Chapter 5

Regression

In this chapter, we will start looking into machine learning techniques. To start with we, look into regression.

5.1 Functions and Basis

Recall from calculus that we can approximate a function with a series, the most popular one being Taylor series approximation, where we approximate a function with it's projections on a polynomial basis:

$$f(x) = \sum_{n=0}^{N} \frac{f^{n}(x_0)(x - x_0)^n}{n!} \tag{5.1}$$

where, $f(x)$ is the function we want to approximate, index n denotes nth function in the basis and it ranges from 0 to N, $f^{n}(.)$ denotes the n^{th} derivative of the function, and $n!$ is the factorial of n.

A question to ask here is why does this approximation or any approximation of this type is good? To answer this question we need to understand the concept of basis. We can approximate any function if we choose a large/rich set of basis. "Basis" can be thought of as component function in a function space with the condition that none of the component function can be obtained through the linear combination of all other functions.

Note: A basis is a set of functions on a function space, if every continuous function on that space can be written as a linear combination of these basis. Elements of basis are linearly independent of each other. Can you guess why?

Note: The set of all functions that can be written as a linear combination of basis functions is called the span of the basis.

5.2 Regression from Data

In the above section, we decomposed a function in polynomial basis when we can calculate all the derivative so the function. What do we do, when we don't have access to the derivative of the function but only values of function at certain points? Think in term of real life, we can get samples of the under lying relation between variables but we don't always have access to the expression of this relation. Suppose, we want to find the relation between change in resistance of a conductor with temperature. We can vary the temperature of the resistors and measure the resistance. Now, we have a set of values $(x_j, r(x_j))$. Can we find as expression for the underlying function $r(x)$ given this data?

The answer is yes, we can try to fit a function to the data. In a way similar to the Taylor series approximation for analytic functions. We start out with an assumption that a certain number of polynomial basis N will be sufficient for the approximation of the underlying function. Then we can setup a loss function on the data and minimise the loss for the optimum projections on these basis.

$$L(\boldsymbol{a}) = \frac{1}{2}\sum_{j=0}^{M}(r(x_j) - \sum_{n=0}^{N} a_n x_j^n)^2 \tag{5.2}$$

Here, $\boldsymbol{a} = [a_0, a_1, ..., a_N]$ are the unknown weights of the polynomial basis, that we are trying to find. $r(x_j)$ and x_j^n are the values of resistance and the values of temperature raised to nth power.

Any ideas as to how we can minimise this loss function? Simple calculus to the rescue :). We need to find the gradient of the loss function a find the points where it is zero, this will give us the critical points of the loss function, we can then look at the points where the Hessian matrix is positive definite.

It is much more convenient to use the matrix notation, but just to give an idea of the method, we will first derive expression for each variable independently. The derivative of the loss function with respect to the k^{th} element of vector $\mathbf{a}$ is

$$\frac{\partial L(\ \mathbf{a})}{\partial a_k} = \frac{1}{2}\sum_{j=0}^{M}(2(r(x_j) - \sum_{i=0}^{N} a_i x_j^i) x_j^k) \tag{5.3}$$

we can write the expression in terms of dot product and eliminate the sum over data samples

$$\frac{\partial L(\ \mathbf{a})}{\partial a_k} = \langle \boldsymbol{r}, \boldsymbol{x^k} \rangle - \sum_{i=0}^{N} a_i \langle \boldsymbol{x^i}, \boldsymbol{x^k} \rangle \tag{5.4}$$

where $\langle .,. \rangle$ represents dot product, $\boldsymbol{x}^i = [x_0^i, x_1^i, ..., x_M^i]^T$, and $\boldsymbol{r} = [r_0^i, r_1^i, ..., r_M^i]^T$. If we set equation 5.4 equal to 0 and rearrange the terms we get,

$$\sum_{i=0}^{N} a_i \langle \boldsymbol{x^i}, \boldsymbol{x^k} \rangle = \langle \boldsymbol{r}, \boldsymbol{x^k} \rangle \tag{5.5}$$

Now, we have an equation in $N+1$ variables, how do we solve it? At the very least we will need $N+1$ equations to solve for these $N+1$ variables. Any ideas, where we will get these extra N equations? We get it by taking the derivative with respect to the other components of $\boldsymbol{a}$, essentially the above expression for all values of $k \in \{0, 1, 2, ..., N\}$.

Let us the write the above, in matrix notation which is more convenient in our notation. We can rewrite 5.4 as

$$[\boldsymbol{x^k}]^T \boldsymbol{X}\boldsymbol{a} = [\boldsymbol{x^k}]^T \boldsymbol{r} \tag{5.6}$$

where $\boldsymbol{X} = [\boldsymbol{x}^0, \boldsymbol{x}^1, ..., \boldsymbol{x}^N]$ and $\boldsymbol{a} = [a_0, a_1, ..., a_N]^T$. We can combine for all k equations, and the expression now becomes.

$$\boldsymbol{X}^T \boldsymbol{X}\boldsymbol{a} = \boldsymbol{X}^T \boldsymbol{r} \tag{5.7}$$

Now this looks like the an expression we can solve, notice that $\boldsymbol{X}$ is a matrix of sampled basis at value of $x = x_j, j \in \{0, 1, , M\}$ the data points where we samples our original function $r(x)$. The column vector $\boldsymbol{r}$ are the value of function $r(x)$ at sampled points. Here we go now, to find vector $\boldsymbol{a}$ which are projections on our basis functions we can solve the above system of linear equations.

