CS 615 – Deep Learning

Artificial Neurons

Slides adapted from material created by E. Alpaydin
Prof. Mordohai, Prof. Greenstadt, Pattern Classification (2nd Ed.),
Pattern Recognition and Machine Learning

Objectives

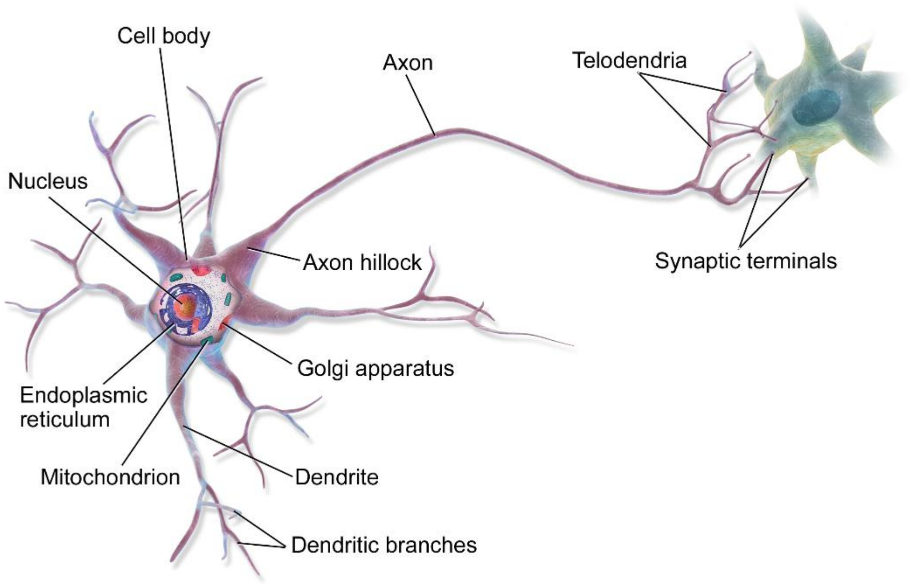
- Artificial Neurons
- Evaluating a classifier
- Logistic activation function
- Log likelihood objective function
- Softmax activation function with cross entropy objective function

Artificial Neurons

- In the previous material (on **linear regression**) we discussed the idea of estimating an output value as $\hat{\mathrm{y}} = g(x) = x\theta$ where θ is a vector of parameters.
- And to learn the values of these parameters, we can start with an initial guess, and iteratively update them based on the gradient of the least squared error.
- But what if the output was a binary value?
  - Or a probability?
- Then we could use it for binary classification!

Artificial Neurons

- A simple "network" that takes the weighed sum of the inputs, $x\theta$ and applies some function to it is called an *artificial neuron*
- These are the building blocks of artificial and deep networks.



Artificial Neurons

- Technically a linear regression module is an artificial neuron.
- The function that processes $x\theta$ is referred to as the activation function
- For linear regression this was the linear activation function:
$$g(x) = x\theta$$
- At this time we're also going to move to the notation $g(x|\theta)$ for reasons seen later.
- A non-linear activation function is the threshold/step function:
$$g(x) = x\theta > t$$
- An artificial neuron that uses this is referred to as a *perceptron.*

Artificial Neurons for Classification

- Now we want to use artificial neurons for classification.
- In doing so we'll have several decisions:
  1. What is our activation function?
  2. What is our objective function?
- Both of these give rise to a particular gradient rule used to learn the model's parameters.
- Let's look at a few common combinations….

Finding the Weights

- Let's start with the squared error objective function.
- Recall that given some function $g(x|\theta)$, **for a single observation**, (x, y), we can generalize the squared error as:

$$J = (y - \hat{y})^2 = \big(y - g(x|\theta)\big)^2$$

- The gradient of this, with respect to a single parameter θ_i, is then:

$$\frac{\partial J}{\partial \theta_i} = -2\left(\frac{\partial}{\partial \theta_i} g(x|\theta)\right)\big(y - g(x|\theta)\big)$$

Gradient of SE

$$\frac{\partial J}{\partial \theta_i} = -2\left(\frac{\partial}{\partial \theta_i} g(x|\theta)\right)\left(y - g(x|\theta)\right)$$

- So we obviously need to find the partial derivative of g with respect to θ_i
    - Note: With linear regression, $g(x|\theta) = x\theta$ and therefore $\frac{\partial}{\partial \theta_i} g(x|\theta) = x_i$
    - Which results in a gradient of $\frac{\partial J}{\partial \theta_i} = -2x_i(y - x\theta)$
- So what is $\frac{\partial}{\partial \theta_i} g(x|\theta)$ when $g(x|\theta) = x\theta > 0$?

Update Rule

$$\frac{\partial J}{\partial \theta_i} = -2\left(\frac{\partial}{\partial \theta_i} g(x|\theta)\right)\left(y - g(x|\theta)\right)$$

- So what is $\frac{\partial}{\partial \theta_i} g(x|\theta)$ when $g(x|\theta) = x\theta > 0$?
- There's basically two cases:
  - Case 1: When $x\theta > 0$
    - $\frac{\partial}{\partial \theta_i} g(x|\theta) = x_i$
  - Case 2: When $x\theta \leq 0$
    - $\frac{\partial}{\partial \theta_i} g(x|\theta) = 0$
- Therefore:

$$\frac{\partial}{\partial \theta_i}(x\theta > 0) = \begin{cases} x_i & if\ x\theta > 0 \\ 0 & otherwise \end{cases}$$

Update Rule

$$\boxed{\frac{\partial}{\partial \theta_i}(x\theta > 0) = \begin{cases} x_i & if\ x\theta > 0 \\ 0 & otherwise \end{cases}}$$

