AIML - CS 337

Lecture 8: Classifiers and Loss Functions, Tutorial 1 Solutions

September 1, 2022

Lecturer: Abir De *Scribe: Group 15 and Group 16*

1 Classifiers

Suppose we are given two datasets - $\mathcal{D}_{\text{Train}} = \{\mathcal{X} \times \mathcal{Y}\}^M$ and $\mathcal{D}_{\text{Test}} = \{\mathcal{X} \times \mathcal{Y}\}^N$. Here $\mathcal{X} \subset \mathbb{R}^d$ is a set consisting of d-dimensional input features, while $\mathcal{Y} \subset \mathbb{R}$ consists of the corresponding labels.

The true classifier h^* satisfies $h^*(x_j) = y_j$ for all $(x_j, y_j) \in \mathcal{D}_{\text{Test}}$. On the other hand, we develop a classifier $\widehat{h}$ using only the training dataset, satisfying $\widehat{h}(x_i) = y_i$ for all $(x_i, y_i) \in \mathcal{D}_{\text{Train}}$. Our aim is to find the best classifier $\widehat{h}$ which can mimic the true classifier h^*.

2 Types of Classifiers

2.1 Complex $\widehat{h}$

The most obvious classifier is simply a dictionary for $\mathcal{D}_{\text{Train}}$, i.e. a **Table Look-up** function. Here, we remember the label of each $x_i \in \mathcal{X}_{\text{Train}}$ in our classifier $\widehat{h}$. However, even though this can represent any function, this classifier will behave terribly on unseen data from the test set, on which we have no prior information about the label. Another disadvantage is that, as the data complexity of the dataset itself increases, storage and maintenance of this type of classifier can become cumbersome. Hence, we discard this classifier.

2.2 Modest $\widehat{h}$

2.2.1 Voronoi classifier

Another class of non-parametric classifiers consists of those using nearest neighbors. For example, the **Voronoi classifier** involves making a convex polyhedron (called the Voronoi Cell) for each point in the training dataset and then finding the polyhedron in which a new data point lies.

2.2.2 Exhaustive k-Nearest Neighbor classifier

Another example is an **Exhaustive k-Nearest Neighbor classifier**. Here, k is a hyperparameter and not a learnable parameter. The idea is, for each new test data point, find the k nearest neighbors from the training data, and give the majority class of these k points. Note that this does not require any processing of the training data, and so the training time is negligible (probably the only $\mathcal{O}(0)$ training time classifier we would see).

