**Definition of ML:**

“A computer program is said to learn from experience with respect to some class of tasks and performance measure , if its performance at tasks in , as measured by , improves with experience .” -Samuel Mitchell

* Experience
  + How the learning algorithm interacts with a dataset
* Performance Measure
  + Loss function
* Tasks
  + Fundamental tasks include:
    - Classification
    - Regression
    - Density Estimation

**Classification:**

* Goal:
  + Specify which of categories some input belongs to.

⇒ Learn “hypothesis” function , such that

**Regression:**

* Goal:
  + Perform continuous function approximation

⇒ Learn “hypothesis” function , such that

**Structured Prediction:**

* Goal:

**Density Estimation:**

* Goal:
  + Estimate a probability density (or mass) function from sampled data

i.e. Perform “estimation” (aka Statistics)

⇒ Learn “hypothesis” function , such that

* + is a Random Vector
    - Continuous case:
      * is a PDF

We are given a **training set** which consists of training examples .

Place the entire training set into a table as follows.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sample Number  (Training Example Number) | feature 1 | feature 2 |  | feature | () target |
| 1 |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Feature Vector:

* One row of features (stored as a column vector)

Target Value:

Target Vector:

Training Example:

**Design Matrix**:

* Feature vectors as rows

**MSE Cost Function**:

:

Un-normalized -norm of is defined as

**MSE** is square of un-normalize -norm:

**Partial Derivative**

measure how changes as only the variable increases at point

**Gradient (Vector Derivatives)**:

Generalizes partial derivative

**Directional Derivative**:

* The directional derivative in the direction of the unit vector is the slope of the function in direction

**Matrix Derivative:**

**From class and online dude’ lectures**

**Jacobian:**

Partial derivative of a function who’s inputs and outputs are both vectors.

**Second Derivative:**

For a function , the derivative with respect to of the derivative of w.r.t.

**Trace**

**Useful Properties of Trace and Matrix Derivative**

* From Ng Lectures on top and ASU lectures below

* The professor’s notation is apparently as follows

Least squares “solution”

To find , take negative, set equal to zero, and solve for .

Note: This assumes that is non-singular.

**Probabilistic interpretation of Linear Regression:**

Now assume that each sample has a target that has a noise term which is a random variable that is drawn from a Guassian distribution with zero mean and variance , that is .

The PDF (not PMF) is as follows (where the parameter is viewed as an unknown value that is not a Random Variable).

Assuming all the ’s are independent, the joint density is given as:

Looking at the joint density as a function of , we arrive at the likelihood function .

We call the probability of the data and we call the likelihood of the parameters.

Next, we take logarithm for convenience:

Find the minimum by computing the gradient with respect to , setting the result equal to zero, and solving for.

From the derivation of the least squares solution, we found that **.**

**Setting**  equal to zero, and solving for , we have

We conclude that .

**Finding Variance**Repeat the process to compute the variance as the estimate.

Looking at the PDF as a function of , we arrive at the likelihood function .

Next, we take logarithm for convenience:

To find the maximum, we take derivative, set equal to zero, and solve for.

Set equal to zero and solve for the estimate of .

In other words, the more examples used the smaller the variance.

**Bayesian Estimation:**

Now we consider the unknown parameter we are trying to estimate is a random variable.

Since is an RV it also a PDF.

Recall Bayes’ Rule:

is the **prior density**

* This is the information we have about prior obtaining any measurements

is the **likelihood function** (if is an unknown constant)

is the **posterior density**

* This is from after we have taken the measurements

**Bayesian Estimation Applied to Linear Regression**

Recall that the likelihood from MLE is as follows.

Joint density:

Looking at the joint density as a function of , we arrived at the likelihood function .

In MLE we simply chose to maximize the log likelihood.

**Covariance:**

The covariance between two random variables and is defined as follows.

We can define the covariance matrix for these two RVs as follows:

Given a random vector , where each element is an output of an experiment, we have

Note that .

Uncorrelated RV’s result in a diagonal matrix, as follows.

**Bivariate Gaussian:**

Consider two independent RV’s:

**Multi-Variate Gaussian**:

**Geometric Interpretation of Least Squares**

**Digression of projection matrices**

Consider a vector . Let’s project a vector onto , i.e. find the point on that is closest to

This is seen geometrically as the error , where , .

Noting that is perpendicular to , we have .

Plugging this result into , we have

We can generalize this result be looking at the projection matrix that acts on to produce .

Because is on the line through , we have .

How is this relevant?

Given that may not have a solution, we know that , but we may have .

If in fact, we simply solve instead, where is the point closest to which is in .

In other words, we project onto the column space of to produce , and solve for *.* This will be the best approximate solution of .

Consider now . If we have a plane which does not lie in, we can project onto this two-dimensional subspace of .

Looking at two vectors that form a basis for the plane we want to project into, we have

Since the two columns of form a basis for the plane, they fully describe the plane we desire to project into.

We know that there is a vector which is in the plane. So we know there is a vector which is perpendicular to the plane.

Since is in the plane described by the basis vector which are the columns of , we have

In other words, our objective is to find the right combination of the columns so that the error vector is perpendicular to the plane.

Given , find .

The key to this problem is that is perpendicular to the plane. Since are in the plane we have the following set of relationships. Note that all vectors in the plane are perpendicular to .

By the structure of the final equation we can see that , because as you recall, the nullspace is defined as the set of vectors that multiply to map to .

Furthermore, the nullspace of is defined the same.

From the geometry of the four fundamental subspaces, we know that .

Looking back at the equation, we have

Now, let’s find the corresponding projection. Recall that . Plugging in our solution for , we have

is our projection matrix that projects into the plane onto the point , which is the point which is in the column space of that is closest to **.**

The extreme cases are:

1. If is in the column pace, then
2. If is in perpendicular to the column pace, then

The fundamental geometry of this is:

1. If then , i.e.
2. If , then

The overall geometrically picture is as follows. We have a vector which has a component and a component in .

is the projection applied of onto , and is the projection onto .

**Logistic Regression Backpropagation with examples**

The cost function is defined as .

Note that and .

initialize

repeat until convergence

{

for i = 1:M

{

}

}

* Note that the accumulator for the cost function is not used in the loop. It is used for testing for convergence.

**Vectorization of Logistic Regression**

Step 1: Place the terms in an array:

repeat until convergence

{

for i = 1:m {

}

}

**Step 2:** Place the inputs in a matrix and the weights in a vector:

Input (Data) Matrix:

Net Input Matrix:

Python notation:

You can use with the above numpy addition and will implicitly be broadcasted.

Activation Matrix:

Output (Target) Vector:

Derivate of loss with respect to net input:

Note: In the previous implementation, we are computing a running sum on (one for each of the -weights) and over the -training examples, as shown below

// Looking at this part of the loop

for i = 1:m {

}

Unrolling the loop, we have

Therefore, we can write this as a summation and vectorize appropriately.

The algorithm now reduces to the following (note that this is only one gradient descent step).

// Initialize

**Numerical Gradient**:

Looking at the definition of the derivative of a generic 1D function and its equivalence to the more numerically stable two sided version, we have

As an example, consider evaluating at with with .

Note that the actual derivative is .

The error of the two-sided numerical derivative is .

The two-sided numerical derivative generally has an error of .

The one-sided numerical derivative generally has an error of .

**Gradient Checking**:

Check Euclidean distance between vectors (square root of sum of squares) with:

if

you are good

else if

likely a bug

else if

error