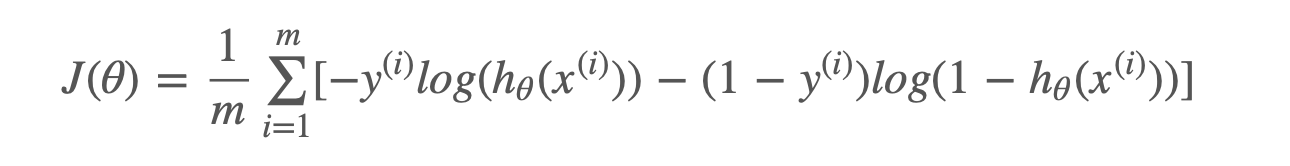
## Results

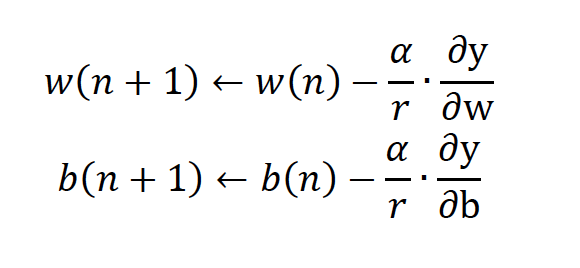
The accuracy is roughly 23%. We believe that for naive bayes classifier, a low accuracy is to be expected, especially in cases which involve continuous features. Moreover, our data set is not strictly balanced as we are making multi class classifier therefore we are expected to lose a bit of accuracy.   
  
However, for the final testing period, we believe that we can increase this accuracy to the mid 30s by adding additive smoothing which would regularize the naive bayes classifier to a certain extent. Therein, our new likelihood probabilities will be:

However, we will only apply this additive smoothing if we discretize the values of the features. Otherwise, we might apply Bessel’s correction to the gaussian distribution or try different distributions such as gamma distribution. But it is important to note that we will try all these methods and present the one which gives us the highest accuracy.

# Logistic Regression

Logistic regression is based on the sigmoid function below:

As stated in the previous report, we decided to use one-vs-all logistic regression. We defined our loss function as the one below: 

Where was the product of our sigmoid function for the dot product of the weights with the feature vector. We used gradient descent to optimize our weights. The formula for it is given below (b(n+1) is our bias term in the weights): 

Where is our learning rate, which we will fine tune by trial and error, and is what ever the bias is, found by gradient descent.

In order to actually implement this formula in the code, we had to apply chain rule to get the jacobian matrix in the correct format:

Therefore, our update rule for the gradient descent algorithm becomes:

Before the training could commence, the HOGValuesTrain and test cell datasets were transformed into matrices of size 48905x1600 and 16421x1600 to make the computations easier. Moreover, a bias column of 1s was added to the sets.

## Training Phase

The data given to the logisticRegressionWeights2 function which would compute the weights. As we were low on time and the weight calculation for the desired 100000 iterations would take at least 12 hours, we tested the algorithm for only 5 epochs with a greater learning of 0.1 to compensate. Ideally we would like to give the algorithm enough iterations with a smaller learning rate to learn.

## Testing Phase

For the testing phase we used the logisticRegressionClassifyforN function which takes the test feature set and the weights as the input. For one-vs-all we had to run this algorithm for each feature vector, a total of 95 times with a different weight each time. This algorithm input the dot product of the weights and the feature vector into the sigmoid function. Then the weight corresponding to maximum of the outputs of the sigmoid function was chosen. And then the feature vector was predicted to belong to the class of the weight with the highest output of the sigmoid function as the sigmoid function represents the probability .

## Results

The accuracy of this algorithm was around 1% which is extremely low, however we did not give the algorithm enough time to train as we did not anticipate that the logistic regression algorithm would take so long to train. Right now we were training for 5 iterations, with an elapsed time of around 30 minutes, but we expect that with 100000 iterations we would get a very high accuracy. Moreover, will also then be willing to implement the mini-batch gradient function instead of the gradient function but as we are getting our highest accuracy for the gradient descent function, we decided to include that in our report.   
  
Before, the next report we will spend more effort on the pre-processing of the dataset with PCA algorithm too to salvage more time and make eliminate the instances in the dataset with very low variances. Another, problem we encountered was the precision of the calculated probabilities. In order to eliminate false 0 probabilities, we will increase the precision of the probabilities to around 20 digits, which is likely to be computationally exhaustive.

