CSCC11 Exam Notes

Table of Contents:

Week 1 Material (Intro to C11)
Week 2 Material (Linear and Non-linear Regression)
Week 3 Material (Logistic Regression)
Week 4 Material (kNN and Decision Trees)
Week 5 Material (Estimation and Class Conditionals)
Week 6 Material (Naive Bayes)
Week 7 Material (Unsupervised Learning)
Week 8 Material (Clustering)
Week 9 Material (MLP and CNN)

Week 1 Material

Intro to Machine Learning (ML):

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

- ML is all about "fitting" a function.

Fields of ML:

- Supervised Learning: Here, the data is labelled with correct answers.
I.e. Supervised learning is the ml task of learning a function that maps an input to an output based on example input/output pairs.

There are 2 main types of supervised learning:
1. Regression: The target/output is a real-valued number.
2. Classification: The target/output is a discrete label.

- Unsupervised Learning: Here, we are given a collection of unlabelled data which we wish to analyze and discover patterns within. There are 3 main types:
1. Density Estimation: Here, we estimate the parameters of a distribution that generated the data.
2. Dimensionality Reduction: Here, we aim to reduce the dimension of high dimensional data (E.g. Images)
3. Clustering: Here, we aim to group data with similar patterns together.

- Reinforcement Learning: Here, a computer seeks to learn the opt actions to take based on the state of the world and hence the consequences of past actions.

- Active Learning: Here, we seek to learn / come up with an algo to determine which training data to acquire. This is used bc obtaining data is expensive.

- 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 diff 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."

Terminology:
- Inputs = $[x_1, x_2, \ldots, x_n]^T$ ← Nx1 vector

Note: Inputs can be multi-dimensional.

E.g. $X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2N} \\ \vdots & & & \\ x_{M1} & x_{M2} & \cdots & x_{MN} \end{bmatrix}$

- Each X_i (row) is referred to as a feature, predictor or indep var.

- Data is the combination of X_i and y_i.

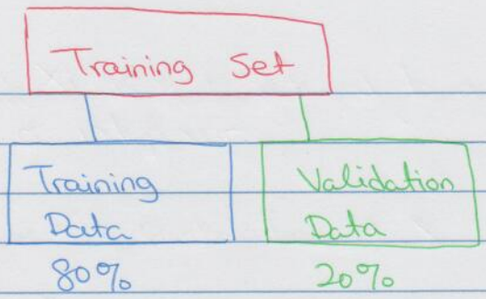
E.g.

	X				y	
	Height	Weight	Age	Gender	Which Person	
X_1						y_1
X_2						y_2
⋮						⋮
X_n						y_n

Each X_i (row) is a feature/predictor/indep var.
(X_i, y_i) is a data point

Training, Validation and Testing Data:
- We have a training set that's split into training and validation and a test set.
- The training set is split into training data (80%) and validation data (20%).
- The training data is used to learn the model's parameters and weights.
- The validation data provides an unbiased evaluation of a model fit on the training data while tuning the model's hyperparameters.
- The test set is used to provide an evaluation of a final model fitted on the training set.
- The error on each set (training and test) is called training error and test error, respectively.

- E.g.



- How to choose your model:

1. Have a training set and split it into training data (80%) and validation data (20%).

2. Train model on training.

3. Test on validation.

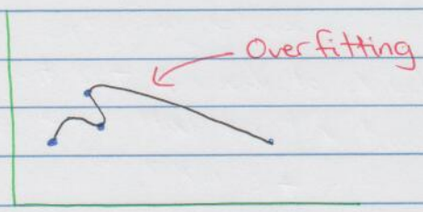
4. Repeat steps 2 and 3 until you find the model that performs best for the objective.

5. Deploy the model on test set.

Overfitting and Underfitting:

- Overfitting occurs when a model learns the data so well, it even learns the noise/outliers that isn't wanted. This negatively impacts the model's ability to generalize.

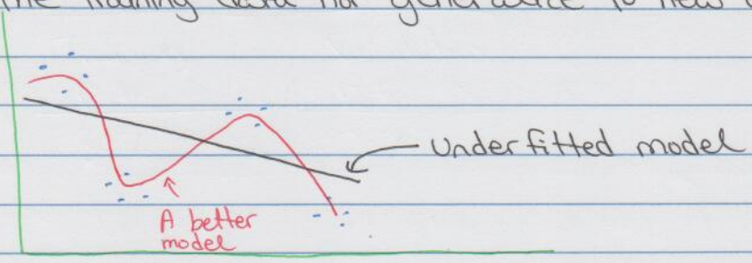
- E.g.



- We don't want to fit perfectly bc there could be noise.

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

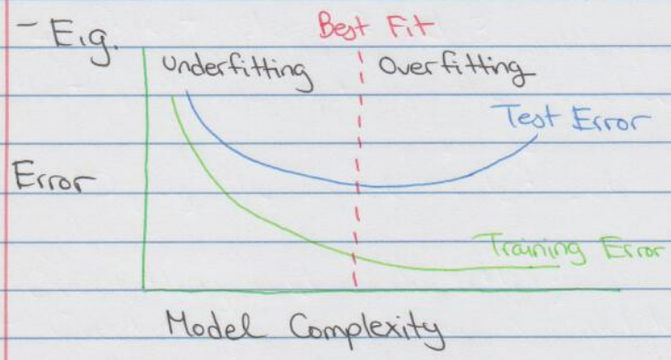
- E.g.



- If the model performs too well on the training data but performance drops significantly over the test set, then its overfitting.

- If the model performs poorly on both the training and test set, then its underfitting.

- E.g.



Likelihood Function:

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

$$L(\theta|x) = L(\theta_1, \dots, \theta_k | x_1, \dots, x_n) = \prod_{i=1}^{n} f(x_i|\theta)$$

- 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.

- To find the Max Likelihood Estimation (MLE):
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.

Eg. 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:

$N = 10$

$$L = \prod_{i=1}^{N} \theta(1-\theta)^{y_i - 1}$$

$$\log(L) = \log\left(\prod_{i=1}^{N} \theta(1-\theta)^{y_i-1}\right)$$

$$= \sum_{i=1}^{N} \log(\theta(1-\theta)^{y_i-1}) \leftarrow \log(a \cdot b) = \log(a) + \log(b)$$

$$= \sum_{i=1}^{N} \log(\theta) + (y_i - 1)\log(1-\theta) \leftarrow \log(b^n) = n\log(b)$$

$$= \sum_{i=1}^{N} \log(\theta) + \sum_{i=1}^{N} (y_i - 1)\log(1-\theta)$$

$$= 10\log(\theta) + 7\log(1-\theta)$$

$$\frac{\partial L}{\partial \theta} = \frac{10}{\theta} + \frac{-7}{1-\theta} = 0 \rightarrow 10(1-\theta) - 7(\theta) = 0$$

$$10 - 10\theta - 7\theta = 0$$

$$-17\theta = -10$$

$$\theta = 10/17$$

Discriminative and Generative Models:

- Discriminative models learn the boundaries btwn classes or labels in a dataset.
- They use $P(y|x) = \frac{P(x|y) \cdot P(y)}{P(x)}$
- Generative models model the actual distribution of each class.
- They use $P(x, y)$

Week 2 Material

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. $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$

Note: The x_i's can be a scalar or vector, but here it's a scalar.

