Hand In 1
Must be sent by mail to jallan@cs.au.dk at the latest 25/9 2014 14.00 (before class)

Version 3

Intro

In this exercise you will build classifiers for optical character recognition (OCR). The models we use can be implemented with relatively few lines of actual code if expressed using matrices and vectors (and matlab is much faster when using matrix product instead of for loops). So the key is in understanding and matrix products. In this document we go through the models in quite a lot of detail to ease the understanding. We use matrix and vector notation heavily since it simplifies the exposition so you might as well get used to it.

You can write your final report in danish if you prefer. The maximal report length is 5 pages. You are allowed to be up to 3 members in a group. You are encouraged to discuss the exercise between groups and help each other as much as possible without of course copying each others work. Particularly, discussing the quality of your classifiers is probably a good idea to get an indication if you are doing it correctly. Bonus questions may be skipped but you are encouraged to try and at least think about answering them.

There are associated data files and matlab examples for you on the website. The files are....

mnistTrain.mat a matlab file that contains the mnist digits.

mnistTest.mat a test set for mnist digits

auTrain.mat a train set with images you produced comes online when we have parsed your inputs

linregdemo.zip A zip file with examples matlab code that will help you get started.

When you hand in your report you must also upload matlab code. We will specify what we need after the description of the exercise.

Be sure discuss all the choices you have made in your implementation in your report.

Logistic Regression

In logistic regression we model the target function as a probability distribution $p(y \mid x)$. This probability distribution is implemented with the logistic function $\sigma(z) = 1/(1 + e^{-z})$ over a linear function of the input data, $\sum_{i=0}^{d} \theta_i x_i = \theta^\intercal x$, that is parameterized by the vector θ. As in the

lectures we code the bias variable into the input x and force $x_0 = 1$. The logistic function is a nice smooth function with simple derivaties, $\frac{\partial \sigma}{\partial z} = (1 - \sigma(z))\sigma(z)$, which makes it pleasing to work with. The model becomes,

$$p(y \mid x, \theta) = \begin{cases} \sigma(\theta^\intercal x) & \text{if } y = 1 \\ 1 - \sigma(\theta^\intercal x) & \text{if } y = 0 \end{cases}$$

, e.g. Probability that $y = 1$ given x, θ is the probability of getting heads with a biased coin, where the bias is $\sigma(\theta^\intercal x)$. Given a fixed θ we can use the function p to make classification by returning the most likely class, e.g.

$$\text{Given } x \text{ return 1 if } \sigma(\theta^\intercal x) \geqslant 0.5, \text{otherwise return 0,}$$

meaning we end up with a linear classifer. So the job at hand is to find a good θ and we will do that using the Maximum Likelihood method.

The input is a labeled dataset $D = (X, Y) = \{(x, y)_i \mid i = 1, \ldots, n\}$, where $x \in \{1\} \times \mathbb{R}^d$ (as stated above we use the first input dimension for bias) is a vector of length $d + 1$ and $y \in \{0, 1\}$ is a number. Think of X as a matrix where each row is an input point (vector) x, and Y as a column vector (matrix with one column) and notice that y's are not probabilities, but actual realisation of events. We assume (as always) that the points in D are independently sampled. This means that under our model, for a fixed vector of parameters θ, the likelihood of the data is

$$p(D \mid \theta) = \prod_{(x,y) \in D} p(y|x, \theta) = \prod_{(x,y) \in D} \sigma(\theta^\intercal x)^y (1 - \sigma(\theta^\intercal x))^{1-y}$$

Notice the last equality, which gives a convenient way of expressing the probability $p(y|x, \theta)$ as a product (you should convince yourself that it is true).

As i talked about in the lecture we use the likelihood as a proxy of $p(\theta|D)$ which is what we really care about.

Computing the Maximum Likelihood Parameters

We want to compute the parameters that makes the likelihood as large as possible, hence the name Maximum Likelihood, and it is defined as

$$\theta_{ml} = \arg\max_\theta p(D \mid \theta).$$

Your task is to find θ_{ml}, or at least parameters that are very close to that. To make this task simpler we minimize the negative log likelihood (NLL) of the data instead. This transforms the likelihood product into a pointwise sum which is a lot easier to handle. The negative log likelihood is

$$NLL(D \mid \theta) = -\sum_{(x,y) \in D} y \log(\sigma(\theta^\intercal x)) + (1 - y) \log(1 - \sigma(\theta^\intercal x))$$

which is called the *cross entropy* error function. Before we can start to implement an algorithm that optimizes the negativ log likelihood we need to know what we are dealing with. Basically, if a function is convex/concave we know we can optimize it efficiently. Luckily that is actually the case, but you should not take my word for it (Bonus Question 1 below).