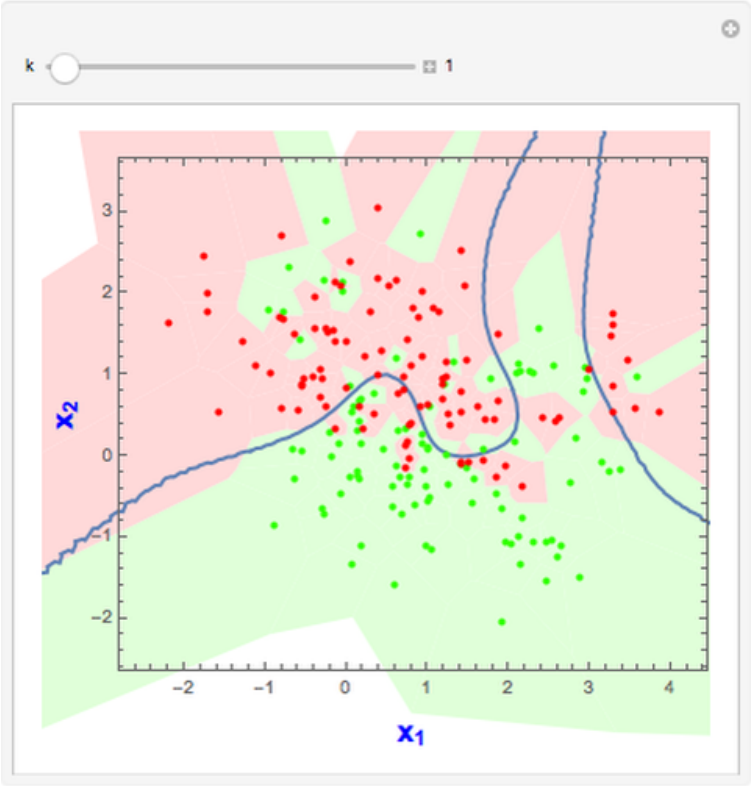

Figure 1: 1-NN classifier for data with 2 classes

The benefit of the k-NN algorithm is that it works nicely with non-linear data. However, during inference, we have to find the k nearest neighbors, which causes linear time blowup (time complexity will be of order $nd + n \log k$). This is undesirable in most situations, as we generally want testing to be fast so that the user does not have to wait for a long duration. Thus, we now move on to parameterised algorithms.

2.3 Simple $\widehat{h}$

A simple function for the classifier is the linear function, i.e. $\widehat{h} = w^T x + b$. Note that this classifier may not work suitably for all possible datasets, and if the given dataset is highly non-linear, then this classifier would perform poorly.

2.3.1 Constant $\widehat{h}$

A possible restriction to linear classifiers is the set of constant classifiers. The best constant classifier $\widehat{h} = c^*$ is given by

$$c^* = \arg\min_c \sum_{(x_i, y_i) \in \mathcal{D}_{\text{Train}}} \mathbb{I}[c \neq y_i]$$

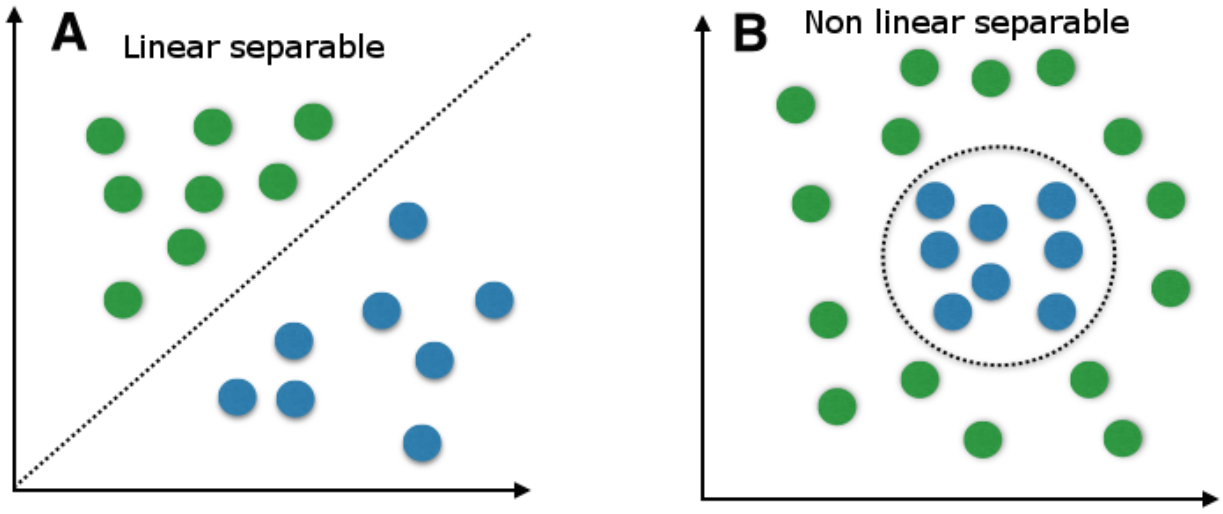

Figure 2: Linear v/s Non-linear data

This can easily be found as the mode label (also known as majority class) of the training data. However, although this classifier is easy to learn, it can perform poorly in the case when the distribution of the training data and the test data differs even slightly.

So our focus is mostly on linear classifiers. Note that we don't go to higher dimensions (like quadratic functions), since the problem at hand is already pretty hard.

2.4 Hypothesis Class

The hypothesis class is the set of classifiers over which we are searching for the optimal. For example, for a linear classifier, this class is given by all possible linear functions, i.e. all possible pairs (w, b) with weight $w \in \mathbb{R}^d$ and bias $b \in \mathbb{R}$.

