9/5

- N-gram model
  - A sentence is a sequence of words (tokens). Each unique word token is called a word type. The set of word types is called the vocabulary.
  - Assumes that the word at i is dependent of the n previous words
- Unigram model
  - Distribution of each word is not depend on any previous words
  - A unigram model generate the likelihood for each word in the vocabulary, choose the one with the maximum likelihood
  - Prob of observing a sequence is P1 * P2 * P3
- Bigram model
  - 📗 Bigram model assumes Markov property:
    $\mathbb{P}\{w_1, w_2, \ldots, w_d\} = \mathbb{P}\{w_1\}\mathbb{P}\{w_2|w_1\}\mathbb{P}\{w_3|w_2\}\ldots\mathbb{P}\{w_d|w_{d-1}\}$.
    📗 Markov property means $\mathbb{P}\{w_t|w_{t-1}, w_{t-2}, \ldots w_0\} = \mathbb{P}\{w_t|w_{t-1}\}$ or the probability distribution of observing w_t only depends on the previous word in the sentence w_{t-1}. A visualiation of Markov chains: Link.
    📗 The maximum likelihood estimator of $\mathbb{P}\{w_t|w_{t-1}\}$ is $\hat{\mathbb{P}}\{w_t|w_{t-1}\} = \frac{c_{w_{t-1}w_t}}{c_{w_{t-1}}}$,
  - P(wt | wt-1) = # of time wt-1wt appears / number of time wt-1 appears
- Maximum likelihood estimation
  - Given a training set, P is estimated by P(wt) = c_{wt}/(c1 + c2 + … + cm)
    - Number of times a word appear in the training set / total words in the training set
- Transition matrix
  - The bigram probabilities can be stored in a transition matrix of markov chain
- Trigram model
  - 📗 The same formula can be applied to trigram models:
    $\hat{\mathbb{P}}\{w_t|w_{t-1}, w_{t-2}\} = \frac{c_{w_{t-2}w_{t-1}w_t}}{c_{w_{t-2}w_{t-1}}}$.
    📗 In a document, some longer sequences of tokens never appear, for example, when $w_{t-2}w_{t-1}$ never appears, the maximum likelihood estimator $\hat{\mathbb{P}}\{w_t|w_{t-1}, w_{t-2}\}$ will be $\frac{0}{0}$ and undefined. As a result, Laplace smoothing (add-one smoothing) is often used:
    $\hat{\mathbb{P}}\{w_t|w_{t-1}, w_{t-2}\} = \frac{c_{w_{t-2}w_{t-1}w_t} + 1}{c_{w_{t-2}w_{t-1}} + m}$, where m is the number of unique words in the document.
    📗 Laplace smoothing can be used for bigram and unigram models too:
    $\hat{\mathbb{P}}\{w_t|w_{t-1}\} = \frac{c_{w_{t-1}w_t} + 1}{c_{w_{t-1}} + m}$ for bigram and $\hat{\mathbb{P}}\{w_t\} = \frac{c_{w_t} + 1}{c_1 + c_2 + \ldots + c_m + m}$ for unigram.
  - Laplace smoothing

- advantages:
  - deal with 0/0 cases
  - Able to generate new phrases
- General laplace smoothing

$$\hat{\mathbb{P}}\{w_t|w_{t-1}, w_{t-2}, \ldots, w_{t-N+1}\} = \frac{c_{w_{t-N+1}, w_{t-N+2}, \ldots, w_t} + \delta}{c_{w_{t-N+1}, w_{t-N+2}, \ldots, w_{t-1}} + \delta m}.$$

-
- As delta increase, more likely to generate new phrases

9/10

- Natural language processing
  - When processing language data, documents need to be first turned into sequences of word tokens
  - Stemming (lemmatization) - looks, looked, looking → look
  - Tokenization
    - Bag of words feature
      - One numerical vector for each document
      - Length = size of the vocabulary
      - xj = xj/(x1+...+xm)
        - x is the # appearance of a word in the document
    - TF IDF features
      - Term frequency - inverse document frequency
      - Term frequency is defined the same way as in the bag of words features, $TF_{ij} = \frac{c_{ij}}{c_{i1} + c_{i2} + \ldots + c_{im}}$.

        Inverse document frequency is defined as $IDF_j = \log\left(\frac{n}{|\{i : c_{ij} > 0\}|}\right)$, where $|\{i : c_{ij} > 0\}|$ is the number of documents that contain word j.

        TF IDF representation of document i is $x_i = (x_{i1}, x_{i2}, \ldots, x_{im})$, where $x_{ij} = TF_{ij} \cdot IDF_j$.