- Plugging this back into $\frac{\partial J}{\partial \theta_i} = -2\left(\frac{\partial}{\partial \theta_i} g(x|\theta)\right)(y - g(x|\theta))$ we get our gradient update rule for SE with perceptrons as:

$$\frac{\partial J}{\partial \theta_i} = \begin{cases} -2x_i(y-1) & if\ x\theta > 0 \\ \cancel{-2x_i y} & \cancel{otherwise} \end{cases} \quad 0$$

- Which we can also write as:

$$\frac{\partial J}{\partial \theta_i} = \begin{cases} 2x_i(1-y) & if\ x\theta > 0 \\ \cancel{-2x_i y}\ 0 & otherwise \end{cases}$$

- We can vectorize the parameters:

$$\xcancel{\frac{\partial J}{\partial \theta} = \begin{cases} 2x^T(1-y) & if\ x\theta > 0 \\ -2x^T y & otherwise \end{cases}}$$

Logistic Activation Function

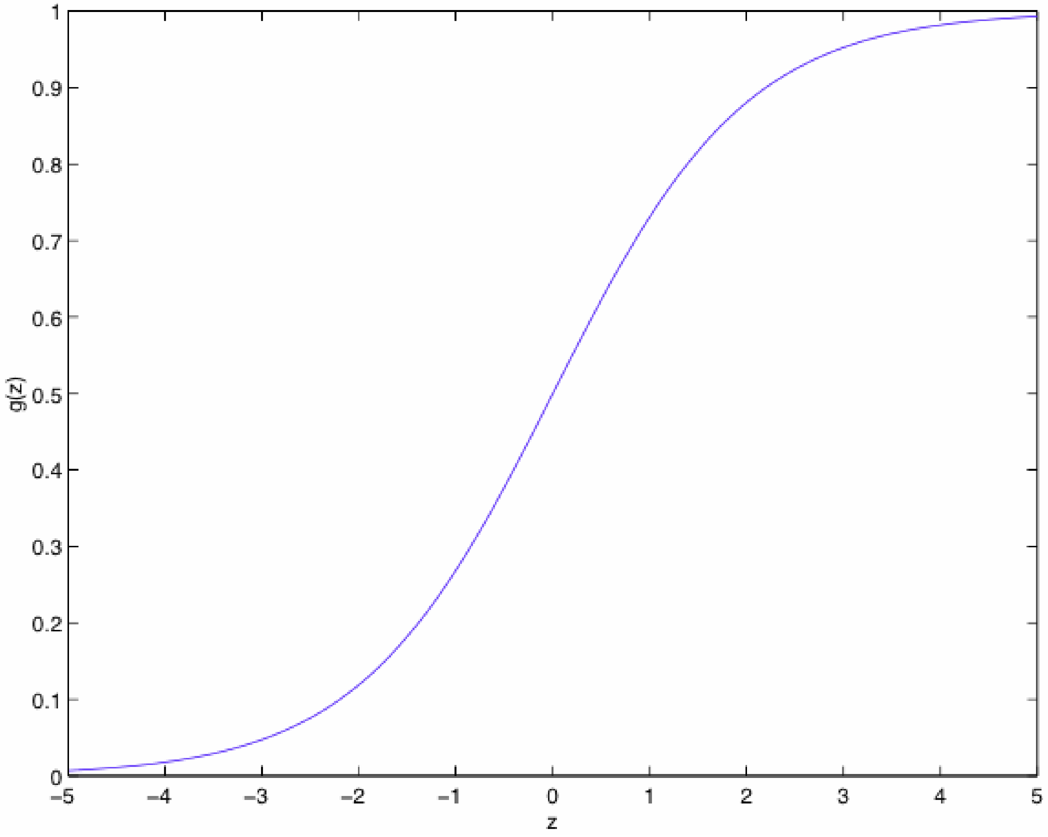
- Let’s take a look at another activation function.
- An activation function this is non-linear, but differentiable, is the *logistic function* (sometimes known as the sigmoid or logit function)*:*

$$g(x|\theta) = \frac{1}{1 + e^{-x\theta}}$$

Logistic Function

- The logistic function, defined as $\boldsymbol{g(z) = \frac{1}{1+e^{-z}}}$ tends towards 0 as z decreases and tends towards 1 as z increases
- For our purposes $z = x\theta$ and therefore our activation function is:

$$g(x|\theta) = \frac{1}{1 + e^{-x\theta}}$$



Logistic Function

- Let’s see how we can use the logistic function, with the least squared error objective function, to define our gradient rule!
- Recall for the LSE:

$$J = (y - \hat{y})^2 = \big(y - g(x|\theta)\big)^2$$

- And the gradient of this is:

$$\frac{\partial J}{\partial \theta_i} = -2\left(\frac{\partial}{\partial \theta_i} g(x|\theta)\right)\big(y - g(x|\theta)\big)$$

- What is $\frac{\partial}{\partial \theta_i} g(x|\theta)$ for the logistic activation function?

