Machine Learning

Lecture 4

Justin Pearson[1]

[1] http://user.it.uu.se/~justin/Hugo/courses/machinelearning/

Today's plan

- Very quick Revision on Linear Regression.
- Logistic Regression — Another classification algorithm
- Possibly more on confusion matrices and F-scores.

Classification and Regression

Remember two fundamental different learning tasks

Regression From input data predict and or learn a numeric value.

Classification From the input data predict or learn what class a class something falls into.

More lingo from statistics. A variable is *Categorical* if it can take one of a finite number of discrete values.

Linear Regression

Given m data samples $x = (x^{(1)}, \dots, x^{(m)})$ and $y = (y^{(1)}, \dots, y^{(m)})$. We want to find θ_0 and theta θ_1 such that $J(\theta_0, \theta_1, x, y)$ is minimised. That is we want to minimise

$$J(\theta_0, \theta_1, x, y) = \frac{1}{2m} \sum_{i=1}^{m} (h_{\theta_0,\theta_1}(x^{(i)}) - y^{(i)})^2$$

Where $h_{\theta_0,\theta_1} = \theta_0 + \theta_1 x$

Linear Regression — Partial Derivatives

$$\frac{\partial}{\partial \theta_0} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_{\theta_0}(x^{(i)}) - y^{(i)})$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_{\theta_0,\theta_1}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

For linear regression you can either find an exact solution minimising $J(\theta)$ by setting the partial derivatives to zero or using gradient descent.

To avoid treating θ_0 as a special case transform your data $(x_1^{(i)}, \ldots, x_n^{()} i)$ to $(1, x_1^{(i)}, \ldots, x_n^{(i)})$ $(x_0^{(i)} = 1)$.

L_2 Regularisation

To avoid over fitting we sometimes want to stop the coefficients becomming too large.

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{i=1}^{n} \theta_i^2$$

There is an exact solution or you can use gradient descent.

L_1 Regularisation

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{i=1}^{n}|\theta_i|$$

Where $|\cdot|$ is the absolute value function. This has no analytic solution. You have to use gradient descent or some other optimisation algorithm.

How do you select your model?

We saw that there are a lot of choices of what model you can choose.

- You can fit higher order polynomials.
- You can have non-linear features such $x_i x_j$ where x_i could be for example the width and x_j could be the breadth and $x_i x_j$ would represent an area.
- You can reduce the number of features.

Picking features is quite complex and we will look at it later. There is also a bigger question, if you have a number of different models how do you decide which to pick? We will look at cross-validation later as well.

Classification and Regression

Remember two fundamental different learning tasks

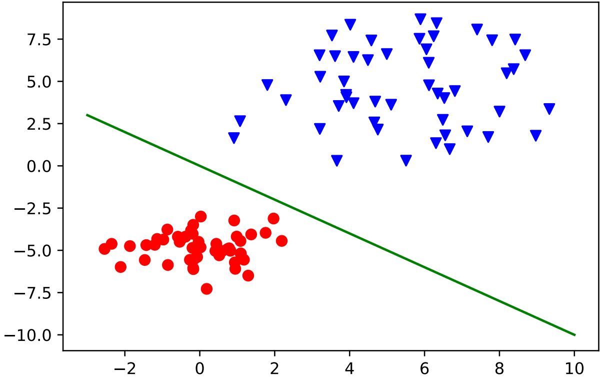
Regression From input data predict and or learn a numeric value.

Classification From the input data predict or learn what class a class something falls into.

More lingo from statistics. A variable is *Categorical* if it can take one of a finite number of discrete values.

Classification

General problem of classification.



Given a number of classes find a way to separate them.

Approaches to Classification

Probabilistic Classification Try to predicate the probability that an put sample x belongs to a class: $P(C \mid x)$.

Learn a hypothesis h_θ such that $h_\theta(x) = 1$ if x belongs to the class and $h_\theta(x) = 0$ otherwise.

With Naive Bayes we calculated $P(C \mid x)$ by looking at $P(x \mid C)P(C)$. Instead we could try to estimate $P(C \mid x)$ directly.