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

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

  Note: x_i and y_i come from the training data.

- Let $L(w, b)$ be the loss function.

$$L(w,b) = \sum_{i=1}^{n} (e_i)^2$$

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

Note: y_i's are the actual values.
$wx_i + b$'s are the predicted values.

- We need to square the diff to remove negative values and to give greater weights to bigger differences.

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

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

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

$$0 = (-2)\left(\sum_{i=1}^{n} y_i - \sum_{i=1}^{n} wx_i + b\right)$$

$$0 = (-2)\left(\sum_{i=1}^{n} y_i - \sum_{i=1}^{n} wx_i - nb\right)$$

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

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

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

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

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

Now, we find w^*.

We first rewrite $L(w,b)$.

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

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

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

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

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

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

Multi-Dimensional Inputs:

- Now, let $x \in \mathbb{R}^D$. (x is now a Dx1 column vector)
  $y \in \mathbb{R}$

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

$$= \sum_{j=1}^{D} w_j x_j + b$$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{bmatrix} \leftarrow \text{1 data point, each with D features}$$

- To make $f(x)$ the result of a dot product we can add b as the last element in w and 1 as the last element of 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 $L(w)$ to be $\sum_{i=1}^{n} (y_i - w^T x_i)^2$

$$= \|\bar{y} - \tilde{X} w\|^2$$

$$\bar{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad \tilde{X} = \begin{bmatrix} — x_1^T — \\ — x_2^T — \\ \vdots \\ — x_n^T — \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1D} \\ 1 & x_{21} & x_{22} & \dots & x_{2D} \\ \vdots & & & & \\ 1 & x_{n1} & x_{n2} & \dots & x_{nD} \end{bmatrix}$$

- To find w^*:

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

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

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

$$\therefore w^* = \underbrace{(\tilde{X}^T\tilde{X})^{-1}}_{\text{Not always invertible}}(\tilde{X}^T\bar{y})$$

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 take to find the min is gradient descent.

$$w_{i+1} = w_i - \underbrace{\alpha}_{\text{Learning rate/Step size}}\left(\frac{\partial L(w)}{\partial(w)}\right)$$

- One problem is that the soln might not be unique.
Imagine having 2 identical features: $y = w_1x_1 + w_2x_1$
$= (w_1 + w_2)x_1 + 0x_1$
If (w_1, w_2) is 1 soln while $(w_1 + w_2, 0)$ is another soln then we have multi-collinearity.

Non-Linear Regression:

- We can introduce non-linearity by adding/using a basis function.
- There are 2 main basis functions:

1. Polynomial Model:

- The basis function, $b_k(x)$, is equal to x^k.

I.e. $b_k(x) = x^k$

- Then, $f(x)$ becomes:

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

2. Radial Basis Function:

- Abbreviated as 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.

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

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

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

Poly - Global fit
RBF - Local fit

- The hyperparameter of the polynomial model is the degree of the polynomial.

- The hyperparameters of RBF are the num of RBFs, μ and σ.

- Generally, you choose the hyperparameters during validation testing by seeing what parameters give the best result.

- 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 num of data points in higher dim input spaces.

2. Place one center at each data point. This limits the num of centers needed but can be expensive if there's too many data points.

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

b) To pick the width:

1. Try diff values and pick the best.

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

Regularization:

- Directly min squared error can lead to overfitting.

There are 2 solns to this:

1. Adding prior knowledge ← We'll use this
2. Handling uncertainty

- We can assume that the model is smooth.
- We can use regularization, which means adding an extra term, to "smooth" models.

- There are 2 types of regularized least squares:

1. L_2 Regularization / Ridge Regression:

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

- Is a way to deal with multicollinearity.
- λ is used to "control" w.
- Increasing λ increases $\|y - Xw\|^2$ but decreases $\|w\|^2$
- Used to prevent overfitting

2. L_1 Regularization / Lasso Regression:

- $L(w) = \|y - Bw\|^2 + \lambda \|w\|$

↑ Lagrange Multiplier variant of a constrained optimizer.

Week 3 Material

Bias-Variance Tradeoff:

- 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.

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

- 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 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 caused by unknown vars and from the chosen framing of the problem.

Classification:

- The label/output y is a category.
- With binary classification there are 2 categories and we classify the elements in 1 of the 2 categories.
- With multiclass classification there are more than 2 categories and we classify the elements in 1 of the categories.
- With multi-label classification we classify elements into 1 or more categories.

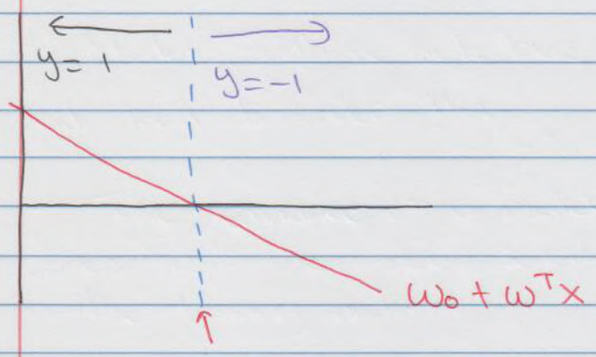
- A reasonable decision rule is:

$$y_i = \begin{cases} 1, & \text{if } f(x_i, w) \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

← Equivalent to

$$y = \text{sign}(w_0 + w^T x_i)$$

E.g.



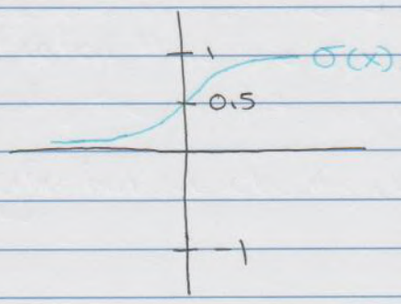
↑ Decision Boundary / Hyperplane

- The decision boundary/hyperplane separates the 2 groups.

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 will use the sigmoid/logistic function, denoted as σ.