Derivative of Logistic Function

- $\frac{\partial}{\partial\theta_i} g(x|\theta) = \frac{\partial}{\partial\theta_i}\left(\frac{1}{1+e^{-x\theta}}\right) = \frac{\partial}{\partial\theta_i}\left(1+e^{-x\theta}\right)^{-1}$
- $= -1\left(0 - x_i e^{-x\theta}\right)\left(1+e^{-x\theta}\right)^{-2} = \frac{x_i e^{-x\theta}}{\left(1+e^{-x\theta}\right)^2}$
- $= x_i \frac{1}{1+e^{-x\theta}} \frac{e^{-x\theta}}{1+e^{-x\theta}}$
- $= x_i g(x|\theta)\left(1 - g(x|\theta)\right)$

Gradient Rule w/ Logistic Function

- So when $g(x|\theta)$ is the logistic function, its partial derivative is:

$$\frac{\partial}{\partial \theta_i} g(x|\theta) = x_i g(x|\theta)\big(1 - g(x|\theta)\big)$$

- Plugging this back into $\frac{\partial J}{\partial \theta_i} = -2\left(\frac{\partial}{\partial \theta_i} g(x|\theta)\right)\big(y - g(x|\theta)\big)$ we get our gradient update rule for the SE of a perceptrons with a logistic activation function to be:

$$\begin{aligned}\frac{\partial J}{\partial \theta_i} &= -2\left(x_i g(x|\theta)\big(1 - g(x|\theta)\big)\right)\big(y - g(x|\theta)\big) \\ &= -2x_i\hat{y}(1 - \hat{y})(y - \hat{y})\end{aligned}$$

Gradient Rule w/ Logistic Function

$$\frac{\partial J}{\partial \theta_i} = -2x_i\hat{y}(1-\hat{y})(y-\hat{y})$$

- Now let's try to vectorize!
- Vectorizing over all parameters is now:

$$\frac{\partial J}{\partial \theta} = -2x^T\hat{y}(1-\hat{y})(y-\hat{y})$$

- And our batch gradient update is:

$$\frac{\partial J}{\partial \theta} = -\frac{2}{N}X^T\hat{Y}\left(1-\hat{Y}\right)^T(Y-\hat{Y})$$

Gradient Rule w/ Logistic Function

$$\frac{\partial J}{\partial \theta} = -\frac{2}{N} X^T \hat{Y}\left(1-\hat{Y}\right)^T \left(Y-\hat{Y}\right)$$

- Ok, at least this isn’t piecewise/conditional!
- But it’s not too pretty.
- There are other objective functions that are (often) better suited for classification.
- One such function is called the *log likelihood*

Log Likelihood

- Given a observation (x, y), we can compute the **likelihood** that we are correct as
$$\ell = (\hat{y})^y (1-\hat{y})^{(1-y)} = \left(g(x|\theta)\right)^y \left(1-g(x|\theta)\right)^{(1-y)}$$
- Our technique will of course be to find the gradient of this with respect to one of the parameters.
- However, this can be made easier if we first take the log of this.
- Recall
  - The log of a product is the sum of its logs: $ln(mn) = ln(m) + ln(n)$
  - The log of an exponent is the exponent times the log of its base: $ln(a^x) = xln(a)$
- Therefore our log likelihood objective function is:
$$J = ln(\ell) = y\, ln\left(g(x|\theta)\right) + (1-y)\, ln\left(1-g(x|\theta)\right)$$

To Maximum Likelihood

$$\boxed{J = y\,ln\big(g(x|\theta)\big) + (1-y)\,ln\big(1-g(x|\theta)\big)}$$

- First off, a reminder…

$$\frac{\partial}{\partial x}(ln\,x) = \frac{1}{x}\cdot\frac{\partial}{\partial x}(x)$$

- Therefore

$$\frac{\partial J}{\partial \theta_i} = \left(\frac{\partial}{\partial \theta_i} g(x|\theta)\right)\frac{y}{g(x|\theta)} + \left(\frac{\partial}{\partial \theta_i}\big(1-g(x|\theta)\big)\right)\frac{(1-y)}{1-g(x|\theta)}$$

- And of course $\frac{\partial}{\partial \theta_i} g(x|\theta)$ depends on what our activation function $g(x|\theta)$ is.

To Maximum Likelihood

$$\boxed{\frac{\partial J}{\partial \theta_i} = \left(\frac{\partial}{\partial \theta_i} g(x|\theta)\right)\frac{y}{g(x|\theta)} + \left(\frac{\partial}{\partial \theta_i}\big(1 - g(x|\theta)\big)\right)\frac{(1-y)}{1-g(x|\theta)}}$$

- If we choose the logistic function to be our activation function, its partial derivative is:

$$\frac{\partial}{\partial \theta_i} g(x|\theta) = x_i g(x|\theta)\big(1 - g(x|\theta)\big)$$

- And therefore our gradient is just:

$$\frac{\partial J}{\partial \theta_i} = x_i\big(y - g(x|\theta)\big) = x_i(y - \hat{y})$$

To Maximum Likelihood

$$\boxed{\frac{\partial J}{\partial \theta_i} = x_i(y - \hat{y})}$$

- Vectorizing this for all parameters we have

$$\frac{\partial J}{\partial \theta} = x^T(y - \hat{y})$$

- And our batch gradient is just:

$$\frac{\partial J}{\partial \theta} = \frac{1}{N} X^T\left(Y - \hat{Y}\right)$$

- What we just derived (maximizing the log likelihood based on the output of a logistic function) is called *logistic regression*
  - *Which is confusing since it's actually used for classification, not regression.*
  - Nevertheless…

Gradient Ascent Rule

- One last thing!
- Since we want to *maximize* the log likelihood, we want to ***add*** some amount of the gradient to the parameters.

$$\theta \coloneqq \theta + \eta \frac{\partial J}{\partial \theta}$$

Evaluation

- How do we evaluate when we're doing classification?
- I suppose we could use the RMSE, but that may make less sense.
- We could just count how often we predict the correct class
  - We call this *accuracy*.
  - Let Y_i be the true class, and $\widehat{Y_i}$ be the predicted class for observation i.

