# Softmax Regression

**Nishant Agarwal** ee1140464@iitd.ac.in
*Department of Electrical Engineering*
*Indian Institute of Technology Delhi*
*Hauz Khas, New Delhi, India*

**Editor:** Nishant Agarwal

## Abstract

There are many problems in the world where we have more than two outcomes (identifying a digit from the MNIST database; etc.). When the goal is to distinguish between multiple classes, softmax regression model is useful. This model is a generalisation of logistic regression for multiclass classification. The hypothesis for this regression model is based on the softmax function, given by

$$\sigma_z(j) = \frac{\exp(z_j)}{\sum_{k=1}^{K} exp(z_k)} \quad for \;\; j = 1..K$$

Here z is a K-dimensional vector squashed to K-dimensional vector $\sigma$(z) of real values in the range (0, 1) that add up to 1. For this assignment I have applied gradient descent method and have analysed the effect of various hyperparameters and optimizers on the accuracy of prediction.

## 1. Introduction

Softmax regression model is commonly used for multinomial classification problems. The hypothesis $h_\theta(x)$ is represented by a softmax function. The hypothesis $h_\theta(x^{(i)})$ represents the probability that $y = k$ on input $x$. Thus, our hypothesis will output a k dimensional vector (whose elements sum to 1), giving us our k estimated probabilities-

$$h_\theta(x^{(i)}) = \begin{bmatrix} p(y^{(i)} = 1|x^{(i)};\theta) \\ p(y^{(i)} = 2|x^{(i)};\theta) \\ \vdots \\ p(y^{(i)} = k|x^{(i)};\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix}$$

For classification, we choose the maximum value in the hypothesis vector, and the index of that value gives us the classification label.

Our aim is to optimize the cost function which is the generalization of model used for logistic regression. So we define a new cost function J($\theta$) for logistic regression as follows-

$$J(\theta) \;=\; \frac{-\sum_{i=1}^{m}\sum_{j=1}^{k}(1(y^{(i)}=j)log(p(y^{(i)}=j|x^{(i)};\theta)}{m}$$

where $1(y^{(i)}=j)$ is the indicator function which has value of 1 if $(y^{(i)}=j)$ else 0 and $p(y^{(i)}=j|x^{(i)};\theta)$ is the probability function given by

$$p(y^{(i)}=j|x^{(i)};\theta) \;=\; \frac{\exp(\theta_j^T x^{(i)})}{\sum_{l=1}^{K} exp(\theta_l^T x^{(i)})} \;\; for \;\; j=1..K$$

For optimization, the above cost function need to be minimised for all parameters $\theta$. By applying Gradient Descent Algorithm for minimising cost function with respect to all the parameters we get the below equation which dictate how each parameter should be updated

$$\theta_j \;=\; \theta_j - \frac{-\alpha\sum_{i=1}^{m}[x^{(i)}(1(y^{(i)}=j)-p(y^{(i)}=j|x^{(i)};\theta))]}{m}$$

where $\alpha$ is the learning rate and m is the number of training sets. Here $\theta_j$ is a k-dimensional vector. All the parameters must be updated simultaneously in each iteration. The parameters are updated in each iteration until cost function converges with respect to number of iterations.

## 2. Assignment and Observations

In this assignment we had to apply softmax regression model on given data set containing 15000 examples with 16 input parameters each and corresponding 26 classification labels. Applying the stochastic gradient descent algorithm on the data set with learning rate $\alpha =$ 14 and decay $\lambda = 0.004$ resulted in highest 77.38% accuracy on training set and 75% accuracy on test data set. To improve the accuracy I tried many different techniques like regularization, cross-validation, stochastic gradient descent etc. The analysis of each of the techniques and effect of various hyper parameters is discussed below.

## 3. Analysis

For applying the learning algorithms I first converted the labeled data $Y$ which was in class 1 to 26 to class 0 to 25 by subtracting one from all the data. I also added a bias vector b to each sample, initially set to 1. I used different gradient descent algorithm for finding parameters $\theta$. The parameters were all taken unmodified from the input data. The accuracy of estimation on both training and testing is dependent on the learning rate along with the method used. For smaller learning rate higher number of iterations are required and vice versa.

### 3.1 Regularization

To improve the accuracy I also tried regularization by adding weighted decay term to the original cost function. The decay parameter is $\lambda$ and the new cost function is given by

$$J(\theta) = \frac{-\sum_{i=1}^{m}\sum_{j=1}^{k}(1(y^{(i)}=j)log(p(y^{(i)}=j|x^{(i)};\theta)}{m} + \lambda\sum_{i=1}^{p}\sum_{j=1}^{k}\theta_{ij}^2$$

where m is the number of samples, p is number of input parameters and k is number of output classes.

