CSCC11 Week 1 Notes

Introduction to Machine Learning:

- Machine Learning (ML) is a set of tools that allows computers to learn how to perform a task by giving examples of how it should be done.

- ML is usually broken into 2 steps:
  1. Training: A model is learned from a collection of training data.
  2. Application/Testing: The model is used to make decisions about some new test data.

- ML is all about "fitting" a function.

- Some fields in ML include:
  1. Supervised Learning: Here, the training data is labelled with correct answers. (Input/output) pairs Supervised learning has 2 main sub-topics:
     a) Regression: The target/output is a real-valued number.
     b) Classification: The target/output are discrete labels.
  2. Unsupervised Learning: Here, we are given a collection of unlabelled data, which we wish to analyze and discover patterns within. There are 3 main sub-topics:
     a) Density Estimation: Here, we estimate the parameters of a distribution that generated the data.
     b) Dimensionality Reduction: Reduce the dimension of high dimensional data (E.g. images)

c) Clustering: Here, we are grouping data with similar patterns together.

3. Reinforcement Learning: Here, an agent (I.e. a robot or controller) seeks to learn the optimal actions to take based on the state of the world, and hence the consequences of past actions.

4. Active Learning: Here, obtaining data is expensive and so an algo must determine which training data to acquire.

5. Meta Learning: Here, models that learned from tasks with large training sets or many tasks with moderate amounts of training data can be used to help constrain learning on related problems that have relatively small data sets.

I.e. Machines observe how different ML approaches perform on a wide range of learning tasks and then learn from this experience to learn new tasks much faster.

It is "learning to learn."

Math Formulas:

1. Mean Squared Error:

- $L(y, \hat{y}) = \left(\sum_{n=1}^{N} (\hat{y}_n - y_n)^2 \right) \cdot \frac{1}{N}$

y is the target and $\hat{y}$ is the prediction.

$\hat{y} = f(x)$

- The Mean Squared Error (MSE) tells you how close a regression line is to a set of points. It does this by taking the differences or distances from the points to the regression line and squaring them. Note: The distances are the "errors". We square them to remove negative numbers and to give more weight to larger differences. The lower the MSE, the better the forecast.

2. Likelihood Function:
   - If $X_1, \ldots X_n$ are an iid (independent and identically distributed) sample from a population with pdf or pmf $f(x|\theta_1, \ldots, \theta_k)$, the likelihood is defined by

$$L(\theta|x) = L(\theta_1, \ldots, \theta_k | X_1, \ldots, X_N)$$
$$= \prod_{i=1}^{N} f(X_i|\theta)$$

   - The likelihood function helps us find the best distribution of the data given a particular value of some feature or some situtation in the data.

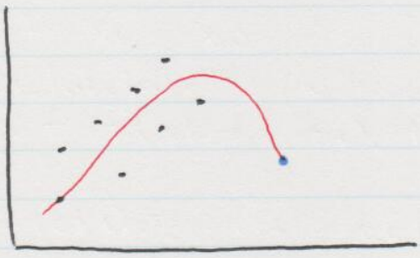
     I.e. Likelihood means to increase the chances of a particular situtation to occur by varying the characteristics of the dataset distribution.

   - A collection of random variables is iid if each r.v. has the same probability distribution as the others and all are mutually independent.
     E.g.
     1. Tossing a coin 10 times and counting the num of H.
     2. Rolling a die 5 times and counting how many times it lands on 6.

Overfitting and Underfitting:

- Often, data sets have outliers/noise that we don't want.
- Overfitting occurs when a model learns the data too well. In this case outliers/noises are also picked up. The problem is that these noises don't apply to new data and negatively impact the model's ability to generalize.

Fig. Suppose we have the data set below:



Here, our model is overfitted as it picked up the outlier.

Note: We don't want to fit perfectly as there could be noise.

- Underfitting refers to a model that can neither model the training data nor generalize to new data.

- This is how to choose your model:
  - Have a training set that's split into training and validation and a test set. Then:
  1. Train model on training.
  2. Test on validation.
  3. Repeat 1 and 2 until you find the model that performs best for the objective.
  4. Deploy such model on test set.

Validation is part of your training set that you set aside.
Test set is data not seen before.

Discriminative and Generative Models:

- Discriminative models learn the boundaries between classes or labels in a dataset.
- The goal of discriminative models is to separate one class from another.
- Discriminative models use $P(y|x)$

  Note: $P(y|x) = \frac{P(x|y) \cdot P(y)}{P(x)}$
- Generative models model the actual distribution of each class.
- They use $P(x, y)$
- E.g.

  One day, a father takes his 2 kids, A and B, to a zoo to see a lion and an elephant only. After seeing both animals, the father takes out a picture and asks both children if the picture is of a lion or elephant.

  kid A draws a picture of a lion and elephant on a piece of paper based on what he saw in the zoo. Then, after comparing his pictures with his father's, he says "A lion".

  kid B only knows the differences based on diff properties of the 2 animals and also says "A lion".

  kid A is an example of a generative model while kid B is an example of a discriminative model.

Probability vs Likelihood

Probability:

- Probability refers to finding the chance of something given a sample distribution of data.

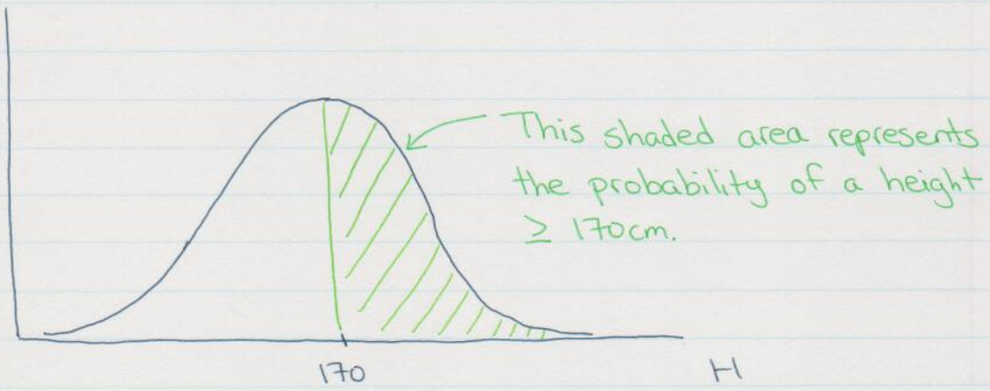
- E.g. Suppose we have a dataset about the heights of people in country A, and that the mean is 170cm and the standard dev is 3.5.

If we want to find the probability of people with height equal or over 170cm, we would do

$P(\text{height} \geq 170\text{cm} \mid \mu = 170, \sigma = 3.5)$

↑ Symbol for mean

↑ Symbol for std dev



With probability, we vary the LHS and keep the RHS constant.

I.e. $P(\theta \mid \beta)$

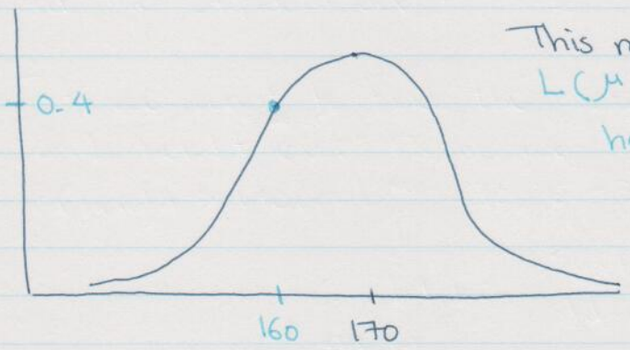
↑ Vary this

↑ keep this constant

Likelihood:

- Likelihood involves calculating the best distribution or best characteristics of a data given a particular feature value or situtation.

- E.g. Using the prev example, we want to find the likelihood of a height being 160cm



This means that
$L(\mu = 170cm, \sigma = 3.51$
$height = 160cm) = 0.4$

With likelihood, we vary the dataset features (mean and std dev) in order to find the maximum likelihood.

- Likelihood means to increase the chance of a particular situtation to happen by varying the characteristics of the dataset.

- I.e. Probability is used to find the chance of occurrence of a particular situtation while likelihood is used to generally maximize the chances of a particular situtation to occur.

Max Likelihood Examples

Steps:

1. Take the product of the pdf or pmf.
2. Take the log of that to turn it into a sum.
3. Take the derivative and set it to 0.

Examples:

1. The PMF of Y is $f(y) = \theta(1-\theta)^{y-1}$, $y = 1, 2, 3, \ldots$
   The observed values for y are 2, 2, 1, 1, 5, 1, 1, 1, 2, 1.
   Find the MLE of θ.

Soln:

$N = 10$

$$L(\theta) = \prod_{i=1}^{10} f(y)$$

$$= \prod_{i=1}^{10} \theta(1-\theta)^{(y-1)}$$

$$\ln(L) = \sum_{i=1}^{10} \ln(\theta(1-\theta)^{(y-1)})$$

$$= \sum_{i=1}^{10} \ln(\theta) + \sum_{i=1}^{10} \ln(1-\theta)^{y-1}$$

$$= \sum_{i=1}^{10} \ln(\theta) + \sum_{i=1}^{10} \ln(1-\theta)(y^{(i)}-1)$$

$$= 10\ln(\theta) + \sum_{i=1}^{10} y^{(i)} \ln(1-\theta) - \sum_{i=1}^{10} \ln(1-\theta)$$

$$= 10\ln(\theta) + 17\ln(1-\theta) - 10\ln(1-\theta)$$
$$= 10\ln(\theta) + 7\ln(1-\theta)$$

$$\frac{\partial L}{\partial \theta} = \frac{10}{\theta} - \frac{7}{1-\theta} = 0$$

$$0 = 10(1-\theta) - 7\theta$$
$$= 10 - 10\theta - 7\theta$$
$$= 10 - 17\theta$$
$$\frac{10}{17} = \theta$$

CSCC11 Week 2 Notes

Linear Regression:

1. Introduction:

- Is a linear approach to modelling the relationship btwn a dep var and an indep var.

2. 1D Linear Regression:

- We want to find $y = f(x) + \varepsilon$ where:
  a) $f(x) = wx + b$
  - w = weight
  - b = bias

  = "w" and = "b" are the parameters of f.

  b) ε is the error term (I.e. noise)

- We want to find/estimate "w" and "b" s.t. $f(x)$ fits the training data as well as possible.

  The training data is just a set of input/output pairs. I.e. $\{(x_1, y_1), \ldots, (x_n, y_n)\}$.

- One way to do this is to min the vertical dist between the actual value and the predicted value. We can do this using the Least Squares Method.

- Let $e_i = y_i - f(x_i)$
  $= y_i - (wx_i + b)$

  The loss function, $L(w,b)$, is equal to $\sum_{i=1}^{n} (e_i)^2$

  $= \sum_{i=1}^{n} (y_i - (wx_i + b))^2$

Note: We need to square the error because of possible negative values.

- Finding the line that minimizes the squared error is equivalent to solving for "w" and "b" that minimize $L(w,b)$. This can be done by setting the derivatives of L w.r.t these parameters to 0 and then solving.

$$\frac{\partial L}{\partial b} = -2 \sum_{i=1}^{N} (y_i - (wx_i + b)) = 0$$

$$0 = \sum_{i=1}^{N} (y_i - wx_i - b)$$

$$= \sum_{i=1}^{N} y_i - \sum_{i=1}^{N} wx_i - \sum_{i=1}^{N} b$$

$$= \sum_{i=1}^{N} y_i - w \sum_{i=1}^{N} x_i - bN$$

$$bN = \sum_{i=1}^{N} y_i - w \sum_{i=1}^{N} x_i$$

$$b^* = \frac{\sum_{i=1}^{N} y_i}{N} - \frac{w \sum_{i=1}^{N} x_i}{N}$$

$$= \hat{y} - w\hat{x}$$

We'll define $\hat{x}$ and $\hat{y}$ as the avgs of the x's and y's respectively.

Now, we can rewrite $L(w, b)$ as:

$$\sum_{i=1}^{N} (y_i - (wx_i + (\hat{y} - w\hat{x})))^2$$

$$= \sum_{i=1}^{N} (y_i - (wx_i + \hat{y} - w\hat{x}))^2$$

$$= \sum_{i=1}^{N} (y_i - \hat{y} - (wx_i - w\hat{x}))^2$$

$$= \sum_{i=1}^{N} ((y_i - \hat{y}) - w(x_i - \hat{x}))^2$$

Using this new form of L, we can try to solve for w.

$$\frac{\partial L}{\partial w} = -2 \sum_{i=1}^{N} ((y_i - \hat{y}) - w(x_i - \hat{x}))(x_i - \hat{x})$$

$$= 0$$

$$0 = \sum_{i=1}^{N} (y_i - \hat{y})(x_i - \hat{x}) - w(x_i - \hat{x})^2$$

$$= \sum_{i=1}^{N} (y_i - \hat{y})(x_i - \hat{x}) - w \sum_{i=1}^{N} (x_i - \hat{x})^2$$

$$w \sum_{i=1}^{N} (x_i - \hat{x})^2 = \sum_{i=1}^{N} (y_i - \hat{y})(x_i - \hat{x})$$

$$w^* = \frac{\sum_{i=1}^{N} (y_i - \hat{y})(x_i - \hat{x})}{\sum_{i=1}^{N} (x_i - \hat{x})^2}$$

w^* and b^* are the least-square estimates for the parameters of the linear regression.

3. Multi-Dimensional Inputs:

- Now, let $x \in R^D$ (x is now a $1 \times D$ column vector.)
  $y \in R$

$$f(x) = w^T x + b = \sum_{j=1}^{D} w_j x_j + b$$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{bmatrix}$$

← 1 data point, each with D features

- To make $f(x)$ the result of a dot product, we can add b as the last element in w and add a 1 to x.

I.e. $w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_D \\ b \end{bmatrix}$, $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \\ 1 \end{bmatrix}$

Then, $f(x) = w^T x$

- We can define our loss function $L(w)$, to be $\sum_{i=1}^{N} (y - w^T x_i)^2$.

$$L(w) = \sum_{i=1}^{N} (y - w^T x_i)^2$$

This is the 2-norm squared

$$= \|\vec{y} - \tilde{X}w\|_2^2 \text{ where}$$

$$\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

$$\tilde{X} = \begin{bmatrix} — & \tilde{x}_1^T & — \\ — & \tilde{x}_2^T & — \\ & \vdots & \\ — & \tilde{x}_N^T & — \end{bmatrix}$$

$$= \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1d} \\ 1 & x_{21} & x_{22} & \cdots & x_{2d} \\ & & \vdots & & \end{bmatrix}$$

- Now, we want to find w^*.

$$L(w) = \|\vec{y} - \tilde{X}w\|_2^2$$
$$= (\vec{y} - \tilde{X}w)^T (\vec{y} - \tilde{X}w)$$
$$= (\vec{y}^T - w^T\tilde{X}^T)(\vec{y} - \tilde{X}w)$$
$$= \vec{y}^T y - \vec{y}^T\tilde{X}w - w^T\tilde{X}^T\vec{y} + w^T\tilde{X}^T\tilde{X}w$$
$$= w^T\tilde{X}^T\tilde{X}w - 2\vec{y}^T\tilde{X}w + \vec{y}^T\vec{y}$$

Recall: $(a \pm b)^T = a^T \pm b^T$

$(ab)^T = b^T a^T$

$(abc)^T = c^T b^T a^T = c^T(ab)^T$

$$\frac{\partial L(w)}{\partial w} = 2(\tilde{X}^T\tilde{X})w - 2\tilde{X}^T\vec{y} + 0 = 0$$

$$0 = (\tilde{X}^T\tilde{X})w - \tilde{X}^T\vec{y}$$

$$(\tilde{X}^T\tilde{X})w = \tilde{X}^T\vec{y}$$

$$w = \underbrace{(\tilde{X}^T\tilde{X})^{-1}\tilde{X}^T}_{\text{Pseudo Inverse}}\vec{y}$$

$$\therefore w^* = (\tilde{X}^T\tilde{X})^{-1}\tilde{X}^T\vec{y}$$

Note: w^* must be a min bc L is convex w.r.t w.

Note: Because finding the inverse is expensive, another approach we can do to find the min is to use the gradient descent.

$$w_{i+1} = w_i - \alpha\left(\frac{\partial L(w)}{\partial w}\right)$$

← Gradient descent formula

↑ Learning rate/step size

Non-linear Regression:

1. Introduction:

- We can introduce non-linearity by adding/using a basis function.

2. Basis Function Regression:

- In basis function regression:

$$f(x) = \sum_{i=1}^{N} w_i b_i(x)$$

E.g.

Linear Model: $b_0(x) = 1, \; b_1(x) = x$

$$f(x) = w_0 b_0(x) + w_1 b_1(x)$$
$$= w_1 x + w_0$$

Polynomial Model: $b_k(x) = x^k$

$$f(x) = \sum_{k=1}^{N} w_k x^k$$

Radial Basis Function (RBF): $b_k(x) = \exp\left(\frac{(x-\mu_k)^2}{2\sigma_k^2}\right)$,

μ_k and σ_k are hyperparameters meaning that they are expensive to choose.

$$f(x) = \sum_{k=1}^{N} w_k \exp\left(\frac{(x-\mu_k)^2}{2\sigma_k^2}\right)$$

Furthermore, μ_k is the center of the basis function and σ_k^2 is the width of the basis function.

- The polynomial model and RBF are 2 common choices of basis functions.

- The polynomial model is more susceptive to outliers but can extrapolate while RBF is not as susceptive to noise but can't extrapolate.

  RBF - local fit
  Polynomial - global fit

- In the above basis functions, there are hyperparameters to decide:
  Polynomial: Degree of polynomial
  RBF: # of RBFs, μ and σ

Generally speaking, how you choose the hyperparameters is to do the training and validation and use the loss to decide the best parameters.

For RBF, we can use the following guidelines

a) To pick the center:
   1. Place the centers uniformly spaced containing the data. This is simple but can lead to empty regions with basis functions and will have an impractical number of data points in higher dimensional input spaces.

2. Place one center at each data point. This is used more often since it limits the num of centers needed although it can be expensive if the num of data points is too big.

3. Cluster the data and use one center for each cluster.

b) To pick the width:

1. Manually try diff values and pick the best.

2. Use the avg squared dist to neighbouring centers, scaled by a constant. This approach also allows you to use diff widths for diff basis functions and it allows the basis functions to be spaced non-uniformly.

- Directly minimizing squared-error can lead to overfitting. There are 2 important solns to this:
1. Adding prior knowledge. ← We'll use this for now.
2. Handling uncertainity.

- In many cases, there is some sort of prior knowledge we can use. A common assumption is that the underlying function is likely to be smooth. We can use regularization. This means we add an extra term, often to encourage smooth models.

- Least Squares / Ordinary Least Squares:

$$L(w) = \|\tilde{y} - Bw\|_2^2$$

- Regularized Least Squares:

$$L(w) = \underbrace{\|\tilde{y} - Bw\|_2^2}_{\text{Data term}} + \underbrace{\lambda \|w\|_2^2}_{\text{Smoothness term}} \quad (\lambda \in \mathbb{R}^+)$$

The smoothness term forces your parameters to be smaller, causing your function to be smoother.

This is also called L2 Regularization or Ridge Regression.

We can also use the L1 term as regularizer. This is called Lasso Regression.

$$L(w) = \|\vec{y} - Bw\|_2^2 - \lambda \|w\|_1$$

↑ Lagrange multiplier variant of a constrained optimizer.

Linear and Non-linear Regression Notes

Linear Regression:

- 1D Case:
- We want to find $y = f(x) + \varepsilon$ where:
  a) $f(x) = wx + b$
  (w: Weight, b: bias)

  "w" and "b" are the parameters of "f".

  b) ε is the error term (noise)

- We want to estimate "w" and "b" s.t. $f(x)$ fits the training data as well as possible.

  The training data is a set of input/output pairs, $\{(x_1, y_1), \ldots, (x_n, y_n)\}$.

  Note: x_i's can be a scalar or vector.

- One way to do this is to minimize the vertical dist btwn the actual value and the predicted value. We can do this using Least Squares Method.