3 Error

For any classifier $\widehat{h}$, the error that we aim to minimize is given by

$$\text{Error}(\widehat{h}) = \sum_{(x_j, y_j) \in \mathcal{D}_{\text{Test}}} \mathbb{I}[\widehat{h}(x_j) \neq y_j]$$

Our aim is to minimize the test error, but our classifier must be learned from the training dataset. In particular, the best classifier h^* satisfies

$$h^* = \arg\min_{\widehat{h} \in H} \sum_{(x_i, y_i) \in \mathcal{D}_{\text{Train}}} \mathbb{I}[\widehat{h}(x_i) \neq y_i]$$

Note that, if the set of classifiers over which we are looking is not exhaustive enough, i.e. we are missing some crucial classifiers, then the error for any classifier in H will be extremely large. So our hypothesis class should be able to generalize on $\mathcal{D}_{\text{Test}}$.

3.1 Linear Hypothesis Class

Our aim is to find the optimal pair $\{w^*, b^*\}$ such that

$$\{w^*, b^*\} = \arg \min_{w \in \mathbb{R}^d,\, b \in \mathbb{R}} \sum_{i=1}^{M} \mathbb{I}[w^T x_i + b \neq y_i]$$

However, since we are searching over all $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$, $w^T x_i + b$ is a continuous function, which means that its probability of actually becoming exactly equal to a real number y_i is 0. In such a situation, the above error will become M with probability 1. Thus, we must change our error function.

3.2 Discrete error - Based on Signum Function

Let's assume that there are only 2 classes, -1 and 1, i.e. for all $1 \leq i \leq M$, we have $y_i \in \{-1, 1\}$. One option is to make our prediction 1 if $w^T x_i + b$ is non-negative, and -1 otherwise. This can be written as the classifier $h(x_i) = (w^T x_i + b)$. Thus, we wish to find the parameters

$$\{w^*, b^*\} = \arg \min_{w \in \mathbb{R}^d,\, b \in \mathbb{R}} \sum_{i=1}^{M} \mathbb{I}[(w^T x_i + b) \neq y_i]$$

Note that the set $\mathcal{D}_{\text{Train}}$ is **scarce** (does not cover a lot of data points), so any discrete classifier will not be able to generalize well over the test set. Hence, it is better to look for **probabilistic classifiers**. Another problem with a discrete classifier is that it will not be possible to find derivatives, which are required during gradient descent.

3.3 Final Proposal for Error - Based on Sigmoid Function

To mimic the discrete classifier, we replace the signum function with the sigmoid function. So our classifier will give the prediction $f(x_i) = \dfrac{1}{1 + e^{-w^T x_i + b}}$. Now we are looking for the parameters

$$\{w^*, b^*\} = \arg \min_{w \in \mathbb{R}^d,\, b \in \mathbb{R}} \sum_{i=1}^{M} \mathbb{I}\left[\frac{1}{1 + e^{-w^T x_i + b}} \neq \frac{y_i + 1}{2}\right]$$

Since the sigmoid function squeezes our prediction between 0 and 1, we have also changed the true prediction -1 to 0, by shifting $y_i \longrightarrow \dfrac{y_i + 1}{2}$.

Finally, the error function is still discrete, and so we replace it with a continuous function, giving the final classification problem proposal:

$$\boxed{\{w^*, b^*\} = \arg \min_{w \in \mathbb{R}^d,\, b \in \mathbb{R}} \sum_{i=1}^{M} \max\left(0, \left(\frac{1}{2} - f(x_i)\right) y_i\right)}$$

4 Solutions to Tutorial 1

Presented below are the solutions to the first tutorial:

4.1 Problem 1

A loss function is a good approximator of $\mathbb{I}[(h(x_i)) \neq y_i]$ if it takes a high value when y_i and $h(x_i)$ are of different signs, while it is close to 0 when they have the same sign.

4.1.1 Part (i)

max{0, 1 - $y_i \cdot h(x_i)$}

If y_i and $h(x_i)$ have the same sign, then $1 - y_i h(x_i)$ is lesser than 1, and so it's max with 0 is close to 0 (or 0 only). On the other hand, when they have different signs, $1 - y_i h(x_i)$ is highly positive, as desired.