This gives the new update function as

$$\theta_j = \theta_j - \frac{-\alpha\sum_{i=1}^{m}[x^{(i)}(1(y^{(i)}=j) - p(y^{(i)}=j|x^{(i)};\theta))]}{m} - 2\alpha\lambda\theta_j$$

The effect of variation of $\lambda$ is shown in below table

| Lambda | Training Accuracy(%) |
|---|---|
| 0.0001 | 74.14 |
| 0.004 | 77.04 |
| 0.005 | 77.04 |
| 0.05 | 75.14 |
| 0.1 | 73.14 |

Figure 1: $\lambda$ vs Training Accuracy

### 3.2 Gradient Descent Algorithm

Firstly I applied the normal gradient descent algorithm for different learning rates and and found the accuracy for each case. But since the sample size is very large, I applied the normal gradient descent algorithm along with the stochastic gradient descent method. Below is the plot showing variation of cost function with iteration for same values of hyerparameters for the three methods. Here, alpha=15, lambda= 0.004, number of iterations= , batch size=1 and 15000

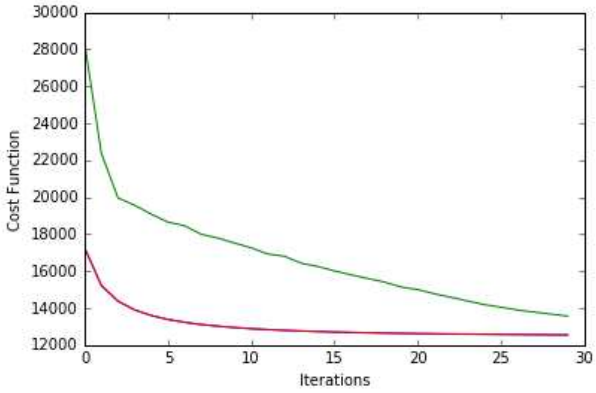


Figure 2: Cost vs Iteration for normal (green) and stochastic(red) gradient descent

As we can see above stochastic gradient descent converges faster and smoothly than normal gradient descent since it updates the cost function after each example instead of waiting for all the examples.

For very high learning rate the cost function does not converge and for very low learning rate it converges very slowly. Below is the cost vs iterations graph for $\alpha = 15$, $\lambda$=0.004.

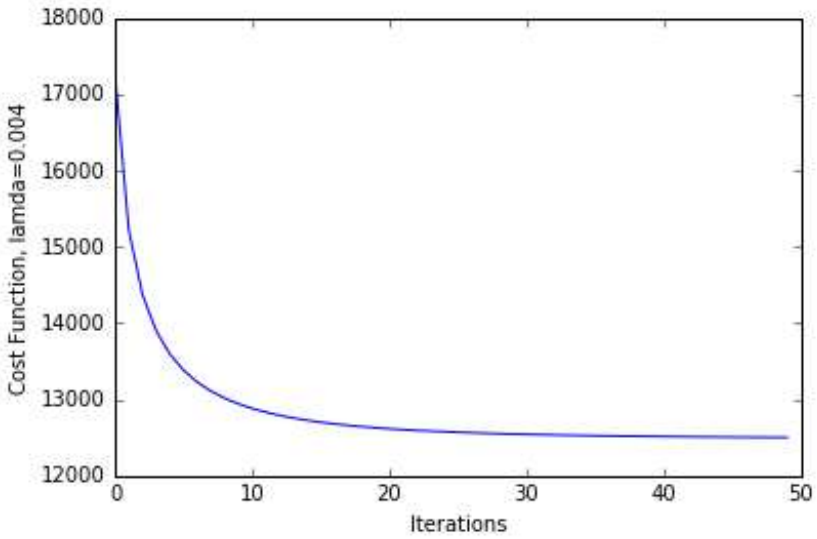


Figure 3: Cost vs Iterations for $\alpha = 15$ $\lambda$=0.004

### 3.3 Gradient Descent Algorithm with Cross validation

For increasing the accuracy of my model I tried the technique of cross validation. Cross validation is a better evaluation model than normal method as it allows gives an indication of how well the algorithm will do when it has to make new predictions for data it has not already seen. K-fold cross validation is another way to improve accuracy. The data set is

divided into k subsets. In each iteration, one of the k subsets is used as the test set and the other k-1 subsets are concatenated to form a training set. Then the accuracy across all k trials is computed. Those $\theta$ were chosen which gave highest accuracy on the test set.