# CODE

%% naive bayes classifier

%calculating mean and std dev of features

%total no. of features for each image: 400x4

%features are x1, x2, x3, x4

piRatios = zeros(95, 1);

featIsolated = {};

mu = {};

stdDev = {};

featCentered = [];

for k = 1:95

piRatios(k) = sum(yTrain == k);

end

clear i j k;

tempMean = 0;

for fruit = 1:95

v = find(yTrain == fruit);

for u = 1:length(v)

featIsolated{u, 1} = HOGValuesTRAIN{v(u), 1};

end

clear v;

for i = 1:20

for j = 1:20

for k = 1:4

for u = 1: length(featIsolated)

tempMean = tempMean +...

featIsolated{u, 1}(i, j, k);

featCentered(u) = featIsolated{u, 1}(i, j, k);

end

mu{fruit, 1}(i, j, k) = tempMean/length(featIsolated);

featCentered = (abs(featCentered - mu{fruit, 1}(i, j, k))).^2;

stdDev{fruit, 1}(i, j, k) = sqrt((sum(featCentered)/length(featIsolated)));

% disp(stdDev{fruit, 1}(i, j, k))

tempMean = 0;

featCentered = [];

end

end

end

end

clear u;

disp("learning means and standard deviations are done")

%% Testing phase

disp("Testing phase")

y\_mle = zeros(95, 1);

y\_hat = zeros(95, 1);

probs = 0;

theta\_fun = @(x, u, sigma) 1/(sqrt(2\*pi)\*sigma)\*exp(-((x-u)^2)\*(1/(2\*sigma^2)));

theta = [];

clear fruit;

count = 1;

for u = 1: length(HOGValuesTEST)

for fruit = 1:95

for i = 1:20

for j = 1:20

for k = 1:4

sigma = stdDev{fruit, 1}(i, j, k);

muu = mu{fruit, 1}(i, j, k);

probs = probs + log(double(1/(sqrt(2\*pi)\*sigma)\*exp(-((HOGValuesTEST{u, 1}(i, j, k)-muu).^2)./(2\*sigma.^2))));

end

end

end

y\_mle(fruit) = log(piRatios(fruit)) + (probs);

% disp(probs)

probs = 0;

end

[D I] = max(y\_mle);

y\_hat(u) = I;

% disp(y\_hat(u))

y\_mle = zeros(95, 1);

end

disp("accuracy is: ")

accuracy = sum(yTest == y\_hat)/length(yTest)

%% confusion matrix plot

plotconfusion(yTest,y\_hat)

%logistic regression script

weight = zeros(1601, 95);

for N = 1:95

disp(N)

y = zeros(size(yTrain));

k = find(yTrain == N);

y(k) = 1;

XTrain = trainF(:, 2:end);

XTest = testF(:, 2:end);

[n, p] = size(XTrain);

w0 = rand(p + 1, 1);

weight(:, N) = logisticRegressionWeights2( XTrain, yTrain, w0, 20, 0.1);

end

%%

y\_hat = ClassifyforN( XTest, weight/100000000 );

disp("done")

%%

err = sum(yTest ~= y\_hat');

accuracy\_log = 1 - err / size(XTest, 1) ;

function [w] = logisticRegressionWeights2( XTrain, yTrain, w\_o, epoch, lr)

[n, m] = size(XTrain);

w = w\_o;

previous = 0;

for j = 1:epoch

disp("iteration number: " + j)

temp = zeros(m + 1,1);

for k = 1:n

temp = temp + (sigmoid([1.0 XTrain(k,:)] \* w) - yTrain(k)) \* [1.0 XTrain(k,:)]';

end

w = w - lr \* temp;

cost = CostFunc(XTrain, yTrain, w);

if j~=0 && abs(cost - precost) / cost <= 0.0001

break;

end

precost = cost;

end

% w = w/1601;

end

function [ res ] = ClassifyforN( XTest, w )

[m n] = size(w);

y\_mle = [];

y\_hat = [];

%this w should be of size(1601, N)

nTest = size(XTest,1);

res = zeros(nTest,1);

for i = 1:nTest

disp(i)

for j = 1:n

w0 = w(:, j);

% y\_mle = vpa([ y\_mle sigmoid([1.0 XTest(i,:)] \* w0) ], 20);

y\_mle = ([ y\_mle sigmoid([1.0 XTest(i,:)] \* w0) ]);

end

[D I] = max(y\_mle);

disp([D I]);

y\_hat(i) = I;

y\_mle=[];

end

res = y\_hat;

end

function g = sigmoid(z)

% z = dot(x, B);

h = 1\*exp(-z)/(1 + exp(-z));

g = h;

end