Hypotheses for Classification

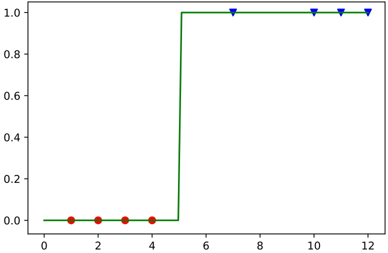
- Learning (and even formulating) hypothesises h_θ such that $h_\theta(x) = 1$ if x belongs to the class and $h_\theta(x) = 0$ otherwise is quite hard.
- It is better to use threshold values and learn an hypotheses such that

$$C_\theta(x) = \begin{cases} 1 & \text{if } h_\theta(x) \leq 0.5 \\ 0 & \text{if } h_\theta(x) > 0.5 \end{cases}$$

Hypotheses for Classification

For the one dimensional case we want to learn some sort of step function

$$h_{\theta_0,\theta_1} = \begin{cases} 1 & \text{if } \theta_0 + \theta_1 x > 0.5 \\ 0 & \text{if } \theta_0 + \theta_1 x \leq 0.5 \end{cases}$$



In general it will be very hard find values of θ_0 and θ_1 that minimise the error on our training set. Gradient descent will not work, and there is no easy exact solution.

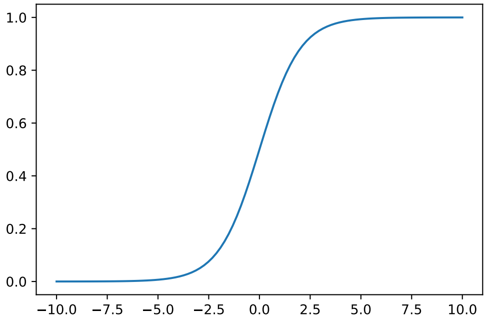
The Logistic-Sigmoid Function

Two(ish) approaches to get (Logisitic)-sigmoid functions

- Try to approximate step functions with a continuous function.
- An argument from probability with log odds ratio.
- Biological motivation neurons and activation functions: Modelling the firing rate of neurons

The Logistic-Sigmoid Function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

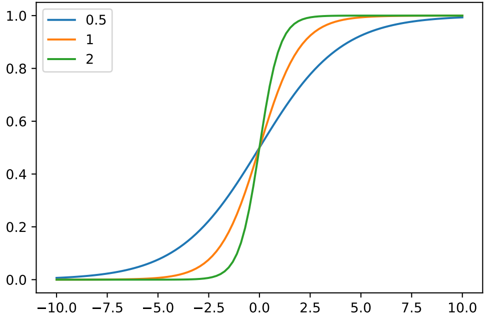


The Logistic-Sigmoid Function

In general we combine it with a linear function

$$h_{\theta_0,\theta_1}(x) = \frac{1}{1 + e^{-\theta_1 x + \theta_0}}$$

As θ_1 gets larger the function looks more a step function.



The Logistic-Sigmoid Function — an informal interpretation

Since for

$$h_{\theta_0,\theta_1}(x) = \frac{1}{1 + e^{-\theta_1 x + \theta_0}}$$

we have that

$$0 \leq h(x) \leq 1$$

We could interpret $h(x)$ is a probability that x belongs to a class.

Derivative of the Sigmoid function

The sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Has a rather nice derivative

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Gradient Descent

Since we can take the derivative sigmoid function it is possible to calculate the partial derivatives of the cost function

$$J(\theta, x, y) = \frac{1}{2m} \sum_{i=1}^{m} (\sigma(\theta^T x^{(i)}) - y^{(i)})^2$$

Where θ is a vector of values.

Neural Networks — Very Briefly — Not examined

A single (artificial) neuron can be modelled as

$$h_{w_1,\ldots,w_k,\theta_0}(x_1,\ldots,x_k) = \frac{1}{1+\exp(\sum_{i=1}^{k} w_i x_i + \theta_0)}$$

For a single neuron apply gradient descent to the function

$$J(w_1,\ldots,w_k,\theta_0) = \frac{1}{2m}\sum_{i=1}^{m}(h_{w_1,\ldots,w_k,\theta_0}(x^{(i)}) - y^{(i)})^2$$

For multiple layer neural networks you just keep applying the chain rule and you get back-propagation.

Neural Networks — Very Briefly — Not examined

- Allow you to do very powerful non-linear regression.
- Even though the cost function is highly non-linear it is generally possible minimise the error.
- Very sensitive to the architecture: the number of layers, how many neurons in each layer.
- For very large networks you need a lot of data to learn the weights. Often you get vanishing gradients. That is for some weight w the quantity $\frac{\partial J}{\partial w}$ can be very small. This can make convergence very slow.

Since tuning neural networks can be hard, try other methods first. With deep learning when they work they work, when they do not work nobody really knows why.

Odds Ratio

Given an event with probability p we can take the odds ratio of p happening and p not happening.

$$\frac{p}{1-p}$$

Odds ratio is another way of expressing Probability

- If the probability of an event is say 0.8 (or 80%) then the odds ratio is

$$0.8/0.2$$

- Or 4 to 1.