Document	Phrase	Number of times
"Guardians of the Galaxy"	"I am Groot"	13
-	"We are Groot"	1
"Guardians of the Galaxy Vol. 2"	"I am Groot"	17
"Guardians of the Galaxy Vol. 3"	"I am Groot"	13
-	"I love you guys"	1

        Answer:

        bag of words: (I, am, groot, we, are , love, you, guys)
        document 1: [13/42, 13/42, 1/3, 1/42, 0, 0, 0]
        document 2: [1/3, 1/3, 1/3, 0, 0, 0, 0, 0]
        TF-IDF
        document 1: [log(3/3)*13/42, log(3/3)*13/42, log(3/3)*13/42, log(3/3)*13/42, log(3/1)*1/3, log(3/1)*1/42, log(3/1)*1/42, 0, 0, 0]
- Supervised learning examples
  - Given emails, predict whether they are spam or ham
  - Given comments, oredict whether they are offensive or not
  - Given reviews, predict whether they are positive/negative

- Discriminative vs generative models
- Conditional prob
  - P(B|A) = P(A, B)/P(A)
- Joint prob
  - P(A, B)
- Marginal prob
  - P(A) = P(x=1, y=1) + P(x=1, y=0)
- Bayes rule
  - P(A|B) = P(B|A)*P(A)/P(B)
  - P(B|A) = P(A, B)/P(A)
  - Prior probability
    - P(A) - the probability of a specific label
  - Likelihood
    - P(B|A) - chance for the
- Naive bayes classifier
  - Simple Bayesian network that assumes the features are independent
  - The key assumption is the independence assumption
    - $\mathbb{P}\{x_i|y\} = \mathbb{P}\{x_{i1}, x_{i2}, \ldots, x_{im}|y\} = \mathbb{P}\{x_{i1}|y\}\mathbb{P}\{x_{i2}|y\}\ldots\mathbb{P}\{x_{im}|y\}.$
- Multinomial naive bayes
  - [4 points] Consider the problem of detecting if an email message is a spam. Say we use four random variables to model this problem: a binary class variable S indicates if the message is a spam, and three binary feature variables: C, F, N indicating whether the message contains "Cash", "Free", "Now". We use a Naive Bayes classifier with associated CPTs (Conditional Probability Table):

Prior	$\mathbb{P}\{S=1\} = 0.87$	-	-
Hams	$\mathbb{P}\{C=1\|S=0\} = 0.35$	$\mathbb{P}\{F=1\|S=0\} = 0.27$	$\mathbb{P}\{N=1\|S=0\} = 0.46$
Spams	$\mathbb{P}\{C=1\|S=1\} = 0.4$	$\mathbb{P}\{F=1\|S=1\} = 0.96$	$\mathbb{P}\{N=1\|S=1\} = 0.3$

  - Compute $\mathbb{P}\{S=1 \mid C=0, F=1, N=0\}$.
  - Answer: Calculate
  - computing prob that the new email is a spam given the email contains free, but not cash and now
    first apply bayes rule
    P(S=1, C=0, F=1, N=0) = P(C=0, F=1, N=0|S=1)*P(S=1)/(P(C=0, F=1, N=0|S=1)*P(S=1) + P(C=0, F=1, N=0|S=0)*P(S=0))
    then apply naive bayes assumption
    = P(C=0|S=1)P(F=1|S=1)P(N=0|S=1)P(S=1)/(P(C=0|S=1)P(F=1|S=1)P(N=0|S=1)*P(S=1) + P(C=0|S=1)P(F=1|S=1)P(N=0|S=1)*P(S=0))
- In this class, log = $\log_e$
- Other naive bayes models
  - Multinomial naive bayes
    - Assume that follow multinomial distribution
  - Gaussian naive bayes
    - If the features are continuous, assume the features follow gaussian distribution
  - Bayesian network
    - A network of relationship between features, so they do not depend only on labels

9/12

- For this course, use 0.5 as the classification threshold
- Unsupervised learning
  - Data are not labeled
  - Clustering - separates items into groups
  - Novelty(outlier) detection - finds items that are different (two groups)
  - Dimensionality reduction - represents each item by a lower dimensional feature vector while maintaining key characteristics
  - Applications
    - Google news
    - Image segmentation
    - Text processing
    - Data visualization
- Principal component analysis
  - math
    - A vector u_k is a unit vector if it has length 1: $\|u_k\|^2 = u_k^\top u_k = u_{k1}^2 + u_{k2}^2 + \ldots + u_{km}^2 = 1$.
    - Two vectors u_j, u_k are orthogonal (or uncorrelated) if $u_j^\top u_k = u_{j1}u_{k1} + u_{j2}u_{k2} + \ldots + u_{jm}u_{km} = 0$.
    - The projection of x_i onto a unit vector u_k is $\left(u_k^\top x_i\right)u_k = (u_{k1}x_{i1} + u_{k2}x_{i2} + \ldots + u_{km}x_{im})u_k$ (it is a number $u_k^\top x_i$ multiplied by a vector u_k). Since u_k is a unit vector, the length of the projection is $u_k^\top x_i$.
    - The (unbiased) estimate of the variance of $x_1, x_2, \ldots, x_n$ in one dimensional space ($m = 1$) is $\frac{1}{n-1}\left((x_1 - \mu)^2 + (x_2 - \mu)^2 + \ldots + (x_n - \mu)^2\right)$, where μ is the estimate of the mean (average) or $\mu = \frac{1}{n}(x_1 + x_2 + \ldots + x_n)$. The maximum likelihood estimate has $\frac{1}{n}$ instead of $\frac{1}{n-1}$.
    - In higher dimensional space, the estimate of the variance is $\frac{1}{n-1}\left((x_1 - \mu)(x_1 - \mu)^\top + (x_2 - \mu)(x_2 - \mu)^\top + \ldots + (x_n - \mu)(x_n - \mu)^\top\right)$. Note that μ is an m dimensional vector, and each of the $(x_i - \mu)(x_i - \mu)^\top$ is an m by m matrix, so the resulting variance estimate is a matrix called variance-covariance matrix.
    - If $\mu = 0$, then the projected variance of $x_1, x_2, \ldots, x_n$ in the direction u_k can be computed by $u_k^\top \Sigma u_k$ where $\Sigma = \frac{1}{n-1}X^\top X$, and X is the data matrix where row i is x_i.
    - ⇨ If $\mu \neq 0$, then X should be centered, that is, the mean of each column should be subtracted from each column.
  - Find the direction that maximize the projected variance
    - $$\max_{u_k} u_k^\top \Sigma u_k \text{ subject to } u_k^\top u_k = 1.$$
  - Pick the largest eigenvalue and the corresponding eigenvector
  - How to pick the principle dimension K
    - Number of non-zero eigenvalues
    - Selected based on prior knowledge
    - Number of eigenvalues larger than some threshold
  - Reduced feature space
    - Image the original data into the new vector space defined by the remaining eigenvectors