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



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

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

$$= \sigma(w^T x)$$

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

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

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

- The decision boundary is given by:
$P(C_1|x) = P(C_2|x)$ or equivalently $\frac{P(C_1|x)}{P(C_2|x)} = 1$
↳ This is where we're not sure if the point belongs in $P(C_1|x)$ or $P(C_2|x)$

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

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

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

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

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

$$= \exp(w^T x)$$

$$\ln\left(\frac{P(C_1|x)}{P(C_2|x)}\right) = \ln(\exp(w^T x))$$

$$= w^T x$$

$$= \ln(1) \leftarrow \text{This is bc we earlier said that our decision boundary is } \frac{P(C_1|x)}{P(C_2|x)} = 1$$

$$= 0$$

→ 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 data set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ we want to model $P(y|x)$ using $P(y_1, \ldots, y_n | x_1, \ldots, x_n, \omega)$ s.t. the likelihood is maxed.

I.e. We want to $\arg\max_{\omega} (P(y_1, \ldots, y_n | x_1, \ldots, x_n, \omega))$

- Since $(x_i, y_i) \overset{iid}{\sim} D$, we can write $\arg\max_{\omega} (P(y_1, \ldots, y_n | x_1, \ldots, x_n, \omega))$ as

$$\arg\max_{\omega} \prod_{i=1}^{n} P(y_i | x_i, \omega)$$

← Note: Since $y_i \in \{0, 1\}$, we can assume it follows a Bernoulli Distribution

$$\Rightarrow \arg\max_{\omega} \prod_{i=1}^{n} P(C_1 | x_i, \omega)^{y_i} (1 - P(C_1 | x_i, \omega))^{1-y_i}$$

$$\Rightarrow \arg\max_{\omega} \prod_{i=1}^{n} P(C_1 | x_i, \omega)^{y_i} \cdot P(C_2 | x_i, \omega)^{1-y_i}$$

If we take the negative log of both sides, we get:

$$L(\omega) = \arg\min_{\omega} - \sum_{i=1}^{n} \left(\log y_i P(C=1 | x_i, \omega) + (1-y_i) \log (P(C=2 | x_i, \omega)) \right)$$

$$= \arg\min_{\omega} - \sum_{i=1}^{n} \left(y_i \cdot \log (P(C=1 | x_i, \omega)) + (1-y_i) \cdot \log (1 - P(C=1 | x_i, \omega)) \right)$$

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

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

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

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

$$\frac{\partial L(w)}{\partial w} = -\sum_{i=1}^{n} \; y_i(1-P_i)x_i - (1-y_i)P_i x_i$$

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

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

- With regularized logistic regression we have

$$L(w) = -\sum_{i=1}^{n} \left(y_i \log p_i + (1-y_i) \log (1-p_i) + \underbrace{\lambda \|w\|^2}_{L2} \right)$$

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

- The new gradient descent is

$$w_i^{(t+1)} = w_i^{(t)} - \alpha \left(\frac{\partial L(w)}{\partial w_i} \right) - 2\alpha\lambda w_i^{(t)}$$

Week 4 Material

k-Nearest Neighbour (kNN):

- In kNN, we classify an unknown point with the most common class "around" the 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: $f(x) = \text{sign}\left(\sum_{i \in N_k(x)} y_i\right)$

- There are 2 popular distance formulas 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 $X_1 = (3,4)$ and $X_2 = (5,3)$.

The Manhattan 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

$$\left(\sum_{j=1}^{p} (X_{1j} - X_{2j})^2\right)^{1/2}$$

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

The Euclidean Distance is $((3-5)^2 + (4-3)^2)^{1/2}$

$= (4+1)^{1/2}$

$= \sqrt{5}$

- Both the Manhattan and Euclidean Distances are part of the Minkowski Distance/ Lm Distance.

Let $a, b \in X$ 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.

- KNN Algo:

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 freq of these k chosen points in their classes.
5. Return the class with the highest freq 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.
Rule of thumb: $k < \sqrt{n}$, where n is the num of points.

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.

Note: Increasing k increases the model's complexity and thus the bias but it decrease variance.

Note: Decreasing k decreases bias but increases variance.

- Final notes on knn:

1. knn is a lazy classifier. It has no real training step as it simply stores the complete data which are needed during prediction.

knn does not need to learn the parameters.

2. Its parameter are the training data.

3. knn requires storing the entire training dataset.

4. As the num of points increase, it becomes computationally more expensive.

5. As the num of points and parameters increase, we call knn a non-parametric model.

6. knn is not based on any distributional or functional assumptions and, in theory, can model data situtations of any complexity.

7. knn's accuracy can be severely degraded by the presence of noisy or irrelevant features.

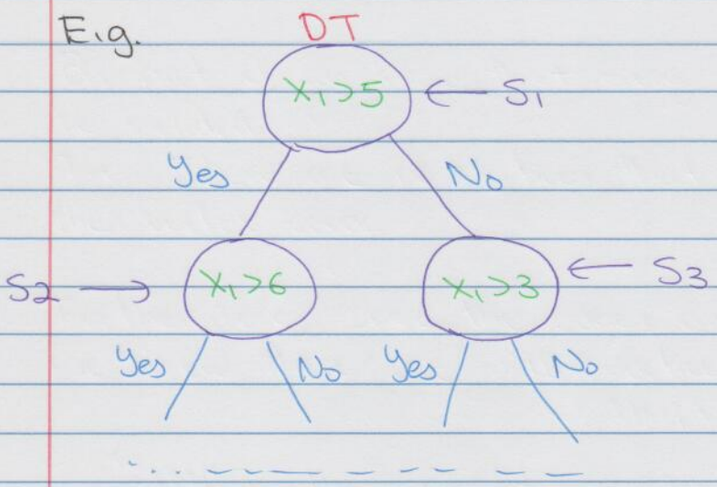
8. We often use the Euclidean distance to measure "closeness" but when the dimension is high, it will often become useless.

Decision Trees

- Decision Trees (DT) are usually binary trees but can be k-ary Trees.

- The idea is to split the space in half until we get a good prediction.

E.g.



- Suppose a binary tree has M internal nodes. These are the split functions.
Suppose there are m+1 leaf nodes.
At an internal node j, we define the split function to be $t_j(x): R^d \rightarrow \{-1, 1\}$.
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 labels.
We can represent it as $P(y = c \mid \text{leaf node } j) = \frac{N_{jc}}{N_j}$
where N_{jc} is 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 DT is NP-hard / NPC.

Algo:
1. Start from an empty DT.
2. Split on the best attribute.
3. Recurse.

- To decide the best split value, we will choose it s.t. the data in the left/right children has the min possible uncertainty.

- Entropy is a measure of uncertainty. It provides a measurement of uncertainty associated with a r.v. or random process.

For a discrete r.v. with possible outcomes $x_1, x_2, ..., x_n$ which occur with probability $P(x_1), P(x_2), ..., P(x_n)$, 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: $H(x) = E\left[-\log_2 P(x)\right]$

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

- E.g. Find the entropy of flipping a fair coin.