Some important things to note here, $\mathbf{X}^T\ \mathbf{X}$ is a square matrix and is full-ranked if $M \geq N$so we can find the inverse for this matrix. Only issue to look into is that matrix is well-conditioned, in order for us to invert it. Keep this point in mind, we will tie it up with the concept of regularisation.

5.3 Under-fitting, Over-fitting, and Regularisation

Now that we have introduced the tools for setting up a regression problem. Let's look into some important concepts associated with regression. The most import part of setting up a regression problem is the choice of the basis. How many basis do we use in regression? are fewer basis better, or are more basis better? and what is the right number of basis?

If we have a very small number of basis, we get an under fit where the basis are not sufficiently diverse to capture the properties of the training data (of course it would not work on test data as well). Simplest example of an under-fit problem is if we are trying to fit a data from quadratic equation with constant and linear basis. Of course a projection on these two basis will not be able to capture the quadratic data properly.

However, what happens if we have too many basis? Imagine if we have 100 data points and we try 100 basis. The regression function will match the data completely but will it be a good fit to the underlying distribution of the data. The answer is NO. Data from real life scenarios is almost always prone to some noise, typically what it means is that any good regression method should be immune to this noise. So a complete fit to the data might not be a good idea. We want the regression solution to fit the underlying distribution while being immune to the noise in the process. Thus, we have to be careful about the selection of the right number of basis.

There are detailed examples in the `Google Colab` notebook, to explain in detail the under-fitting and over-fitting scenarios.

Regularisation

The main idea of regularisation is to add a term to the energy, which penalises the projection on the basis which are not related to data.

$$L(\boldsymbol{a}) = \frac{1}{2}\sum_{j=0}^{M}(r(x_j) - \sum_{i=0}^{N} a_i x_j^i)^2 + \lambda \sum_{i=0}^{N} a_i^2 \tag{5.8}$$

Now, if we do the same steps for this equation that we did for the unregulariszed version of the equation we will get the expression for optimum weighs as

$$[\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}]\boldsymbol{a} = [\boldsymbol{X}]^T\boldsymbol{r} \tag{5.9}$$

Some important observations, The regularisation term puts a penalty on norm of the weight vector. Here we are using the second norm of the weight vector because it is the simplest norm to understand and implement. However, in practice it might not be the best regularisation to use in all case. Another popular norm used in regularization is the first norm of the weight vector. It has the desirable property of sparsity.

Note: To understand why the L2 regularization works. Assume that 2 columns of X are correlated then for we can have many values of projections on these two components for which the residual will be minimum (both components can cancel each other). Putting a penalty on weight of the projection ensures that the values for projection on correlation are not complementary.

!

Note: Another interpretation of the L2 regularization is this. If columns in X are highly correlated then the condition number for X^TX is very high (X^TX is not well conditioned). Adding the weight penalty converts the term into $X^TX + \lambda I$ which is well conditioned if λ is sufficiently large.

!

5.4 Logistic Regression

So far, we have seen how to regress data to a continuous function. In many applications in real life, however, we require to output discrete categories for data. The techniques that we have studied above will not work in these cases as we will not be able to define the derivative for function with discrete outputs.

To solve problem like these we use logistic regression. The core idea of logistic regression is: instead of finding the discrete class labels, we instead find the probability of data belonging to a certain class.

In this chapter we will setup a simple binary classification problem with logistic regression and provide some examples. In order to convert a linear regression model to output probability values for a class we use sigmoid function, defined as:

$$\sigma(x) = \frac{1}{1+exp(-x)} \tag{5.10}$$

The output of the sigmoid function lies in the range (0,1), so it can be interpreted as the probability of belonging to a class.

Next we setup a linear logistic model. Let's assume we have data points ($\boldsymbol{x}_i$,y_i) where $\boldsymbol{x}_i$ are input points and y_i is the corresponding class label. In the case of binary classification $y_i \in \{0,1\}$. Our model is

$$\hat{y} = \sigma(h_\theta(x)) \tag{5.11}$$

where,

$$h_\theta(x) = \sum_{j=0}^{N} a_j x^j \tag{5.12}$$

Note: In our example, we are assuming x to be one-dimensional. It can be multi-dimensional as well.