  Let $e_i = y_i - f(x_i)$
  $= y_i - (wx_i + b)$

  Note: x_i and y_i are from training data.

  The loss function, $L(w,b)$, is equal to $\sum_{i=1}^{n} (e_i)^2$

  $$= \sum_{i=1}^{n} (y_i - wx_i - b)^2$$

- We need to square the error because of negative values.

- Finding the line that minimizes the squared error is equivalent to solving for "w" and "b" that minimizes $L(w,b)$. This can be done by setting the derivatives of L w.r.t "w" and "b" to 0 and then solving.

For b:

$$\frac{\partial L}{\partial b} = -2 \sum_{i=1}^{n} (y_i - wx_i - b) = 0$$

$$0 = \sum_{i=1}^{n} y_i - w \sum_{i=1}^{n} x_i - \sum_{i=1}^{n} b$$

$$= \sum_{i=1}^{n} y_i - w \sum_{i=1}^{n} x_i - bn$$

$$bn = \sum_{i=1}^{n} y_i - w \sum_{i=1}^{n} x_i$$

$$b^* = \frac{\sum_{i=1}^{n} y_i}{n} - w \frac{\sum_{i=1}^{n} x_i}{n}$$

$$= \hat{y} - w\hat{x}$$

For w:

First, we can rewrite L by substituting $\hat{y} - w\hat{x}$ for b.

$$L = \sum_{i=1}^{n} (y_i - wx_i - (\hat{y} - w\hat{x}))^2$$

$$= \sum_{i=1}^{n} ((y_i - \hat{y}) - w(x_i - \hat{x}))^2$$

$$\frac{\partial L}{\partial w} = -2 \sum_{i=1}^{n} ((y_i - \hat{y}) - w(x_i - \hat{x}))(x_i - x) = 0$$

$$0 = \sum_{i=1}^{n} (y_i - \hat{y})(x_i - \hat{x}) - w(x_i - \hat{x})^2$$

$$= \sum_{i=1}^{n} (y_i - \hat{y})(x_i - \hat{x}) - \sum_{i=1}^{n} w(x_i - x)^2$$

$$w^* = \frac{\sum_{i=1}^{n} (y_i - \hat{y})(x_i - \hat{x})}{\sum_{i=1}^{n} (x_i - \hat{x})^2}$$

- Multi-Dimensional Inputs:
- Now, let $x \in \mathbb{R}^D$. I.e. $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{bmatrix}$ ← 1 data point with D features.

- $f(x) = w^T x + b$

$$= \sum_{i=1}^{n} w_i x_i + b$$

We can add b to w and 1 to x to "absorb" b.

$$w = \begin{bmatrix} b \\ w_1 \\ \vdots \\ w_D \end{bmatrix}, \quad x = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_D \end{bmatrix}$$

Now, $f(x) = w^T x$

$$= \sum_{i=1}^{n} w_i x_i$$

- $L(w) = \sum_{i=1}^{N} (y_i - w^T x_i)^2$

$= \|y - Xw\|^2$ where $y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ x_N \end{bmatrix}$

$$X = \begin{bmatrix} — x_1^T — \\ — x_2^T — \\ \vdots \\ — x_N^T — \end{bmatrix}$$

$\|y - Xw\|^2$

$$L(w) = (y - Xw)^T (y - Xw)$$
$$= (y^T - w^T X^T)(y - Xw)$$
$$= y^T y - y^T Xw - w^T X^T y + w^T X^T Xw$$
$$= w^T X^T Xw - 2y^T Xw + y^T y$$

- Now, to find w^*:

$$\frac{\partial L}{\partial w} = 2(X^T X)w - 2X^T y + 0 = 0$$

$$0 = (X^T X)w - X^T y$$
$$(X^T X)w = X^T y$$
$$w^* = \underbrace{(X^T X)^{-1} X^T}_{\text{Pseudo Inverse}} y$$

Note: $(X^T X)$ isn't always invertible.

Non-linear Regression:

- In basis function regression, we introduce a basis function denoted by $b_k(x)$.

- 2 common basis functions are the polynomials and radia basis functions (RBF).

- For polynomical, we have $b_k(x) = x^k$.

$$f(x) = \sum_{i=1}^{N} w_i b_i(x) \leftarrow \text{General basis function representation.}$$
$$= \sum_{i=1}^{N} w_i x^i$$

- For RBF, we have $b_k(x) = \exp\left(-\frac{(x-\mu_k)^2}{2\sigma_k^2}\right)$

μ_k is the center of the basis function.

σ_k^2 is the width of the basis function.

- Examples of polynomial basis function:

1. Let $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$ be data points.

$$f(x) = \sum_{i=0}^{k=2} w_i b_i(x)$$

$$= \sum_{i=0}^{k=2} w_i x^i$$

$$= w_0 x^0 + w_1 x + w_2 x^2$$

$$= w_0 + w_1 x + w_2 x^2$$

$$= \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ & \vdots & \\ 1 & x_N & x_N^2 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$$

Basis function matrix, B

$B_{i,j} = b_j(x_i)$

$B \in R^{N,K}$

Each row in B corresponds to 1 data point.

2. Let $\{([x_{11} \dots x_{1D}], y_1), ([x_{21} \dots x_{2D}], y_2), \dots ([x_{N1} \dots x_{ND}], y_N)\}$ be the data points.

$$f(x) = \sum_{i=0}^{k=2} w_i b_i(x)$$

$$= \sum_{i=0}^{k=2} w_i x^i$$

$$= \begin{bmatrix} 1 & [x_{11}, x_{12}, \dots, x_{1D}]^1 & [x_{11}, x_{12}, \dots, x_{1D}]^2 \\ & \vdots & \\ 1 & [x_{N1}, x_{N2}, \dots, x_{ND}]^1 & [x_{N1}, x_{N2}, \dots, x_{ND}]^2 \end{bmatrix}$$

Basis function matrix

$$\begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$$

$$- L(w) = \sum_i (y_i - f(x_i))^2$$

$$= \sum_i \left(y_i - \sum_j w_j b_j(x) \right)^2$$

$$= \| y - Bw \|^2$$
$$= (y - Bw)^T (y - Bw)$$
$$= (y^T - w^T B^T)(y - Bw)$$
$$= w^T B^T B w - 2 y^T B w + y^T y$$

$$\frac{\partial L}{\partial w} = 2(B^T B)w - 2B^T y + 0 = 0$$

$$w^* = (B^T B)^{-1} B^T y$$

- Regularized Least Squares:
- $L(w) = \underbrace{\|y - Bw\|^2}_{\text{Data Term}} + \underbrace{\lambda \|w\|^2}_{\text{Smoothness Term}}$

$$= (y - Bw)^T (y - Bw) + \lambda w^T w$$
$$= w^T B^T B w - \quad + \lambda w^T w + y^T y$$

$$\frac{\partial L}{\partial w} = 2B^T Bw - 2B^T y + 2\lambda w = 0$$

$$0 = B^T Bw - B^T y + \lambda w$$
$$= (B^T B + \lambda I) w - B^T y$$
$$(B^T B + \lambda I) w = B^T y$$
$$w^* = \underbrace{(B^T B + \lambda I)^{-1}}_{\text{Always invertible}} B^T y$$

CSCC11 Week 3 Notes

Bias - Variance Tradeoff:

- How we train a model:
  1. Collect data
  2. Train a model M
  3. Predict using a trained model

Note: Data collection is a sampling process. It can be sampled from a data generating distribution.

To determine the generalizability of M, we want to see how well M predicts on avg when we sample D from the distribution. We care about the expected squared error shown below.

- $E[(y-\hat{f})^2]$

$= E[(f+\varepsilon-\hat{f})^2]$ Recall: $y = f(x) + \varepsilon$

$= E[(f+\varepsilon-\hat{f}+E[\hat{f}]-E[\hat{f}])^2]$

$= E[(f-E[\hat{f}])^2] + E[\varepsilon^2] + E[(E[\hat{f}]-\hat{f})^2] +$
$2E[(f-E[\hat{f}])\varepsilon] + 2E[\varepsilon(E[\hat{f}]-\hat{f})] +$
$2E[(E[\hat{f}]-\hat{f})(f-E[\hat{f}])]$

$= (f-E[\hat{f}])^2 + E[\varepsilon^2] + E[(E[\hat{f}]-\hat{f})^2] + 2(f-E[\hat{f}])E[\varepsilon]$
$+ 2E[\varepsilon]E[E[\hat{f}]-\hat{f}] + 2E[E[\hat{f}]-\hat{f}](f-E[\hat{f}])$

$= (f-E[\hat{f}])^2 + E[\varepsilon^2] + E[(E[\hat{f}]-\hat{f})^2]$

$= \text{Bias}[\hat{f}]^2 + \text{Var}[\varepsilon] + \text{Var}[\hat{f}]$

$= \underbrace{\text{Bias}[\hat{f}]^2}_{\text{Bias}} + \underbrace{\sigma^2}_{\text{Bayes Error}} + \underbrace{\text{Var}[\hat{f}]}_{\text{Variance}}$

- Some properties:

1. Let X be a discrete random variable with a finite number of outcomes $x_1, x_2, \ldots, x_k$ occurring with probabilities $p_1, p_2, \ldots, p_k$ respectively. The expected value of X, denoted as $E[X]$ is

$$E[X] = \sum_{i=1}^{k} x_i p_i$$
$$= x_1 p_1 + x_2 p_2 + \ldots + x_k p_k$$

Note: Expected value is another term for the mean.

2. Variance is the avg of how much each point differs from the mean.

$$\text{Var}[X] = E[X^2] - E[X]^2$$

3. Standard deviation measures how far apart a group of numbers is from the mean. A low std dev indicates that the values tend to be close to the mean while a high std dev indicates that the values are spread out over a wider range. It is denoted as σ.

$$\sigma = \sqrt{\text{Var}[X]}$$

- The bias of a model is the error that comes from the potentially wrong prior assumptions in the model. These assumptions cause the model to miss important info about the relationship btwn the feature and targets for a ML problem.

- The variance of a model is the error that comes from the model's sensitivity to small variations in the training data.

- Models with a high bias are often too simple and lead to underfitting.

  Models with a high variance are often too complex and lead to overfitting.

- Baye's Error / Irreducible Error is something we have no control over. It is the error that is introduced from the chosen framing of the problem and may be caused by unknown variables.

Ordinary Least Squares Regression Through Maximum Likelihood:

- Recall: $y = f(x) + \varepsilon$
  - $\varepsilon \sim N(0, C)$ where C is the covariance matrix.

Suppose we're dealing with linear models.

$y = w^T x + \varepsilon$

y: Random Var; ε: Random Var

$y \sim N(w^T x, C)$

The goal is to find w^* s.t. $p(y|w,x)$ is maximized

$p(y|w,x)$: Max likelihood function

- Given $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$, we have $P(y_1, y_2, \ldots, y_n | w, x_1, \ldots, x_n)$. Further, the data set is an i.i.d, so

Gaussian/Normal Distribution

$$P(y_1, \ldots, y_n | w, x_1, \ldots, x_n) = \prod_{i=1}^{n} P(y_i | w, x_i)$$

$$= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi |c|^d}} \exp\left(-\frac{1}{2}(y_i - w^T x_i)^T C^{-1} (y_i - w^T x_i)\right)$$

Precison Matrix

Take the log likelihood

$$= \sum_{i=1}^{n} \log\left(P(y_i | w, x_i)\right)$$

$$= \sum_{i=1}^{n} \log\left(\frac{1}{\sqrt{2\pi |c|^d}}\right) + \frac{-1}{2}(y_i - w^T x_i)^T C^{-1} (y_i - w^T x_i)$$

Classification:

- Supervised Learning: The label/output y is a category.
  - If there's 2 categories, it's called Binary Classification.
  - If there's > 2 categories, it's called Multiclass Classification.
  - With multi-label classification, multiple labels may be assigned to 1 instance.

Note: Binary classification means classifying the elements into 2 groups.

Multi-label classification means classifying the elements into more than 2 groups.

We'll be mainly focused on binary classification.

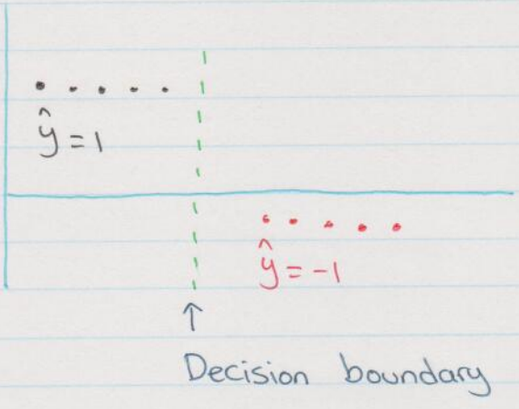
- Let's try linear regression for the first model.

$$y = w^T x$$

- A reasonable decision rule is:

$$\hat{y} = \begin{cases} 1, & \text{if } f(x) \geq 0 \\ -1, & \text{if otherwise} \end{cases}$$

← This is equivalent to $\hat{y} = \text{sgn}(w^T x)$



- The decision boundary separates the 2 groups. It is also called a hyperplane.

- In 1D, the decision boundary is a threshold.
  In 2D, it is a line.
  In 3D, it is a plane.

- For our loss function, we can no longer use least squares. We have to use something else.

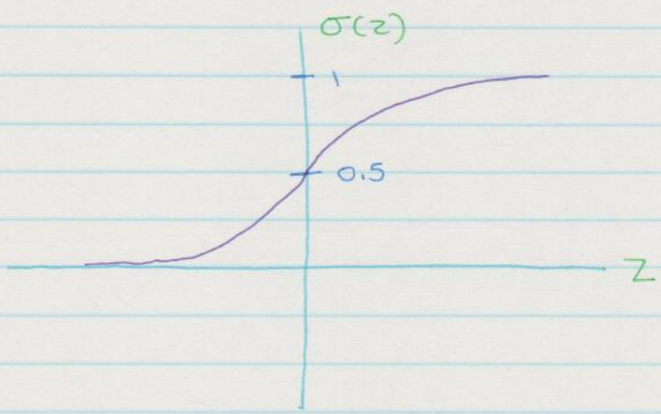
Something else we can use is ↗ Zero/One Loss function.

$$L_{0\text{-}1}(f(x), y) = \begin{cases} 0, & \text{if } f(x) = y \\ 1, & \text{if } f(x) \neq y \end{cases}$$

However, this isn't great either.

- Instead, we can use this: the sigmoid / logistic function, denoted as σ.

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



- Now, let's define the class probability input:

$$P(C_1|x) = \frac{1}{1+\exp(-(w^T x))}$$
$$= \sigma(w^T x)$$

$$P(C_2|x) = 1 - \sigma(w^T x)$$

- The decision boundary is given by:
$P(C_1|x) = P(C_2|x)$ ← This is where we're not sure if the point belongs in $P(C_1|x)$ or $P(C_2|x)$.

$$\Rightarrow \frac{P(C_1|x)}{P(C_2|x)} = 1$$

If $\frac{P(C_1|x)}{P(C_2|x)} > 1$, then choose C_1.

If $\frac{P(C_1|x)}{P(C_2|x)} < 1$, then choose C_2.

- $$\frac{P(C_1|x)}{P(C_2|x)} = \frac{\sigma(w^T x)}{1-\sigma(w^T x)}$$

$$= \frac{\frac{1}{1+\exp(-w^T x)}}{1 - \frac{1}{1+\exp(-w^T x)}}$$

$$= \frac{\frac{1}{1+\exp(-w^T x)}}{\frac{1+\exp(-w^T x) - 1}{1+\exp(-w^T x)}}$$

$$= \frac{1}{\exp(-w^T x)} = \exp(w^T x)$$

Continuing from the prev page, I'll take the ln of both sides.

$$\ln\left(\frac{P(C_1|x)}{P(C_2|x)}\right) = w^T x = \underbrace{\ln(1)}_{\text{This is because we earlier said that our decision boundary is } \frac{P(C_1|x)}{P(C_2|x)} = 1} = 0$$

$\Rightarrow$ Linear Decision Boundary

- Given $P(C_1|x) = \frac{1}{1+\exp(-w^T x)}$, w is the parameter. Hence, we want to find w^* that gives us the best performance.

- Given the training set $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, we want to model $P(y|x)$ using $P(\underbrace{y_1, \ldots, y_N}_{\text{Training set}} | x_1, \ldots, x_N, w)$ s.t. the likelihood is maximized.

  I.e. $\arg\max_w \left(P(y_1, \ldots, y_N | x_1, \ldots, x_N, w)\right)$

- Recall: $(x_i, y_i) \overset{iid}{\sim} D$, so we can write $\arg\max_w \left(P(y_1, \ldots, y_N | x_1, \ldots, x_N, w)\right)$ as

  $\arg\max_w \prod_{i=1}^{N} P(y_i | x_i, w)$ Note: Since $y_i \in \{0, 1\}$, we can assume the output follows a Bernoulli Distribution

  $\Rightarrow \arg\max_w \prod_{i=1}^{N} P(C_1|x_i, w)^{y_i} \left(1 - (P(C_1|x_i, w))\right)^{1-y_i}$

$$\Rightarrow \arg\max_{w} \prod_{i=1}^{N} P(C=1|X_i,w)^{y_i} \cdot P(C=2|X_i,w)^{1-y_i}$$

Then, if we take the negative log, we get:

$$L(w) = \arg\min_{w} -\sum_{i=1}^{N} \left(\begin{array}{l} (y_i \log(P(C=1|X_i,w))) + \\ (1-y_i)\log(P(C=2|X_i,w)) \end{array} \right)$$

$$\Rightarrow \arg\min_{w} -\sum_{i=1}^{N} \left(\begin{array}{l} y_i \log(P(C=1|X_i,w)) + \\ (1-y_i)(\log(1- (P(C=1|X_i,w)))) \end{array} \right)$$

Now, if we let $P_i = P(C=1|X_i,w)$, we get:

$$L(w) = \arg\min_{w} -\sum_{i=1}^{N} \underbrace{(y_i \log P_i + (1-y_i)(\log(1-P_i)))}_{\text{This term is called Cross-Entropy}}$$

Cross-entropy is a convex function.
Cross-entropy loss has no closed form soln, so we need to use gradient descent.

- Recall: $w_i^{(t+1)} = w_i^{(t)} - \alpha \left(\frac{\partial L(w)}{\partial w_i} \right)$ (Step size)

  is the formula for gradient descent.

- We can stop the procedure when:
  1. $w^{(t+1)} \approx w^{(t)}$
  2. Reached max iterations
  3. Validation loss is going up

$$-\frac{\partial P_i}{\partial w_i} = \frac{\partial P_i}{\partial \sigma(w^T x_i)} \cdot \frac{\partial \sigma(w^T x_i)}{\partial w_i}$$

$$= \sigma(w^T x_i)(1-\sigma(w^T x_i))$$

$$-\frac{\partial L(w)}{\partial w} = -\sum_{i=1}^{N} \left(y_i(1-P_i)x_i - (1-y_i)P_i x_i \right)$$

$$= -\sum_{i=1}^{N} (y_i - P_i)x_i$$

$$= -\sum_{i=1}^{N} (y_i - \sigma(w^T x))x_i$$

- Regularized Logistic Regression:

$$L(w) = -\sum_{i=1}^{N} (y_i \log P_i + (1-y_i)\log(1-P_i)) + \underbrace{\lambda \|w\|_2^2}_{L2}$$

Note: $\|w\|_2^2 < c$

- The new gradient descent is:

$$w_i^{(t+1)} = w_i^{(t)} - \alpha \left(\frac{\partial L(w)}{\partial w_i} \right) - \alpha \lambda w_i^{(t)}$$

- How to tune λ:
  1. Partition training data into training set and validation set.
  2. Use training set to learn the weights $(w_1, \ldots, w_n)$.
  3. Use validation set to estimate the tuning parameter. We typically choose the tuning parameter when the model performs the best on the validation set.
  4. Note: Never use test data for tuning the hyper-parameters.

Normal Distribution:

- Also called Gaussian Distribution.
- The general form of its probability density function (pdf) is:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

- The general notation of a normal distribution is: $N(\mu, \sigma^2)$.