This method has one serious advantage that every data point gets to be in a test set exactly once, and gets to be in a training set k-1 times. So the variance of the resulting estimate is reduced as k is increased. But since it it has to run k times all over again it is computationally very expensive.

For this algorithm we have hyper parameters $\alpha$, number of iterations and k -number of subsets in which data is divided. The hyper parameters $\alpha$, number of iterations were fixed as before (14, 50) and k was varied.

| Value of K | Training Accuracy |
|---|---|
| 3 | 77.04 |
| 4 | 76.95 |
| 5 | 77.1 |

Figure 4: Training Accuracy vs k

As we can see from the above table, there was not any significant effect on the accuracy of prediction by changing the number of subsets.

## Acknowledgments

I would like to acknowledge the support of Prof. Jayadeva, Department of Electrical Engineering IIT Delhi and help of various online resources for not only helping me in understanding softmax regression, but also learning implementation of its algorithm as a python code.

## Appendix

### CODE

```
'''Softmax Regression Assignment'''

"""
Assignment 4
"""
import numpy as np
from numpy import *
from pylab import plot, show, xlabel, ylabel, scatter

def hypothesis (theta_m,x):
    m= int (theta_m.shape[0]); #k
    hypo=zeros(shape=(m,1))
    for i in range(m):
        hypo[i]=x.dot(theta_m[i].T)
    ll=np.amax(hypo)
    gypo=np.exp(hypo-ll)
    return gypo,np.sum(gypo,axis=0)

def indic (y,k):
    if y==k :
        return 1
    else :
        return 0

def gradbatch (x_m,y_m,theta_1,alpha,num_iter,num2,lamb):
    num = int (x_m.shape[0]) #rows or no. of samples
    sz = int (x_m.shape[1]) #no. of input parameters
    m= int (theta_1.shape[0]) #no. of classes in output

    J_hist=zeros(shape=(num_iter,1))
    z_m=zeros(shape=(num,sz+1))
    z_m[:,0]=y_m[:,0]
    z_m[:,1:]=x_m[:,:]

    aa=np.split(z_m,num2,axis=0)
    theta_mn=theta_1

    J_grad= zeros(shape=(m,sz))
    for z in range(num_iter):
        cost1=0
        theta_m=theta_mn
        for epoch in aa:
```

```
            x=epoch[:,1:]
            y=epoch[:,0]
            ssz= int(epoch.shape[0])

            for i in range(ssz):
                hypo, norm = hypothesis(theta_m,x[i])
                for k in range(m):
                    J_grad[k]+= x[i]*(hypo[k]/norm-indic(y[i],k))
            theta_mn=theta_m-alpha*(J_grad/num+lamb*2.0*theta_m)
            theta_m=theta_mn

        for i in range(num):
            hypo, norm = hypothesis(theta_mn,x_m[i])
            for k in range(m):
                cost1+= indic(y_m[i],k)*(-log(hypo[k])+log(norm))
        J_hist[z]=cost1+lamb*np.sum(np.square(theta_mn))

        if (J_hist[z]-J_hist[z-2]>100):
            if (z>2):
                break

    return J_hist,theta_m

def crossval1 (x_m,y_m,theta_1,alpha,lamda,num_iter,numepoch,numcross):
    num = int (x_m.shape[0]) #rows or no. of samples
    sz = int (x_m.shape[1]) #no. of input parameters
    m= int (theta_1.shape[0]) #no. of classes in output

    J_hist=zeros(shape=(0,1))
    z_m=zeros(shape=(num,sz+1))
    z_m[:,0]=y_m[:,0]
    z_m[:,1:]=x_m[:,:]

    theta=[]
    aa=np.split(z_m,numcross,axis=0)

    out=zeros(shape=(numcross,1))
    for ll in range (aa.__len__()):
        aa1=zeros(shape=(0,sz+1))
        for jj in range(aa.__len__()):
            if (jj==ll):
                jj+=1
            else:
                aa1=np.concatenate((aa1,aa[jj]),axis=0)
        x= aa1[:,1:]
```

```
        y=aa1[:,0]
        J,theta_m=gradstoc(x,y,theta_1,alpha,num_iter,numepoch,lamda)
        out[ll]=findacc(aa[ll],theta_m)
        print ll,out[ll]