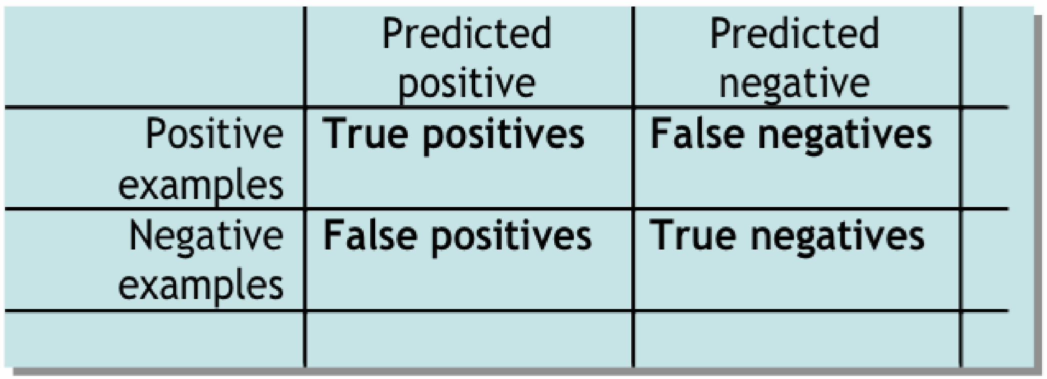
$$accuracy = \frac{1}{N}\sum_{i=1}^{N}\left(Y_i == \widehat{Y_i}\right)$$

Class Priors

- It’s also important to have some sort of baseline accuracy.
- The *class prior* is the likelihood (probability) of a class.
- So for our baseline, we can just use the highest of the class priors.
  - Since naively assigning all test samples that class, would result in that accuracy.

Binary Classification Error Types

- If we're doing binary classification (just two classes), then there's some additional evaluations we can do.
- In binary classification, often we're focused on attempting to "find" one particular class.
  - We refer to this as the positive class.
  - The other data is called the negative class.
- From this we can describe four different possibilities:
  - True positive = Hit
  - True negative = Correct rejection
  - False positive = False Alarm (Type 1 error)
  - False negative = Miss (Type 2 error)



	Predicted positive	Predicted negative	
Positive examples	**True positives**	**False negatives**	
Negative examples	**False positives**	**True negatives**	

Evaluating your Classifier

- From the four error types, we can establish some binary-classification-specific measurements:
- *Precision* – percentage of things that were classified as positive and actually were positive
  - $Precision \ = \frac{TP}{TP+FP}$
- *Recall* – the percentage of true positives (*sensitivity*) correctly identified
  - $Recall \ = \frac{TP}{TP+FN}$
- *f-measure* – The weighted harmonic mean of precision and recall
  - $F_1 = \frac{2*precision*recall}{precision+recall}$

Evaluating your Classifier

- Related to this, we sometimes will look at the true positive rate vs the false positive rate
  - The true positive rate is