- Recall: On page 3, we said: $y \sim N(w^T x, c)$. Hence, subbing $w^T x$ for μ and c for σ^2, we get:

$$\frac{1}{\sqrt{2\pi|c|^d}} \exp\left(-\frac{1}{2}(y_i - w^T x_i)^T c^{-1} (y_i - w^T x_i)\right)$$

Bernoulli Distribution:

- Can be thought of as a model for the set of possible outcomes of a single experiment that asks a Yes/No question.

- PDF: $\begin{cases} q = 1-p, & \text{if } k=0 \\ p, & \text{if } k=1 \end{cases}$

or as

$$p^k (1-p)^{1-k}$$

More Bias-Variance Notes:

- A model with a high bias fails to properly fit the training data. (Underfitting)

- A model with a high variance fits the training data so well, it memorizes it and fails to correctly apply what is has learned to new real-world data. (Overfitting)

- The optimal model has enough bias to avoid memorizing the training data and enough variance to actually fit the patterns in the training data.

Basic Probability Notes

Terminology:

1. The probability of A, denoted as P(A) is a real number between 0 and 1 inclusive.

   $P(A) = 0$ means that A is false.
   $P(A) = 1$ means that A is true.
   $0 < P(A) < 1$ correspond to varying degrees of certainty.

2. The joint probability of A and B, denoted as $P(A, B)$ is the prob that both A and B are true.

   $$P(A, B) = P(B, A)$$

3. The conditional probability of A given B, denoted as $P(A|B)$ is the prob we would assign to A if we knew B to be true.

   $$P(A|B) = \frac{P(A, B)}{P(B)}$$

   Rearranging the above eqn, we get:
   $P(A, B) = P(A|B) \cdot P(B)$ ← Product Rule

   Similarly, we have
   $$P(B|A) = \frac{P(B, A)}{P(A)} \rightarrow P(B, A) = P(B|A) \cdot P(A)$$
   $$\rightarrow P(A, B) = P(B|A) \cdot P(A)$$

4. Sum Rule: $P(A) + P(\bar{A}) = 1$

The prob of a statement being true and the prob of a statement being false sum to 1.

Now, suppose we have a set of mutually exclusive statements, A_i, exactly one of which must be true, we have:

$$\sum_i P(A_i) = 1$$

5. Conditioning Rule: $P(A|B) + P(\bar{A}|B) = 1$

More Probability Formulas:

1. $P(A|B)P(B) + P(\bar{A}|B)P(B) = P(B)$

Proof:

$$\begin{aligned} LHS &= P(A|B)P(B) + P(\bar{A}|B)P(B) \\ &= P(B)\underbrace{(P(A|B) + P(\bar{A}|B)}_{=1 \text{ By conditioning rule}} \\ &= P(B) \\ &= RHS \end{aligned}$$

2. $P(A, B) + P(\bar{A}, B) = P(B)$

Proof:

$$\begin{aligned} LHS &= P(A, B) + P(\bar{A}, B) \\ &= P(A|B)P(B) + P(\bar{A}|B)P(B) \\ &= P(B) \quad \text{(See above eqn)} \\ &= RHS \end{aligned}$$

3. $\sum_i P(A_i \mid C) = 1$

Here, the A_i's are a mutually exclusive set, exactly one of which must be true.

4. $P(A, B \mid C) = P(A \mid B, C) \cdot P(B \mid C)$

5. $P(B) = \sum_i P(A_i \mid B)$ ← Marginalization

Independence:

- 2 statements, A and B are independent iff $P(A, B) = P(A) \cdot P(B)$.

- Furthermore, if A and B are independent, then $P(A \mid B) = P(A)$.

Proof:

$$\begin{aligned} \text{LHS} &= P(A \mid B) \\ &= \frac{P(A, B)}{P(B)} \\ &= \frac{P(A) \cdot \cancel{P(B)}}{\cancel{P(B)}} \\ &= P(A) \\ &= \text{RHS} \end{aligned}$$

Random Variable:

- A random variable (r.v.) is a variable taking on numerical values determined by the outcome of a random phenomenon.

- A discrete random variable has a countable number of possible values.

- A continuous random variable takes on all the values in some interval of numbers.

- Discrete random variables use Probability Mass Function (PMF) to describe their distributions.

The notation $P_X(x)$ refers to the PMF of the r.v. X.
$P_X(x) = P(X = x)$

Properties of PMFs:
1. $0 \leq P_X(x) \leq 1$ (PMFs are always btwn 0 and 1, inclusive)
2. $\sum_{-\infty}^{\infty} P_X(x) = \sum_{x \in X} P_X(x) = 1$

- Continuous r.v. use Probability Density Function (PDF) to describe their distributions.

- We use the notation $f_X(x)$ to refer to the PDF of a r.v. X.

Properties of PDFs:
1. $0 \leq f_X(x)$
2. $\int_{-\infty}^{\infty} f_X(x)\,dx = \int_{x \in X} f_X(x)\,dx = 1$
3. $P(a \leq x \leq b) = \int_a^b f_X(x)\,dx$

- For discrete random variables, the expected value, denoted as $E(X)$, is:

$$E(X) = \sum_i P(r_i)\, x_i$$

where r_i is the outcome of X_i.

The variance denoted as $Var(X)$ is:

$$Var(X) = \sum_{i=1}^{n} P(r_i) \cdot (x_i - \mu)^2$$

where μ is the expected value
I.e. $\mu = \sum_i P(r_i)\, x_i$

The standard deviation, denoted as σ, is:

$$\sigma = \sqrt{Var(X)}$$

- For continuous random variables:

$E(X) = \int x p(x)\, dx$ where $p(x)$ is the PDF.

$Var(X) = \int x^2 p(x)\, dx - \mu$ where $p(x)$ is the PDF and μ is the expected value.

$$\sigma = \sqrt{Var(X)}$$

PMF Notes

Introduction:

- Used for discrete r.v.
- Denoted as $P_X(x)$ or $P(X=x)$
- E.g. Consider the probability of rolling a die.
  $P(X=1) = \frac{1}{6}$

Properties:

1. $\sum_i P(X=x) = 1$

2. $0 \leq P(X=x) \leq 1$

Distributions:

1. Bernoulli Distribution:
   - $P(X=1) = p$
   - $P(X=0) = 1-p$
   - $\begin{cases} q = 1-p, & \text{if } k=0 \\ p, & \text{if } k=1 \end{cases}$
   - $p^k (1-p)^{1-k}$

2. Binomial Distribution:
   - Is the number of successes in a sequence of n independent experiments, each asking a yes-no question and each with its own boolean-valued outcome.
   - Note: When n=1, we get Bernoulli Distribution.
   - Let k be the number of successes.
     Let n be the number of trials.

$$\binom{n}{k} p^k q^{n-k}$$

p^k: Probability of success; q^{n-k}: Prob of failure

3. Multi-nomial Distribution:
   - Is a generalization of the binomial dist.
   - Is the number of successes in a seq of n indep experiments, each with k mutually exclusive outcomes having probability p_i.
   - $\frac{n!}{X_1!\, X_2! \dots X_k!} p_1^{X_1} p_2^{X_2} \dots p_k^{X_k}$

PDF Notes

Introduction:

- Used for continuous r.v.
- Denoted as $p(x)$ or $f(x)$

Properties:

1. $p(x) \geq 0 \quad \forall x$

2. $\int_{-\infty}^{\infty} p(x)\,dx = 1$

3. $P(a \leq x \leq b) = \int_{a}^{b} p(x)\,dx$

Distributions:

1. Uniform Distribution:

$$p(x) = \begin{cases} \frac{1}{x_1 - x_0}, & \text{if } x_0 \leq x \leq 1 \\ 0, & \text{otherwise} \end{cases}$$

- States that all values within a given range, $[x_0, x_1]$, are equally likely.

2. Normal / Gaussian Distribution:

$$p(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\left(\frac{(x-\mu)^2}{\sigma^2}\right)\right)$$

μ = mean
σ^2 = variance

CSCC11 Week 4 Notes

Logistic Regression Recap:

- Decision boundary, denoted as $d(x)$, splits the points into 2 classes, C_1 and C_2.

$d(x) = 0$ ← Decision Boundary

$d(x) > 1$ ← Correspond to C_1
$d(x) < 1$ ← Correspond to C_2

- If a point is on the decision boundary, then we are uncertain about the class for that point.

- If d is a linear function, then we have a linear decision boundary.

- Logistic regression has a linear decision boundary.

$$P(C=1|x) = \sigma(w^T x)$$
$$= \frac{1}{1 + \exp(-w^T x)}$$

$$P(C=2|x) = 1 - P(C=1|x)$$
$$= 1 - \sigma(w^T x)$$

k-Nearest Neighbour:

- In KNN, we classify an unknown point with the most common class "around" this point.
  Note: "around" means k closest points

- Can be used for both regression and clarification

- The set containing the k closest points to x in the training data is called the k-neighbourhood of x, denoted as $N_k(x)$.

- Let $y \in \{-1, 1\}$. Then: $f(x) = \text{sign}\left(\sum_{i \in N_k(x)} y_i\right)$

  Again, $N_k(x)$ is the set of k closest neighbours.

- There are 2 popular distance formulas to use for calculating "closeness":

  1. Manhattan Distance:
     - Let $x_1, x_2 \in X$ have p numeric features.
     - Then, the Manhattan Distance formula is:

       $$\sum_{j=1}^{p} |x_{1j} - x_{2j}|$$

     - E.g. Say $X_1 = (3,4)$, $X_2 = (5,3)$
       The Mahattan distance is:
       $|3-5| + |4-3|$
       $= 2+1$
       $= 3$

  2. Euclidean Distance:
     - Again, let $x_1, x_2 \in X$ have p numeric features
     - Then, the Euclidean Distance formula is:

       $$\sqrt{\sum_{j=1}^{p} (x_{1j} - x_{2j})^2}$$

     - E.g. Say $X_1 = (3,4)$ and $X_2 = (5,3)$. Then, we get:

       $\sqrt{(3-5)^2 + (4-3)^2}$
       $= \sqrt{(-2)^2 + 1^2}$
       $= \sqrt{4+1}$
       $= \sqrt{5}$

- Note: Both the Manhattan and Euclidean distances are part of the Minkowski distance or Lm Distance.

Let a and b $\in X$ and have p numeric features. Then, the Minkowski distance formula is:

$$\|a-b\|_q = \left(\sum_{j=1}^{p} |a-b|^q \right)^{1/q}$$

When $q=1$, we have the L_1 distance or Manhattan distance.

When $q=2$, we have the L_2 distance or Euclidean distance.

- Algorithm:
  1. For each test data, i, calculate the dist btwn data i and each training data point.
  2. Rank the dist from smallest to largest.
  3. Select the smallest k points.
  4. Calculate the frequency of these k chosen points in their classes.
  5. Return the class with the highest frequency as the predicted label.

- For classification in g groups, a majority vote is used:

$$\hat{h}(x) = \underset{\ell \in \{1,\dots,g\}}{\arg\max} \sum_{i : x^{(i)} \in N_k(x)} (y^{(i)} = \ell)$$

Furthermore, posterior probabilities can be calculated with:

$$\hat{\pi}_\ell(x) = \frac{1}{k} \sum_{i : x^{(i)} \in N_k(x)} (y^{(i)} = \ell)$$

- We choose k based on hyperparameter search in validation.

In general:
1. As $k \to \infty$, the decision boundary becomes smoother.
2. As $k \leftarrow 1$, the decision boundary becomes rough and is susceptible to noisy data.

Rule of thumb: $k < \sqrt{N}$, where N is the number of points.

- Final notes on kNN:
1. kNN is a lazy classifier. It has no real training step, it simply stores the complete data which are needed during prediction.
2. Its parameters are the training data.
3. kNN requires storing the whole training set.
4. As the number of points increase, it becomes more computationally expensive. However, when $k=1$, we don't need to train.

Furthermore, as the number of parameters increase with the number of training points, we call kNN a non-parametric model.

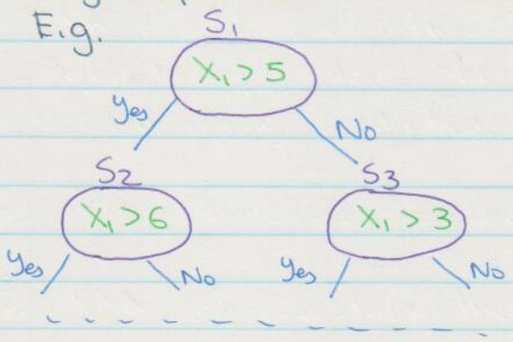
5. kNN is not based on any distributional or functional assumption and can, in theory, model data situtations of any complexity.

6. The smaller k is, the more rough / wiggly the decision boundary becomes.

7. The accuracy of kNN can be severely degraded by the presence of noisy or irrelevant features.

8. We often use the Euclidean distance to measure "closeness" but when the dimension is high, it will often become useless.

Decision Trees:
- Are usually binary trees but can be k-ary trees.
- The idea is to split the space in halves until we get a good prediction.
- E.g.



- Consider a binary tree:
  - Suppose it has M internal nodes. These are the split functions.
  - Suppose it has M+1 leaf nodes.
  - At an internal node, j, we define the split function $t_j(x): \mathbb{R}^d \to \{-1, 1\}$.

    If $t_j(x) = -1$, x is directed to the left child node
    If $t_j(x) = 1$, x is directed to the right child node

  - Leaf nodes have a categorical distribution over class label.

    Can represent as $P(y = c \mid \text{leaf node } j) = \frac{N_{jc}}{N_j}$

    where N_{jc} means the num of class c in node j and N_j means the num of points in node j.

    Note: $\frac{N_{jc}}{N_j}$ is the max likelihood estimate

- Learning the simplest/smallest decision tree is NP-hard or NP-complete problem.
  - We start from an empty decision tree.
  - We split on the best attribute.
  - Recurse

- To decide the best split value, we will choose it such that the data in the left/right children has the minimal possible uncertainty.

* There is a formula for conditional entropy, too.

$$H(y|x=x) = -\sum_{y} p(y|x) \log_2(p(y|x))$$

- Entropy is a measure of uncertainty.

$$H = -\sum_{c=1}^{k} P_c \log_2 P_c, \text{ where } P_c = P(y=c) \quad *$$

- Information gain (IG) is the reduction in entropy produced by partitioning the data according to a split test.

$$IG(y|x) = H(y) - H(y|x)$$

$$IG(D_j, t_j) = H(D_j) - \frac{NL}{NJ} H(D_L) - \frac{NR}{NJ} H(D_R)$$

NL is the num of data in the left child.
Nr is the num of data in the right child.

Note: Information gain is also called the mutual information of y and x.

- E.g.

$x = \{\text{Raining, Not raining}\}$
$y = \{\text{Cloudy, Not cloudy}\}$

	Cloudy	Not Cloudy
Raining	24/100	1/100
Not Raining	25/100	50/100

What is the entropy of cloudiness, given the knowledge of whether or not its raining?

Soln:

$$H(Y|X) = \sum_{x \in X} P(x) \cdot H(Y|X=x)$$

Conditional

$$= P(x = \text{raining}) \cdot H(Y|X=x) + P(x = \text{not raining}) \cdot H(Y|X=x)$$

Conditional

$$= \frac{25}{100} H(Y|X=x) + \frac{75}{100} H(Y|X=x)$$

$$\approx 0.75 \text{ bits}$$

$$H(Y|\text{raining}) = -\sum_{y} P(y|x) \log_2(P(y|x))$$

$$= -\frac{24}{25} \cdot \log_2\left(\frac{24}{25}\right) - \frac{1}{25} \cdot \log_2\left(\frac{1}{25}\right)$$

$$\approx 0.24229$$

$$H(Y|\text{Not raining}) = -\sum_{y} P(y|x) \cdot \log_2(P(y|x))$$

$$= -\frac{25}{75} \cdot \log_2\left(\frac{25}{75}\right) - \frac{50}{75} \cdot \log_2\left(\frac{50}{75}\right)$$

$$\approx 0.91829$$

about cloudiness

How much info do we get [redacted] by discovering whether it is raining?

Soln:

$$IG(y|x) = H(y) - H(y|x)$$
$$= 0.25$$

$$H(y) = -\sum_{c=1}^{k} P_c \cdot \log(P_c)$$

$$= -P(\text{cloudy}) \cdot \log(P(\text{cloudy})) - P(\text{not cloudy}) \cdot \log(P(\text{not cloudy}))$$
$$= -\frac{49}{100} \cdot \log\left(\frac{49}{100}\right) - \frac{51}{100} \cdot \log\left(\frac{51}{100}\right)$$
$$\approx 1$$

- Some attributes, such as age and height, are continuous. In this case, to find the threshold for splitting, we do:

1. Sort the continuous attribute in increasing order.
2. Calculate the middle values between adj values.
3. For each middle point, calculate the entropy.
4. Choose the threshold with max reduction in entropy.

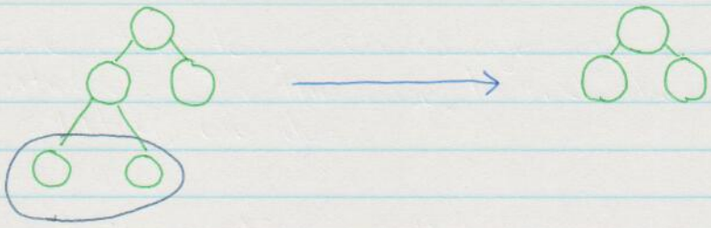
- As we grow the trees deeper, the model becomes more prone to overfitting.
  I.e. bias decreases while variance increases.

- There are 2 techniques we can use to reduce overfitting:
  1. Pruning the tree
  2. Ensemble method

- Pruning the tree:
  - One way we can prune the tree is by stop growing the tree if the info gain doesn't exceed a specific threshold. However, this is short sighted because a seemingly worthless split early on might be followed by a really good split later.

  - A better approach is to prune leaf nodes by combining them if the prediction accuracy on the validation set is not worse than the current accuracy.

E.g.



If removing these 2 nodes don't worsen the accuracy, remove them.

Algo:
Start at the leaves and recursively eliminate splits.
1. Evaluate the performance of the tree on validation data.

2. Prune the tree if the performance does not worsen.

- Ensemble Method:
  - Also known as Random Forests
  - The idea is that we learn a bunch of trees and since each tree is randomly generated, each tree will provide a diff prediction. We aggregate the predictions by averaging.
  - Random forest is a bootstrapping method and does bootstrap aggregation/bagging.
  - Algo:
    1. For each tree:
       a) Bootstrap the data.
          I.e. Sample a subset of the training data with replacement.
       b) Build the tree using a subset of the data and only use a subset of features
    2. During prediction, each tree gives an output. We take the avg prediction. (aggregation)
  - Hyper parameters:
    1. Num of data to sample.
    2. Num of features to look at.
       Rule of thumb: If D is the number of features, we look at $\sqrt{D}$ features.
    3. Num of trees.
  - When we do bootstrapping, we increase the bias, although slowly, but we reduce the correlation of the trees and hence the variance.

- How to choose the hyperparameters:
  - Recall that we want to use the model to make the prediction. (Generalization)

  - First, we will use 1 of the following techniques to search over the hyperparameter space:

    1. Grid Search
       E.g. If the learning rate $\alpha \in (0, 1]$, we can do $\alpha = (0.1, 0.2, \ldots, 1)$.

    2. Random Search

    3. Grad Student Descent

- Now, we'll go to validation.
  1. For a simple approach, we can use a metric that we care about.

     E.g. train using cross-entropy, validate using accurate

     The problem with this approach is that if we don't have many training data, it's hard to split.

  2. We can use k-Fold Cross Validation to bypass the limit/flaw of the first approach.

     Steps:
     1. Partition the training data into k partitions.
     2. For each partition, train on the remaining k-1 partitions.

3. Validate each partition.
   Say L_i is the validation of the i^{th} partition.

4. Average L_i for overall performance.

- One special case of k-Fold Cross Validation is Leave One Out Cross-Validation (LOOCV).
The idea behind this is with small training datasets, we have as much partitions as data points.
I.e. If we have N training points, we have N partitions.
The rest of LOOCV works like k-Fold Cross Validation.

LOOCV is useful for small datasets but very expensive.