Soln:

$P_1 = P_2 = \frac{1}{2}$ $\quad P_1 \to P(X=H),\ P_2 \to P(X=T)$

$$H(X) = -\sum_{i=1}^{2} P(X_i) \log_2 P(X_i)$$

$$= -\left(P(X=H) \cdot \log_2 P(X=H) + P(X=T) \cdot \log_2 P(X=T) \right)$$
$$= -2\left(P(X=H) \cdot \log_2 P(X=H) \right)$$
$$= -2\left(\tfrac{1}{2} \cdot \log_2 \left(\tfrac{1}{2}\right) \right)$$
$$= -\left(\log_2 (1) - \log_2 (2) \right)$$
$$= -(0-1)$$
$$= 1$$

- E.g. Suppose we toss an unfair coin now s.t. $P(X=H) = 70\%$ and $P(X=T) = 30\%$. Find the new entropy.

Soln:

$P_1 = 0.7,\ P_2 = 0.3$

$$H(X) = -\sum_{i=1}^{2} P(X_i) \log_2 P(X_i)$$

$$= -\left(0.7 \cdot \log_2 (0.7) + 0.3 \log_2 (0.3) \right)$$

≈ 0.88 ← Since one side comes up more frequently, there is reduced uncertainty and hence entropy.

- E.g. Suppose we roll a fair six-sided die. Find the entropy.

Soln:

$P_1 = P_2 = \dots = P_6 = \frac{1}{6}$

$$H(X) = -\sum_{i=1}^{6} P(X_i) \cdot \log_2(P(X_i))$$

$$= -6\left(\frac{1}{6} \cdot \log_2\left(\frac{1}{6}\right)\right)$$
$$= -(\log_2(1) - \log_2(6))$$
$$= \log_2 6$$
$$\approx 2.58$$ ← Since the probability of rolling a die and landing on any side is less than the probability of flipping a coin, we have more uncertainty and thus more entropy.

- An entropy value of 0 means that there is no uncertainty.

- An entropy value of ∞ means there's a lot of uncertainty.

- The formula for conditional entropy is

$$H(y|X=x) = -\sum_{y} P(y|x) \cdot \log_2 P(y|x)$$

- Information gain/ Mutual information is a measure of how much the state of one var is known when it's conditioned on another var.

I.e. It is the reduction in entropy produced by partitioning the data according to a split test.

$$IG(Y|X) = H(Y) - H(Y|X)$$

$$IG(D_j, t_j) = H(D_j) - \frac{N_L}{N_j} H(D_L) - \frac{N_r}{N_j} H(D_r)$$

N_L is the num of data in the left child.
N_R is the num of data in the right child.

- Eg. $X = \{\text{Raining, Not Raining}\}$
$Y = \{\text{Cloudy, Not Cloudy}\}$

	Cloudy	Not Cloudy
Raining	$\frac{24}{100}$	$\frac{1}{100}$
Not Raining	$\frac{25}{100}$	$\frac{50}{100}$

a) What is the entropy of cloudiness given the knowledge of whether or not it's raining?

Soln:

$$H(\text{Cloudiness}) = H(\text{Cloudiness} | \text{Raining}) \cdot P(\text{Raining}) + H(\text{Cloudiness} | \text{Not Raining}) \cdot P(\text{Not Raining})$$

$$H(\text{Cloudiness} \mid \text{Raining}) = \sum_{P} (\text{Cloudy} \mid \text{Raining}) \cdot \log_2 P(C \mid R) + P(\text{Not Cloudy} \mid R) \cdot \log_2 (P(NC \mid R))$$

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

$$H(\text{Cloudiness} \mid \text{Not Raining}) = P(C \mid NR) \cdot \log_2 P(C \mid NR) + P(NC \mid NR) \cdot \log_2 P(NC \mid NR)$$

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

$$H(\text{Cloudiness}) \approx 0.75$$

b) How much info about cloudiness do we get by discovering whether it's raining.

Soln:

$$IG(y|x) = H(y) - H(y|x) \leftarrow \text{Got this from part a}$$

$$= 1 - 0.75$$

$$= 0.25$$

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

$$= -P(\text{Cloudy}) \cdot \log_2 P(\text{Cloudy}) - P(NC) \cdot \log_2 P(NC)$$

$$= -\frac{49}{100} \cdot \log_2\left(\frac{49}{100}\right) - \frac{51}{100} \cdot \log_2\left(\frac{51}{100}\right)$$

$$\approx 1$$

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

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

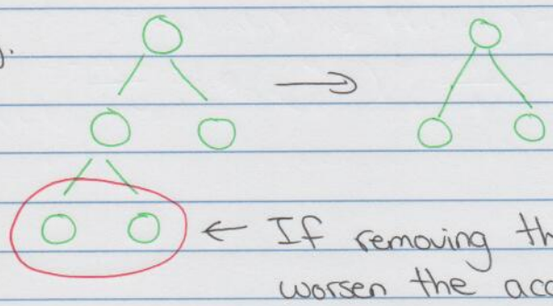
- As we grow the trees deeper, the model becomes more prone to overfitting.
I.e. Variance decreasing while variance increases.

- There are 2 ways we can reduce overfitting:
1. 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.

- This is short-sighted bc a seemingly worthless split can be followed by a good split.

- 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 doesn't worsen.

2. Ensemble Method/Random Forest:

- 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 (RF) 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)

- The hyperparameters are:
1. Num of data to sample.
2. Num of features to look at.
   Rule of thumb: If D is the total num of features, we look at $\sqrt{D}$ features.
3. Num of trees.

- When we do bootstrapping, we slowly inc the bias but we decorrelate the trees and hence the variance.

- To choose the hyperparameters, we want the model to make the prediction (Generalization). We will first use 1 of the following techniques to search over the hyperparameter space:
1. Grid Search
2. Random Search
3. Grad Student Descent

- For validation:
1. For a simple approach, we can use a metric we care about.
E.g. We can train using cross-entropy and validate using accuracy.
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 flaws of the first approach.
Algo:
a) Partition the training data into k partitions.
b) For each partition, train on the remaining k-1 partitions.
c) Validate each partition. Let L_i be the validation of the i^{th} partition.
d) Avg L_i for overall performance.

3. Leave One Out Cross-Validation (LOOCV) is a special case of k-Fold Cross Validation. The idea is that 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 the same as k-Fold.

LOOCV is useful for small datasets but is very expensive.

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

While k-Fold is very simple and empirical, 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. Safest to use random partitions to avoid the possibility that there are unmodeled correlations in the data.

Week 5 Material:

Baye's Rule:

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

Estimation:

- Estimation is determining the values of some unknown var from observed data. It is interested in finding point estimates.
I.e. It is finding a single estimate of the value of an unknown parameter.