Note that the term 1 is also added here, to ensure that small noises do not lead to significant errors in the function output.

y	h(x)	Loss
+1	> 0	0
-1	> 0	> 0
+1	< 0	> 0
-1	< 0	0

Answer - YES

4.1.2 Part (ii)

$$min\{0, 1 - y_i \cdot h(x_i)\}$$

This works complementary to the first subpart. So when y_i and $h(x_i)$ have different signs, the term $1 - y_i h(x_i)$ is positive, making the loss 0. While, in the other case, the loss is either 0 or a negative value. Although this follows the trend to some extent, it does not help in penalizing the bad cases sufficiently. Hence, this is a bad loss function.

Answer - NO

4.1.3 Part (iii)

$$\frac{\exp(-y_i \cdot h(x_i))}{1 + \exp(-y_i \cdot h(x_i))}$$

The given loss function assumes high values ($\rightarrow 1$) when y_i and $h(x_i)$ are of opposite signs, and takes values closer to 0 when they have the same sign. Thus, this is a valid approximator.

Answer - YES

4.1.4 Part (iv)

$$\frac{1}{1+\exp(-y_i \cdot h(x_i))}$$

This behaves the exact reverse of (3) ($\frac{1}{1+\exp(-y_i \cdot h(x_i))} = 1 - \frac{1}{1+\exp(y_i \cdot h(x_i))}$) and clearly its a wrong approximation.

Answer : NO

4.2 Problem 2

The problem is a replication of Problem 1. It simply involves replacing $h(x_i)$ by $w^T x_i + b$ for all x_i in the training dataset. All the trends remain the same as mentioned above. Hence, one possible optimization problem in this case is:

$$\{w^*, b^*\} = \arg\min_{w,b} \sum_{i=1}^{N} \max\{0, 1 - y_i(w^T x_i + b)\}$$

4.3 Problem 3

Only **options (a) and (b)** are correct. This is because for negative values of $yf(x)$ (i.e. the prediction is different from the true label), the loss $\ell(yf(x))$ must be high. On the other hand, when the predicted and true labels have the same sign, then the loss must be low. Both (a) and (b) satisfy this, while (d) and (e) are complementary to them, and so they don't work. Finally, (c) has a low loss value in both cases, which is not desirable either. These trends are similar to what we explained in problems 1 and 2.

4.4 Problem 4

4.4.1 Part (i)

In our Binary classification Task, the Training set is *imbalanced*, i.e, D_{Train} has $\eta_{Train} = 90\%$ examples of class $+1$, and 10% of class -1. Here we are working with family of constant models $\mathcal{H} = \{+1, -1\}$

We only have access to the Training Set, so to minimize the Training error, we choose $\hat{h}$ to be the *Majority Estimator*, i.e, the Dominant class. So $\hat{h}(x) = +1$

4.4.2 Part (ii)

$$Error(h^*) - Error(\hat{h})$$

If the test set is well balanced (i.e. positive and negative labels both form 50% of the data), then any constant classifier will have a test error of 50%. Since h^* and $\widehat{h}$ are both chosen from the constant hypothesis class, so their error will be equal, giving the result Error(h^*) − Error$(\widehat{h}) = 0$.

4.4.3 Generalized Problem

Suppose the composition of $\mathcal{D}_{\text{Train}}$ is η_{Train} for class $+1$, and the distribution for $\mathcal{D}_{\text{Test}}$ is η_{Test} for class $+1$. Now, we train the constant classifier $\widehat{h}$ on $\mathcal{D}_{\text{Train}}$, and let h^* be the best possible constant classifier. Then $\widehat{h} = \begin{cases} +1 & \text{if } \eta_{\text{Train}} \geq 0.5 \\ -1 & \text{if } \eta_{\text{Train}} < 0.5 \end{cases}$ and $h^* = \begin{cases} +1 & \text{if } \eta_{\text{Test}} \geq 0.5 \\ -1 & \text{if } \eta_{\text{Test}} < 0.5 \end{cases}$.