- In general, if there are m hyperparameters each taking C values, there are C^m models.
If we do k-Fold Cross Validation, there are $k \cdot C^m$ models.

- While k-Fold Cross Validation is very simple and empirical way of comparing models, it has some issues:
  1. Can be time consuming and expensive.
  2. Because a reduced dataset is used for training, there must be sufficient training data so that all relevant phenomena of the problem exist in both the training and test data.
  3. It is safest to use a random partition to avoid the possibility that there are unmodeled correlations in the data.

Entropy and Information Gain Notes

Entropy:

- Entropy provides a measurement of uncertainity associated with a random variable or random process.

  Note: This is the definition of entropy under/in the context of information theory.

- For a discrete r.v. X with possible outcomes $x_1, x_2, \ldots, x_n$ which occur with probability $P(x_1), P(x_2), \ldots, P(x_n)$, the entropy of X, denoted as $H(X)$, is defined as:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_2 P(x_i)$$

  Note: $H(X) = E\left[-\log_2 P(X)\right]$, where E is expected value.

- Note: When $P(x_i) = 0$, for some x_i, we take $P(x_i) \log_2 P(x_i)$ to be 0, which is consistent with its limit.
  I.e. $\lim_{p \to 0} p \log(p) = 0$

- E.g. Suppose we flip a fair coin.

$$P_1 = P_2 = \frac{1}{2}$$

$$H(X) = -(2)\left(\frac{1}{2} \log_2\left(\frac{1}{2}\right)\right)$$
$$= -\log_2\left(\frac{1}{2}\right)$$
$$= -(\log_2(1) - \log_2(2))$$
$$= -(-1)$$
$$= 1$$

- E.g. Say we toss an unfair coin now.
Suppose the probability of getting heads is 70%.
Then, the entropy becomes:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_2 P(x_i)$$

$$= -(0.7 \cdot \log_2(0.7) + 0.3 \cdot \log_2 0.3)$$
$$\approx 0.8816$$ ← Since one side comes up more frequently, there is reduced uncertainty and hence entropy.

- E.g. The entropy of rolling a fair die is:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \cdot \log_2 P(x_i)$$

$$= -(6)\left(\tfrac{1}{6}\right)\left(\log_2\left(\tfrac{1}{6}\right)\right)$$
$$= -(\log_2(1) - \log_2(6))$$
$$= \log_2(6)$$
$$= 2.58$$ ← Since the probability of rolling a die (1/6) is smaller than the prob of flipping a coin (1/2), its entropy will be higher.

- We also have conditional entropy.
- $H(x|y) = -\sum_{i,j} P(x_i, y_j) \log_2 P(x_i|y_j)$

$$= -\sum_{j} P(y_j) \sum_{i} P(x_i|y_j) \log_2 P(x_i|y_j)$$

$$= \sum_{j} P(y_j) H(x|y_j)$$

Mutual Information:

- Mutual information is a measure of the info shared by 2 r.v.'s.
  I.e. It is a measure of how much about the state of one such var is known when it is conditioned on the state of the other.

- $I(x;y) = H(x) - H(x|y)$
  $= H(y) - H(y|x)$

- Also called information gain.

CSCC11 Week 7 Notes

Review of Baye's Rule:

- Baye's Rule: $P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$

Introduction to Estimation:

- Estimation is determining the values of some unknown variables from observed data. I.e. It is finding a single estimate of the value of an unknown parameter.

- There are 2 main types of estimation:
  1. Maximum Likelihood Estimation (MLE)
  2. Bayesian Estimation

Maximum Likelihood Estimation (MLE):

- Uses the frequentist view/ frequentist approach which says that an event's probability is the limit of its relative frequency in many trials.

- It uses $P(D|M)$ where D is the data and M the model. It's too focused on the training data.

- Recall the likelihood function if all data points are iid is $L(\theta|D) = f(D|\theta)$

$$= \prod_{i=1}^{N} f(x_i|\theta)$$

- To find the maximum likelihood we would do:

$$\hat{\theta} = \arg\max_{\theta} \left(\prod_{i=1}^{N} f(x_i|\theta) \right)$$

$$= \frac{\partial}{\partial \theta} \left(\prod_{i=1}^{N} f(x_i|\theta) \right)$$

However, this is not ideal because getting or taking the derivative of products is very messy.

Instead, we will use the log likelihood function. We can do this because:

1. The log of a product is the sum of logs.
2. Taking the log of any function may change its values but does not change where the max of that function occurs.

Log Likelihood: $\ell(\theta) = \ln(f(D|\theta))$

$$= \ln\left(\prod_{i=1}^{N} f(x_i|\theta) \right)$$

$$= \sum_{i=1}^{N} \ln(f(x_i|\theta))$$

$$\therefore \hat{\theta} = \arg\max_{\theta} \ell(\theta)$$

Bayesian Estimation:

- Uses the Bayesian View/Bayesian Approach where probability is defined as a degree of belief in an event. This degree of belief may be based on prior knowledge.

  I.e. The idea behind Bayesian Estimation is that before we've seen any data, we already have some prior knowledge about the distribution it came from. Such prior knowledge comes from experience or past experiments.

- $P(\theta|D) = \frac{P(D|\theta)\,P(\theta)}{P(D)}$

$$= \frac{P(D|\theta)\,P(\theta)}{\int P(D|\theta)\,P(\theta)\,d\theta} \longrightarrow P(D) = \int P(D|\theta)\cdot P(\theta)\,d\theta$$

$P(\theta|D)$ is called the posterior distribution and it describes our knowledge of the model based on both the data and the prior.

$P(D|\theta)$ is called the likelihood distribution/function describes the likelihood of the observations assuming the data is correct.

$P(\theta)$ is called the prior distribution and it describes our assumptions about the model without having observed any data.

$P(D)$ is called the evidence. It is usually expensive to calculate and is used to normalize the posterior.

- The problem with MLE is that if there's too little data, it can overfit.

- In MLE, the observations / data, D, is treated as random vars but the parameters, θ, are not.

  In Bayesian Estimation, both the data and observation are treated as random variables.

- Since MLE depends solely on observed data, it can overfit if the data is minimal.

  E.g. Suppose I flip a fair coin 3 times and on all 3 times, it lands on heads. Then, MLE tells you that $P(H)=1$ and $P(T)=0$.

  In situations where data is sparse, having some prior information can help.

  However, unreliable priors can lead to biased models, so make sure they are well defined.

- If the Bayesian prior is non-informative, for example it is uniform over all values, then the Bayesian prediction will be very similar, if not equal, to MLE predictions.

More on Estimation:

- Estimation: Interested in finding point estimates
- Inference: Interested in $P(M|D)$.
- In this course we focus on 3 types of estimations:
  1. MLE
  2. Maximum a Posteriori (MAP)
  3. Bayes' Estimate

Maximum a Posteriori (MAP):

- $\hat{\theta} = \arg\max_{\theta} P(\theta|D)$

$$= \arg\max_{\theta} P(D|\theta) \cdot \underbrace{P(\theta)}_{\text{Prior}}$$

- Note: We don't need $P(D)$ for MAP since it doesn't depend on θ and may therefore be treated as a constant for MAP estimation.
- Note: If the prior, $P(\theta)$, is uninformative (I.e. It is uniform), then MAP becomes MLE. I previously referenced this on page 4.
- $\hat{\theta} = \arg\max_{\theta} P(D|\theta) \cdot P(\theta)$

$$= \arg\min_{\theta} -\ln(P(D|\theta) \cdot P(\theta))$$

$$= \arg\min_{\theta} -\ln(P(D|\theta)) - \ln(P(\theta))$$

- Both MLE and MAP are optimization problems.

- Furthermore, both MAP and MLE ignore uncertainty in the parameters. This means we are choosing to put all our faith in the most probable model, but sometimes, this has surprising and undesirable consequences.

Bayes' Estimate:

- Formula: $\theta_{Bayes} = \int p(\theta|D) \cdot \theta \, d\theta$
$= E_{p(\theta|D)}[\theta]$

This subscript is used to be explicit about which distribution we're using.

Class Conditionals:

- Here, we're modelling the distribution over the features themselves. These models are called generative models.
- E.g. In the case of binary classification, suppose we have 2 mutually-exclusive classes C_1 and C_2. The prior probability of a data vector coming from class C_1 is $P(C_1) \equiv P(y = C_1)$ and $P(C_2) \equiv P(y = C_2)$
$\equiv 1 - P(y = C_1)$

Each class has its own distribution for the feature vectors, specifically $P(x|C_1)$ and $P(x|C_2)$. (These are the data likelihood distributions for the 2 classes.)

Then, the prob of a data point can be written as:

$P(x) = P(x, C_1) + P(x, C_2)$
$= P(x|C_1)P(C_1) + P(x|C_2)P(C_2)$

- If one had such a model, one could draw data samples from the model in the following way:
  1. One would randomly choose a class according to the probabilities $P(C_1)$ and $P(C_2)$.

  2. Then, conditioned on the class, one can sample a data point, x, from the associated likelihood distribution.

- For the learning problem we are given a set of labelled training data $\{(x_i, y_i)\}$ and our goal is to learn the parameters of the generative model.
  I.e. We want to:
  1. Estimate the conditional likelihood distribution for each class.
  2. Estimate $P(C_1)$ by computing the ratio of the number of elements of class 1 to the total number of elements.

- Once we have learned the parameters of our generative model, we perform classification by comparing the posterior class probabilities: $P(C_1|x) > P(C_2|x)$?

  If $P(C_1|x) > P(C_2|x)$, then we classify the input as to belonging to class C_1.

  If $P(C_1|x) < P(C_2|x)$, then we classify the input as C_2.

  If $P(C_1|x) = P(C_2|x)$, then it's on the decision boundary.

Equivalently, we can compare their ratio to 1.

I.e. $\frac{P(C_1|x)}{P(C_2|x)} = 1 \longrightarrow$ On decision boundary

$\frac{P(C_1|x)}{P(C_2|x)} > 1 \longrightarrow$ Classify as C_1

$\frac{P(C_1|x)}{P(C_2|x)} < 1 \longrightarrow$ Classify as C_2

- $P(C_i|x) = \frac{P(x|C_i) \cdot P(C_i)}{P(x)}$ ← Bayes' Rule

$$\frac{P(C_1|x)}{P(C_2|x)} = \frac{\frac{P(x|C_1) \cdot P(C_1)}{\cancel{P(x)}}}{\frac{P(x|C_2) \cdot P(C_2)}{\cancel{P(x)}}}$$

$$= \frac{P(x|C_1) \cdot P(C_1)}{P(x|C_2) \cdot P(C_2)}$$

- Note: These computations are typically done in the logarithmic domain as its faster and more numerically stable.

I.e. we check if $\log\left(\frac{P(C_1|x)}{P(C_2|x)}\right) > 0$

- Recap:
  - Class conditionals are used for generative models.
  - Want to model $P(x) = P(x, c_1) + P(x, c_2)$
  - $P(x, c_i) = \underbrace{P(c_i)}_{\text{Prior}} \cdot \underbrace{P(x|c_i)}_{\text{Likelihood}}$

$$= \underbrace{P(x)}_{\text{Evidence}} \cdot \underbrace{P(c_i|x)}_{\text{Posterior}}$$

  - $P(c_i|x) = \dfrac{P(x|c_i) \cdot P(c_i)}{P(x)}$

$$= \frac{P(x|c_i) \cdot P(c_i)}{\sum_{i=1}^{2} P(x|c_i) P(c_i)}$$

Can be used to classify input x

- How to find/learn the priors:

$$P(c_1) = \frac{\#\text{ of class } c_1}{N}$$

$$P(c_2) = \frac{\#\text{ of class } c_2}{N}$$

- How to find/learn the likelihoods:
  1. Partition based on class
  2. Use all x_i's s.t. $y_i = c_j$ to learn $P(x|c_j)$. This is usually done via MLE.

- Class conditions can:
  1. Make predictions
  2. Generate data
     - → 1. Sample class $\hat{c}$ from prior $p(c)$.
     - → 2. Sample data $\hat{x}$ from likelihood $p(x|\hat{c})$.

Gaussian Class Conditionals:

- Also called Linear Discriminate Analysis (LTA).
- Assume likelihoods to be Gaussian.
- For each class i, we model the i^{th} likelihood to be:

$$N(\vec{x}, \vec{M_i}, \Sigma_i) = \frac{1}{\sqrt{2\pi|\Sigma|^d}} \exp\left(\frac{-1}{2}(\vec{x}-\vec{M_i})\Sigma_i^{-1}(\vec{x}-\vec{M_i})\right)$$

(Mean: $\vec{M_i}$, Covariance: Σ_i)

Hence, $P(X_{1:N} | M, \Sigma) = \prod_{i=1}^{N} P(X_i | M, \Sigma)$

$$= \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi|\Sigma|^d}} \exp\left(\frac{-1}{2}(\vec{x}-\vec{M_i})\Sigma_i^{-1}(\vec{x}-\vec{M_i})\right)$$

Note: d is the dimensionality of the X_i's.

- It is often easier to find the min log likelihood.

$$L(M, \Sigma) = -\ln(p(X_{1:N} | M, \Sigma))$$
$$= -\sum_i \ln(p(X_i | M, \Sigma))$$
$$= \sum_i \frac{(X_i - M)^T \Sigma^{-1} (X_i - M)}{2} + \frac{N}{2}\ln|\Sigma| + \frac{Nd}{2}\ln(2\pi)$$

Solving for M and Σ involves setting $\frac{\partial L}{\partial M} = 0$ and $\frac{\partial L}{\partial \Sigma} = 0$.

$$M^* = \frac{\sum_i X_i}{N}, \quad \Sigma^* = \frac{\sum_i (X_i - M^*)(X_i - M^*)^T}{N}$$

- For the decision boundary, for simplicity, assume $P(C_1) = P(C_2) = \frac{1}{2}$.
Then, the decision boundary occurs at $d(x) = 0$.

Recall that:

$$d(x) = \ln\left(\frac{P(C_1|x)}{P(C_2|x)}\right)$$

$$= \ln\left(\frac{P(x|C_1) \cdot P(C_1)}{P(x|C_2) \cdot P(C_2)}\right) \frac{P(C_1)}{P(C_2)} \quad = 1 \text{ since } P(C_1) = P(C_2) = \frac{1}{2}$$

$$= \ln(P(x|C_1)) - \ln(P(x|C_2))$$

$$= \ln\left(\frac{1}{\sqrt{2\pi|\Sigma_1|}}\right) - \ln\left(\frac{1}{\sqrt{2\pi|\Sigma_2|}}\right) \leftarrow \text{Constants w.r.t. } \vec{x}$$

$$-\frac{1}{2}(\vec{x} - \vec{M_1})^T \Sigma_1^{-1} (\vec{x} - \vec{M_1}) + \frac{1}{2}(\vec{x} - \vec{M_2})^T \Sigma_2^{-1} (\vec{x} - \vec{M_2}) \quad \} \text{ Quadratic w.r.t. } \vec{x}$$

$$= 0$$

$d(x)$ is a quadratic function.

If $\Sigma_1 = \Sigma_2$, then we have a linear decision boundary.

CSCC11 Week 8 Notes

Review of Class Conditionals:

- Want to model $P(x) = P(x|C_1)P(C_1) + P(x|C_2)P(C_2)$
- If $P(C_1|x) > P(C_2|x)$, then we classify the input as belonging to class 1.
  If $P(C_1|x) < P(C_2|x)$, then we classify the input as belonging to class 2.
  $P(C_1|x) = P(C_2|x)$ is the decision boundary.
- Can also do $\frac{P(C_1|x)}{P(C_2|x)} > 1$

  or equivalently $\ln\left(\frac{P(C_1|x)}{P(C_2|x)}\right) > 0$
- Recall that $P(C_i|x) = \frac{P(x|C_i)P(C_i)}{P(x)}$

  Hence, $\ln\left(\frac{P(C_1|x)}{P(C_2|x)}\right) = \ln\left(\frac{P(x|C_1) \cdot P(C_1)}{P(x|C_2) \cdot P(C_2)}\right)$

  Note: The $P(x)$ term cancels out.
- We can model $P(x|C_1)$, $P(x|C_2)$ as Gaussians. Assuming d features for each input x (I.e. $x \in \mathbb{R}^d$), we have $O(d^2)$ parameters. This is from the Covariance Matrix.

Naive Bayes (NB)

- Naive bayes aims to simplify the estimation problem by assuming that the diff input features (the diff elements of the input vector) are conditionally independent.

I.e. $P(X|C) = \prod_{i=1}^{d} P(X_i|C)$

- With this assumption, rather than estimating 1 d-dimension density, we estimate d 1-dimension densities. This is important bc each 1D Gaussian only has 2 parameters (mean and variance) both of which are scalars. Hence, the model has 2d unknowns. In the Gaussian case, the NB model replaces the dxd covariance matrix by a diagonal matrix. The i^{th} entry is the variance of $X_i|C$.

Discrete Input Features

- In discrete NB, the inputs are a discrete set of features.

- Right now, we'll assume that each input either has or does not have each feature.

- Each data vector is described by a list of discrete features (I.e. $F_{1:d} = [F_1, ..., F_d]$) and for simplicity, we'll assume that each feature is binary (I.e. $f_i = \{0,1\}$).

- Consider this: we want to solve $P(F_1, F_2, F_3 | C=1)$.
  Without using naive bayes, we would get

Note: → $P(F_1, F_2, F_3 | C=1) = P(F_1 | F_2, F_3, C=1) \cdot P(F_2 | F_3, C=1) \cdot P(F_3 | C=1)$

This formula comes from the chain rule.

For $P(F_3 | C=1)$, Since we know $F_3 = \{0,1\}$, we can model it with 1 number.

For $P(F_2 | F_3, C=1)$, F_2 depends on F_3 and we know that F_3 has 2 possible values, we need to model 2 diff distributions.

For $P(F_1 | F_2, F_3, C=1)$, we need to model 4 diff distributions.

For d-dimensional binary inputs, there are $d(2^d - 1)$ parameters one needs to learn.

With Naive Bayes, only d parameters have to be learned.

This is because $P(F_{1:d} | C=j) = \prod_i P(F_i | C=j)$

- Continuing with NB's way:
  - Let $a_{ij} \equiv P(F_i = 1 | C=j)$
  - Let $b_j \equiv P(C=j)$ ← Prior
  - $P(C=j | F_{1:d}) = \dfrac{P(F_{1:d} | C=j) P(C=j)}{P(F_{1:d})}$

$$= \frac{\left(\prod_i P(F_i | C=j)\right)\left(P(C=j)\right)}{\sum_{\ell=1}^{k} P(F_{1:d}, C=\ell)}$$

$$= \frac{\left(\prod_{i:F_i=1} a_{ij} \prod_{i:F_i=0} (1-a_{ij})\right) b_j}{\sum_{\ell=1}^{k} \left(\prod_{i:F_i=1} a_{i\ell} \prod_{i:F_i=0} (1-a_{i\ell})\right) b_\ell}$$

- If we wish to find the class with max posterior prob, we only need to compute the numerator.

- The computation shown on the prev page can lead to underflow.
To avoid these issues, it's safer to perform the computations in the log-domain:

$$d_j = \left(\sum_{i:F_i=1} \ln a_{ij} + \sum_{i:F_i=0} \ln(1-a_{ij}) \right) + \ln b_j$$

$$r = \min_j d_j$$

$$P(C=j \mid F_{1:d}) = \frac{\exp(d_j - r)}{\sum_l \exp(d_l - r)}$$

- Now, consider we have N training vectors F_k, each, associated class label c_k.
with an

Suppose there are N_j training examples of class j and N examples total. Then

$b_j = \frac{N_j}{N} \longrightarrow b_j = \frac{N_j + \beta}{N + k\beta}$, where β is some constant and k is the num of classes

regularization

Suppose that class j has N_{ij} examples for which the i^{th} feature is 1. Then

$$a_{ij} = \frac{N_{ij}}{N_j} \longrightarrow a_{ij} = \frac{N_{ij} + \alpha}{N_j + 2\alpha}, \text{ for some small value } \alpha.$$

Regularization

- E.g. Suppose we observe N examples of class 0 and M examples of class 1, what is the probability of observing class 0?