Now that you know the negative log likelihood function is convex you know that a local minimum is a global minimum and at a local minimum the gradient vanishes. The gradient is (skipping all the derivations)

$$\nabla NLL(\theta) = \frac{\partial NLL}{\partial \theta} = \ldots = X^\intercal(Y - \sigma(X\theta)).$$

You should of course verify that this is true. Notice how we compute the summed error in one matrix product. Unfortunately, we cannot analytically solve to find a zero for this gradient (as we could for linear regression). Luckily, the function is twice differentiable and we can turn to gradient descent optimizations algorithms.

Bonus Question 1: Convexity Show that the negative log likelihood function for logistic regression is convex.

Question 2: Gradient Descent and OCR Implement gradient descent for logistic regression on apply to the digits data.

See matlab help sheet for how to read the data.

Use your logistic regression implementation to train a classifier for digit recognition (for instance 2 and 7) that when given x returns $\arg\max_y p(y \mid x, \theta)$. In this case we care about the percentage of correct classifications event if this is not the error function we optimized. What line search method have you used?

How does the weight vector θ look as the algorithm progresses and when the algorithm has terminated (plot as a 2D image)? Try different pairs of digits. Does the weight vector θ behave like you would expect?

You can now implement a full classifier for digit recognition using the one-vs all technique. How large a percentage of images do you classify correctly from the test set.

Question 3: Sanity Check What happens if we randomly permute the pixels in each image (with the same permutation) before you train the classifier. Will you get a classifier that is better,worse, or the same. Give short explaination.

Bonus Question 4: Linear Separable If the data is linearly separable what happens to weights when we implement logistic regression with gradient descent. Assume that we have full precision (ignore floating point errors). We can run gradient descent on the data set for as long as we want. Now what will happen with the weights in the limit. Do they converge to some fixed number (fluctuate around it) or do they keep decreasing or increasing.

Code Requirements

Cost Function and Gradient Matlab function named logCost in file logCost.m Takes 3 inputs, data X, target y, parameters θ. X is a $n \times (d+1)$ matrix, y is an $n \times 1$ column vector and θ is a $(d+1) \times 1$ column vector. The function must return the cost function E_{in} and the gradient ∇E_{in} when applied to data X, y with parameters θ.

Gradient Descent Algorithm Matlab function logRun in file logRun.m that takes three arguments X, y, θ as above and runs gradient descent and return the best parameters found on the data. The algorithm should be optimized for the auDigits training set.

Best parameters Found A simple matlab script named logBestParams.m, that defines two parameters logThetaMnist and logThetaAu the best parameters you have found on the two training sets.

Multinomial/Softmax Regression

In this exercise we generalize logistic regression to handle K classes instead of 2. This is known as Multinomial or Softmax regression and we will use the exact same approach as for logistic regression, it just becomes a little more technical due to the extra classes. In Softmax regression we are classifying into K classes (instead of 2 for logistic regression). We encode the target values, y, as a vector of length K with all zeros exept one which corresponds to the class. If an example belong to class 3 and there are five classes then $y = [0, 0, 1, 0, 0]$. So Y is now a matrix of size $n \times K$, (X is unchanged).

We can no longer use the logistic function as the probability function. Instead we use a generalization which is the *softmax* function. Softmax takes as input a vector of length t (think number of classes K) and outputs another vector of the same length t mapping the t numbers into t *probabilities* that sum to one. It is defined as

$$\text{softmax}(x)_j = \frac{e^{x_j}}{\sum_{i=i}^{t} e^{x_i}} \text{ for } j = 1, \ldots, t.$$

Notice that the denominator acts as a normalization term that ensures that the probabilitie sum to one. Again we get nice derivatives (something we like)

$$\frac{\partial \,\text{softmax}(x)_i}{\partial x_j} = (\delta_{i,j} - \text{softmax}(x)_j)) \,\text{softmax}(x)_i,$$

where $\delta_{i,j} = 1$ if $i = j$ and zero otherwise. As before we use a linear model for each class. The parameter set θ is represented as a $(d+1) \times K$ matrix giving $d+1$ parameters ($+1$ for the bias) for each of the K classes (parameters for class c is column c), meaning that $\theta = [\theta_1, \ldots, \theta_K]$. We get

$$p(y \mid x, \theta) = \begin{cases} \text{softmax}(\theta^\intercal x)_1 & \text{if } y = 1 \\ \ldots & \\ \text{softmax}(\theta^\intercal x)_K & \text{if } y = K \end{cases}.$$

Think of the probability distribution over y as throwing a K-sided die where the probability of landing on each of the K sides is stored in the vector $\theta^\intercal x$ (which is a vector of length K) and the probability of landing on side i is $\text{softmax}(\theta^\intercal x)_i$.