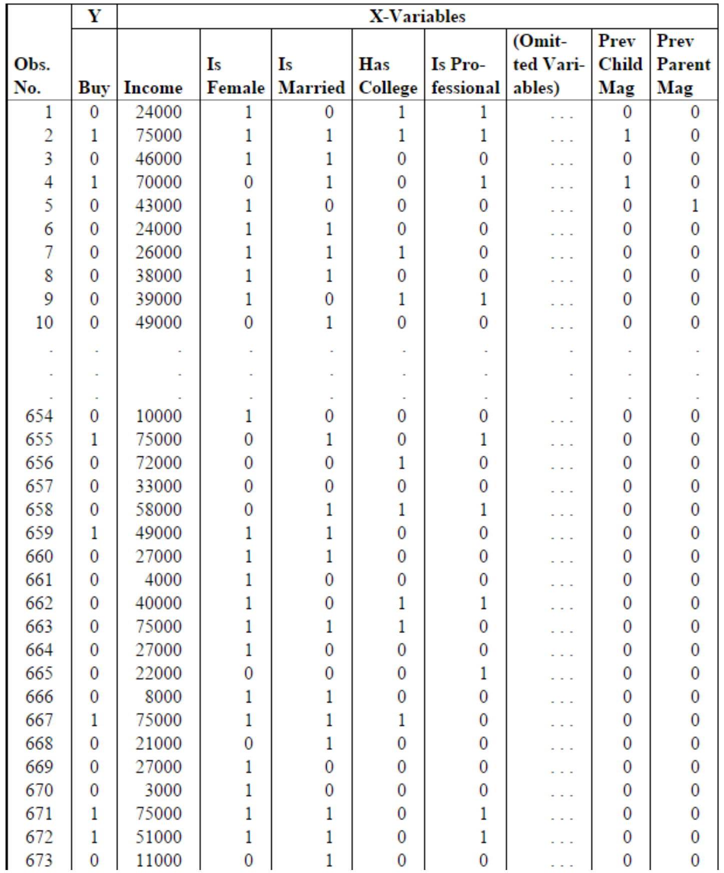
$$TPR = Recall = \frac{TP}{TP + FN}$$

  - The false positive rate is

$$FPR = \frac{FP}{FP + TN}$$

Logistic Regression Example

- Let’s classifying whether a person will buy a product or not



	Y	X-Variables							
Obs. No.	Buy	Income	Is Female	Is Married	Has College	Is Professional	(Omitted Variables)	Prev Child Mag	Prev Parent Mag
1	0	24000	1	0	1	1	. . .	0	0
2	1	75000	1	1	1	1	. . .	1	0
3	0	46000	1	1	0	0	. . .	0	0
4	1	70000	0	1	0	1	. . .	1	0
5	0	43000	1	0	0	0	. . .	0	1
6	0	24000	1	1	0	0	. . .	0	0
7	0	26000	1	1	1	0	. . .	0	0
8	0	38000	1	1	0	0	. . .	0	0
9	0	39000	1	0	1	1	. . .	0	0
10	0	49000	0	1	0	0	. . .	0	0
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
654	0	10000	1	0	0	0	. . .	0	0
655	1	75000	0	1	0	1	. . .	0	0
656	0	72000	0	0	1	0	. . .	0	0
657	0	33000	0	0	0	0	. . .	0	0
658	0	58000	0	1	1	1	. . .	0	0
659	1	49000	1	1	0	0	. . .	0	0
660	0	27000	1	1	0	0	. . .	0	0
661	0	4000	1	0	0	0	. . .	0	0
662	0	40000	1	0	1	1	. . .	0	0
663	0	75000	1	1	1	0	. . .	0	0
664	0	27000	1	0	0	0	. . .	0	0
665	0	22000	0	0	0	1	. . .	0	0
666	0	8000	1	1	0	0	. . .	0	0
667	1	75000	1	1	1	0	. . .	0	0
668	0	21000	0	1	0	0	. . .	0	0
669	0	27000	1	0	0	0	. . .	0	0
670	0	3000	1	0	0	0	. . .	0	0
671	1	75000	1	1	0	1	. . .	0	0
672	1	51000	1	1	0	1	. . .	0	0
673	0	11000	0	1	0	0	. . .	0	0

KidCreative.csv

Logistic Regression Example

- Make some design decisions:
  - Randomize data
  - Use 2/3 training, 1/3 testing
  - Standardize features
  - Add bias feature
  - Initialize parameters to random values in the range [-1, 1]
  - Since our equation is based on log likelihood, let's terminate when change in sum of log likelihoods doesn't change more than eps
    - Recall the log likelihood of an example being correct is

$$y\ln\left(\frac{1}{1+\mathrm{e}^{-x\theta}}\right)+(1-y)\ln\left(1-\frac{1}{1+\mathrm{e}^{-x\theta}}\right)$$

    - But be careful… $\log(0) = -Inf$. So you might need to deal with this somehow
  - Let's dynamically allow the learning parameter η to adapt!
  - Let's do batch regression

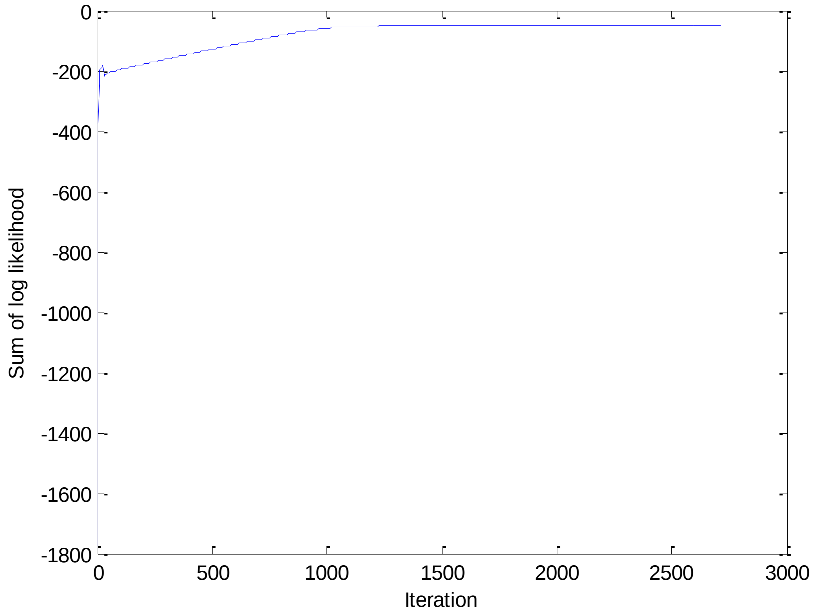
Example

θ =

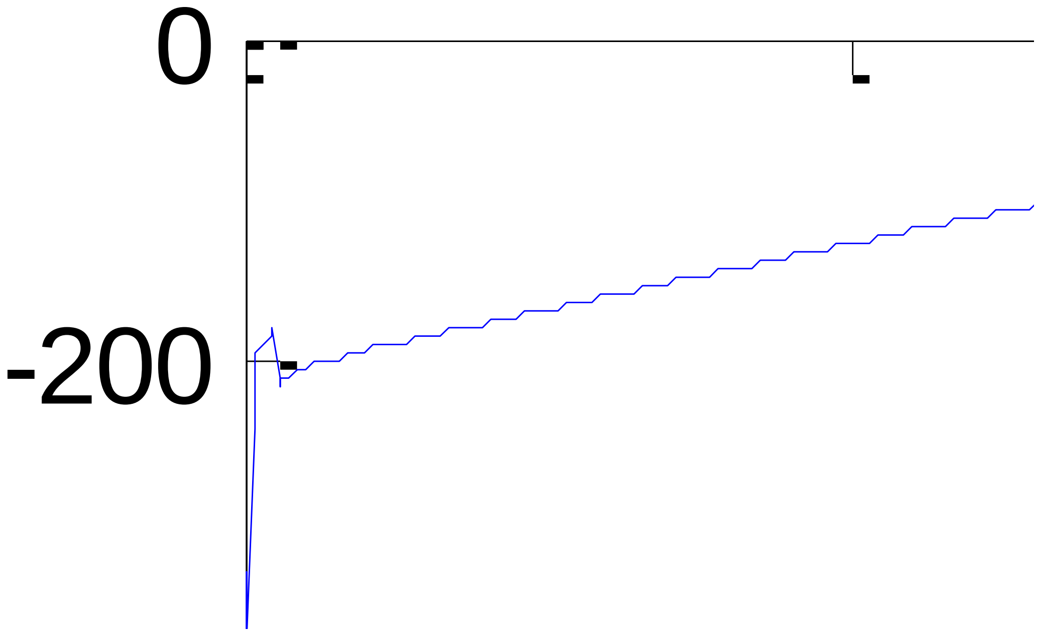
-12.7663
5.1987
0.9057
0.6136
0.0880
-0.3606
-0.4920
-29.3976
0.2140
-0.0329
0.7355
0.3957
0.0054
0.9061
1.1647
0.1887
0.1881