Soln:

$$\prod_i P(C_i = j) = \left(\prod_{i:C_i=0} P(C_i=0)\right)\left(\prod_{i:C_i=1} P(C_i=1)\right)$$

$$= b_0^N \cdot b_1^M$$

$$= b_0^N (1-b_0)^M$$

$$L(b_0) = N\ln(b_0) + M\ln(1-b_0)$$

$$\frac{\partial L}{\partial b_0} = \frac{N}{b_0} - \frac{M}{1-b_0} = 0$$

$$0 = N(1-b_0) - Mb_0$$

$$= N - Nb_0 - Mb_0$$

$$-N = -Nb_0 - Mb_0$$

$$N = Nb_0 + Mb_0$$

$$b_0^* = \frac{N}{N+M}$$

Pros and Cons:

1. Pros
   - Works fast due to the conditional independence assumptions.
   - Works well with high-dimensional data

2. Cons
   - The assumptions may not be easy to satisfy.

CSCC11 Week 9 Notes

Unsupervised Learning:

- With supervised learning algos you want to produce the desired outputs for the given inputs. Also, you're given both the inputs and outputs during training.

- With unsupervised learning algos, only the inputs are given during training. The labels/outputs are unknown.

- Types of unsupervised learning:
  1. Dimension Reduction
  2. Clustering
  3. Data Density Modelling

Dimension Reduction:

- Increasing the num of features will not always improve performance. It may even lead to worse performance.

- Generally, the num of training data required increases exponentially with dimensionality to avoid overfitting.

- The goal is to choose an optimum set of features to lower dimensionality.

- Feature extraction: Finds a [redacted] set of new features through some mapping f(x) from existing features

$$\bar{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \xrightarrow{f(x)} Y = \begin{bmatrix} y_1 \\ \vdots \\ y_K \end{bmatrix}, \quad K \ll N$$

Mapping could be linear or non-linear.

- Feature Selection: Chooses a subset of the original features.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \rightarrow y = \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{ik} \end{bmatrix}$$

$k << N$

Intro to Principal Component Analysis (PCA):

- Is a technique for dimensionality reduction. It aims to find a low-dimensional representation of high dimensional data.

- Uses and motivations of PCA:

1. Visualization: High-dim data are extremely hard to visualize (I.e. To see how disjoint 2 diff categories of feature vectors might be or to see how noisy some measurements are.) PCA provides a way to project high-dim data onto 2^d or 3^d for purposes of easy visualization.

2. Pre-processing: Learning regression and classification models from high-dim data is often very slow and prone to overfitting. This is called the curse of dimensionality.

3. Compression: One of PCA's earliest uses was data compression. If one can find a low-dim representation of a high-dim image, for example, then one can use such a representation to store and transmit data more efficiently.

PCA Intuition and Steps:

- Assume each data point y_i has dimension p, so $y_i \in R^p$.
  The aim is to reduce the dimension.
  There are 2 ways to do so.

- Maximum Variance Formulation:
  - Find the orthogonal projection of the data into a lower dim linear space s.t. the variance of the projected data is maximised.

  - Consider the projection onto a 1-D space.
  - The linear projection is $u_1^T y_i$, $u_1 \in R^p$
  - For convenience, we choose the unit vector. I.e. $u_1^T u_1 = 1$
  - Assume y_i is standardized.
    ↳ $\frac{y - \bar{y}}{SD(y)}$ ← The mean of y.
  - Then, the sample variance of the projected data is

$$\frac{\sum_{i=1}^{N} (u_1^T y_i)^2}{N} = u_1^T S u_1$$

where $S = \frac{1}{N} \sum_{i=1}^{N} y_i \cdot y_i^T$

  - We want to max $u_1^T S u_1$ w.r.t u_1, given the constraint $u_1^T u_1 = 1$.

- We have $\arg\max_{U_1} \left(U_1^T S U_1 + \lambda_1 (1 - U_1^T U_1) \right)$

The soln is: $SU_1 = \lambda_1 U_1$

$\rightarrow U_1^T S U_1 = \lambda_1$ (Multiply both sides by U_1^T)

- The variance is max when U_1 = the eigenvector having the largest eigenvalue λ_1.
Such $U_1^T y_i$ is called the first principal component.

- We can extend this to k-Dim.
Let $U = [U_1, U_2, \ldots, U_k] \rightarrow U \in R^{p \times k}$
The k^{th} principal component for the i^{th} sample is $U_k^T y_i$. We can find/get this by ranking the eigenvalues.

The new (dimension reduced) data is:

$$x_i = \begin{bmatrix} U_1^T y_i \\ U_2^T y_i \\ \vdots \\ U_k^T y_i \end{bmatrix} = U^T y_i \in R^k$$

Principal components are uncorrelated:

$$E(U^T y y^T U) = U^T E(y y^T) U$$
$$= U^T (U D U^T) U$$
$$= (U^T U) D (U^T U)$$
$$= D \leftarrow \text{A diagonal matrix}$$

- Minimum Error Formulation:
  - Find a linear projection that min the error btwn the data points and their projection $y = Wx + b$ where W is a $p \times k$ matrix and x is a k-dim vector.

$$W = [w_1, \ldots, w_k]$$

  - One way to learn the model is to solve the following problem:

$$\arg\min_{W, b, \{x_i\}} \sum_i \| y_i - (Wx_i + b) \|^2$$

Subject to $W^T W = \underbrace{I_{k \times k}}_{\text{Identity matrix of size } k.}$

This constraint requires that we obtain an orthonormal mapping W.
I.e.

$$w_i^T w_j = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

Without this, the problem would be underconstrained.
E.g.
Assume $\alpha \neq 0$

$$y_i = w^* x_i$$
$$= (\alpha w^*)\left(\tfrac{1}{\alpha} x_i\right)$$

I can change w^* and x_i but still get the same error.

Steps to solve the problem:

1. Let $b = \frac{\sum_i y_i}{N}$

2. Compute the data co-variance matrix
   $C = \frac{\sum_i (y_i - b)(y_i - b)^T}{N}$

3. Let $VDV^T = C$ be the eigenvector decomposition of C. D is a diagonal matrix of eigenvalues (I.e. $D = \text{diag}(\lambda_1, \ldots, \lambda_d)$) and $V = [v_1, \ldots, v_d]$ contains the orthonormal eigenvectors.
   $V^T V = I_d$

4. Assume the eigenvalues are sorted from largest to smallest. If this is not the case, sort them along with their corresponding eigenvectors.

5. Let W be a matrix containing the first k eigenvectors.
   I.e. $W = [v_1, \ldots, v_k]$

6. Let $x_i = W^T(y_i - b) \; \forall i$

Representation and Reconstruction of New Data:

- Suppose we have learned a PCA model and are given a new y_{new} value. To estimate its corresponding x_{new} value, do:
  1. Minimize $\|y_{new} - (W x_{new} + b)\|^2$.
     However, since W is orthonormal, the solution simplifies to
  2. $x^*_{new} = W^T(y_{new} - b)$

Properties of PCA:

1. **Mean Zero Coefficients:** The PCA coefficients/latent coordinates, $\{x_i\}_{i=1}^n$, have a mean of 0.

Proof:

$$\text{Mean}(x) = \frac{\sum_{i=1}^{N} x_i}{N}$$

$$= \frac{\sum_{i=1}^{N} W^T(y_i - b)}{N} \leftarrow \text{By step 6 on pg. 6}$$

$$= \frac{W^T}{N}\left(\sum_{i=1}^{N}(y_i - b)\right)$$

$$= \frac{W^T}{N}\left(\underbrace{\sum_{i=1}^{N} y_i}_{\text{Equals } 0} - Nb\right)$$

$$= 0 \leftarrow \text{By step 1 on pg. 6.}$$

We set $b = \frac{\sum_i y_i}{N}$

$$Nb = \sum_i y_i$$

2. Max Var Formulation and Min Error Formulation are equivalent.

3. Out of subspace error: The total variance in the data is given by the sum of the eigenvalues of the sample co-variance matrix. The variance captured by PCA is the sum of the first k eigenvalues. The total amount of variance lost is given by the sum of the remaining eigenvalues.

One can show that the least-squares error in the approx to the original data provided by the opt model params w^*, $\{x_i^*\}$, and b^* is:

$$\sum_i \|y_i - (w^* x_i^* + b^*)\|^2$$

$$= \sum_{j=k+1}^{d} \lambda_j$$

When learning a PCA model, it is common to use the ratio of the total LS error and total variance in the training data (I.e. The sum of all eigenvalues). One needs to choose a k large enough s.t. this ratio is small (often 0.1 or less).

4. Proportion of Variance:

$$\text{Proportion of variance} = \frac{\sum_{j=1}^{k} \lambda_j}{\sum_{j=1}^{p} \lambda_j}$$

$$\sum_{j=1}^{p} \lambda_j = \sum_{j=1}^{p} \text{Var}(x_j) = p$$

PCA Final Comments:

1. Generally, it's critical to perform Standardization prior to PCA.

   PCA is very sensitive to the variances of the inital variables.
   I.e. If there are large diff btwn the range of the initial variables, the variables with the larger ranges will dominate over those with small ranges.

2. Pros:
   - Removes correlated features. Used a lot in the multi-collinearity issue.
   - Can speed up algos with fewer features.
   - Reduces overfitting with fewer features.

3. Cons:
   - Less interpretable since it transforms the original data.
   - Data standardization is a must.
   - Info loss w/o proper number of components

CSCC11 Week 10 Notes

Clustering:

- Clustering is an unsupervised learning problem in which the goal is to discover clusters in the data. A cluster is a collection of data points that are similar in some way.

K-Means:

- A simple method for clustering.
- Given N input data vectors $(\{y_i\}_{i=1}^N)$ we wish to label each vector as belonging to 1 of k-clusters. This labelling will be specified with a binary matrix L, the elements are given by

$$L_{ij} = \begin{cases} 1, & \text{if data point } i \text{ belongs to cluster } j \\ 0, & \text{otherwise} \end{cases}$$

- We also assume that each data point is assigned to only 1 cluster.
  I.e. $\forall$ points i, $\sum_i L_{ij} = 1$

- We also want to find a representative for each cluster, c_j. For example, it could be the mean of the points assigned to that cluster.

The obj-fun for k-Means clustering is:

$$E(\{c_j\}_{j=1,\dots,k}, L) = \sum_{i,j} L_{ij} \|y_i - c_j\|^2$$

This obj func penalizes the squared Euclidean dist btwn each data point and the center of the cluster to which it is assigned. To minimize this error, we want to bring cluster centers close to the points within their clusters and we want to assign each data point to the nearest cluster.

This optimization problem is NP-hard and can't be solved in close form. Furthermore, because it includes discrete variables (the labels L), we can't use gradient-based methods. Instead, we'll use block coordinate descent.

General Algo:

1. Initalize C_j's
2. Assign each point y_i to the closest C_j. Closest is computed using Euclidean dist
3. Update C_j to be the mean of the data points assigned to cluster j.
4. Repeat (2) and (3) until assignments are unchanged.

Note: There's a chance that you can get trapped in a local minima with this algo.

- In step 1 of the algo on the previous page we initalized c_j's. Poor initalization can lead to poor results. Here are a few strategies that can be used for initalization:

1. **Random Labelling:** Initalize the labelling L randomly and then run step (3) of the gen algo to determine the initial centers. This approach isn't recommended bc the initial centers will likely end up just being very close to the mean of the entire dataset.

2. **Random Initial Centers:** Choose the inital center values randomly. But it's very likely that some of the centers will fall into empty regions of the feature space and will be assigned no data. Getting a good initalization is difficult using this method.

3. **Random data points as centers:** Choose a random subset of the data points as the initial centers. Works somewhat better.

4. **Multiple restarts:** Run k-Means multiple times, each time with a different random initalization and keep the soln that gives the lowest value of the obj func.

5. **k-Means++:** The goal is to choose the initial centers to be relatively far from each other.
   1. Choose 1 data point at random to be the first center.
   2. Compute the dist btwn each point and its closest center denoted $D(y_i)$ for the i^{th} data point.
   3. Choose the next center from the remaining data points with prob proportion to $D(y_i)^2$ for the i^{th} data point. 4. Repeat 2 and 3

One of the simplest and best ways for initalization →

Mixture of Gaussians (MoG):

- Also called Gaussian Mixture Model (GMM)
- Is a generalization of k-Means clustering. While k-Means works well with / for well-separated clusters that are more or less spherical, MoG can handle the wider class of oblong clusters and it does an excellent job when clusters are overlapping.

- MoG model compries a linear combination of k Gaussian distributions, each with its own mean and covariance $\{(\mu_j, C_j)\}_{j=1}^{k}$. Each Gaussian component also has an associated prior m_j s.t. $\sum_j m_j = 1$. These probabilities, often called mixing probabilities, represent the fraction of the data generated by the diff Gaussian components.

- As a shorthand, it is convenient to capture all model parameters with a single variable $\theta = \{m_{1:k}, \mu_{1:k}, C_{1:k}\}$

- We can now write the likelihood of y being generated by θ as

$$P(y|\theta) = \sum_{j=1}^{k} P(y, \ell=j|\theta) \leftarrow \text{Marginalization}$$

$$= \sum_{j=1}^{k} P(y|\theta, \ell=j) \cdot P(\ell=j|\theta)$$

Note: ℓ is the j^{th} Gaussian that generates y.

- Prior: $m_j = P(\ell = j \mid \theta)$

- Likelihood: $P(y \mid \theta, \ell = j) = G(y; \mu_j, C_j)$

- Going to the eqn on the prev page, we can write $P(y|\theta)$ as:

$$P(y|\theta) = \sum_{j=1}^{k} \underbrace{m_j}_{\text{Prior}} \underbrace{\frac{1}{\sqrt{(2\pi)^d |C_j|}} \exp\left(-\tfrac{1}{2}(y-\mu_j)^T C_j^{-1} (y-\mu_j)\right)}_{\text{Gaussian Dist}}$$

- Our goal is to learn θ s.t. it maximizes $P(y|\theta)$

Learning Parameters:

$L(\theta) = P(y_{1:N}|\theta)$

$L'(\theta) = -\ln(P(y_{1:N}|\theta))$ ← Want to min negative log likelihood

$$= -\ln \prod_{i=1}^{N} P(y_i|\theta)$$ ← Assume $y_i \overset{iid}{\sim} D$

$$= -\sum_{i=1}^{N} \ln(P(y_i|\theta))$$

$$= -\sum_{i=1}^{N} \ln \sum_{j=1}^{k} m_j \cdot G(y_i; \mu_j, C_j)$$ ← No closed form soln

- Since we require $m_j \geq 0$, $\Sigma_j m_j = 1$ and C_j must be a symmetric, positive-definite matrix, this is a constrained optimization.

- We could use gradient descent to optmize for $l'(\theta)$, but there are a lot of stuff we have to be careful about.

- We could also reparameterize the problem to be unconstrained.
  I.e. Reparameterize μ_j and C_j s.t. they always satisfy the constraints.

- Alternatively, we can do Expectation-Maximization algorithm (EM algo).

- EM is a general algo for "hidden variable" or "missing data" problems. In this case, the missing data are the labels ℓ.

- EM algo uses 2 steps:
  1. E-Step: Compute the posterior probability that each Gaussian generates each data point.

  2. M-Step: Assuming that the data was generated this way, change the parameters of each Gaussian to max the probability that it would generate the data it is currently responsible for.

- Let r_{ij}, called the ownership probability, correspond to the probability that data point i came from cluster j.
  I.e. r_{ij} is meant to estimate $P(\ell=j|y_i, \theta)$

- In EM, we opt both θ and r_{ij}.

- The algo alternates btwn the E-step, which updates r and the M-step, which updates θ.

- Algo for GMM:

1. Initalize r_{ij}'s and θ.

2. For each point i, compute r_{ij} as shown below. This is E-step.

E-Step:

$$r_{ij} = P(\ell = j \mid y_i, \theta)$$

$$= \frac{P(y_i \mid \ell = j, \theta) \cdot P(\ell = j \mid \theta)}{\sum_{j=1}^{k} P(y_i \mid \ell = j, \theta) \cdot P(\ell = j \mid \theta)}$$

3. For each cluster j, compute θ_j (as shown below). This is M-Step.

M-Step:

$$m_j = \frac{\Sigma_i r_{ij}}{N}, \quad \mu_j = \frac{\Sigma_i r_{ij} \cdot y_i}{\Sigma_i r_{ij}}$$

$$c_j = \frac{\Sigma_i r_{ij} (y_i - \mu_j)(y_i - \mu_j)^T}{\Sigma_i r_{ij}}$$

4. Repeate 2 and 3 until termination condition is met. (Unchanged assignments, unchanged likelihoods, etc)

Relation to k-Means:

- MoG is very similar to k-Means.
- If
  1. $m_j = \frac{1}{k}$ ← Uniform prior
  2. $C_j = \sigma^2 I \ \forall j$ ← Spherical Gaussian
  3. $\sigma^2 \to 0$ ← σ^2 is infinitesimal (Hard assignment)

  then GMM collapses to k-Means.

CSCC11 Week 11 Notes

Multilayer Perceptrons (MLP):

- Consider $\phi\left(\sum_j w_j x_j + b\right)$. We've seen variations of it.

For linear regression, $\phi(z) = z$.

For logistic regression, ϕ is the logistic function $\sigma(z) = \frac{1}{1+e^{-z}}$

A neural network is just a combination of these.

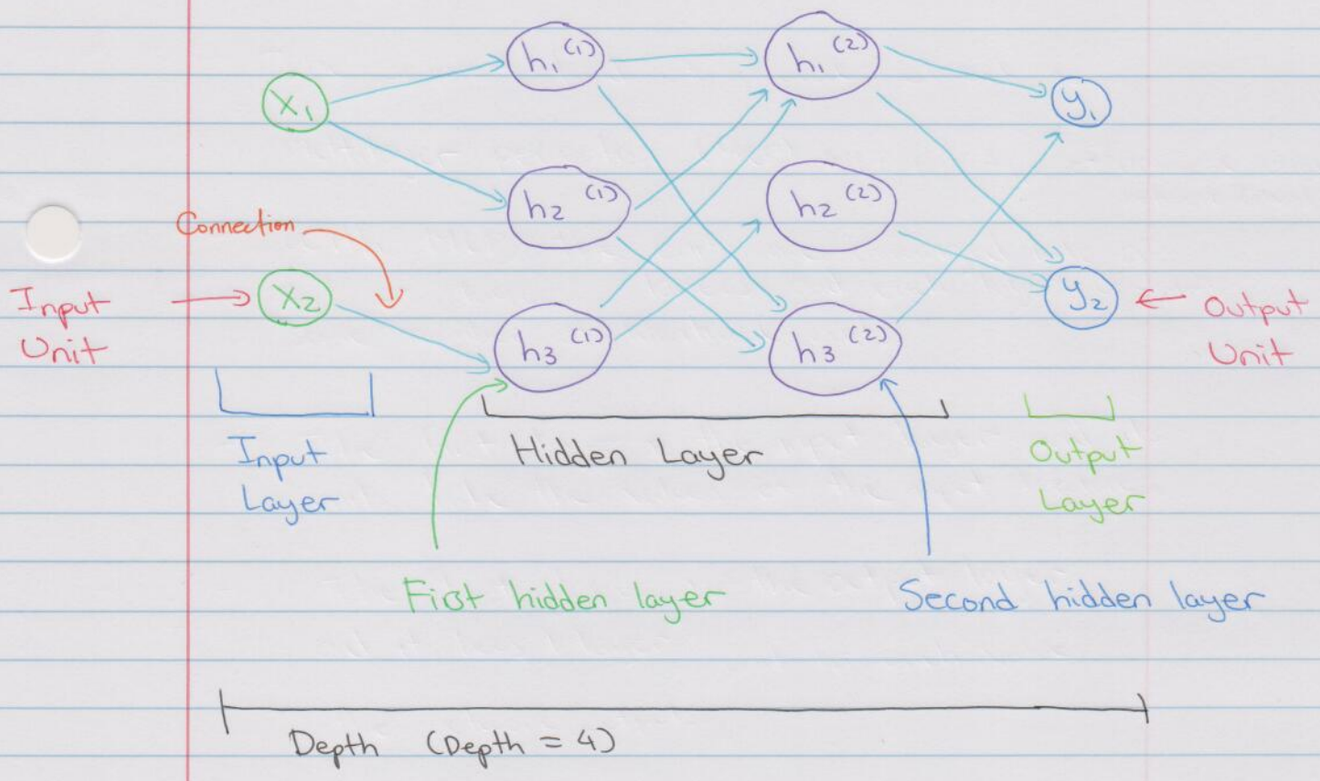
- The simplest kind of neural network is multilayer perceptron (MLP). MLP is a type of artificial neural network (ANN).
- With MLP, the units are arranged into a set of layers and each layer contains some number of identical units.
- The first layer is the input layer and its units take the values of the input features.
- The last layer is the output layer and it has 1 unit for each value the network outputs.
- All layers in btwn the input and output layers are known as hidden layers bc we don't know ahead of time what these units should compute and this needs to be discovered during learning.

known as

- The number of layers is the depth.
  If depth = 1, we have shallow nn. If depth > 1, deep nn.
- The number of units in a layer is known as the width.