An original item is in the m dimensional feature space: $x_i = (x_{i1}, x_{i2}, \ldots, x_{im})$.
The new item is in the K dimensional space with basis $u_1, u_2, \ldots, u_k$ has coordinates equal to the projected lengths of the original item: $(u_1^\top x_i, u_2^\top x_i, \ldots, u_k^\top x_i)$.
Other supervised learning algorithms can be applied on the new features.

- u are unit vectors

- Reconstruction

The original item can be reconstructed using the principal components. If all m principal components are used, then the original item can be perfectly reconstructed: $x_i = (u_1^\top x_i)u_1 + (u_2^\top x_i)u_2 + \ldots + (u_m^\top x_i)u_m$.
The original item can be approximated by the first K principal components: $x_i \approx (u_1^\top x_i)u_1 + (u_2^\top x_i)u_2 + \ldots + (u_K^\top x_i)u_K$.
⇨ Eigenfaces are eigenvectors of face images: every face can be written as a linear combination of eigenfaces. The first K eigenfaces and their coefficients can be used to determine and reconstruct specific faces: Link, Wikipedia.

9/17

- Hierarchical clustering
  - Items of similar feature belongs to the same group
  - Every point start as its own clustering, drag one point to another point to merge them into one cluster
- Distance between points

Distance between points in m dimensional space is usually measured by Euclidean distance (also called L_2 distance): Wikipedia.
Distances can also be measured by L_1 or L_∞ distances.
⇨ Manhattan distance (L_1): $\|x_i - x_j\|_1 = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \ldots + |x_{im} - x_{jm}|$: Wikipedia.
⇨ Chebyshev distance (L_∞): $\|x_i - x_j\|_\infty = \max\{|x_{i1} - x_{j1}|, |x_{i2} - x_{j2}|, \ldots, |x_{im} - x_{jm}|\}$: Wikipedia

  - Single linkage distance
    - Shortest distance from any item in one cluster to any item in another cluster
  - Complete linkage distance
    - the longest distance from any item in one cluster to any item in the other cluster
  - Average linkage distance
    - the average distance from any item in one cluster to any item in the other cluster (average of distances, not distance between averages)
  - Cluster by distance matrix

$$d = \begin{bmatrix} 0 & 28 & 50 & 39 & 77 \\ 28 & 0 & 95 & 5 & 49 \\ 50 & 95 & 0 & 70 & 6 \\ 39 & 5 & 70 & 0 & 57 \\ 77 & 49 & 6 & 57 & 0 \end{bmatrix}$$

    - Step1 - find the two points closest to each other - point2 and point4
    - Step2 - merge {2, 4} into one cluster, recompute the pairwise distance table
    - Then repeat the algorithm
- Choosing number of clusters

- The number of clusters should be chosen based on prior knowledge about the dataset.
- The algorithm can also stop merging as soon as all the between-cluster distances are larger than some fixed threshold.
- The binary tree generated by hierarchical clustering is often called dendrogram

9/19

- K means clustering
  - Start with a fixed k
  - Start from k random cluster centers
  - Compare distance from each point to the cluster centers, assign cluster based on the shortest distance
  - Relocate the centers to the actual center of the clusters
  - Re-iterate the algorithm until it converges
- Total distortion
  - K means clustering try to minimize total distortion
  - Suppose the cluster centers are $c_1, c_2, \ldots, c_K$, and the cluster center for an item x_i is $c(x_i)$ (one of $c_1, c_2, \ldots, c_K$), then the total distortion is $\|x_1 - c(x_1)\|_2^2 + \|x_2 - c(x_2)\|_2^2 + \ldots + \|x_n - c(x_n)\|_2^2$.
  - Minimizing total distortion is similar to gradient descent
- Number of clusters
  - Based on prior knowledge
  - Cannot be chosen by minimizing total distortion since the total distortion is always minimized at 0 when k=n (number of clusters = number of training items)
  - K can be chosen by minimizing total distortion plus some regularizer, for e.g., c*mKlog(n) where c is a fixed constant and m is the number of features for each item
    - C large if lower k
    - C small if higher k
- Initial clusters
  - The initial cluster centers can be randomly chosen in the domain.
  - The initial cluster centers can be randomly chosen as K distinct items.
  - The first cluster center can be a random item, the second cluster center can be the item that is the farthest from the first item, the third cluster center can be the item that is the farthest from the first two items, …
- K nearest neighbor
  - The K Nearest Neighbor algorithm (not related to K Means) is a simple supervised learning algorithm that uses the items from the training set that is the closest to a new item to predict the label of the new item
  - 1 nearest neighbor copies the label of the closest item.