As for logistic regression we compute the likelihood of the data given a fixed matrix of parameters. We use the notation $[z]$ for the indicator function e.g. $[z]$ is one if z is true.

$$P(D \mid \theta) = \prod_{(x,y) \in D} \prod_{j=1}^{K} \text{softmax}(\theta^\intercal x)_j^{[y_i = 1]}$$

This way of expressing is the same as we did for logistic expression. The product over the K classes will have one element that is different than one namely the y_i'th element (y is a vector of $K-1$ zeros and one 1). The remaining probabilities are raised to a power of zero and has the value one.

For convenience we minimize the negative log likelihood of the data instead of maximizing the likelihood of the data and get a pointwise sum.

$$NLL(D \mid \theta) = -\sum_{(x,y)\in D}\sum_{j=1}^{K}[y_i = 1]\log(\text{softmax}(\theta^\intercal x)_j)$$

As for logistic regression this is a convex function but we cannot solve for a zero analytically and turn to gradient methods. To use gradient descent as before all you really need is the gradient of the negative log likelihood function. This gradient is a *simple* generalization of the one for logistic regression since we now have a set of parameters for each class θ_j for $j = 1, \ldots, K$ (the j'th column in the parameter matrix θ). Luckily some nice people tells you what it is:

$$\nabla NLL(\theta_j) = ... = X^\intercal(Y^j - \text{softmax}(\theta^\intercal X)_j)$$

for $j = 1, \ldots, K$, where Y^j is the j'th column of Y e.g. the vector that indicates whether input point i has class j for $i = 1, \ldots, |D|$. You should, of course, verfify this yourself but you do not have to prove it. This should look awfully familiar to you from the logistic regression Exercise. With this in place you should be able to make a gradient descent implementation for softmax regression, like you did for logistic regression.

Numerical Issues with Softmax

There are some numerical problems with the softmax function.

$$\text{softmax}(x)_j = \frac{e^{x_j}}{\sum_{i=i}^{t} e^{x_i}} \text{ for } j = 1, \ldots, t.$$

because we are computing a sum of exponentials (before taking logs again), and when we exponentiate numbers they tend to become very big giving numerical problems. Lets look at the function for a fixed j, $\frac{e^{x_j}}{\sum_{i=i}^{t} e^{x_i}}$. Since logarithm and the exponential functions are inverse we may write $e^{x_j - \log(\sum_{i=1}^{t} e^{x_i})}$ The problematic part is the logarithm of the sum. However, we can move something e^c for any constant outside the sum e.g

$$\log\left(\sum_i e^{x_i}\right) = \log\left(e^c \sum_i e^{x_i - c}\right) = c + \log\left(\sum_i e^{x_i - c}\right).$$

We need to find a good c, and we choose $c = \max_i x_i$ since e^{x_i} is the dominant term in the sum. We are less concerned with values being unwantedly made decreased to zero since that does not change the value of the sum significantly.

Implement Multinomial Regression with Gradient Descent

Train the classifier on the OCR training data and report the error. How does the 10 weight matrices look like (one for each digit). Any surprises. Is it better or worse than one vs. all training using logistic regression. Is it easier or harder to optimize.

You have to hand in code just like for logistic regression. To be precise.

Code Requirements

Cost Function and Gradient Matlab function named softCost in file softCost.m Takes 3 inputs, data X, target y, parameters θ. X is a $n \times (d+1)$ matrix, y is an $n \times 1$ column vector and θ is a $(d+1) \times K$ column vector. The function must return the cost function E_{in} and the gradient ∇E_{in} when applied to data X, y with parameters θ.

Gradient Descent Algorithm Matlab function softRun in file softRun.m that takes three arguments X, y, θ as above and runs gradient descent and return the best parameters found on the data. The algorithm should be optimized for the auDigits training set.

Best parameters Found A simple matlab script named softBestParams.m, that defines two parameters softThetaMnist and softThetaAu the best parameters you have found on the two training sets.