Log Odds Ratio

For various reasons it is better to study

$$\log\left(\frac{p}{1-p}\right)$$

Log-Odds make non-linear things slightly more linear and more symmetric.

Log Odds classifier

If we use log-odds we are interested in the quantity $P(C \mid x)$ the probability that we are in the class C given the data x. If we look at the log-odds ratio and use a linear classifier $h_\theta(x) = \sum_{i=1}^{m} \theta_i x_i + \theta_0$.

$$\log\left(\frac{P(C \mid x)}{P(\overline{C} \mid x)}\right) = \log\left(\frac{P(C \mid x)}{1 - P(C \mid x)}\right) = h_\theta(x)$$

A bit of algebra

$$\left(\frac{P(C \mid x)}{P(\overline{C} \mid x)}\right) = \left(\frac{P(C \mid x)}{1 - P(C \mid x)}\right) = \exp(h_\theta(x))$$

More algebra

$$\left(\frac{P(C \mid x)}{1 - P(C \mid x)}\right) = \exp(h_\theta(x))$$

Gives

$$P(C \mid x) = \exp(h_\theta(x))(1 - P(C \mid x))$$

Thus with a bit more algebra we can get

$$P(C \mid x) = \frac{\exp(h_\theta(x))}{1 + \exp(h_\theta(x))} = \frac{1}{1 + \exp(-h_\theta(x))}$$

Logistic regression

Thus if

$$P(C \mid x) = \frac{1}{1 + \exp(-h_\theta(x))}$$

We are modelling the log-odds ratio, which is a good thing.

Cross Entropy Cost

The standard cost/loss/error function

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (\sigma(h_\theta(x^{(i)})) - y^{(i)})^2$$

Is not really suitable if the expected values $y^{(i)}$ can only be 0 or 1. We really want to count the number of miss-classifications. We would also like something convex (one minimum as in linear regression)

Cross Entropy Cost

$$\mathrm{Cost}_\theta(x) = \begin{cases} -\log(\sigma(h_\theta(x))) & \text{if } y = 1 \\ -\log(1-\sigma(h_\theta(x))) & \text{if } y = 0 \end{cases}$$

There are lots of ways of motivating this. One it to use information theory, another is via maximum likelihood estimation. Most importantly (although the proof is outside the scope of the course) it is concave and hence gradient descent will converge to the global minimum.

Cross Entropy Cost — Intuitive Picture

$$\mathrm{Cost}_\theta(x) = \begin{cases} -\log(\sigma(h_\theta(x))) & \text{if } y = 1 \\ -\log(1 - \sigma(h_\theta(x))) & \text{if } y = 0 \end{cases}$$

- Suppose our target value y is equal to 1, and $\sigma(h_\theta(x))$ is close to 1 then $\mathrm{Cost}_\theta(x)$ will be close to 0. Remember $log(1) = 0$.
- Again when $y = 1$ if when $\sigma(h_\theta(x))$ gets closer to 0 then $-\log(\sigma(h_\theta(x)))$ gets larger and larger. We heavily penalise values away from 1.

Cross Entropy Cost

Since y can only be 0 or 1 in a classification task we can rewrite the cost function as

$$\mathrm{Cost}_\theta(x) = -y\log(\sigma(h_\theta(x))) + (1-y)\log(1-\sigma(h_\theta(x)))$$

So our total cost/error function is now (I've made a factor of $\frac{1}{2}$ go away, don't worry).

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} -y\log(\sigma(h_\theta(x))) - (1-y)\log(1-\sigma(h_\theta(x)))$$

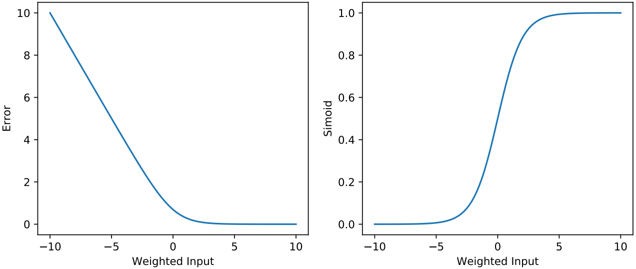
Unlike linear regression there is no analytic solution for the minimum.

Cross Entropy Cost

Just looking at

$$-y\log(\sigma(h_\theta(x)))$$

We are trying to force the term $\sigma(h_\theta(x))$ to be 1. The larger the value of θx the less the error.