    - 3 nearest neighbors find the majority label of the three closest items.
    - N nearest neighbor uses the majority label of the training set (of size N) to predict the label of every new item
- Training set accuracy
    - For 1NN, the accuracy of the prediction on the training set is always 100 percent.
    - When comparing the accuracy of KNN for different values of K (called hyperparameter tuning), training set accuracy is not a great measure.
    - K fold cross validation is often used instead to measure the performance of a supervised learning algorithm on the training set.
        - The training set is divided into K groups (K can be different from the K in KNN).
        - Train the model on K - 1 groups and compute the accuracy on the remaining 1 group.
        - Repeat this process K times.
    - K fold cross validation with K=n is called Leave One Out Cross Validation (LOOCV).

9/24

- Decision tree
    - We want to make the tree in a way to minimize the uncertainty
    - We want to maximize information gain
    - Find the feature that is most informative
    - Split the training set into subsets based on this feature
    - Repeat on each of the subset recursively until all features or labels in the subset are the same.
- Uncertainty
    - Let p_0 be the fraction of items in a training with label 0 and p_1 be the fraction of items with label 1.
      ⇨ If $p_0 = 0, p_1 = 1$, the outcome is certain, so there is no uncertainty, the measure of uncertainty should be 0.
      ⇨ If $p_0 = 1, p_1 = 0$, the outcome is certain, so there is no uncertainty, the measure of uncertainty should be 0.
      ⇨ If $p_0 = \frac{1}{2}, p_1 = \frac{1}{2}$, the outcome is the most uncertain, so the measure of uncertainty should be at its maximum value, for example 1.
      One measure of uncertainty that satisfies the above condition is entropy: $H = p_0 \log_2\left(\frac{1}{p_0}\right) + p_1 \log_2\left(\frac{1}{p_1}\right)$ or $H = -p_0 \log_2(p_0) - p_1 \log_2(p_1)$.
    -
        - The entropy formula is the only place we use $\log_2 x$
        - Max at 1
- Entropy
    - In general, if there are K classes ($y = \{1, 2, \ldots, K\}$), and p_y is the fraction of the training set with label y, then the entropy of y is $H(y) = -p_1 \log_2(p_1) - p_2 \log_2(p_2) - \ldots - p_K \log_2(p_K)$.
      Conditional entropy is the entropy of the conditional distribution: $H(y|x) = q_1 H(y|x = 1) + q_2 H(y|x = 2) + \ldots + q_{K_x} H(y|x = K_x)$, where K_x is the number of possible values of the feature x and q_x is the fraction of training data with feature x.
      $H(y|x = k) = -p_{y=1|x=k} \log_2\left(p_{y=1|x=k}\right) - p_{y=2|x=k} \log_2\left(p_{y=2|x=k}\right) - \ldots - p_{y=K|x=k} \log_2\left(p_{y=K|x=k}\right)$, where $p_{y|x}$ is the fraction of training data with label y among the items with feature x.
- Information gain

- The information gain from a feature x is defined as the difference between the entropy of the label and the conditional entropy of the label given that feature: $I(y|x) = H(y) - H(y|x)$.
- The larger the information gain, the larger the reduction in uncertainty, and the better predictor the feature is.
- A decision tree iteratively splits the training set based on the feature with the largest information gain. This algorithm is called ID3 (Iterative Dichotomiser 3): Wikipedia.
  - ⇨ Find feature j so that $I(y|x_{ij})$ is the largest.
  - ⇨ Split the training set into the set with $x_{ij} = 1$, $x_{ij} = 2$, ..., $x_{ij} = K_j$.
  - ⇨ Repeat the process on each of the subsets to create a tree.
- For continuous features, construct all possible splits and find the one that yields the largest information gain: this is the same as creating new variables $z_{ij} = 1$ if $x_{ij} \leq t_j$ and $z_{ij} = 0$ if $x_{ij} > t_j$.
  - ⇨ In practice, the efficient way to create the binary splits uses the midpoint between items with different labels.
- [4 points] "It" has a house with many doors. A random door is about to be opened with equal probability. Doors 1 to 4 have monsters that eat people. Doors 5 to 7 are safe. With sufficient bribe, Pennywise will answer your question "Will door 1 be opened?" What's the information gain (also called mutual information) between Pennywise's answer and your encounter with a monster?