Then $\text{Error}(h^*) = \min(\eta_{\text{Test}}, 1 - \eta_{\text{Test}})$, and $\text{Error}(\widehat{h}) = \begin{cases} 1 - \eta_{\text{Test}} & \text{if } \eta_{\text{Train}} \geq 0.5 \\ \eta_{\text{Test}} & \text{if } \eta_{\text{Train}} < 0.5 \end{cases}$.

Thus, the final result is

$$\text{Error}(h^*) - \text{Error}(\widehat{h}) = \begin{cases} 2 \times \eta_{\text{Test}} - 1 & \text{if } \eta_{\text{Train}} \geq 0.5, \eta_{\text{Test}} < 0.5 \\ 1 - 2 \times \eta_{\text{Test}} & \text{if } \eta_{\text{Train}} < 0.5, \eta_{\text{Test}} \geq 0.5 \\ 0 & \text{if } (\eta_{\text{Train}} - 0.5) \times (\eta_{\text{Test}} - 0.5) \geq 0 \end{cases}$$

4.5 Problem 5

Weighting is introduced to handle bias from the training dataset. In particular, if we simply minimize the unweighted loss function, then there is a bias towards the majority class of training dataset, assuming the situation when the training set is highly imbalanced, while the test set is nearly balanced. In such a situation, a good approach is to give more weight to the loss from the minority class. Thus the r_i's need to be divided in the opposite ratio, i.e. 0.9 for labels -1 and 0.1 for labels $+1$.

An important point to keep in mind is that this weighing can only be done is we know about the distribution of the test set to some extent. In practice, this is generally not the case, as the test set is only revealed at the end of training. Most ML models work under the assumption that the validation set will be close enough to the test set, to generate reasonable models.

4.6 Problem 6

4.6.1 Part (i)

The table alongside shows the behavior of the cross-entropy loss for the four different situations, i.e., for each of the two values that y can take, whether $h(x)$ is greater than the threshold or not. Observing the loss behavior in each case, we can say that it is indeed a valid loss function.

y	h(x)	Loss
+1	$> 0.5 (\rightarrow 1)$	$\rightarrow 0$
0	$> 0.5 (\rightarrow 1)$	$\rightarrow \infty$
+1	$\leq 0.5 (\rightarrow 0)$	$\rightarrow \infty$
0	$0.5 (\rightarrow 0)$	$\rightarrow 0$

4.6.2 Part (ii)

For the training loss to be 0, when $y_i = 1$, we need $h(x_i)$ also as 1, and similarly, when $y_i = 0$, we need $h(x_i) = 0$. However, for the sigmoid function to attain 0 or 1, we need the argument $w^T x$ to become either $-\infty$ or ∞. Since x is finite (unit-norm), this is only possible when $||w|| = \infty$.

4.6.3 Part (iii)

If $h(x_i) = \dfrac{1}{1+e^{w^T x_i}}$, then $1 - h(x_i)$ is the same as the original model. Thus, we simply need to make the change $h(x_i) \longrightarrow 1 - h(x_i)$ in the loss function. This gives our new cross-entropy loss as

$$\sum_{(x_i,y_i)\in\mathcal{D}_{\text{Train}}} -\{y_i \log(1 - h(x_i)) + (1 - y_i) \log h(x_i)\}$$

4.7 Problem 7

- if $\tau = 0$, model always predicts +1
- if $\tau = 1$, model always predict 0
- if $\tau = 0.5$, the model predicts a mixture of $\{0, 1\}$, but the accuracy is not optimal.

The optimal value of τ can be determined as:

$$\tau^* = argmin_{\tau\in[0,1]} \sum_{(x_i,y_i)\in\mathcal{D}_{\text{Test}}} (Sign(h(x_i) - \tau) \neq y_i)$$