- There are 2 main types of estimation:

1. Max 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 is the model. It's too focused on the training data.

- The problem with MLE is that if there's too little data, it can overfit. This is bc MLE solely depends on observed data.

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

- In MLE, the data/observations, D, is treated as r.v. but the model isn't.

2. 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 such as past experience or past experiments.

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

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

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

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

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

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

- In Bayesian Estimation, both the data and model are treated as r.v.

- In situtations where data is sparse, having some prior info/knowledge can be useful. However, unreliable priors can lead to biased models.

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

- There are 2 types of Bayesian Estimation:

a) 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 therefore may be treated as a constant.

Note: If $P(\theta)$ is non-informative (I.e. It is uniform), then MAP becomes MLE.

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

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

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

- Both MLE and MAP are optimization problems.

- Furthermore, both MLE and MAP ignore uncertainty in the parameters. This means we're choosing to put all our faith in the most probable model, but this may have surprising and undesirable results.

b) Bayes' Estimate:

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

Class Conditions:

- Here, we're modelling the distribution over the features themselves. These models are called generative models.

- In the case of binary classification, suppose we have 2 mutually exclusive classes C_1 and C_2. The prior prob of a data vector coming from class C_1 is $P(C_1) = P(y = C_1)$ and from class C_2 is $P(C_2) = P(y = C_2)$
  $= 1 - P(C_1)$

Each class has its own distribution for the feature vectors $P(X|C_1)$ and $P(X|C_2)$.

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

$P(X) = P(X, C_1) + P(X, C_2)$
$= P(X|C_1) \cdot P(C_1) + P(X|C_2) \cdot P(C_2)$ ← Generating data

- If one had such a model, one could draw data samples from it in the following way:
1. Randomly choose a class according to $P(C_1)$ and $P(C_2)$.
2. Conditioned on the class, we 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 num of elements in class 1 to the total number of elements.

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

$P(C_1|x) > P(C_2|x)$ or $\frac{P(C_1|x)}{P(C_2|x)} > 1 \rightarrow$ classify x to C_1

$P(C_1|x) < P(C_2|x)$ or $\frac{P(C_1|x)}{P(C_2|x)} < 1 \rightarrow$ classify x to C_2

$P(C_1|x) = P(C_2|x)$ or $\frac{P(C_1|x)}{P(C_2|x)} = 1 \rightarrow$ On decision boundary

- $P(C_i|x) = \frac{P(x|C_i) \cdot P(C_i)}{P(x)}$ ← 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 log domain as it's faster and more numerically stable.

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

- $P(C_i|x) = \frac{P(x|C_i) \cdot P(C_i)}{P(x)}$

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

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

- $P(C_1) = \frac{\#\text{ of class } C_1}{N}$

$P(C_2) = \frac{\#\text{ of class } C_2}{N}$

- To find the likelihoods:
1. Partition based on class.
2. Use all x_i's s.t. $y_i = C_j$ to learn $P(X|C_j)$
   Usually done via MLE.

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

Gaussian Class Conditionals:
- Also called Linear Discriminate Analysis (LDA).
- Assumes the likelihoods are Gaussians.
- For each class i, we want to model the i^{th} likelihood to be

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

($\bar{\mu}_i$: Mean, Σ_i: Co-variance)

It has $O(d^2)$ parameters where $\bar{x} \in R^d$.
To learn $\bar{\mu}_i$ and Σ_i, we use MLE.

Note: It is often easier to find the min log likelihood.

$$L(\mu, \Sigma) = -\ln(P(X_1, \ldots, X_N|\mu, \Sigma))$$
$$= -\sum_i \ln(P(X_1, \ldots, X_N|\mu, \Sigma))$$
$$= \sum_i \frac{(X_i-\mu)^T \Sigma^{-1}(X_i-\mu)}{2} + \frac{N}{2}\ln|\Sigma| + \frac{Nd}{2}\ln(2\pi)$$

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

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

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

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

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

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

$$-\frac{1}{2}(\bar{x} - \bar{\mu}_1)^T \Sigma_1^{-1} (\bar{x} - \bar{\mu}_1) + \frac{1}{2}(\bar{x} - \bar{\mu}_2)^T \Sigma_2^{-1} (\bar{x} - \bar{\mu}_2) \Big\} \text{ Quadratic w.r.t } \bar{x}$$

$$= 0$$

$\alpha(x)$ is a quadratic function.

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

Week 6 Material:

Intro to Naive Bayes:

- Naive Bayes (NB) aims to simplify the estimation problem by assuming that the diff input features (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 one d-dim density, we estimate d 1-dim 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 $d \times d$ covariance matrix with a diagonal matrix. The i^{th} entry is the variance of $P(X_i|C)$.

NB With Discrete Input Features:

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

- In this example, we'll assume that each input either has or doesn't have each feature.

Let each data vector be a list of discrete features (E.g. $[F_1, \ldots, F_d]$) and for simplicity, assume each feature is binary (I.e. $F_i = \{0,1\}$)

If we want to solve $P(F_1, F_2, F_3 | C=1)$ without using NB, we would get:

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

↑ 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 $P(F_2|F_3, C=1)$, since F_2 depends on F_3 and F_3 has 2 possible values (0,1), we need to model 2 diff distributions.

For $P(F_1|F_2, F_3, C=1)$, there are 4 possible values since F_2 and F_3 can have 2 possible values each, so we need to model 4 diff distributions.

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

With NB, only d parameters have to be learned. This is bc each feature is indep.

Using NB:

Let $a_{ij} = P(F_i = 1 | C = j)$

Let $b_j = P(C=j)$ ← Prior

$$P(C=j|F_{1:d}) = \frac{P(F_{1:d}|C=j)\,P(C=j)}{P(F_{1:d})}$$

$$= \frac{\pi_i P(F_i|C=j) \cdot P(C=j)}{\sum_\ell P(F_{1:d}, C=\ell)}$$

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

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

Because the prev computation can lead to underflow, it's safer to compute it in the log domain.

Eg. Suppose we observe N examples of class 0 and M examples of class 1. What is the prob of observing class 0?

Soln:

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

$$= b_1^M \cdot b_0^N$$
$$= b_0^N (1-b_0)^M$$

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

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

$$0 = N(1-b_0) - M \cdot b_0$$
$$= N - N b_0 - M b_0$$
$$= N - b_0(N+M)$$
$$(M+N) b_0 = N$$
$$b_0^* = \frac{N}{M+N}$$

NB Regularization:

- Consider we have N training vectors, F_k, each with an associated class label C_k.