  Answer: Calculate

  feature: x=1 if door 1 is opened, x=0 if one of doors 2 to 7 is opened
  label: y=1 if there is a monster, y=0 if no monster
  pr{y=0} = 3/7, and pr{y=1} = 4/7
  entropy[Y] = H(Y) = -3/7 * log2(3/7) - 4/7*log2(4/7) = 0.9852
  pr{Y=0|x=0} = pr{no monster given doors 2 to 7 are opened} = 1/2 = Pr{Y=1|x=0}
  entropy[Y|x=0] = H(Y|x=0) = -1/2 * log2(1/2) - 1/2*log2(1/2) = 1
  entropy[Y|x=1] = H(Y|x=1) = 0
  pr{x=0} = 6/7 and pr{x=1}=1/7
  entropy[Y|x]= 6/7*H(Y|x=0) + 1/7 * H(Y|x=1) = 6/7
  iinformation gain = IG(Y|x) = H(Y) - H(Y|x) = 0.128
- We assume log0 = 0

- Pruning
  - Decision trees can be pruned by replacing a subtree by a leaf when the accuracy on a validation set with the leaf is equal or higher than the accuracy with the subtree. This method is called Reduced Error Pruning
  - A validation set is a subset of the training set that is set aside when training the decision tree and only used for pruning the tree.
  - The items used to train the decision tree cannot be used to prune the tree.
- Random forest
  - Smaller training sets can be created by sampling from the complete training set, and different decision trees can be trained on these smaller training sets (and only using a subset of the features). This is called bagging (or Bootstrap AGGregatING)
    - Training items are sampled with replacement.
    - Features are sampled without replacement.
  - The label of a new item can be predicted based on the majority vote from the decision trees training on these smaller training sets. These trees form a random forest
- Adaptive boosting
  - Decision trees can also be trained sequentially. The items that are classified incorrectly by the previous trees are made more important when training the next decision tree.

- Each training item has a weight representing how important they are when training each decision tree, and the weights can be updated based on the error made by the previous decision trees. This is called AdaBoost (ADAptive BOOSTing)

9/26

- Linear classifier
  - Linear threshold unit
    - W1xi1 + w2xi2 + … wmxim + b >=0

    Given a training set, the weights $w_1, w_2, \ldots, w_m$ and bias b can be estimated based on the data $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$. One algorithm is called the Perceptron Algorithm.
    ⇨ Initialize random weights and bias.
    ⇨ For each item x_i, compute the prediction $\hat{y}_i$.
    ⇨ If prediction is $\hat{y}_i = 0$ and the actual label is $y_i = 1$, increase the weights by $w \leftarrow w + \alpha x_i, b \leftarrow b + \alpha$, α is a constant called learning rate.
    ⇨ If prediction is $\hat{y}_i = 1$ and the actual label is $y_i = 0$, decrease the weights by $w \leftarrow w - \alpha x_i, b \leftarrow b - \alpha$.
  - ⇨ Repeat the process until convergent (weights are no longer changing).
- Perceptron algorithm
  - Wikipedia.

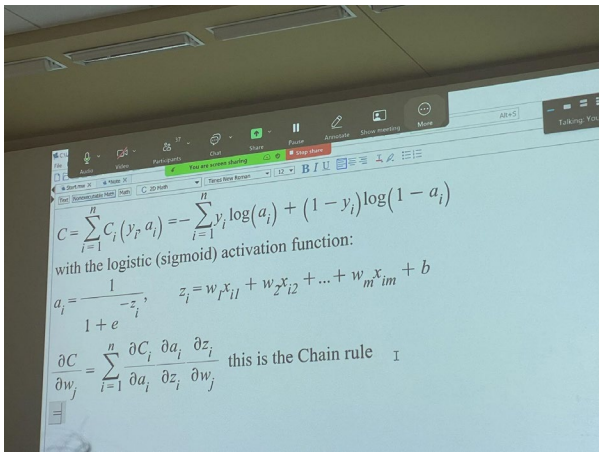
    The percetron algorithm update can be summarized as $w \leftarrow w - \alpha\left(\hat{y}_i - y_i\right)x_i$ (or for $j = 1, 2, \ldots, m, w_j \leftarrow w_j - \alpha\left(\hat{y}_i - y_i\right)x_{ij}$) and $b \leftarrow b - \alpha\left(\hat{y}_i - y_i\right)$, where $\hat{y}_i = 1$ if $w_1x_{i1} + w_2x_{i2} + \ldots + w_mx_{im} + b \geq 0$ and $\hat{y}_i = 0$ if $w_1x_{i1} + w_2x_{i2} + \ldots + w_mx_{im} + b < 0$: Wikipedia.
    The learning rate α controls how fast the weights are updated.
    ⇨ α can be constant (usually 1).
    ⇨ α can be a function of the iteration (usually decreasing), for example, $\alpha_t = \frac{1}{\sqrt{t}}$.
- Activation function
  - Wikipedia.