After 0 we do not really care we just want to force move the input over to the right

Cross Entropy Cost — Gradients

If you do lots of algebra then

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(\sigma(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Thus the gradient descent algorithm is almost the same as for linear regression.

Logistic Regression — Linear Features

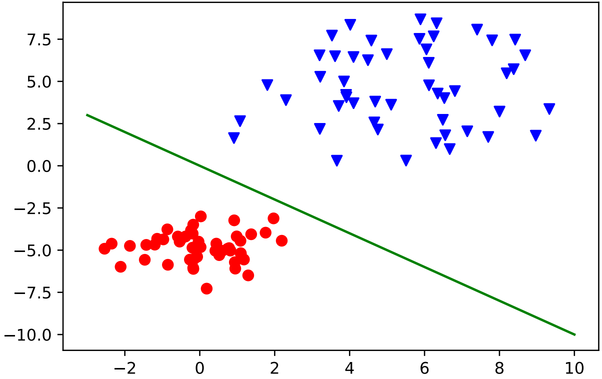
When learning the parameters for

$$\sigma(h_\theta(x))$$

Where h_θ is a linear function. You are essentially finding separating hyperplanes.

Separating Hyperplane

Everything on one side of the hyperplane gets classified in the class and everything on the other side gets classified as not belonging to the class.



Classes have to be linearly separable.

Linear separability

There are lots of things that are not linearly separable. For example

x	y	class
0	0	0
0	1	1
1	0	1
1	1	0

This was noticed in the 60s and was taken as a proof of the limitation of the single perception. They did not know how to train multi-layer networks, and this was partly responsible for shifting AI research to more symbolic AI.
Again it is possible to use non-linear features (polynomials) as we did with linear regression.

Multiclass classification — One vs All

We have only provided a classifier that estimates the probability that

$$P(C \mid x)$$

A simple way of combining these is picking the i such that $P(C_i \mid x)$ is maximised, but we have to be a bit careful.

Multiclass classification — One vs All

Given n classes we need to train n classifiers

$$\sigma(h_{\theta_1}(x)), \ldots, \sigma(h_{\theta_i}(x)), \sigma(h_{\theta_n}(x))$$

Our data set consists of data points $x^{(j)}$ and labels $y^{(j)}$.
For the classifier $\sigma(h_{\theta_i}(x))$ the label $y'^{(j)}$ is defined as

$$y'^{(j)} = \begin{cases} 1 & \text{if } y^{(j)} = i \\ 0 & \text{otherwise} \end{cases}$$

Thus we build a classifier where the positive class is the class we are interested in and the negative class is all the rest.

Confusion Matrices

Given some input x, four things can happen:

True Positive x belongs to the class and we predict that.

False Negative x is in the class, but we predict not.

False Positive x is not in the class, but we predict it is.

True Negative x is not in the class , and we predict that it is not in the class.

True Positive and True Negative are the good things. We want to minimise False Positives and False Negatives. Sometimes we cannot minimise both.

Classification — Confusion Matrices

We can put this into a table:

		Prediction outcome		
		p	**n**	**total**
actual value	$\mathbf{p}'$	True Positive	False Negative	P'
	$\mathbf{n}'$	False Positive	True Negative	N'
	total	P	N	

Classification — Confusion Matrices

What do we minimise? False positive or false negative?

- If you are classifying cancerous tumours , then a false positive means we predict a tumour but there was none. If this is the case we just have to investigate more to find the true situation. We want to avoid false negatives where there is a tumour, but the algorithm predicts that there is none.
- If our spam detector classifies as spam and non-spam. A false positive means that a non-spam message gets classified as spam. This means that you might miss important email that is put into the spam filter.

Confusion Matrices — Accuracy

Accuracy is defined as

$$\frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{TF} + \mathrm{FN}}$$

This is the fraction of the time that we get things correct.

Confusion Matrices — Precision

Precision is defined as

$$\frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

Of all the positive predictions what fraction are actually correct.

Confusion Matrices — Recall

Recall is defined as

$$\frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

The quantity $\mathrm{TP} + \mathrm{FN}$ is the actual number of instances in the class, and so recall gives you the fraction of the time you are catching something in the class.

Confusion Matrices — F-Score

Combine precision and recall into one quantity.

$$2\frac{\mathrm{Precision}\cdot\mathrm{Recall}}{\mathrm{Precision}+\mathrm{Recall}}$$

It is actually the harmonic mean of the precision and the recall. Since you are they are both ratios it only makes sense to take the harmonic mean.

Maximising the F score maximises both the precision and the recall.