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

$b_j = \frac{N_j}{N} \longrightarrow b_j = \underbrace{\frac{N_j + \beta}{N + k\beta}}_{\text{Regularization}}$ β is some constant and k is the num of classes

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

$a_{ij} = \frac{N_{ij}}{N_j} \longrightarrow a_{ij} = \underbrace{\frac{N_{ij} + \alpha}{N_j + 2\alpha}}_{\text{Regularization}}$ for some small value α

Pros and Cons of NB:

Pros:

1. Works fast bc of the conditional indep assumptions.
2. Works well with high dim data.

Cons:

1. The assumptions may not be easy to satisfy.

Week 7 [illegible] and 8 Material

Intro to Unsupervised Learning:

- With supervised learning, you're given both the inputs and outputs during training. Furthermore, the idea/algo is to produce the desired outputs for the given inputs.

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

- There a few types of unsupervised learning:
1. Dimension Reduction
2. Clustering
3. Data Density Modelling/Density Estimation

Dimension Reduction:

- Used to remove irrelevant/redundant info.
- Increasing the num of features will not always improve performance. Sometimes, it may even worsen it.
- Generally, the num of training data required increases exp with dim to avoid overfitting.
- The goal is to choose an opt set of features to lower dim.

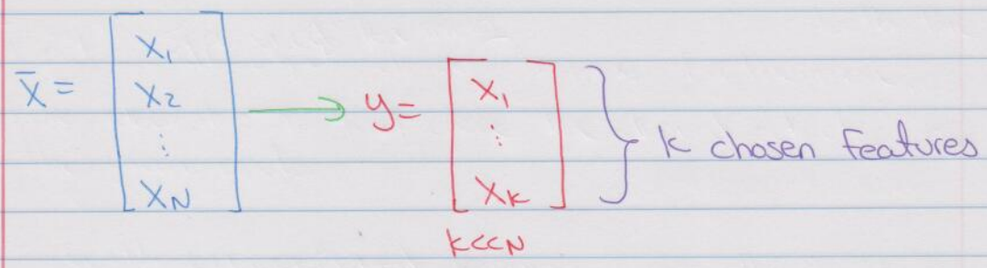
- Feature extraction finds a set of new features thru some mapping $f(\cdot)$ from existing features.

$$\bar{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \xrightarrow{f(x)} y = \begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix}$$

Could be linear or non-linear

$k << N$

- Feature selection chooses a subset of the original feature.



- Principal Component Analysis (PCA) is a technique for dim reduction. It aims to find a low-dim representation of high-dim data.

- Some uses and motivations of PCA are:

1. Visualization: High-dim data is very hard to visualize. PCA provides a way to project high-dim data onto 2^d or 3^d for easier 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 then we can use such representation to store and transmit data more efficiently.

PCA Algo:

- Assume each data point y_i has dim p.
I.e. $y_i \in R^p$

- There are 2 ways to reduce the dim:

1. 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 maximized.

- 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
↳ $\frac{y - \bar{y}}{SD(y)}$ ← $\bar{y}$ is the mean of y

- Then, the sample variance of the projected data is

$$\frac{\sum_{i=1}^{N} (U_1^T y_i)^2}{N} = U_1^T S U_1$$

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

- 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: $S u_1 = \lambda_1 u_1$

$u_1^T S u_1 = \lambda_1$ (Multiply both sides by u_1^T)

- The variance is max when u_1 = Eigenvector with largest eigenvalue.

Such $u_1^T y_i$ is called the First Principal Component.

- We can extend this to k-dim. Let $U = [u_1, \ldots, u_k]$ $(U \in R^{P \times k})$ The k^{th} principal component is $u_k^T y_i$. We can find this by ranking the eigenvalues.

The new (dimension reduced) data is:

$$x_i = \begin{bmatrix} u_1^T y_1 \\ u_2^T y_1 \\ \vdots \\ u_k^T y_1 \end{bmatrix} = U^T y_i \in R^k$$

Note: Principal components are uncorrelated.

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

2. 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 $W^T W = I_{k \times k}$

Identity matrix of size k

This constraint requires that we obtain an orthonormal mapping.

I.e. $w_i^T w_j = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}$

Without this, the problem would be underconstrained.

E.g. Assume $\alpha \neq 0$

$y_i = w^* x_i$

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

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

Steps:

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

2. Compute the data covariance 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. $D = \text{diag}(\lambda_1, \ldots, \lambda_d)$
   $V = [v_1, \ldots, v_d]$ contains the orthonormal eigenvectors. I.e. $V^T V = I_d$

4. Assume the eigenvalues are sorted from largest to smallest. If that's not the case, sort it along with its corresponding eigenvector from largest to smallest.

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

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

- Suppose we learned a PCA model and are given a new y_{new} value. To estimate its corresponding x_{new} value, do:

1. Min $\|y_{new} - (W x_{new} + b)\|^2$. However, since W is orthonormal, the soln simplifies to $x_{new}^* = W^T(y_{new} - b)$

- Some properties of PCA are:

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

2. Max Variance 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's common to use the ratio of the total LS error and the total variance in the training data (The sum of all eigenvalues). One needs to choose a k large enough s.t. this ratio is small (often ≤ 0.1).

Proportion of Variance: The eigenvalues of the covariance matrix give the variance of each component.

$$\frac{\sum_{j=1}^{k} \lambda_j}{\sum_{j=1}^{p} \lambda_j}, \quad \sum_{j=1}^{p} \lambda_j = \sum_{j=1}^{p} \text{Var}(X_j) = p$$

- Some final comments about PCA are:

1. Generally, it's critical to perform standardization prior to PCA. PCA is very sensitive to the variances of the initial vars.
Ie. If there are large diff btwn the range of the initial variables, the variables with the larger ranges will dominate over those with smaller ranges.

2. Pros:
   a) Removes correlated features. Used a lot to deal with multi-collinearity.
   b) Fast if there's few features.
   c) Reduces overfitting if there's few features.

3. Cons:
   a) Less interpretable as it transforms the original data.
   b) Must do standardization
   c) Info loss w/o proper num of components

Clustering:

- Used to group points into clusters, which are collections of points that are similar in some way.

- 2 main techniques:

1. K-Means:

- 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.

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

We assume that each data point belongs to 1 cluster.

- We also want to find a representative of each cluster, c_j. Let c_j be the mean of the points assigned to that cluster.

Then, the obj-fun is:

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

The obj fun penalizes the squared Euclidean dist btwn each data point and the center of the cluster to which it's assigned. To min this error, we want to bring the cluster centers close to the points within their clusters and we want to assign each data point to the nearest cluster.

- The opt problem is NP-hard and can't be solved in closed form. Furthermore, we can't use gradient descent as it includes discrete labels (elements of L). 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 [illegible] chance that you can get stuck at a local minima with this algo.

- In step 1 of the algo, 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 L randomly and run step 3 to determine the inital centers. This isn't very good bc the initial centers will likely end up being very close to the mean of the entire dataset.