4.8 Problem 8

1. $f(x) = w_1 * x_1 + w_2 * x_2$ Linear in x and linear in w
   First property
   $f(\overline{w}, \overline{x}) = w_1x_1 + w_2x_2$
   $f(\overline{w}, \overline{x} + \overline{y}) = w_1(x_1 + y_1) + w_2(x_2 + y_2)$
   $f(\overline{w}, \overline{x} + \overline{y}) = w_1x_1 + w_1y_1 + w_2x_2 + w_2y_2$
   $f(\overline{w}, \overline{x} + \overline{y}) = f(\overline{w}, \overline{x}) + f(\overline{w}, \overline{y})$
   Second property
   $f(\overline{w}, \alpha\overline{x}) = w_1\alpha x_1 + w_2\alpha x_2$
   $f(\overline{w}, \alpha\overline{x}) = \alpha(w_1x_1 + w_2x_2)$
   $f(\overline{w}, \alpha\overline{x}) = \alpha f(\overline{w}, \overline{x})$
   Similarly, it can be shown for $\overline{w}$

2. $f(x) = w_1x_1^2 + w_2x_2^3$
   Linear in w and non-linear in x

3. $f(x) = w_1 \ln x_1 + w_2e^{x_2}$
   Linear in w and non-linear in x

4. $f(x) = x_1 \ln w_1 + x_2 e^{w_2}$
   Linear in x and non-linear in w

5. $f(x) = w^T x \quad w, x \in \mathbb{R}^d$
   Linear in both w and x

6. $f(x) = w^T x + b$
   If we introduce another notation $x_0 = 1$ such that $\overline{x} = [x_0 x_1 ... x_d]^T$ and call $w_0 = b$ then the above equation becomes $f(x) = [w_0 w_1 ... w_d][x_0 x_1 ... x_d]^T$
   $f(x) = w_{new}^T x_{new}$ is linear now in w_{new} and x_{new}

4.9 Problem 9

L2 loss is defined as:

$$E = \Sigma_{i=1}^{N} (y_i - b - w_1 x_1)^2$$

Taking its gradient with respect to b:

$$\frac{\partial E}{\partial b} = \Sigma_{i=1}^{N} 2 * (y_i - b - w_1 x_1)(-1)$$

Equating this to 0, we get

$$\begin{aligned} & \Sigma_{i=1}^{N} (y_i - b - w_1 x_1) = 0 \\ \implies & n\overline{y} - nb - n\overline{x} w_1 = 0 \\ \implies & b + \overline{x} w_1 = \overline{y} \\ \implies & [\overline{x}] b + [\overline{x}^2] w_1 = \overline{x}\ \overline{y} \end{aligned} \tag{1}$$

Taking L2 Loss gradient with respect to w_1:

$$\frac{\partial f}{\partial w_1} = \Sigma_{i=1}^{N} (y_i - b - w_1 x_1)(-x_i)$$

Equating this to 0, we get

$$\begin{aligned} & \Sigma x_i y_i - b(\Sigma x_i) - w_1(\Sigma x_i^2) = 0 \\ \implies & \Sigma x_i y_i - bn\overline{x} - w_1(\Sigma x_i^2) = 0 \\ \implies & [n\overline{x}] b + [\Sigma x_i^2] w_1 = \Sigma x_i y_i \\ \implies & [\overline{x}] b + [\frac{\Sigma x_i^2}{n}] w_1 = \frac{\Sigma x_i y_i}{n} \end{aligned} \tag{2}$$

Subtracting Equation (2) from Equation (1) gives

$$w_1 = \frac{\frac{\Sigma x_i y_i}{n} - \overline{x}\ \overline{y}}{\frac{\Sigma x_i^2}{n} - \overline{x}^2}$$

$$b = \overline{y} - \overline{x}w_1$$

By the way equation of line was: $y = b + w_1 x$

4.10 Problem 10

We will **design** the **X** matrix in the following way with each x_i vector being the data sample vector,

$$X = \begin{bmatrix} \mathrm{x}_1^T & \mathrm{x}_2^T & \mathrm{x}_3^T & . & . & . & . & . & \mathrm{x}_n^T \end{bmatrix}^T$$

Now with matrix multiplication:

$$Xw = \begin{bmatrix} \mathrm{x}_1^T w & \mathrm{x}_2^T w & \mathrm{x}_3^T w & . & . & . & . & . & \mathrm{x}_n^T w \end{bmatrix}^T$$

$$Y = \begin{bmatrix} \mathrm{y}_1 & \mathrm{y}_2 & \mathrm{y}_3 & . & . & . & . & . & \mathrm{y}_n \end{bmatrix}^T$$