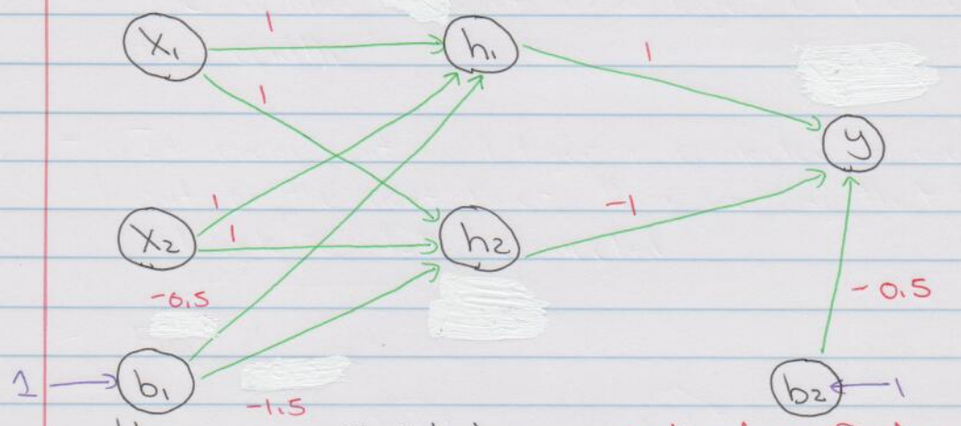
- If every unit in 1 layer is connected to every unit in the next layer, then we say that the network is fully connected.

- E.g. 1



Depth (Depth = 4)

- E.g. 2 This is an MLP that computes XOR



Here, we will introduce an activation function. An activation function is just a non-linear function applied to a node to get its output value from its inputs.

For this example, we'll define the activation function to be:

$$\phi(x) = \begin{cases} 1, & \text{if } w_1 x_1 + w_2 x_2 + b_1 \cdot w_{b1} \geq 1 \\ 0, & \text{if } w_1 x_1 + w_2 x_2 + b_1 \cdot w_{b1} < 1 \end{cases}$$

Consider $x_1 = 0$ and $x_2 = 0$.

$h_1 = x_1 w_1 + x_2 w_2 + b_1 w_{b1}$
$= (0)(1) + (0)(1) + (1)(-0.5)$
$= -0.5$
$= 0$ ← Bc of activation function

$h_2 = x_1 w_1 + x_2 w_2 + b_1 w_{b1}$
$= (0)(1) + (0)(1) + (1)(-1.5)$
$= -1.5$
$= 0$ ← Bc of activation function

$y = h_1(1) + h_2(-1) + b_2(-0.5)$
$= (0)(1) + (0)(-1) + (1)(-0.5)$
$= -0.5$
$= 0$ ← Bc of activation function

Now, consider $x_1 = 1$ and $x_2 = 0$
$h_1 = x_1(1) + x_2(1) + b_1(-0.5)$
$= 1 + 0 - 0.5$
$= 0.5$
$= 1$

$h_2 = x_1(1) + x_2(1) + b_1(-1.5)$
$= -1.5$
$= 0$

$y = h_1(1) + h_2(-1) + b_2(-0.5)$
$= 1 - 0.5$
$= 0.5$
$= 1$

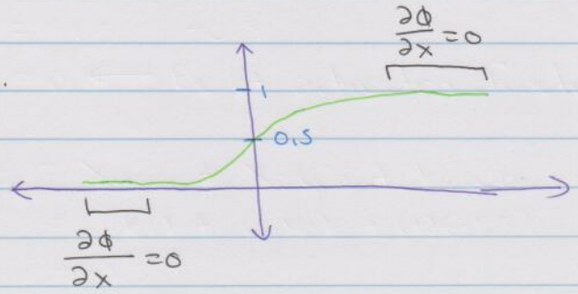
- Some activation functions are:

1. Sigmoid Function:

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

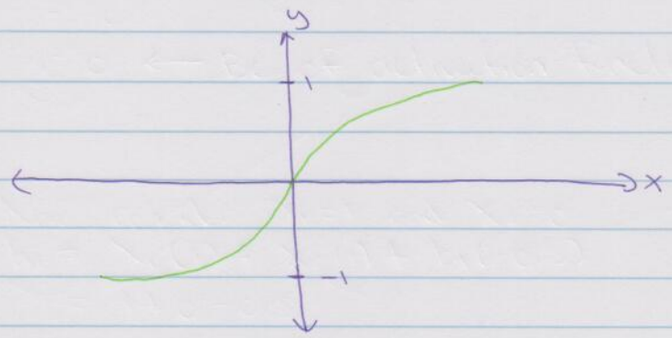
This isn't very good bc the derative of the tail is 0.

I.e.



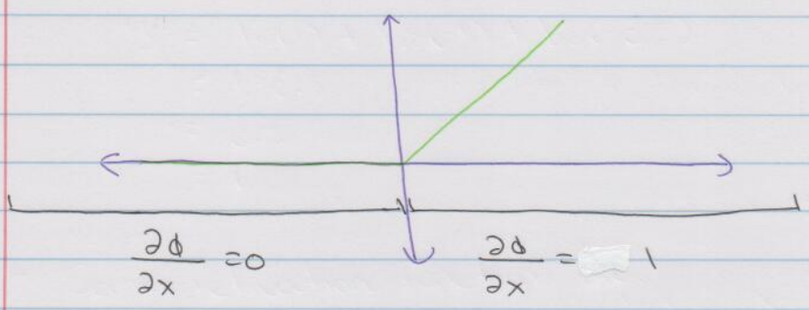
2. tanh function:

$\phi(x) = \tanh(x)$



3. Rectified Linear Unit (ReLU)

$$\phi(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$



This is a popular function

4. Swish Function:

$\phi(x) = x \cdot \text{sigmoid}(x)$

- For hidden layer 1:

$$h_i^{(1)} = \phi_i^{(1)}\left(\sum_j w_{ij}^{(1)} x_j + b_i^{(1)}\right)$$

E.g.

$$h_1^{(1)} = \phi_1^{(1)}\left(\sum_j w_{1j}^{(1)} x_j + b_1^{(1)}\right)$$

Note: We can "move" b into w. If we do,

$$h_i^{(1)} = \phi_i^{(1)}\left(\sum_j w_{ij}^{(1)} x_j\right)$$ but now,

$$w = \left[w_{11}^{(1)}, \ldots, w_{1N}^{(1)}, \underbrace{b_1^{(1)}}_{\text{New term}}\right]$$

$$x = [x_1, \ldots, x_N, \underbrace{1}_{\text{New term}}]$$

Suppose the width = k.

I.e. k = # of hidden units

$$h^{(1)} = \phi^{(1)}(w^{(1)} x) \text{ where}$$

$$w^{(1)} = \begin{bmatrix} w_{11}^{(1)} & \cdots & w_{1N}^{(1)} & b_1^{(1)} \\ w_{21}^{(1)} & \cdots & w_{2N}^{(1)} & b_2^{(1)} \\ \vdots & \vdots & \vdots & \vdots \\ w_{k1}^{(1)} & \cdots & w_{kN}^{(1)} & b_k^{(1)} \end{bmatrix}$$

$$\phi^{(1)} = \begin{bmatrix} \phi_1^{(1)} \\ \vdots \\ \vdots \\ \phi_k^{(1)} \end{bmatrix}$$

- For hidden layer 2:

$$h^{(2)} = \phi^{(2)}(h^{(1)} W^{(2)T})$$
$$= \phi^{(2)}(W^{(2)} h^{(1)})$$
$$= \phi^{(2)}(W^{2}(\phi^{(1)}(W^{(1)} x)))$$

Note: If $\phi^{(i)}$'s are linear, then we'll just have a bunch of matrix multiplications. Furthermore, if you don't have an activation function or all the activation functions are identity, then you get linear regression.

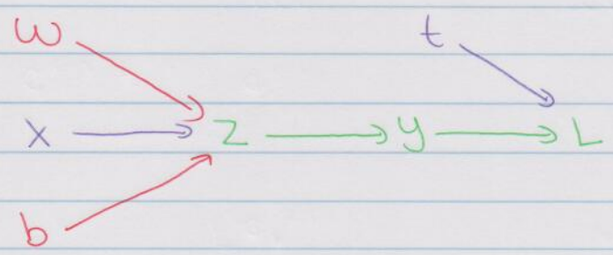
- For hidden layer L:

$$h^{(L)} = \phi^{(L)}(W^{(L)} h^{(h-1)})$$

- Now, we'll talk about how we can learn $W^{(i)}$'s.

Consider a regression problem with 1D input/output (x, t) with loss function L. We'll construct a model $z = wx + b$ followed by an activation function ϕ s.t. $y = \phi(z)$.

I.e.



- — are params
- — are input/output
- — are operations

Forward pass is to compute the loss (L).
Backward pass / Backpropagation is to compute the gradient.

We need to use gradient descent to learn the $w^{(i)}$'s.

For the gradients, we want to compute $\frac{\partial L}{\partial w}$, and $\frac{\partial L}{\partial b}$.

First, we have to do forward pass:

1. $z = wx + b$
2. $y = \phi(z)$
3. $L = \frac{1}{2}(y-t)^2$

Now, we can do backpropagation:

1. $\frac{\partial L}{\partial y} = y - t$
2. $\frac{\partial y}{\partial z}$ ← We don't know what ϕ is, so can't say much for now.
3. $\frac{\partial z}{\partial w} = x$

$\frac{\partial z}{\partial b} = 1$ ← Because b is a constant

We need to do these 3 steps bc

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w}$$

and similar for $\frac{\partial L}{\partial b}$

Chain Rule

Convolutional Neural Networks:

- Artifical Neural Network (ANN) and Multilayer perceptron (MLP) in particular is not very good with analyzing visual images.

The first reason is that with MLP, because it usually involves fully connected networks. (Ie. Each unit in one hidden layer is connected with each unit in the next layer.) This makes it prone to overfitting.

The second reason is that it uses too many parameters and computations. Consider a 224x224 pixel image such that each pixel can be one of 3 colors. We'll need 224 x 224 x 3 or 150, 528 weights.

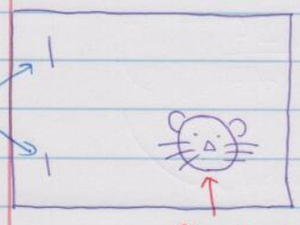
- CNN is used for classification of images and computer vision tasks.

This → assumption is called inductive bias.

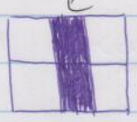
- CNN assumes that the inputs are images and uses this fact to find patterns. We can use these patterns to reduce the number of weights.

E.g. Suppose we have an image like the one shown below and we want to see if there's a picture of 1 in the image.

Image →



Called filter/kernel



We can overlap on the image and see if it matches anywhere on the pic. If it does, we know that place has a 1.

- The layers in CNN have the neurons arranged in 3 dimensions (length, width, depth) where they refers to the RGB values of the image
- There are 3 main layers in CNN:
1. Convolutional Layer
2. Pooling Layer
3. Fully-connected Layer (FC Layer)

Convolutional Layer:

- In this layer, we take the input data and a filter/kernel.
- The filter/kernel is a 2-D array of weights that represents a part of the image.
- The filter is applied over an area of the image and we take the dot product btwn the input pixels and the filter. The final output from the series of dot products is called the feature map/activation map/convolved feature.
- In the above point, we shift the filter by a stride until the entire input is covered.

The stride → determines the amount of movement for the filter.

- E.g. $\text{Input} = \begin{bmatrix} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 1 \\ 7 & 8 & 9 & 1 \end{bmatrix}$, $\text{filter} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

$$\begin{bmatrix} 1 & 2 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = 1\times1 + 2\times1 + 4\times1 + 5\times1 = 12$$

$$\begin{bmatrix} 2 & 3 \\ 5 & 6 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = 2\times1 + 3\times1 + 5\times1 + 6\times1 = 16$$

$$\begin{bmatrix} 3 & 1 \\ 6 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = 3\times1 + 1\times1 + 5\times1 + 6\times1 = 11$$

$$\begin{bmatrix} 4 & 5 \\ 7 & 8 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = 4\times1 + 5\times1 + 7\times1 + 8\times1 = 24$$

$$\begin{bmatrix} 5 & 6 \\ 8 & 9 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = 5\times1 + 6\times1 + 8\times1 + 9\times1 = 28$$

$$\begin{bmatrix} 6 & 1 \\ 9 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = 6\times1 + 1\times1 + 9\times1 + 1\times1 = 17$$

Feature map → $\begin{bmatrix} 12 & 16 & 11 \\ 24 & 28 & 17 \end{bmatrix}$

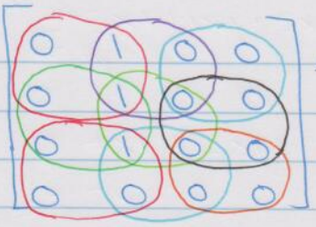
Here are the strides:

$$\begin{bmatrix} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 1 \\ 7 & 8 & 9 & 1 \end{bmatrix}$$

The red circle is the first step.
I.e. We apply the filter over the red part first. We then shift to the blue part, then the purple part, then the light-blue part, then the turquoise part and finally the green part. Afterwards, we've covered the entire input array.

- Note: The stride doesn't have to one single part of the input array. It can be $\begin{bmatrix} 1 & 1 \\ 7 & 1 \end{bmatrix}$ for example.

- E.g. Input = $\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$, filter = $\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}$



1. $\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = 0\times0 + 1\times1 + 0\times0 + 1\times1 = 2$

2. $\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = 1\times0 + 0\times1 + 1\times0 + 0\times1 = 0$

3. $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = 0$

4. $\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = 2$

5. $\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = 0$

6. $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = 0$

7. $\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = 1$

8. $\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = 0$

9. $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = 0$

Feature Map

$$\begin{bmatrix} 2 & 0 & 0 \\ 2 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Pooling:

- Is used to reduce the dimensionality of the feature map.
- It combines a set of values into a smaller number of values.
- This layer serves 2 purposes:
  1. Reduce the number of parameters/weights.
  2. Control overfitting.
- An ideal pooling method is expected to extract only useful info and discard irrelevant details.
- There are 2 types of pooling:

1. Average Pooling:

- We divide the feature map into rectangular sections/ rectangular pooling regions and computing the avg values of each region.

- E.g.

Feature Map

1	4	2	7
2	6	8	5
3	4	0	7
1	2	3	1

→

3.25	5.5
2.5	2.75

Region 1: $\frac{(1+2+4+6)}{4} = \frac{13}{4} = 3.25$

Region 2: $\frac{(2+5+7+8)}{4} = \frac{22}{4} = 5.5$

Region 3: $\frac{(1+2+3+4)}{4} = \frac{10}{4} = 2.5$

Region 4: $\frac{(1+3+7)}{4} = \frac{11}{4} = 2.75$

2. Max Pooling:

- We divide the feature map into rectangular pooling regions and get the max of each region.

- E.g.

Feature Map

1	4	2	7
2	6	8	5
3	4	0	7
1	2	3	1

→

6	8
4	7

Note: There could be some overlaps for the rectangular pooling regions.

Note: The window size of the pooling region is just another hyperparameter. If we use an extremely large region, we may lose out on some key information.

Linear Algebra Review

Table of Contents

Basic Notation:

- $A \in \mathbb{R}^{m \times n}$ denotes a matrix with m rows and n columns where the entries of A are real numbers.

- $x \in \mathbb{R}^n$ denotes a vector with n entries. Usually, a vector x will denote a column vector, a matrix with n rows and 1 column. We typically write x^T to represent a row vector, a matrix with 1 row and n columns.

I.e. $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ ← Column vector

$x^T = \begin{bmatrix} x_1, x_2, \ldots, x_n \end{bmatrix}$ ← Row Vector

- x_i denotes the i^{th} element of vector x.

- A_{ij} / a_{ij} denotes the element in matrix A at the i^{th} row and j^{th} column.

- a_j / $A_{:,j}$ denotes the j^{th} column in matrix A.

- a_i^T / $A_{i,:}$ denotes the i^{th} row in matrix A.

Identity Matrix:

- The identity matrix, denoted $I \in \mathbb{R}^{n \times n}$, is a square matrix with 1's on the main diagonal and 0's elsewhere.
- It has the property that for all $A \in \mathbb{R}^{m \times n}$, $AI = A = IA$, where the size of I is determined by the dimensions of A so matrix multiplication is possible.
- E.g. $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ is a 3 by 3 identity matrix.

Diagonal Matrix:

- A diagonal matrix is a matrix where all elements not on the main diagonal are 0.
- We typically denote a diagonal matrix using $\text{diag}(d_1, d_2, \ldots, d_n)$.
- The identity matrix, I, is equal to $\text{diag}(1, 1, \ldots, 1)$.
- E.g. $\begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$, $\begin{bmatrix} 6 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$, $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ are all diagonal matrices.

Note: There is no restriction that elements along the main diagonal can't be 0.

Square Matrix:

- A square matrix is a matrix with the same number of rows and columns.
- E.g. $\begin{bmatrix} 1 \end{bmatrix}$, $\begin{bmatrix} 3 & 0 \\ 1 & 7 \end{bmatrix}$, $\begin{bmatrix} 2 & 1 & 6 \\ 0 & 9 & 7 \\ 3 & 5 & 4 \end{bmatrix}$ are all square matrices.

Main Diagonal:

- The main diagonal of matrix A is the list of entries A_{ij} where $i=j$.
- E.g. $\begin{bmatrix} 1 \end{bmatrix}$, $\begin{bmatrix} 2 & 3 \\ 1 & 5 \end{bmatrix}$, $\begin{bmatrix} 7 & 0 \\ 1 & 6 \\ 3 & 2 \end{bmatrix}$, $\begin{bmatrix} 7 & 1 & 3 \\ 2 & 4 & 6 \end{bmatrix}$

All elements in green are part of their array's main diagonal.

Transpose:

- The transpose of a matrix, denoted as A^T, results from flipping the rows and cols.

- Let $A \in R^{m \times n}$. Then, $A^T \in R^{n \times m}$.

- $(A^T)_{ji} = A_{ij}$

- E.g. $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$

- Properties:
  1. $(A^T)^T = A$
  2. $(AB)^T = B^T A^T$
  3. $(A+B)^T = A^T + B^T$

Symmetric Matrices:

- A square matrix is symmetric if that matrix is equal to its transpose.
  I.e. $A \in R^{n \times n}$ is symmetric if $A = A^T$.

- A square matrix is anti-symmetric if that matrix is equal to the negative of its transpose.
  I.e. $A \in R^{n \times n}$ is anti-symmetric if $A = -A^T$.