Similar to the procedure for linear regression we can minimize the residual of the logistic linear regression

$$\sum_i (y_i - \hat{y}_i)^2 = \sum_i (y_i - \sigma(h_\theta(x_i)))^2 \tag{5.13}$$

To find the minima we set the derivative of the residual equal to zero and solve for the unknowns.

$$\frac{\partial}{\partial a_j} \sum_i (y_i - \hat{y}_i)^2 = 2\sum_i (y_i - \hat{y}_i) * \sigma'(h_\theta(x)) * x^j \tag{5.14}$$

Kullback–Leibler Divergence and Cross-Entropy Loss

Since, the output of the sigmoid is interpreted as the probability of belonging to a certain class. For most application we will use metrics on probability like Kullback-Leibler divergence (KL divergence). Let's assume we have two discrete probability distributions P(x) and Q(x). The Kullback–Leibler divergence $D_{KL}(P||Q)$ is defined as:

$$D_{KL}(P||Q) = -\sum_x P(x) log \frac{Q(x)}{P(x)} \tag{5.15}$$

Some interesting properties of KL divergence:

- KL divergence is always non-negative
- In general $D_{KL}(P||Q) \neq D_{KL}(Q||P)$

- $D_{KL}(P||Q) = 0$ if and only if P and Q are identical
- KL divergence in convex in P and Q

Note: What happens to the value $P(x)log\frac{Q(x)}{P(x)}$ as $P(x) \rightarrow 0^+$? Why aren't we interested in $P(x) \rightarrow 0^-$?

In most machine learning applications, we will focus on the expression of cross entropy (a special case of KL divergence)

$$L_{CE}(y,\hat{y}) = -\sum_i(\sum_k y_i^k log(\hat{y}_i^k)) \tag{5.16}$$

where, i represents the data sample, k represents the number of out class $y^k = 1$ if y belongs to class k and is zero otherwise.

Note: Can you show that cross entropy loss is a special case to KL divergence.

For binary classification we will get

$$L = -\sum_i y_i^k log(\hat{y}_i^k) + (1 - y_i^k) log(1 - \hat{y}_i^k) \tag{5.17}$$

The derivative of the loss is

$$\frac{\partial L}{\partial a_j} = -\frac{y_i}{\hat{y}_i^k} * \sigma'(h_\theta(x)) * x^j + \frac{1 - y_i^k}{1 - \hat{y}_i^k} * \sigma'(h_\theta(x)) * x^j \tag{5.18}$$

which can be further simplified as:

$$\frac{\partial L}{\partial a_j} = -(y - h_\theta(x)) * x_j \tag{5.19}$$

Now, that we have defined the derivative of the loss function with respect to the unknowns, we can setup an iterative scheme for finding the minima of the problem. The details of the optimization tools to setup such schemes follow in the next chapter.

to be added: appropriate figures and exercises

Exercises

Exercise 5.1: Perform linear regression on data provided for the exercise. You can access the data by using the following commands:

```
!git clone https://github.com/naeem872/AICWM.git
import pandas as pd
import matplotlib.pyplot as plt
data=pd.read_csv('AICWM/Data/Q1trainData.csv')
x=data['x'].to_numpy()
y=data['y'].to_numpy()
plt.plot(x,y)
```

- using the above code snippet load the training data from Q1trainData.csv

- using the above code snippet load the test data from Q1testData.csv
- plot both the test and train data on the same plot
- perform the linear regression with 2 up to 10 basis and show the test error as a function of the number of basis
- can you observe over-fitting and under-fitting?
- perform the l_2 regularization and see how it changes the results

Exercise 5.2: Let's try to write a code to produce noisy version of a function and then try to perform polynomial regression on it:

```
x=np.linspace(0,10,1000)
x=x.T
f=0.2
pi=np.pi
m,b=1,1
y=np.sin(2*pi*f*x)+np.random.rand(x.size)
plt.plot(x,y)
```

- use the above code to generate test and train samples for the following functions: $\exp x$, $\log x$ (do we need to change the domain of the function), $u(x)$step function at $x = 5$, $\delta(x)$ delta function at $x = 5$

Exercise 5.3: **Linear Regression for images:** In this exercise we will show that our formulation of linear regression can be applied to images as well. First we create a test image:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

im=np.zeros((28,28))
im[10:15,10:15]=1

plt.imshow(im)
```

Next we define basis for linear regression. We will use the following basis:

$$b_{i,j} = \cos(\frac{\pi i x}{N})\cos(\frac{\pi j y}{M}) \tag{5.20}$$

where N, M is the size of the image. x, y represent the pixel coordinate. Vary the number of basis from $i, j = \{5, 10, 15, 25\}$ and see how the resultant image changes.

Exercise 5.4: Logistic Regression: Perform logistic regression on the data generated by the following code:

```
num_data=100 # data points per class

x1=np.random.randn(2,num_data)+1
x0=np.random.randn(2,num_data)

y1=np.ones((1,num_data))
y0=np.zeros((1,num_data))

X=np.concatenate((x1,x0),axis=1)
y=np.concatenate((y1,y0), axis=1)

print(X.shape)
print(y.shape)


plt.plot(X[0,:100],X[1,:100],'b*')
plt.plot(X[0,100:],X[1,100:],'r*')
X=X.T
y=y.T
```