2. Random Initial Centers: Choose the inital center values randomly. This also isn't very good bc some centers will fall into empty regions of the feature space.

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

4. Multiple Restarts: Run k-Means multiple times, each time with a diff random initialization and keep the soln that gives the lowest value of the obj fun.

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

2. Mixture of Gaussians (MoG) / Gaussian Mixture Models (GMM):

- A generalization of k-Means. k-Means works well for well-separated clusters that are more or less spherical. MoG works better if the clusters are oval/oblong shaped and can also handle overlapping.

- MoG comprises a linear comb 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 prob, often called mixing prob, represent the fraction of the data generated by the diff Gaussian components.

- We often use a shorthand to capture all the parameters with a single var
$\Theta = \{m_{1:k}, \mu_{1:k}, C_{1:k}\}$

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

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

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

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

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

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

- Going back to the eqn, we can now write $P(y|\theta)$ as:

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

(m_j: Prior)

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

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

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

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

$= -\sum_{i=1}^{N} \ln(P(y_i | \theta))$ ← No closed form soln

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

- We will use Expectation-Maximization (EM) algo.

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

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

In EM, we opt both θ and r_{ij}.

- EM algo has 2 major steps:

1. E-Step: Compute the posterior prob that each Gaussian generates each data point. (updates r)

2. M-Step: Assuming that the data was generated this way, change the parameters of each Gaussian to max the prob that it would generate the data it's responsible for. (update θ)

- The algo alternates btwn the E-Step and M-Step.

- Algo for GMM:

1. Initialize r_{ij}'s and θ

2. For each point i, compute r_{ij} as shown below:

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

$$= \frac{P(y_i \mid l=j, \theta) \cdot P(l=j \mid \theta)}{\sum_{j=1}^{k} P(y_i \mid l=j, \theta) \cdot P(l=j \mid \theta)} \quad \text{E-Step}$$

3. For each cluster j, compute θ_j as shown below:

$$m_j = \frac{\sum_i r_{ij}}{N}, \quad \mu_j = \frac{\sum_i r_{ij} \cdot y_i}{\sum_i r_{ij}} \quad \text{M-Step}$$

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

4. Repeat 2 and 3 until termination condition.

- 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.

Week 9 Material

Multilayer Perceptrons:

- 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 (NN) is just a combination of these.

- The simplest kind of NN is Multilayer Perceptron (MLP). MLP is a type of artifical 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.

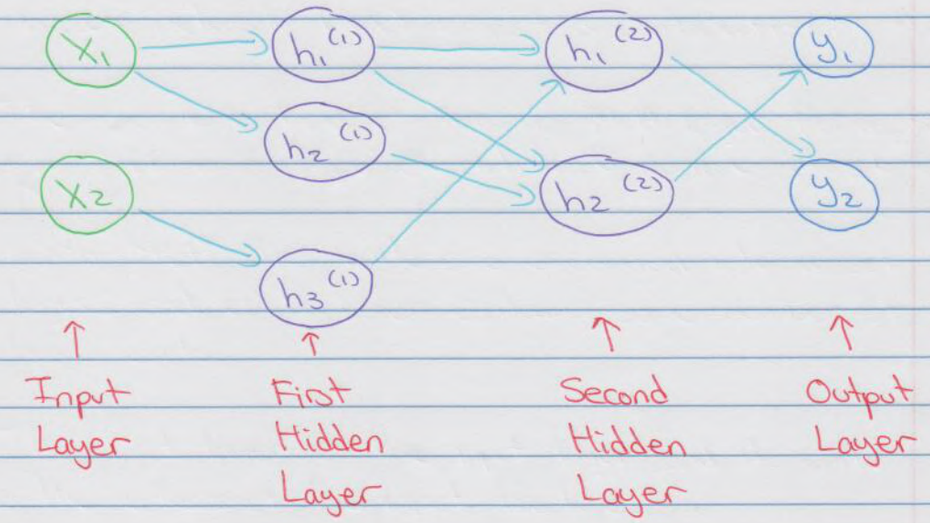
- All layers in btwn the input and output layers are known as hidden layers bc we don't know ahead of time what these units should compute and this needs to be discovered during learning.

- The num of layers is called the depth.
If depth = 1, we have a shallow NN.
If depth > 1, we have a deep NN.

- The num of units in a layer is the width.

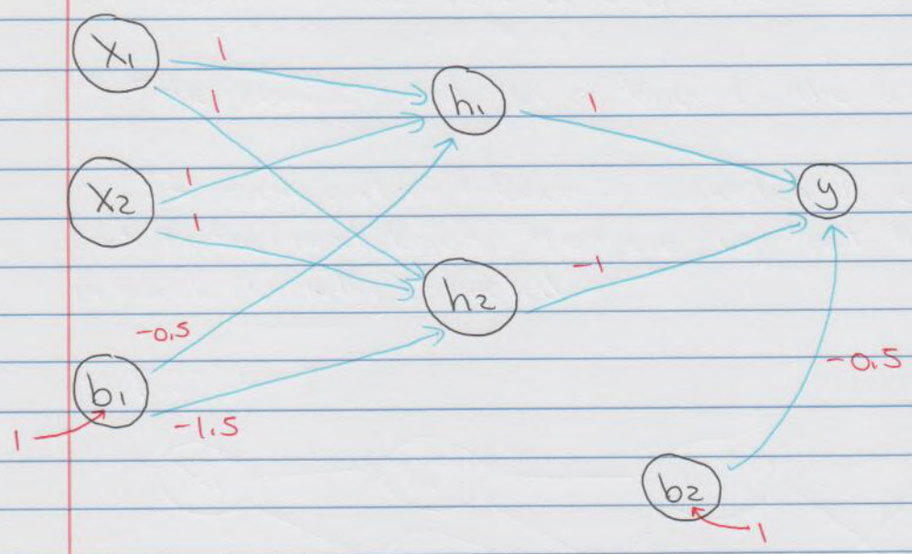
- If every unit in 1 layer is connected to every unit in the next layer, then we say that the network is fully connected.

- E.g. 1



Depth = 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

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

Consider $X_1 = 0$ and $X_2 = 0$

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

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

$y = w_{h1} \cdot h_1 + w_{h2} \cdot h_2 + w_{b2} \cdot b_2$
$= (1)(0) + (-1)(0) + (-0.5)(1)$
$= -0.5$
$= 0$ ← Bc of activation function

Now, consider $X_1 = 1$ and $X_2 = 0$.