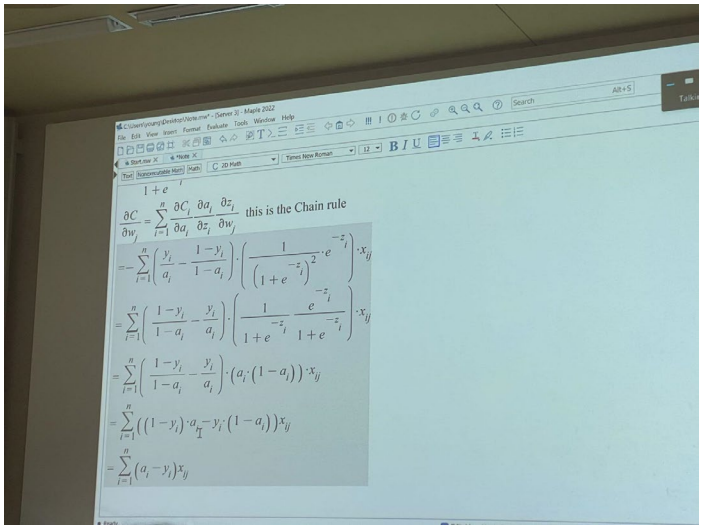
    More generally, a non-linear activation function can be added and the classifier will still be a linear classifier:
    $\hat{y}_i = g\left(w_1x_{i1} + w_2x_{i2} + \ldots + w_mx_{im}\right)$ for some non-linear function g called the activation function: Wikipedia.
    ⇨ LTU: the special case where $g(z) = 1$ if $z \geq 0$ and $g(z) = 0$ otherwise.
    ⇨ Linear regression (not commonly used for classification problems): $g(z) = z$, usually truncated to a number between $[0, 1]$ to represent probability that the predicted label is 1: Wikipedia.
    ⇨ Logistic regression: $g(z) = \frac{1}{1 + e^{-z}}$: Wikipedia.
    There are other activation functions often used in neural networks (multi-layer perceptrons):
    ⇨ ReLU (REctified Linear Unit): $g(z) = \max(0, z)$: Wikipedia.
    ⇨ tanh (Hyperbolic TANgent): $g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$.
- Loss function
  - Wikipedia.

    The weights and biases need to be selected to minimize a loss (or cost) function $C = C\left(\hat{y}_1, y_1\right) + C\left(\hat{y}_2, y_2\right) + \ldots + C\left(\hat{y}_n, y_n\right)$ or $C = C(a_1, y_1) + C(a_2, y_2) + \ldots + C(a_n, y_n)$ (in case $\hat{y}_i$ is a prediction of either 0 or 1 and a_i is a probability prediction in $[0, 1]$), the sum of loss from the prediction of each item: Wikipedia.
    ⇨ Zero-one loss counts the number of mistakes: $C(a_i, y_i) = 1$ if $a_i \neq y_i$ and 0 otherwise.
    ⇨ Square loss: $C(a_i, y_i) = \frac{1}{2}(a_i - y_i)^2$.
    ⇨ Hinge loss: $C(a_i, y_i) = \max\{0, 1 - (2a_i - 1) \cdot (2y_i - 1)\}$: Wikipedia.
  - ⇨ Cross entropy loss: $C(a_i, y_i) = -y_i \log(a_i) - (1 - y_i) \log(1 - a_i)$: Wikipedia.

10/1

- Gradient descent example

- 

- 

- Linear regression
  - The regression coefficients are usually estimated by $w = \left(X^\top X\right)^{-1} X^\top y$, where X is the design matrix whose rows are the items and the columns are the features (a column of 1s can be added so that the corresponding weight is the bias): Wikipedia.
  - Gradient descent can be used with squared loss and the weights should converge to the same estimates:
    $w = w - \alpha\left((a_1 - y_1)x_1 + (a_2 - y_2)x_2 + \ldots + (a_n - y_n)x_n\right)$ and $b = b - \alpha\left((a_1 - y_1) + (a_2 - y_2) + \ldots + (a_n - y_n)\right)$.
    ▼Math Note
    - The coefficients are sometimes derived from $y = Xw$, implying $X^\top y = X^\top X w$ or $w = \left(X^\top X\right)^{-1} X^\top y$.
    - Gradient descent step is given by $C_i = \frac{1}{2}(a_i - y_i)^2$ and $a_i = w_1 x_{i1} + w_2 x_{i2} + \ldots + w_m x_{im}$, so,
      $\frac{\partial C_i}{\partial w_j} = \frac{\partial C_i}{\partial a_i} \frac{\partial a_i}{\partial w_j}$
      $= (a_i - y_i) x_{ij}$,
      combining the w_j for $j = 1, 2, \ldots, m$, $\nabla_w C_i = (a_i - y_i) x_i$,
      and combining the C_i for $i = 1, 2, \ldots, n$, $\nabla_w C = (a_1 - y_1)x_1 + (a_2 - y_2)x_2 + \ldots + (a_n - y_n)x_n$.
  - Do not use linear regression for classify problems
    - Loss function does not make sense
- Model interpretation
  - Given a new item x_k with features $(x_{k1}, x_{k2}, \ldots, x_{km})$, the predicted y_k is given by $\hat{y}_k = w_1 x_{k1} + w_2 x_{k2} + \ldots + w_m x_{km} + b$.
  - The weight (coefficient) for feature j is usually interpreted as the expected (average) change in y when x_{kj} increases by one unit with the other features held constant.
  - The bias (intercept) is usually interpreted as the expected (average) value of y when all features have value 0, or $x_{k1} = x_{k2} = \ldots = x_{km} = 0$.
    ⇨ This interpretation assumes that 0 is a valid value for all features (or 0 is in the range of all features).
- Margin and support vectors
- Multi-class SVM