- E.g. $\begin{bmatrix} 1 & 7 & 3 \\ 7 & 4 & 5 \\ 3 & 5 & 0 \end{bmatrix}$ is symmetric

  $\begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 3 \\ -2 & -3 & 0 \end{bmatrix}$ is anti-symmetric

- For any square matrix $A \in R^{n \times n}$, $A + A^T$ is symmetric and $A - A^T$ is anti-symmetric. Hence, we can write any square matrix $A \in R^{n \times n}$ as $A = \frac{1}{2}(A + A^T) + \frac{1}{2}(A - A^T)$

The Trace

- The trace of a square matrix $A \in R^{n \times n}$, denoted as $tr(A)$ / trA is the sum of the elements along the main diagonal.

$$tr(A) = \sum_{i=1}^{n} A_{ii}$$

- Properties:
  1. For $A \in R^{n \times n}$, $tr(A) = tr(A^T)$
  2. For $A, B \in R^{n \times n}$, $tr(A+B) = tr(A) + tr(B)$
  3. For $A \in R^{n \times n}$, $t \in R$, $tr(tA) = t \cdot tr(A)$
  4. For A, B s.t. $AB \in R^{n \times n}$, $tr(AB) = tr(BA)$
  5. For A, B, C s.t. $ABC \in R^{n \times n}$, $tr(ABC) = tr(BAC) = tr(CAB) = \ldots$

Norm:

- The norm of a vector, denoted as $\|x\|$, informally is the measure of the length of the vector.

  More formally, a norm is any function $f: R^n \to R$ that satisfies 4 properties:
  1. $\forall x \in R^n$, $f(x) \geq 0$
  2. $f(x) = 0$ iff $x = 0$
  3. $\forall x \in R^n$, $t \in R$, $f(tx) = |t| f(x)$
  4. $\forall x, y \in R^n$, $f(x+y) \leq f(x) + f(y)$

- Let $p \geq 1$ be a real number. The p-norm/lp-norm of vector x is

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

For $p=1$, we get the taxicab norm.

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|$$

For $p=2$, we get the Euclidean norm.

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} (x_i)^2}$$

Note: $(\|x\|_2)^2 = x^T x$

For $p=\infty$, we get the infinity norm.

$$\|x\|_\infty = \max_i |x_i|$$
$$= \max(|x_1|, |x_2|, \ldots, |x_n|)$$

- Norms can also be defined for matrices, such as the Frobenius Norm.

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} (A_{ij})^2}$$

The square root of the sum of the abs squares of its elements.

$$= \sqrt{tr(A^T A)}$$

Linear Independence and Rank:

- A set of vectors $\{x_1, x_2, \ldots, x_n\}$ is linearly independent if no vector can be represented as a linear combination of the remaining vectors. Conversely, a vector that can be represented as a linear combination of the remaining vectors is linearly dependent.

  Another way to think about this is a set of vectors is linearly dependent if there is a non-trivial linear combination of the vectors that equal the zero vector. If no such linear combination exists, then the vectors are linearly independent.

- E.g. Are the vectors $v_1 = \begin{bmatrix} 2 \\ 5 \\ 3 \end{bmatrix}$, $v_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$, $v_3 = \begin{bmatrix} 4 \\ -2 \\ 0 \end{bmatrix}$ linearly dep or indep?

Soln:

$$\begin{bmatrix} 2 & 1 & 4 \\ 5 & 1 & -2 \\ 3 & 1 & 0 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$R_1 - R_3$

$$\left[\begin{array}{ccc|c} -1 & 0 & 4 & 0 \\ 5 & 1 & -2 & 0 \\ 3 & 1 & 0 & 0 \end{array}\right]$$

$R_2 - R_3$

$$\left[\begin{array}{ccc|c} -1 & 0 & 4 & 0 \\ 2 & 0 & -2 & 0 \\ 3 & 1 & 0 & 0 \end{array}\right]$$

Note: A set of vectors is linearly indep if you only get the trivial soln when you try to find coefficients s.t.

$a_1\vec{v_1} + a_2\vec{v_2} + \ldots + a_n\vec{v_n} = \vec{0}$

$R_3 + 3 \cdot R_1$

$$\left[\begin{array}{ccc|c} -1 & 0 & 4 & 0 \\ 2 & 0 & -2 & 0 \\ 0 & 1 & 12 & 0 \end{array}\right]$$

$R_2 / -2$

$$\left[\begin{array}{ccc|c} -1 & 0 & 4 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 1 & 12 & 0 \end{array}\right]$$

$R_2 - R_1$

$$\left[\begin{array}{ccc|c} -1 & 0 & 4 & 0 \\ 0 & 0 & -3 & 0 \\ 0 & 1 & 12 & 0 \end{array}\right]$$

$R_2 / -3$

$$\left[\begin{array}{ccc|c} -1 & 0 & 4 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 12 & 0 \end{array}\right]$$

$R_1 \cdot (-1)$

$$\left[\begin{array}{ccc|c} 1 & 0 & -4 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 12 & 0 \end{array}\right]$$

$R_2 \leftrightarrow R_3$

$$\left[\begin{array}{ccc|c} 1 & 0 & -4 & 0 \\ 0 & 1 & 12 & 0 \\ 0 & 0 & 1 & 0 \end{array}\right]$$

Since we get the trivial soln, the 3 vectors are linearly independent.

- E.g. Are the vectors $v_1 = \begin{bmatrix} 4 \\ 1 \\ -2 \end{bmatrix}$, $v_2 = \begin{bmatrix} -3 \\ 0 \\ 1 \end{bmatrix}$

$v_3 = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}$?

Soln:

$$\left[\begin{array}{ccc|c} 4 & -3 & 1 & 0 \\ 1 & 0 & -2 & 0 \\ -2 & 1 & 1 & 0 \end{array}\right]$$

$R_3 - R_1$

$$\left[\begin{array}{ccc|c} 4 & -3 & 1 & 0 \\ 1 & 0 & -2 & 0 \\ -6 & 4 & 0 & 0 \end{array}\right]$$

$R_1 \cdot 2 + R_2$

$$\left[\begin{array}{ccc|c} 9 & -6 & 0 & 0 \\ 1 & 0 & -2 & 0 \\ -6 & 4 & 0 & 0 \end{array}\right]$$

$R_1 / -3$

$$\left[\begin{array}{ccc|c} -3 & 2 & 0 & 0 \\ 1 & 0 & -2 & 0 \\ -6 & 4 & 0 & 0 \end{array}\right]$$

$R_3 / 2$

$$\left[\begin{array}{ccc|c} -3 & 2 & 0 & 0 \\ 1 & 0 & -2 & 0 \\ -3 & 2 & 0 & 0 \end{array}\right]$$

These 2 rows are the same.

$R_1 - R_3$

$$\left[\begin{array}{ccc|c} 0 & 0 & 0 & 0 \\ 1 & 0 & -2 & 0 \\ -3 & 2 & 0 & 0 \end{array}\right]$$

← Because there's a column without a pivot, the vectors are lin dep.

- The column rank of matrix A is the largest number of cols of A that constitute a linearly indep set.

- The row rank of a matrix A is the largest number of rows of A that constitute a linearly indep set.

Note: For any matrix A, column rank (A) = row rank (A). As a result, this quantity is simply referred to as the rank of A, denoted as rank(A).

- Properties of rank:
  1. For $A \in \mathbb{R}^{m \times n}$, $\text{rank}(A) \leq \min(m, n)$. If $\text{rank}(A) = \min(m, n)$, then A is said to be full rank.
  2. For $A \in \mathbb{R}^{m \times n}$, $\text{rank}(A) = \text{rank}(A^T)$.
  3. For $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{n \times p}$, $\text{rank}(AB) \leq \min(\text{rank}(A), \text{rank}(B))$
  4. For $A, B \in \mathbb{R}^{m \times n}$, $\text{rank}(A+B) \leq \text{rank}(A) + \text{rank}(B)$

Inverse Matrix:

- The inverse of a square matrix $A \in \mathbb{R}^{n \times n}$, denoted as A^{-1}, is the unique matrix s.t.

$$A^{-1}A = AA^{-1} = I$$

- Not all square matrices have an inverse. In this case, we say that the matrix is non-invertible or singular. If the matrix does have an inverse, we say that the matrix is invertible or non-singular.

- It is possible to show that A^{-1} exists iff A is full rank.

- Properties (Assume A and B are square and non-singular):
  1. $(A^{-1})^{-1} = A$
  2. If $Ax = b$, we can multiply by A^{-1} on both sides to get $x = A^{-1}b$.
  3. $(AB)^{-1} = B^{-1}A^{-1}$
  4. $(A^{-1})^T = (A^T)^{-1}$. For this reason, this matrix is often denoted as A^{-T}.

Orthogonal Matrices:

- 2 vectors are orthogonal if they are perpendicular to each other. (That means their dot product is 0.)

- E.g. Let $A = (1, 0, -1)$ and $B = (-1, 0, -1)$

$$A \cdot B = (1)(-1) + (0)(0) + (-1)(-1)$$
$$= -1 + 1$$
$$= 0$$

$\therefore$ A and B are orthogonal.

- A set of vectors, $\{v_1, v_2, \ldots, v_n\}$ are mutually orthogonal if every pair of vectors are orthogonal.

  I.e. $v_i \cdot v_j = 0 \quad \forall i \neq j$

- E.g. Let $A = (1, 0, -1)$, $B = (1, \sqrt{2}, 1)$, $C = (1, -\sqrt{2}, 1)$

$$A \cdot B = (1)(1) + (0)(\sqrt{2}) + (-1)(1)$$
$$= 1 - 1$$
$$= 0$$

$$A \cdot C = (1)(1) + (0)(-\sqrt{2}) + (-1)(1)$$
$$= 1 - 1$$
$$= 0$$

$$B \cdot C = (1)(1) + (\sqrt{2})(-\sqrt{2}) + (1)(1)$$
$$= 1 - 2 + 1$$
$$= 0$$

$\therefore \{A, B, C\}$ are mutually orthogonal.

- A vector $\hat{x}$ is normalized if $\|x\|_2 = 1$.

- E.g. Let $A = (1, 0, -1)$

$$\|A\|_2 = \sqrt{(1)^2 + (0)^2 + (-1)^2}$$
$$= \sqrt{2}$$

$\therefore$ A is not normalized.

- To normalize a vector, we just divide the vector by its magnitude.

- E.g. Let $A = (1, 0, -1)$.

We've previously show that A is not normalized.

Let $B = \frac{A}{\|A\|_2}$

$$B = \frac{1}{\sqrt{2}}(1, 0, -1)$$

$$= \left(\frac{1}{\sqrt{2}}, 0, -\frac{1}{\sqrt{2}}\right)$$

$$\|B\|_2 = \sqrt{\left(\frac{1}{\sqrt{2}}\right)^2 + (0)^2 + \left(-\frac{1}{\sqrt{2}}\right)^2}$$

$$= \sqrt{\frac{1}{2} + 0 + \frac{1}{2}}$$
$$= \sqrt{1}$$
$$= 1$$

$\therefore$ B is normalized.

- A set of vectors is orthonormal if every vector in the set is normalized and the set of vectors are mutually orthogonal.

- A square matrix is orthogonal if all of its cols are orthogonal to each other.

- The cols of a square matrix is orthonormal if all of its cols are normalized and it is orthogonal.

- Let U be a square, orthogonal matrix. Then:
  1. $U^T U = I = U U^T$
  2. $U^T = U^{-1}$

- Given a square, orthogonal matrix U and a vector x, we have the following property:

$$\|Ux\|_2 = \|x\|_2$$

Note: The dimensions of U and x must be compatible.

Range and Nullspace:

- The span of a set of vectors is the set of all vectors that can be expressed as a linear comb of it.

  I.e. The span of a set of vectors, S, is denoted as $\text{span}(S)$ and is the set of all linear comb of these vectors.

  I.e. Let $\{x_1, x_2, \ldots, x_n\}$ be a set of vectors

$$\text{span}(\{x_1, \ldots, x_n\}) = \left\{ V : V = \sum_{i=1}^{n} a_i x_i, \; a_i \in R \right\}$$

- If $\{x_1, \ldots, x_n\}$ is a set of n linearly indep vectors where each $x_i \in R^n$, then $\text{span}(\{x_1, \ldots, x_n\}) = R^n$

- E.g. $\text{span}(\{(1,0) \text{ and } (0,1)\}) = R^2$

- The projection of a vector $y \in R^m$ onto the span of $\{x_1, \ldots, x_n\}$ where each $x_i \in R^m$ is the vector $v \in \text{span}(\{x_1, \ldots, x_n\})$ s.t. v is as close as possible to y.

I.e. $\text{Proj}(y; \{x_1, \ldots x_n\}) = \text{argmin}_{v \in \text{span}(\{x_1, \ldots x_n\})} \|y - v\|_2$

Let $S = \{x_1, \ldots, x_n\}$. If S is mutually orthogonal, then we can find the projection of vector v this way:

Dot product

$$\text{Proj}_w{}^v = \frac{v \cdot x_1}{x_1 \cdot x_1} x_1 + \frac{v \cdot x_2}{x_2 \cdot x_2} x_2 + \ldots + \frac{v \cdot x_n}{x_n \cdot x_n} x_n$$

where $w = \text{span}(\{x_1, \ldots, x_n\})$.

E.g. Find the projection of $(7, 2, 2, 1)$ on $\text{span}(\{(1, 2, 2, 4), (4, -2, 8, -4)\})$.

Soln:

$$(1, 2, 2, 4) \cdot (4, -2, 8, -4) = 4 + (-4) + 16 + (-16) = 0$$

$$\frac{(7, 2, 2, 1) \cdot (1, 2, 2, 4)}{(1, 2, 2, 4) \cdot (1, 2, 2, 4)} (1, 2, 2, 4) = \frac{7 + 4 + 4 + 4}{1 + 4 + 4 + 16} (1, 2, 2, 4) = \frac{19}{25} (1, 2, 2, 4)$$

$$\frac{(7, 2, 2, 1) \cdot (4, -2, 8, -4)}{(4, -2, 8, -4) \cdot (4, -2, 8, -4)} (4, -2, 8, -4) = \frac{28 - 4 + 16 - 4}{16 + 4 + 64 + 16} (4, -2, 8, -4) = \frac{36}{100} (4, -2, 8, -4)$$

$$= \frac{19}{25}\begin{pmatrix}1\\2\\2\\4\end{pmatrix} + \frac{9}{25}\begin{pmatrix}4\\-2\\8\\-4\end{pmatrix}$$

$$= \begin{pmatrix}\frac{19}{25}\\ \frac{38}{25}\\ \frac{38}{25}\\ \frac{76}{25}\end{pmatrix} + \begin{pmatrix}\frac{36}{25}\\ \frac{-18}{25}\\ \frac{72}{25}\\ \frac{-36}{25}\end{pmatrix}$$

$$= \frac{1}{25}\begin{pmatrix}55\\20\\110\\40\end{pmatrix}$$

E.g. Find the projection of $(2, 9, -4)$ on $\text{span}(\{(1,2,2), (2,1,-2)\})$.

Soln:

$(1, 2, 2) \cdot (2, 1, -2) = 2+2-4$
$= 0$

$$\frac{(2,9,-4)\cdot(1,2,2)}{(1,2,2)\cdot(1,2,2)}(1,2,2) = \frac{2+18-8}{1+4+4}(1,2,2)$$
$$= \frac{12}{9}(1,2,2)$$

$$\frac{(2,9,-4)\cdot(2,1,-2)}{(2,1,-2)\cdot(2,1,-2)}(2,1,-2) = \frac{4+9+8}{4+1+4}(2,1,-2)$$
$$= \frac{21}{9}(2,1,-2)$$

$$\frac{4}{3} \qquad \frac{7}{3}$$

$$= \frac{12}{9}\begin{pmatrix} 1 \\ 2 \\ 2 \end{pmatrix} + \frac{21}{9}\begin{pmatrix} 2 \\ 1 \\ -2 \end{pmatrix}$$

$$= \frac{1}{3}\begin{pmatrix} 4+14 \\ 8+7 \\ 8-14 \end{pmatrix}$$

$$= \frac{1}{3}\begin{pmatrix} 18 \\ 15 \\ -6 \end{pmatrix} = \begin{pmatrix} 6 \\ 5 \\ -2 \end{pmatrix}$$

If S is not mutually orthogonal, then we need to Gram-Schmidt process to make S mutually orthogonal.

Gram-Schmidt process:
Let $x_1, x_2, \ldots, x_n$ be a set of vectors.

$$u_1 = x_1$$

$$u_2 = x_2 - \text{proj}_{u_1} x_2$$

$$u_3 = x_3 - \text{proj}_{u_1} x_3 - \text{proj}_{u_2} x_3$$

$$\vdots$$

$$u_k = x_k - \sum_{j=1}^{k-1} \text{proj}_{u_j} x_k$$

$u_1, u_2, \ldots, u_k$ is the set of mutually orthogonal vectors.

E.g. Project $(4, 9)$ on $\text{span}\{(3,1), (2,2)\}$.

Soln:

$$(3,1)\cdot(2,2) = 6+2 = 8$$

Need to use Gram-Schmidt

$$u_1 = (3,1)$$

$$u_2 = (2,2) - \text{Proj}_{u_1}(2,2)$$

$$= (2,2) - \frac{(3,1)\cdot(2,2)}{(3,1)\cdot(3,1)}(3,1)$$

$$= (2,2) - \frac{8}{10}(3,1)$$

$$= \begin{pmatrix}2\\2\end{pmatrix} - \begin{pmatrix}12/5\\4/5\end{pmatrix}$$

$$= \begin{pmatrix}-2/5\\6/5\end{pmatrix}$$

$$(3,1)\cdot(-2/5,\ 6/5) = -6/5 + 6/5 = 0$$

$$\frac{(4,9)\cdot(3,1)}{(3,1)\cdot(3,1)}(3,1) = \frac{21}{10}(3,1)$$

$$\frac{(4,9)\cdot(-2/5,\ 6/5)}{(-2/5,\ 6/5)\cdot(-2/5,\ 6/5)}\left(-\tfrac{2}{5}, \tfrac{6}{5}\right) = \frac{23}{4}\left(-\tfrac{2}{5}, \tfrac{6}{5}\right)$$

Final soln: $\frac{21}{10}\begin{pmatrix}3\\1\end{pmatrix} + \frac{23}{4}\begin{pmatrix}-\frac{2}{5}\\6/5\end{pmatrix}$

- The range/columnspace of a matrix $A \in \mathbb{R}^{m \times n}$, denoted as $R(A)$, is the span of the cols of A.

$$R(A) = \{v \in \mathbb{R}^m : v = Ax, x \in \mathbb{R}^n\}$$

E.g. Let $A = \begin{bmatrix} 1 & 0 & 2 \\ 0 & -1 & 3 \end{bmatrix}$

$$R(A) = \left\{ \underbrace{\begin{bmatrix} 1 & 0 & 2 \\ 0 & -1 & 3 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a \\ b \\ c \end{bmatrix}}_{x} \right\}$$

- The nullspace of a matrix $A \in \mathbb{R}^{m \times n}$, denoted as $N(A)$ is the set of all vectors that equal 0 when multiplied by A.

I.e. $N(A) = \{x \in \mathbb{R}^n : Ax = 0\}$

Note: The nullspace is always non-empty since it always contains the 0 vector.

- $R(A^T)$ and $N(A)$ are disjoint subsets that together span the entire space of $\mathbb{R}^n$. Sets of this type are called orthogonal complements and are denoted as $R(A^T) = N(A)^{\perp}$.

Determinant:

The

- The determinant of a square matrix, denoted as $|A|$ or $\det A$, is the following:

$$A = [a_1] \quad (1 \text{ by } 1 \text{ matrix})$$
$$|A| = a_1$$

$$A = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \quad (2 \text{ by } 2 \text{ matrix})$$
$$|A| = a_1 \cdot a_4 - a_2 \cdot a_3$$

$$A = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix}$$
$$|A| = a_1(a_5 \cdot a_9 - a_6 \cdot a_8) - a_2(a_4 \cdot a_9 - a_6 \cdot a_7) + a_3(a_4 \cdot a_8 - a_5 \cdot a_9)$$

- Properties:
  1. $|I| = 1$
  2. $|tA| = t|A|$
  3. If we swap any 2 rows, then the det is $-|A|$.
  4. $|A| = |A^T|$
  5. $|AB| = |A||B|$
  6. $|A| = 0$ iff A is singular (non-invertible)
  7. If A is invertible, then $\frac{1}{|A|} = |A^{-1}|$

- The adjoint of matrix $A \in \mathbb{R}^{n \times n}$, denoted as $\text{adj}(A)$, is the transpose of its cofactor matrix.