        theta.append(theta_m)
        J_hist=np.concatenate((J_hist,J),axis=0)

    print out
    jk=np.argmax(out)
    print out[jk]*numcross/15000
    return J_hist,theta[jk]


def predict (x_m,theta_m,strg):
    num = int (x_m.shape[0]) #rows or no. of samples
    out=zeros(shape=(num,1))
    for i in range(num):
        hypo, norm = hypothesis(theta_m,x_m[i])
        l=np.argmax(hypo/norm)
        out[i]=l+1

    np.savetxt(strg,out,delimiter=';')
    return out

def findacc (aa_m,theta_m):
    num = int (aa_m.shape[0]) #rows or no. of samples
    x_m=aa_m[:,1:]
    y_m=aa_m[:,0]
    out=zeros(shape=(num,1))
    acc=0
    for i in range(num):
        hypo, norm = hypothesis(theta_m,x_m[i])
        l=np.argmax(hypo/norm)
        out[i]=l
        if (y_m[i]-l==0):
            acc+=1
    return acc

def gradnorm (x_m,y_m,theta_1,alpha,num_iter,num2,lamb):
    num = int (x_m.shape[0]) #rows or no. of samples
    sz = int (x_m.shape[1]) #no. of input parameters
    m= int (theta_1.shape[0]) #no. of classes in output

    J_hist=zeros(shape=(num_iter,1))
    theta_mn=theta_1
```

```
    J_grad= zeros(shape=(m, sz))
    for z in range(num_iter):

        cost1=0
        theta_m=theta_mn
        for i in range(num):
            hypo, norm = hypothesis(theta_m,x_m[i])
            for k in range(m):
                J_grad[k]+= x_m[i]*(hypo[k]/norm-indic(y_m[i],k))
                cost1+= indic(y_m[i],k)*(-log(hypo[k])+log(norm))

        theta_mn=theta_m-alpha*(J_grad/num+lamb*2.0*theta_m)
        theta_m=theta_mn

        J_hist[z]=cost1+lamb*np.sum(np.square(theta_mn))

    return J_hist,theta_m

def gradstoc (x_m,y_m,theta_1,alpha,num_iter,lamb):
    num = int (x_m.shape[0]) #rows or no. of samples
    sz = int (x_m.shape[1]) #no. of input parameters
    m= int (theta_1.shape[0]) #no. of classes in output

    J_hist=zeros(shape=(num_iter,1))
    theta_mn=theta_1

    J_grad= zeros(shape=(m, sz))
    for z in range(num_iter):

        cost1=0
        theta_m=theta_mn
        for i in range(num):
            hypo, norm = hypothesis(theta_m,x_m[i])
            for k in range(m):
                J_grad[k]+= x_m[i]*(hypo[k]/norm-indic(y_m[i],k))

            theta_mn=theta_m-alpha*(J_grad/num+lamb*2.0*theta_m)
            theta_m=theta_mn

        for i in range(num):
            hypo, norm = hypothesis(theta_mn,x_m[i])
            for k in range(m):
                cost1+= indic(y_m[i],k)*(-log(hypo[k])+log(norm))
        J_hist[z]=cost1+lamb*np.sum(np.square(theta_mn))
```

```
    return J_hist,theta_m

'''Plot cost vs iteration graph'''
def plot_cost(iterations,alpha, J):
    plot(arange(iterations), J[:,0])
    xlabel('Iterations')
    ylabel('Cost Function, lamda='+str(alpha))
    show()


'''Import files'''
data_train = genfromtxt('train_data.csv', delimiter=',')
data_label = genfromtxt('train_label.csv', delimiter=',')


m=data_label.size #number of samples, data_train.shape[0]
n=int(data_train.shape[1]+1) #total no. of features
data_label.shape=(m,1)

'''X and Y defined'''
#converting Y data in class 0 and 1 by subtracting 1 from all samples
Y=data_label-1
X=ones(shape=(m,n))


X[:,1:]=data_train[:,:]


'''Hyper parameters'''
k=26
theta_begin=ones(shape=(k,n))
alpha=15
num_iter=50
lamda=0.004

numepoch=15000
numcross=5

str3='train_data.csv'
str4='Predicted Test Labels.csv'

J_hist, theta=gradstoc(X,Y,theta_begin,alpha,num_iter,lamda)

#np.savetxt('theta.txt',theta,delimiter=';')
```

```
'''bias vector is first column of theta matrix and weight matrix is transpose of

bias=theta[:,0].T
np.savetxt('bias.txt',bias,delimiter=';')


weight=theta[:,1:].T
np.savetxt('weight.txt',weight,delimiter=';')
```