- Train multiple SVMs
- One vs one - (1/2)K(K - 1) classifiers
- One vs all - K classifiers (treat one as a class and the rest as a class)

10/3

- Feature map
  - If the classes are not separable, more features can be created so that in higher dimensions the items might be linear separable
  - Given a feature map φ, the new items $(\varphi(x_i), y_i)$ for $i = 1, 2, \ldots, n$ can be used to train perceptrons or support vector machines.
  - When applying the resulting classifier on a new item x, $\varphi(x)$ should be used as the features too.
- Kernel trick
  - Using non-linear feature maps for support vector machines (which are linear classifiers) is called the kernel trick since any feature map on a data set can be represented by a $n \times n$ matrix called the kernel matrix (or Gram matrix):
    $K_{ij} = \varphi(x_i)^\top \varphi(x_j) = \varphi_1(x_i)\varphi_1(x_j) + \varphi_2(x_i)\varphi_2(x_j) + \ldots + \varphi_m(x_i)\varphi_m(x_j)$, for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n$.
    ⇨ If $\varphi(x_i) = \begin{bmatrix} x_{i1}^2 \\ \sqrt{2}x_{i1}x_{i2} \\ x_{i2}^2 \end{bmatrix}$, then $K_{ij} = \left(x_i^\top x_j\right)^2$.
    ⇨ If $\varphi(x_i) = \begin{bmatrix} \sqrt{2}x_{i1} \\ x_{i1}^2 \\ \sqrt{2}x_{i1}x_{i2} \\ x_{i2}^2 \\ \sqrt{2}x_{i2} \\ 1 \end{bmatrix}$, then $K_{ij} = \left(x_i^\top x_j + 1\right)^2$.
- Kernel matrix
  - A matrix is a kernel for some feature map φ if and only if it is symmetric positive semi-definite (positive semi-definiteness is equivalent to having non-negative eigenvalues).
  - Some kernel matrices correspond to infinite-dimensional feature maps.
    ⇨ Linear kernel: $K_{ij} = x_i^\top x_j$.
    ⇨ Polynomial kernel: $K_{ij} = \left(x_i^\top x_j + 1\right)^d$.
    ⇨ Radial basis function (Gaussian) kernel: $K_{ij} = e^{-\frac{1}{\sigma^2}(x_i - x_j)^\top (x_i - x_j)}$. In this case, the new features are infinite dimensional (any finite data set is linearly separable), and dual optimization techniques are used to find the weights (subgradient descent for the primal problem cannot be used).
- MLP
  - Multiple layers of logistic regressions
  - Output is non-linear in the original features
- Neuro network
  - A 2 layer network can approximate any continuous function arbitrarily closely with enough hidden units
  - A 3-layer network can approximate any function arbitrarily closely with enough hidden units

10/8

- Gradient descent

- Gradient descent will be used, and the gradient will be computed using chain rule. The algorithm is called backpropogation: Wikipedia.
- For a neural network with one input layer, one hidden layer, and one output layer:
  - ⇨ $\frac{\partial C_i}{\partial w_j^{(2)}} = \frac{\partial C_i}{\partial a_i^{(2)}} \frac{\partial a_i^{(2)}}{\partial w_j^{(2)}}$, for $j = 1, 2, \ldots, m^{(1)}$.
  - ⇨ $\frac{\partial C_i}{\partial b^{(2)}} = \frac{\partial C_i}{\partial a_i^{(2)}} \frac{\partial a_i^{(2)}}{\partial b^{(2)}}$.
  - ⇨ $\frac{\partial C_i}{\partial w_{j'j}^{(1)}} = \frac{\partial C_i}{\partial a_i^{(2)}} \frac{\partial a_i^{(2)}}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial w_{j'j}^{(1)}}$, for $j' = 1, 2, \ldots, m, j = 1, 2, \ldots, m^{(1)}$.
  - ⇨ $\frac{\partial C_i}{\partial b_j^{(1)}} = \frac{\partial C_i}{\partial a_i^{(2)}} \frac{\partial a_i^{(2)}}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial b_j^{(1)}}$, for $j = 1, 2, \ldots, m^{(1)}$.
- More generally, the derivatives can be computed recursively.
  - ⇨ $\frac{\partial C_i}{\partial b_j^{(l)}} = \left(\frac{\partial C_i}{\partial b_1^{(l+1)}} w_{j1}^{(l+1)} + \frac{\partial C_i}{\partial b_2^{(l+1)}} w_{j2}^{(l+1)} + \ldots + \frac{\partial C_i}{\partial b_{m^{(l+1)}}^{(l+1)}} w_{jm^{(l+1)}}^{(l+1)} \right) g' \left(a_{ij}^{(l)} \right)$, where $\frac{\partial C_i}{\partial b^{(L)}} = \frac{\partial C_i}{\partial a_i^{(L)}} g' \left(a_i^{(L)} \right)$.
  - ⇨ $\frac{\partial C_i}{\partial w_{j'j}^{(l)}} = \frac{\partial C_i}{\partial b_j^{(l)}} a_{ij'}^{(l-1)}$.
- Gradient descent formula is the same: $w = w - \alpha \left(\nabla_w C_1 + \nabla_w C_2 + \ldots + \nabla_w C_n \right)$ and $b = b - \alpha \left(\nabla_b C_1 + \nabla_b C_2 + \ldots + \nabla_b C_n \right)$ for all the weights and biases.