Choosing Class 1 if $g_\theta(x) \geq 0.5$ we get:

Precision: 0.7708
Recall: 0.7551
F-Measure: 0.7629



Example



Multiple Classes

- What if we had multiple classes to decide among?
- There are ways that we can make this decision using a bunch of binary classifiers, but let’s look at how we can do this directly.

Multi-Class MLE + Logistic Activation

- In order to do multi-class classification, there will be three main changes:
  1. Our target for each observation will now be a *binary vector*. Therefore $Y \in \mathcal{R}^{N\times K}$
  2. Likewise, our estimations will be a vector: $\hat{y} \in \mathcal{R}^{1\times K}$
  3. And finally, our parameters will now be a $D \times K$ **matrix**
- Given all of this, lets adapt our maximum likelihood with logistic activation equations and gradient rules for multi-class classification:

$$\ell_\theta(x, y) = \prod_{k=1}^{K} g(x|\theta_{:,k})^{y_k}\left(1 - g(x|\theta_{:,k})\right)^{(1-y_k)}$$

- Taking the log of this we get our objective function:

$$J = \sum_{k=1}^{K} y_k \ln\left(g(x|\theta_{:,k})\right) + (1 - y_k)\ln\left(1 - g(x|\theta_{:,k})\right)$$

Multi-Class MLE + Logistic Activation

$$\boxed{J = \sum_{k=1}^{K} y_k \ln\left(g\left(x|\theta_{:,k}\right)\right) + (1 - y_k)\ln\left(1 - g\left(x|\theta_{:,k}\right)\right)}$$

- Now we need to find the gradient with respect to each $\theta_{i,k}$, i.e $\frac{\partial J}{\partial \theta_{i,k}}$
- Fortunately, the only term of the summation that will contribute is when $k = k$.
- Therefore we have:

$$\frac{\partial J}{\partial \theta_{i,k}} = x_i\left(y_k - g\left(x|\theta_{:,k}\right)\right) = x_i(y_k - \hat{y}_k)$$

Multi-Class MLE + Logistic Activation

$$\boxed{J = \sum_{k=1}^{K} y_k \ln\left(g\left(x|\theta_{:,k}\right)\right) + (1 - y_k)\ln\left(1 - g\left(x|\theta_{:,k}\right)\right)}$$

$$\frac{\partial J}{\partial \theta_{i,k}} = x_i(y_k - \hat{y}_k)$$

- Vectorizing this for all i, k we get the matrix:

$$\frac{\partial J}{\partial \theta} = x^T(y - \hat{y})$$

- And the batch version would be

$$\frac{\partial J}{\partial \theta} = \frac{1}{N} X^T\left(Y - \hat{Y}\right)$$

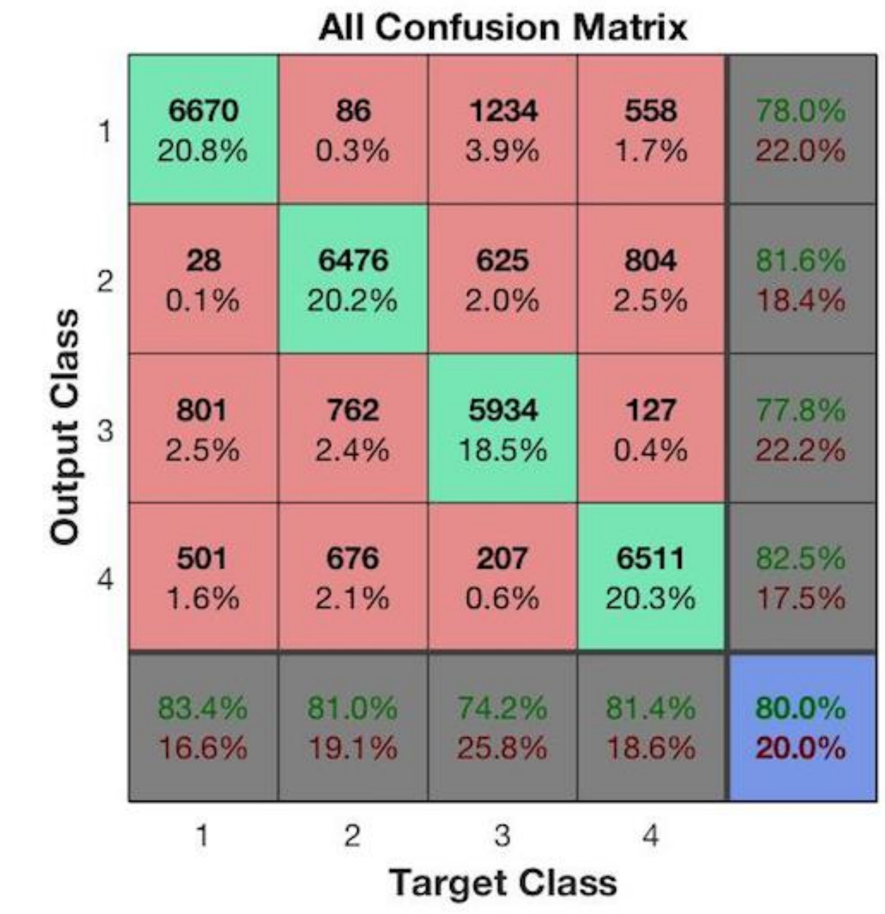
Multi-Class Evaluation

- Just like binary classification, we can evaluate the accuracy of a multi-class classifier:

$$accuracy = \frac{1}{N}\sum_{i=1}^{N}(Y_i == \widehat{Y_i})$$

- In addition, particular to multi-class classification, we may be interested in investigating which classes get confused with which other classes
- To observe this, we can look at a *confusion matrix*