$h_1 = w_1 x_1 + w_2 x_2 + w_{b1} \cdot b_1$
$= (1)(1) + (1)(0) + (-0.5)(1)$
$= 0.5$
$= 1$

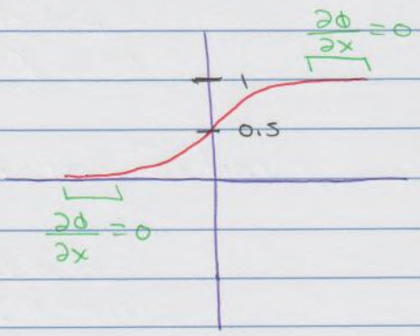
$h_2 = w_1 x_1 + w_2 x_2 + w_{b1} \cdot b_1$
$= 0$

$y = w_{h1} \cdot h_1 + w_{h2} \cdot h_2 + w_{b2} \cdot b_2$
$= (1)(1) + (-1)(0) + (-0.5)(1)$
$= 1$

- Some activation functions are:

1. Sigmoid Function:

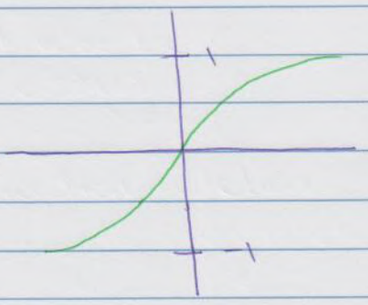
$$\phi(x) = \frac{1}{1+e^{-x}}$$



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

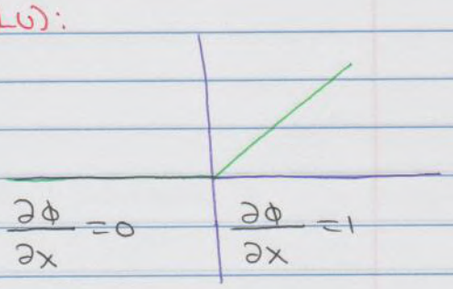
2. Tanh Function:

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



3. Rectified Linear Unit (ReLU):

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



4. Swish Function:

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

- For hidden layer i: $h_i^{(1)} = \phi_i^{(1)}\left(\sum_j w_{ij}^{(1)} x_j + b_i^{(1)}\right)$

E.g.

$h_1^{(1)} = \underbrace{\phi_1^{(1)}}_{\text{Activation Function}}\left(\sum_j w_{ij}^{(1)} \cdot x_j + b_1^{(1)}\right)$

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

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

$w_1 = \left[w_{11}^{(1)}, \ldots, w_{1N}^{(1)}, b_1^{(1)}\right]$

$X = \left[x_1, \ldots, x_N, 1\right]$

Suppose the width is k. (I.e. There are k hidden units).

$h^{(1)} = \phi^{(1)}(w^{(1)} X)$

$$w^{(1)} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & \cdots & w_{1N}^{(1)} & b_1^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & & w_{2N}^{(1)} & b_2^{(1)} \\ \vdots & \vdots & & \vdots & \\ w_{K1}^{(1)} & w_{K2}^{(1)} & \cdots & w_{KN}^{(1)} & b_K^{(1)} \end{bmatrix}$$

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

- For hidden layer 2, we have:

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

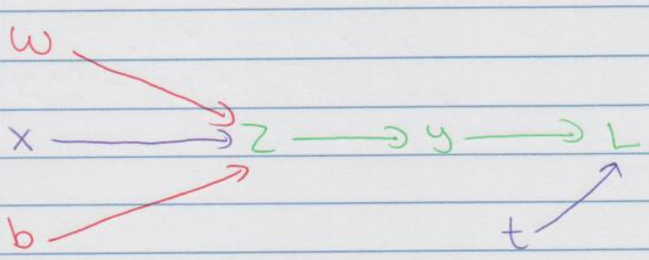
Note: If the $\phi^{(i)}$'s are linear, then we'll just have a bunch of matrix multiplications. Furthermore, if you don't have an activation function or all the activation functions are identity, you get linear regression.

- For hidden layer L: $h^{(L)} = \phi^{(L)}(W^{(L)} h^{(L-1)})$

- To learn the W_i's, we need to do forward pass and backward pass/backpropagation.

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 act func ϕ s.t. $y = \phi(z)$.

I.e.



- are params
- are input/output
- are operations

Forward pass is computing 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 $\frac{\partial L}{\partial b}$.

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

To do backpropagation:
1. $\frac{\partial L}{\partial y} = y - t$

2. $\frac{\partial y}{\partial z}$ ← we don't know what ϕ is so we can't say much for now

3. $\frac{\partial z}{\partial w} = x$

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

Convolutional Neural Networks (CNN):

- ANN and MLP in particular are not very good with analyzing visual images.

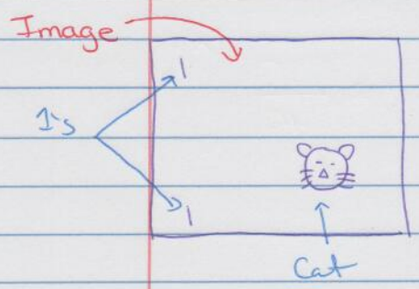
The first reason is that with MLP, because it usually involves fully connected networks, it's prone to overfitting.

The second reason is that it uses too many parameters and computations. Consider a 224 x 224 pixel image s.t. each pixel can be one of 3 colors. We'll need 3x224x224 or 150, 528 weights.

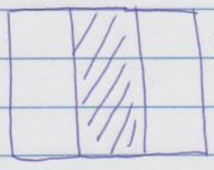
- CNN is used for classification of images and computer vision tasks.

- CNN assumes that the inputs are images and uses this fact to find patterns. We can use these patterns to reduce the num of weights. This assumption is called inductive bias.

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



We can overlap a filter/kernel on the image and move it around to see if it matches anywhere on the pic.



- The layers in CNN have the neurons arranged in 3 dims (length, width, depth) where they refer 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)

- In the convolutional 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 filter and input pixels. Then, we shift the filter by a stride (the stride determines the amount of movement for the filter) and do the dot product. The final output from the series of dot product is called the feature map/activation map/convolved feature.

Note: The stride doesn't have to be a continguous piece. It could be made up of elements nowhere near each other.

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

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

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

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

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

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

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

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

- In the pooling layer we try to reduce the dimensionality of the feature map. It combines a set of values into a smaller set by extracting only useful info and discarding irrelevant details.

- This layer serves 2 purposes:
1. Reduce the num of parameters/weights.
2. Control overfitting.

- There are 2 main pooling methods:
1. Average Pooling:
- We divide the feature map into rectangular regions/ rectangular pooling regions and compute the avg value of each region.

- E.g.

Feature Map

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

→

Reduced Feature Map

3.25	5.5
2.5	2.75

2. Max Pooling:
- We divide the feature map like previously, but this time, we get the max of each region.

- In the example above,

6	8
4	7

would be the reduced feature map.

- Note: There could be some overlaps in the pooling regions.
- Note: The pooling region size is another hyperparameter. If we use an extremely large region, we may lose out on key info.