- E.g.

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

$$\text{adj}(A) = \begin{bmatrix} a_{22} & -a_{21} \\ -a_{12} & a_{11} \end{bmatrix}$$

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

$$\text{adj}(A) = \begin{bmatrix} +\begin{vmatrix} e & f \\ h & i \end{vmatrix} & -\begin{vmatrix} d & f \\ g & i \end{vmatrix} & +\begin{vmatrix} d & e \\ g & h \end{vmatrix} \\ -\begin{vmatrix} b & c \\ h & i \end{vmatrix} & +\begin{vmatrix} a & c \\ g & i \end{vmatrix} & -\begin{vmatrix} a & b \\ g & h \end{vmatrix} \\ +\begin{vmatrix} b & c \\ e & f \end{vmatrix} & -\begin{vmatrix} a & c \\ d & f \end{vmatrix} & +\begin{vmatrix} a & b \\ g & h \end{vmatrix} \end{bmatrix}$$

- For any invertible $A \in \mathbb{R}^{n \times n}$, $A^{-1} = \frac{1}{|A|} \text{adj}(A)$.

Quadratic Forms and Positive Semidefinite Matrices:

- The quadratic form of a square matrix A and vector x, denoted as $x^T Ax$ is:

$$x^T Ax = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} x_i x_j$$

- E.g. Let $A = \begin{bmatrix} 5 & -5 \\ -5 & 1 \end{bmatrix}$, $x = (x_1, x_2, x_3)$

The quadratic form is

$$(x_1, x_2, x_3) \begin{bmatrix} 5 & -5 \\ -5 & 1 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

- Note: $x^T Ax = (x^T Ax)^T$
$= x^T A^T x$
$= x^T \left(\frac{1}{2}A + \frac{1}{2}A^T\right)x$

- Let A be a symmetric matrix: $(A \in S^n)$

1. A is positive definite (PD), denoted as $A \succ 0$ or $A > 0$, if for all non-zero vectors $x \in \mathbb{R}^n$, $x^T Ax > 0$. The set of all pd matrices is denoted as S^n_{++}.

2. A is positive semidefinite (PSD), denoted as $A \succeq 0$ or $A \geq 0$, if $\forall$ vectors $x \in \mathbb{R}^n$, $x^T Ax \geq 0$. The set of all PSD matrices is denoted as S^n_+.

3. A is negative definite (ND), denoted as $A \prec 0$ or $A < 0$ if $\forall$ non-zero vectors $x \in \mathbb{R}^n$, $x^T A x < 0$.

4. A is negative semidefinite (NSD), denoted as $A \succeq 0$ or $A \leq 0$ if $\forall$ vectors $x \in \mathbb{R}^n$, $x^T A x \leq 0$.

5. A is indefinite if it is neither PSD nor NSD.
   I.e. A is indefinite if $\exists\ x_1, x_2 \in \mathbb{R}^n$ s.t.
   $x_1^T A x_1 > 0$ and $x_2^T A x_2 < 0$.

- Note:
  1. If A is PD, then $-A$ is ND and vice versa.
  2. If A is PSD, then $-A$ is NSD and vice versa.
  3. If A is indefinite, then so is $-A$.
  4. PD and ND matrices are always invertible.

- Given any matrix $A \in \mathbb{R}^{m \times n}$, the matrix $G = A^T A$, called the Gram Matrix is always PSD.

  Further, if $m \geq n$ and A is full rank, then $G = A^T A$ is PD.

Eigenvectors:

- Let $A \in \mathbb{R}^{n \times n}$ be a square matrix. $\lambda \in \mathbb{C}$ is an eigenvalue of A and $x \in \mathbb{C}^n$ is an eigenvector if $Ax = \lambda x$, $x \neq 0$.

- $Av = \lambda v$
  $= \lambda I v$
  $Av - \lambda I v = 0$
  $(A - \lambda I)v = 0$

  If v is non-zero, then we can solve for λ using the determinant (I.e. $|A - \lambda I| = 0$)

- E.g. Let $A = \begin{bmatrix} -6 & 3 \\ 4 & 5 \end{bmatrix}$. Find the eigenvalue(s).

Soln:

$$\left| \begin{bmatrix} -6 & 3 \\ 4 & 5 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right| = 0$$

$$\left| \begin{bmatrix} -6-\lambda & 3 \\ 4 & 5-\lambda \end{bmatrix} \right| = 0$$

$$(-6-\lambda)(5-\lambda) - 12 = 0$$
$$-30 + 6\lambda - 5\lambda + \lambda^2 - 12 = 0$$
$$\lambda^2 + \lambda - 42 = 0$$
$$(\lambda + 7)(\lambda - 6) = 0$$

$\lambda = 6$ or -7

- Properties:
  Let A be a square matrix.

  1. $\operatorname{tr} A = \sum_{i=1}^{n} \lambda_i$

  2. $|A| = \prod_{i=1}^{n} \lambda_i$

  3. The rank of A = num of non-zero eigenvalues.

  4. If A is invertible then $1/\lambda_i$ is an eigenvalue of A^{-1} with an associated eigenvector x_i.
     I.e. $A^{-1}x_i = (1/\lambda_i)x_i$.

  5. The eigenvalues of a diagonal matrix $D = \operatorname{diag}(d_1, \ldots, d_n)$ are just the diagonal entries $d_1, \ldots, d_n$.

- We can write all eigenvector eqns simultaneously as $AX = X\Lambda$ where:
  $X \in \mathbb{R}^{n \times n}$ are the eigenvectors of A
  Λ is a diagonal matrix whose entries are the eigenvalues of A.

  If the eigenvectors of A are linearly indep, then matrix X will be invertible. Then, $A = X\Lambda X^{-1}$.
  A matrix that can be written in this form is called diagonalizable.

- Let A be a symmetric matrix and $A \in S^n$.
  Then:
  1. The eigenvalues of A are real.
  2. The eigenvectors of A are orthonormal.
     (We'll replace X with U in this case.)

$$A = U \Lambda U^{-1}$$
$$= U \Lambda U^T \leftarrow \text{The inverse of an orthogonal matrix is just its transpose.}$$

Using this, we can show that the definiteness of a matrix depends entirely on the sign of its eigenvalues.

$$x^T A x = x^T U \Lambda U^T x$$
$$= y^T \Lambda y \leftarrow y = U^T x$$
$$= \sum_{i=1}^{n} \lambda_i \underbrace{y_i^2}_{\text{Always positive}}$$

If all λ_i's > 0, then the matrix is PD.
If all $\lambda_i \geq 0$, then the matrix is PSD.
If all $\lambda_i < 0$, then the matrix is ND.
If all $\lambda_i \leq 0$, then the matrix is NSD.
If some $\lambda_i \geq 0$ and other $\lambda_i < 0$, then the matrix is indefinite.

Matrix Operations:

1. Matrix Addition and Subtraction:

- We can add (or subtract) 2 matrices if they have the same dimensions.

- E.g.

1. $\begin{bmatrix} 1 & 7 \\ 2 & 0 \end{bmatrix} + \begin{bmatrix} 3 & 1 \\ 0 & 9 \end{bmatrix} = \begin{bmatrix} 4 & 8 \\ 2 & 9 \end{bmatrix}$

2. $\begin{bmatrix} 3 & 6 \end{bmatrix} - \begin{bmatrix} 2 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 4 \end{bmatrix}$

- For matrix addition / subtraction, you simply add or subtract the elements at the same index.

2. Matrix Multiplication:

- We can only multiply 2 matrices if the number of cols of the first matrix equals the number of rows of the second matrix. The resulting matrix has the same number of rows as the first matrix and the same number of cols as the second.

I.e. Let $A \in R^{m \times n}$, $B \in R^{n \times p}$, we can multiply them because A has n cols and B has n rows.

$C = A \cdot B$, then $C = R^{m \times p}$
C gets m rows from A and p cols from B.

- To multiply 2 matrices together, we need to dot product each row of the first matrix with each col of the second.

- E.g.

1. Multiply $\begin{bmatrix} 3 & 7 \\ 2 & 1 \end{bmatrix}$ with $\begin{bmatrix} 5 & 3 \\ 0 & 1 \end{bmatrix}$

Soln:

Dot product $(3, 7)$ with $(5, 0)$.

$(3, 7) \cdot (5, 0) = 3 \cdot 5 + 7 \cdot 0$

$= 15$

Dot product $(3, 7)$ with $(3, 1)$.

$(3, 7) \cdot (3, 1) = 3 \cdot 3 + 7 \cdot 1$

$= 16$

Dot product $(2, 1)$ with $(5, 0)$.

$(2, 1) \cdot (5, 0) = 2 \cdot 5 + 1 \cdot 0$

$= 10$

Dot product $(2, 1)$ with $(3, 1)$.

$(2, 1) \cdot (3, 1) = 2 \cdot 3 + 1 \cdot 1$

$= 7$

Resulting matrix: $\begin{bmatrix} 15 & 16 \\ 10 & 7 \end{bmatrix}$

2. Multiply $\begin{bmatrix} 3 & 2 \\ 7 & 1 \\ 0 & 6 \end{bmatrix}$ with $\begin{bmatrix} 9 & 4 & 6 \\ 1 & 7 & 0 \end{bmatrix}$

Soln:

$(3, 2) \cdot (9, 1) = 3 \cdot 9 + 2 \cdot 1$
$= 29$

$(3, 2) \cdot (4, 7) = 3 \cdot 4 + 2 \cdot 7$
$= 26$

$(3, 2) \cdot (6, 0) = 3 \cdot 6 + 2 \cdot 0$
$= 18$

$(7, 1) \cdot (9, 1) = 64$

$(7, 1) \cdot (4, 7) = 35$

$(7, 1) \cdot (6, 0) = 42$

$(0, 6) \cdot (9, 1) = 6$

$(0, 6) \cdot (4, 7) = 42$

$(0, 6) \cdot (6, 0) = 0$

Resulting matrix: $\begin{bmatrix} 29 & 26 & 18 \\ 64 & 35 & 42 \\ 6 & 42 & 0 \end{bmatrix}$

Multi-Variable Calculus Notes

Table of Contents:

The Gradient:

- Suppose $f: R^{m \times n} \to R$ is a function that takes as input a matrix A of size $m \times n$ and returns a real value. Then, the gradient of f with respect to $A \in R^{m \times n}$ is the matrix of partial derivatives.

I.e. $$\nabla f(A) = \begin{bmatrix} \frac{\partial f(A)}{\partial A_{11}} & \frac{\partial f(A)}{\partial A_{12}} & \cdots & \frac{\partial f(A)}{\partial A_{1N}} \\ \vdots & & & \vdots \\ \frac{\partial f(A)}{\partial A_{m1}} & \cdots & & \frac{\partial f(A)}{\partial A_{mn}} \end{bmatrix}$$

- Find the gradient in each of the next examples.

E.g. 1 $f(x, y, z) = xy + z$

Soln:

$$\frac{\partial f}{\partial x} = y \quad \frac{\partial f}{\partial y} = x \quad \frac{\partial f}{\partial z} = 1$$

$$\therefore \nabla f(x, y, z) = [y, x, 1]$$

E.g. 2 $f(x, y, z) = xy + yz + xz$

Soln:

$$\frac{\partial f}{\partial x} = y + z, \quad \frac{\partial f}{\partial y} = x + z \quad \frac{\partial f}{\partial z} = y + x$$

$$\therefore \nabla f(x, y, z) = [y + z, x + z, x + y]$$

- Some properties of gradients:
  1. $\nabla(f(x)+g(x)) = \nabla f(x) + \nabla g(x)$
  2. For $t \in \mathbb{R}$, $\nabla(tf(x)) = t(\nabla f(x))$

The Hessian:

- Suppose that $f: \mathbb{R}^n \to \mathbb{R}$ is a function that takes a vector in $\mathbb{R}^n$ and returns a real number. The Hessian matrix with respect to x, denoted as $\nabla_x^2 f(x)$ or H, is the $n \times n$ matrix of partial derivatives.

  I.e.

$$H = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \vdots & & & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n x_1} & \cdots & \cdots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{bmatrix}$$

- Find the Hessian matrix for each of the examples below:

  E.g. 1 $f(x,y) = x^3 - 2xy - y^6$

  Soln:

$$f_x = 3x^2 - 2y$$
$$f_y = -2x - 6y^5$$
$$f_{xx} = 6x$$
$$f_{yy} = -30y^4$$
$$f_{xy} = f_{yx} = -2$$

$$\therefore H = \begin{bmatrix} 6x & -2 \\ -2 & -30y^4 \end{bmatrix}$$

Eg. 2 $f(x,y) = y^4 + x^3 + 3x^2 + 4y^2 - 4xy - 5y + 8$

Soln:

$f_x = 3x^2 + 6x - 4y$

$f_y = 4y^3 + 8y - 4x - 5$

$f_{xx} = 6x + 6$

$f_{yy} = 12y^2 + 8$

$f_{xy} = f_{yx} = -4$

$$\therefore H = \begin{bmatrix} f_{xx} & f_{xy} \\ f_{yx} & f_{yy} \end{bmatrix} = \begin{bmatrix} 6x+6 & -4 \\ -4 & 12y^2+8 \end{bmatrix}$$

Eg. 3 $f(x,y) = x^2y + y^2x$

Soln:

$f_x = 2xy + y^2$

$f_y = x^2 + 2yx$

$f_{xx} = 2y$

$f_{yy} = 2x$

$f_{xy} = f_{yx} = 2x + 2y$

$$H = \begin{bmatrix} 2y & 2x+2y \\ 2x+2y & 2x \end{bmatrix}$$

- Note: The Hessian matrix is always symmetrical.

Gradients and Hessians of Quadratic and Linear Functions:

- $\nabla b^T x = b$
- $\nabla x^T A x = 2Ax$ if A is symmetric
- $\nabla^2 x^T A x = 2A$ if A is symmetric

Least Squares

- Let $A \in R^{m \times n}$ have a full rank.
  Let $b \in R^m$ s.t. $b \notin R(A)$
  In this situtation, we want to find a vector $x \in R^n$ s.t. Ax is as close to b as possible, as measured by the square of the Euclidean norm $(\|Ax-b\|_2)^2$.

  Using the fact $(\|x\|_2)^2 = x^T x$, we have:

$$(\|Ax-b\|_2)^2 = (Ax-b)^T(Ax-b)$$
$$= x^T A^T x - 2b^T Ax + b^T b$$

  Taking the gradient w.r.t x, we have:

$$\nabla_x (x^T A^T x - 2b^T Ax + b^T b)$$
$$= \nabla_x (x^T A^T x) - \nabla_x (2b^T Ax) + \nabla_x (b^T b)$$
$$= 2A^T Ax - 2A^T b$$

  Setting the last expression to 0, and solving for x, we get:

$$x = (A^T A)^{-1} A^T b$$

Gradients of the Determinant:

- Let $A \in \mathbb{R}^{n \times n}$. We want to find $\nabla_A |A|$.

$$\frac{\partial}{\partial A_{k\ell}} |A| = \frac{\partial}{\partial A_{k\ell}} \sum_{i=1}^{n} (-1)^{i+j} A_{ij} |A_{\setminus i \setminus j}|$$

$$= (-1)^{k+\ell} |A_{\setminus k \setminus \ell}|$$

$$= (\text{adj}(A))_{\ell k}$$

$$= (\text{adj}(A))^T$$

$$= |A| A^{-T}$$

Eigenvalues as Optimization:

- If we want to optimize (min or max) $f(x,y,z)$ subject to the constraint $g(x,y,z) = k$, we can do:
  1. $\nabla f(x,y,z) = \lambda \nabla g(x,y,z)$
  2. $g(x,y,z) = k$
  3. Plug all solns into $f(x,y,z)$ and find the max and min.

- $L(x, \lambda) = f(x) - \lambda g(x)$ is called Lagrange function.
- λ is called Lagrange Multiplier.
- Note: $\nabla g \neq 0$ at the point

- E.g.

1. Find the max and min of $f(x,y) = 5x - 3y$ subject to the constraint $x^2 + y^2 = 136$.

Soln:

$$\nabla f(x,y) = \begin{bmatrix} 5 \\ -3 \end{bmatrix} \quad \lambda \nabla g(x,y) = \begin{bmatrix} 2\lambda x \\ 2\lambda y \end{bmatrix}$$

$$5 = 2\lambda x$$
$$-3 = 2\lambda y$$
$$x^2 + y^2 = 136$$

$$x = \frac{5}{2\lambda}, \quad y = \frac{-3}{2\lambda}$$

$$\left(\frac{5}{2\lambda}\right)^2 + \left(\frac{-3}{2\lambda}\right)^2 = 136$$

$$\frac{25}{4\lambda^2} + \frac{9}{4\lambda^2} = 136$$

$$\frac{34}{4\lambda^2} = 136$$

$$\frac{17}{2\lambda^2} = 136$$

$$17 = 272\lambda^2$$

$$\frac{17}{272} = \lambda^2$$

$$\frac{1}{16} = \lambda^2$$

$\lambda = \pm \frac{1}{4}$

When $\lambda = \frac{1}{4}$:
$\rightarrow x = 10$
$\rightarrow y = -6$

When $\lambda = -\frac{1}{4}$:
$\rightarrow x = -10$
$\rightarrow y = 6$

$f(10, -6) = 68$ Max
$f(-10, 6) = -68$ Min

2. Find the max and min of $f(x,y,z) = x+z$ subject to the constraint $x^2+y^2+z^2 = 1$.

Soln:
$1 = 2\lambda x$
$0 = 2\lambda y$
$1 = 2\lambda z$
$x^2+y^2+z^2 = 1$

$x = \frac{1}{2\lambda}$, $y = 0$, $z = \frac{1}{2\lambda}$

$\left(\frac{1}{2\lambda}\right)^2 + (0)^2 + \left(\frac{1}{2\lambda}\right)^2 = 1$

$\frac{2}{4\lambda^2} = 1$

$\frac{1}{2\lambda^2} = 1$

$\lambda^2 = \frac{1}{2}$
$\lambda = \pm \frac{1}{\sqrt{2}}$

$x = \pm\frac{\sqrt{2}}{2} = \pm\frac{1}{\sqrt{2}} = z, \quad y = 0$

$\left(-\frac{1}{\sqrt{2}}, 0, -\frac{1}{\sqrt{2}}\right)$ and $\left(\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}\right)$ are the 2 points.

$f\left(\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}\right) = \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}$

$= \frac{2}{\sqrt{2}}$ Max

$f\left(-\frac{1}{\sqrt{2}}, 0, -\frac{1}{\sqrt{2}}\right) = -\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}}$

$= -\frac{2}{\sqrt{2}}$ Min

3. Find the max and min of $f(x,y,z) = x - y + z$ subject to the constraint $x^2 + y^2 + z^2 = 2$

$1 = 2\lambda x$

$-1 = 2\lambda y$

$1 = 2\lambda z$

$x^2 + y^2 + z^2 = 2$

$x = \frac{1}{2\lambda}, \quad y = \frac{-1}{2\lambda}, \quad z = \frac{1}{2\lambda}$

$3\left(\frac{1}{2\lambda}\right)^2 = 2$

$\frac{1}{4\lambda^2} = \frac{2}{3}$

$\lambda^2 = 3/8$

$\lambda = \pm\sqrt{3/8}$

When $\lambda = \sqrt{\frac{3}{8}}$

$$x = \frac{1}{2\lambda}$$
$$= \frac{1}{2\sqrt{3/8}}$$
$$= \frac{1}{\sqrt{12/8}}$$
$$= \frac{1}{\sqrt{3/2}}$$
$$= \sqrt{2/3}$$

$$y = -\sqrt{2/3}$$

$$z = \sqrt{2/3}$$

When $\lambda = -\sqrt{3/8}$

$$x = -\sqrt{2/3}$$

$$y = \sqrt{2/3}$$

$$z = -\sqrt{2/3}$$

$f(\sqrt{2/3}, -\sqrt{2/3}, \sqrt{2/3}) = \sqrt{2/3}$ Max

$f(-\sqrt{2/3}, \sqrt{2/3}, -\sqrt{2/3}) = -\sqrt{2/3}$ Min