Stochastic gradient descent

- The gradient descent algorithm updates the weight using the gradient which is the sum over all items, for logistic regression: $w = w - \alpha \left((a_1 - y_1) x_1 + (a_2 - y_2) x_2 + \ldots + (a_n - y_n) x_n \right)$.
- A variant of the gradient descent algorithm that updates the weight for one item at a time is called stochastic gradient descent. This is because the expected value of $\frac{\partial C_i}{\partial w}$ for a random i is equal to $\frac{\partial C}{\partial w}$: Wikipedia.
  - ⇨ Gradient descent (batch): $w = w - \alpha \frac{\partial C}{\partial w}$ or $w = w - \alpha \left(\frac{\partial C_1}{\partial w} + \frac{\partial C_2}{\partial w} + \ldots + \frac{\partial C_n}{\partial w} \right)$.
  - ⇨ Stochastic gradient descent: for a random $i \in \{1, 2, \ldots, n\}$, $w = w - \alpha \frac{\partial C_i}{\partial w}$.
- Instead of randomly pick one item at a time, the training set is usually shuffled, and the shuffled items will be used to update the weights and biases in order. Looping through all items once is called an epoch.

Softmax layer

- For both logistic regression and neural network, the output layer can have K units, $a_{ik}^{(L)}$, for $k = 1, 2, \ldots, K$, for K-class classification problems: Link,
- The labels should be converted to one-hot encoding, $y_{ik} = 1$ when the true label is k and $y_{ik} = 0$ otherwise.
  - ⇨ If there are $K = 3$ classes, then all items with $y_i = 1$ should be converted to $y_i = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$, and $y_i = 2$ to $y_i = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$, and $y_i = 3$ to $y_i = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$.
- The last layer should normalize the sum of all K units to 1. A popular choice is the softmax operation: $a_{ik}^{(L)} = \frac{e^{-z_{ik}}}{e^{-z_{i1}} + e^{-z_{i2}} + \ldots + e^{-z_{iK}}}$, where $z_{ik} = w_{1k} a_{i1}^{(L-1)} + w_{2k} a_{i2}^{(L-1)} + \ldots w_{m^{(L-1)}k} a_{im^{(L-1)}}^{(L-1)} + b_k$ for $k = 1, 2, \ldots, K$: Wikipedia.

Function approximator

- Neural networks can be used in different areas of machine learning.
  - ⇨ In supervised learning, a neural network approximates $\mathbb{P}\{y|x\}$ as a function of x.
  - ⇨ In unsupervised learning, a neural network can be used to perform non-linear dimensionality reduction. Training a neural network with output $y_i = x_i$ and fewer hidden units than input units will find a lower dimensional representation (the values of the hidden units) of the inputs. This is called an autoencoder: Wikipedia.
  - ⇨ In reinforcement learning, there can be multiple neural networks to store and approximate the value function and the optimal policy (choice of actions): Wikipedia.

10/10

- Generalization error
  - With a large number of hidden layers and units, a neural network can overfit a training set perfectly. This does not imply the performance on new items will be good
  - More data can be created for training using generative models or unsupervised learning techniques

- A validation set can be used to train the network until the loss of the validation set begins to increase
- Dropout
  - randomly omit units (random pruning of weights) during training so the rest of the units will have a better performance

- Regularization
  - A simpler model (with fewer weights, or many weights set to 0) is usually more general and would not overfit the training set as much. A way to achieve that is to include an additional cost for non-zero weights during training.
  - ⇨ L_1 regularization adds L_1 norm of the weights and biases to the loss, or $C = C_1 + C_2 + \ldots + C_n + \lambda \left\| \begin{bmatrix} w \\ b \end{bmatrix} \right\|_1$, for example, if there are no hidden layers, $\left\| \begin{bmatrix} w \\ b \end{bmatrix} \right\|_1 = |w_1| + |w_2| + \ldots + |w_m| + |b|$. Linear regression with L_1 regularization is also called LASSO (Least Absolute Shrinkage and Selector Operator): Wikipedia.
  - ⇨ L_2 regularization adds L_2 norm of the weights and biases to the loss, or $C = C_1 + C_2 + \ldots + C_n + \lambda \left\| \begin{bmatrix} w \\ b \end{bmatrix} \right\|_2^2$, for example, if there are no hidden layers, $\left\| \begin{bmatrix} w \\ b \end{bmatrix} \right\|_2^2 = (w_1)^2 + (w_2)^2 + \ldots + (w_m)^2 + (b)^2$. Linear regression with L_2 regularization is also called ridge regression:
  - ⇨ λ is chosen as the trade-off between the loss from incorrect prediction and the loss from non-zero weights.
  - ⇨ L_1 regularization often leads to more weights that are exactly 0, which is useful for feature selection.
  - ⇨ L_2 regularization is easier for gradient descent since it is differentiable.