Confusion Matrix

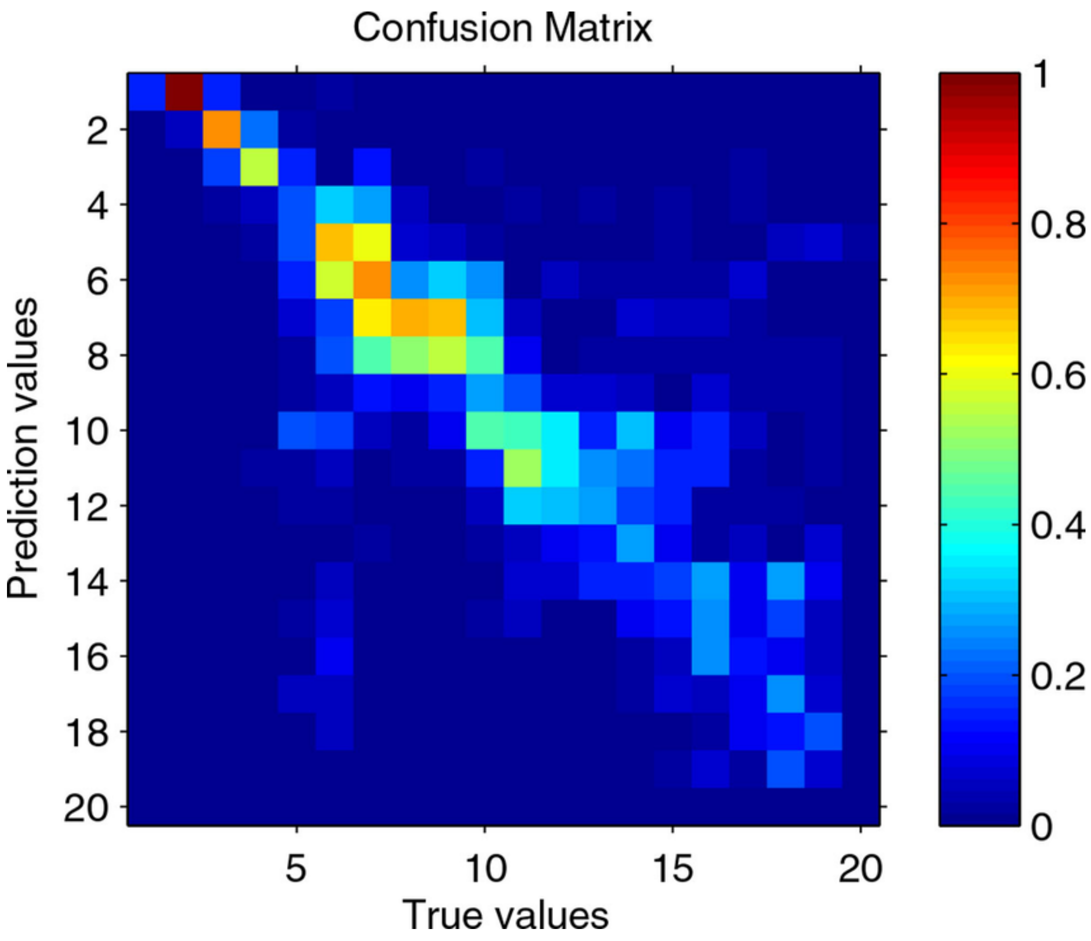


All Confusion Matrix

	1	2	3	4	
1	6670 20.8%	86 0.3%	1234 3.9%	558 1.7%	78.0% 22.0%
2	28 0.1%	6476 20.2%	625 2.0%	804 2.5%	81.6% 18.4%
3	801 2.5%	762 2.4%	5934 18.5%	127 0.4%	77.8% 22.2%
4	501 1.6%	676 2.1%	207 0.6%	6511 20.3%	82.5% 17.5%
	83.4% 16.6%	81.0% 19.1%	74.2% 25.8%	81.4% 18.6%	80.0% 20.0%

Output Class
Target Class

Confusion Matrix



Additional Objective Functions

- Another common objective function for multi-class classification is *cross entropy*
- So let’s take a look at that.
- But to do this, we must first learn about the *softmax* activation function.