Now,

$$||Xw - Y||_2^2 = || \begin{bmatrix} \mathrm{x}_1^T w - y_1 & \mathrm{x}_2^T w - y_2 & \mathrm{x}_3^T w - y_3 & . & . & . & . & . & \mathrm{x}_n^T w - y_n \end{bmatrix}^T ||^2$$

$$\implies ||Xw - Y||_2^2 = \Sigma (X_i^T w - y_i)^2$$

Now,

$$L(w) = ||Xw - Y||_2^2 = (Xw - Y)^T (Xw - Y)$$

$$\implies L(w) = w^T X^T X w - w^T X^T Y - Y^T X W + Y^T Y$$

Now derivative of above equation for loss in terms of **w** gives,

$$\nabla_w [w^T X^T X w - w^T X^T Y - Y^T X W + Y^T Y] = 0$$

$$\implies 2X^T X w - X^T Y - X^T Y + 0 = 0$$

$$\implies X^T X w = X^T Y$$

4.11 Problem 11

So X is the *design matrix*, of training inputs stacked as rows and X.shape = [N, d]
We can prove that if X is a full column rank Matrix, then X^TX is invertible.

- X is full column rank, means all its coulmns are Linearly Independent, $rank(X) = d$, So its Null Space is $\{\mathbf{0}\}$, or simply $Xv = 0 \iff v = 0$
- X^TX is a dxd matrix, to show that X^TX is invertible, we need to show that its *Null Space* $\mathcal{N}(X)$ is $\{\mathbf{0}\}$/ dimension 0.
- So say there exist a $\mathbf{v}$, so that $X^TX\mathbf{v} = 0 \implies \mathbf{v^T}X^TX\mathbf{v} = 0 = (X\mathbf{v})^T(X\mathbf{v}) = ||X\mathbf{v}||_2^2$, So it implies $X\mathbf{v} = \mathbf{0}$, which inturn implies $\mathbf{v} = 0$
- Hence Nullspace of X^TX is $\{\mathbf{0}\}$, so it is full row/column rank. Hence X^TX is invertible.

4.12 Problem 12

M is a Positive Definite Matrix $\iff \mathbf{v^T}M\mathbf{v} > 0 \; \forall \mathbf{v} \in \mathrm{R^n}\backslash\{\mathbf{0}\}$
Determinant of Positive Definite matrix is Non Zero, So its determinant exists.

We need to show $(X^TX + \lambda I)$ is Positive Definite, so consider an arbitary $\mathbf{v} \neq 0$
$\mathrm{v}^T(X^TX + \lambda I)v = v^TX^TXv + \lambda v^Tv = (Xv)^TXv + \lambda v^Tv = ||Xv||_2^2 + \lambda||v||_2^2 > 0$
Hence, $X^TX + \lambda I$ is a Positive Definite Matrix

4.13 Problem 13

Linear regression problem can be modelled as $Y_i = w^Tx_i + \epsilon_i$, $\epsilon \sim \mathcal{N}(0, \sigma^2)$.
So $Y_i \sim \mathcal{N}(f(x_i), \sigma^2)$, where $f(x_i) = w^Tx_i$

The given dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$

We need to maximize the Likelihood of Observing the generated y_i data with our Model, and then try to Maximise the Likelihood. $L(w) = P(y|x)$

$$P(y_i|x_i) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_i - f(x_i))^2}{2\sigma^2}}$$

So the Likelihood, $L(w) = \prod_i P(y_i|x_i)$, we assume all samples are independently drawn

$$L(w) \propto \prod_i e^{-\frac{(y_i - f(x_i))^2}{2\sigma^2}} = e^{\frac{-1}{2\sigma^2}\sum_i (y_i - f(x_i))^2}$$

So maximizing Likelihood means minimizing $-log(L(.))$ Negative log-likelihood, so

$$argmax_w L(w) \iff argmin_w \sum_i (y_i - f(x_i))^2 = \mathbf{L2}\ loss$$

Hence we can see that Maximizing the likelihood is equivalent to minimizing the L2 loss for the regression problem.