Softmax

- It is sometimes advantageous to be able to think of the output values as class *probabilities.*
- To be a valid distribution, we want these outputs to sum to one.
- The softmax function does this!
- The softmax function gives us back the probability of an observation belonging to class j, which we'll indicate as $\hat{y}_j$.
- This is computed as:

$$P(y=j)=\hat{y}_j=\frac{e^{x\theta_{:,j}}}{\sum_{k=1}^{K} e^{x\theta_{:,k}}}$$

Entropy

- Entropy measure the randomness in a system.
- Given a probability distribution $p = (p_1, p_2, \ldots p_K)$ such that $\sum_{k=1}^{K} p_k = 1$, the entropy is computed as:

$$H = \sum_{k=1}^{K} -p_k \ln p_k$$

Cross Entropy Loss

- If we have two distributions, a, b and want to compare them, we can use *cross entropy:*

$$H = \sum_{k=1}^{K} -a_k \ln b_k$$

- If we consider our target outputs to be and our output is from the softmax activation function (or really any activation function the provides a valid output probability), then we can compute the ***cross-entropy loss objective function*** :

$$J = -\sum_{k=1}^{K} y_k \ln(\hat{y}_k)$$

Cross Entropy Loss

$$J = -\sum_{k=1}^{K} y_k \ln(\hat{y}_k)$$

- Since typically our y distribution will be all zeros, with just one location having a probability of one (say $y_a = 1$) then this simplifies to:

$$J = -\ln(\hat{y}_a)$$

Gradient of Softmax with Cross Entropy

- Now we need to find the update rule for all the parameters $\theta_{i,j}$, i.e $\frac{\partial J}{\partial \theta_{i,j}}$, when we have a softmax function and cross entropy objective function.
- In order to do this, we need to write our cross-entropy objective function in terms of θ

$$J = -\ln(\hat{y}_a)$$

$$J = -\ln\left(\frac{e^{x\theta_{:,a}}}{\sum_{k=1}^{K} e^{x\theta_{:,k}}}\right)$$

Gradient of Cross Entropy

$$J = -\ln\left(\frac{e^{x\theta_{:,a}}}{\sum_{k=1}^{K} e^{x\theta_{:,k}}}\right)$$

- Now we can take the partial gradient of this with respect to a parameters $\theta_{i,j}$:

$$\frac{\partial J}{\partial \theta_{i,j}} = -\frac{\sum_{k=1}^{K} e^{x\theta_{:,k}}}{e^{x\theta_{:,a}}}\left(\frac{\partial}{\partial \theta_{i,j}}\left(\frac{e^{x\theta_{:,a}}}{\sum_{k=1}^{K} e^{x\theta_{:,k}}}\right)\right)$$

- Now again there will be two cases

Gradient of Cross Entropy

$$\frac{\partial J}{\partial \theta_{i,j}} = -\frac{\sum_{k=1}^{K} e^{x\theta_{:,k}}}{e^{x\theta_{:,a}}}\left(\frac{\partial}{\partial \theta_{i,j}}\left(\frac{e^{x\theta_{:,a}}}{\sum_{k=1}^{K} e^{x\theta_{:,k}}}\right)\right)$$

- Case 1: $j \neq a$

$$\frac{\partial J}{\partial \theta_{i,j}} = -\frac{\sum_{k=1}^{K} e^{x\theta_{:,k}}}{e^{x\theta_{:,a}}}\left(\frac{\left(0\sum_{k=1}^{K} e^{x\theta_{:,k}}\right) - \left(e^{x\theta_{:,a}} x_i e^{x\theta_{:,j}}\right)}{\left(\sum_{k=1}^{K} e^{x\theta_{:,k}}\right)^2}\right)$$

$$= \frac{\sum_{k=1}^{K} e^{x\theta_{:,k}}}{e^{x\theta_{:,a}}}\left(\frac{e^{x\theta_{:,a}} x_i e^{x\theta_{:,j}}}{\left(\sum_{k=1}^{K} e^{x\theta_{:,k}}\right)^2}\right) = \frac{x_i e^{x\theta_{:,j}}}{\sum_{k=1}^{K} e^{x\theta_{:,k}}} = x_i \hat{y}_j$$

Gradient of Cross Entropy

$$\frac{\partial J}{\partial \theta_{i,j}} = -\frac{\sum_{k=1}^{K} e^{x\theta_{:,k}}}{e^{x\theta_{:,a}}}\left(\frac{\partial}{\partial \theta_{i,j}}\left(\frac{e^{x\theta_{:,a}}}{\sum_{k=1}^{K} e^{x\theta_{:,k}}}\right)\right)$$

- Case 2: $j = a$

$$\frac{\partial J}{\partial \theta_{i,j}} = -\frac{\sum_{k=1}^{K} e^{x\theta_{:,k}}}{e^{x\theta_{:,a}}}\left(\frac{\left(x_i e^{x\theta_{:,j}} \sum_{k=1}^{K} e^{x\theta_{:,k}}\right) - \left(e^{x\theta_{:,j}} x_i e^{x\theta_{:,j}}\right)}{\left(\sum_{k=1}^{K} e^{x\theta_{:,k}}\right)^2}\right)$$

$$= \frac{x_i\left(e^{x\theta_{:,j}} - \sum_{k=1}^{K} e^{x\theta_{:,k}}\right)}{\sum_{k=1}^{K} e^{x\theta_{:,k}}} = x_i\left(\hat{y}_j - 1\right)$$

- Thus:

$$\boxed{\frac{\partial J}{\partial \theta_{i,j}} = \begin{cases} x_i \hat{y}_j & if\ j \neq a \\ x_i\left(\hat{y}_j - 1\right) & Otherwise \end{cases}}$$

a: The right class.

Gradient of Cross Entropy

$$\frac{\partial J}{\partial \theta_{i,j}} = \begin{cases} x_i \hat{y}_j & if\ j \neq a \\ x_i(\hat{y}_j - 1) & Otherwise \end{cases}$$

- Since our target y is a binary vector, then we can write this as:

$$\frac{\partial J}{\partial \theta_{i,j}} = x_i(\hat{y}_j - y_j)$$

- If $\hat{y}$ is the vector of predicted class probabilities, then we can vectorize this to update all the parameters at once as:

$$\frac{\partial J}{\partial \theta} = x^T(\hat{y} - y)$$

- And to do batch updates:

$$\boxed{\frac{\partial J}{\partial \theta} = X^T(\hat{Y} - Y)}$$