Summary of CPEN 355

Tom Wang

Fall, 2023

1 ML Basics

1.1 How to learn from data?

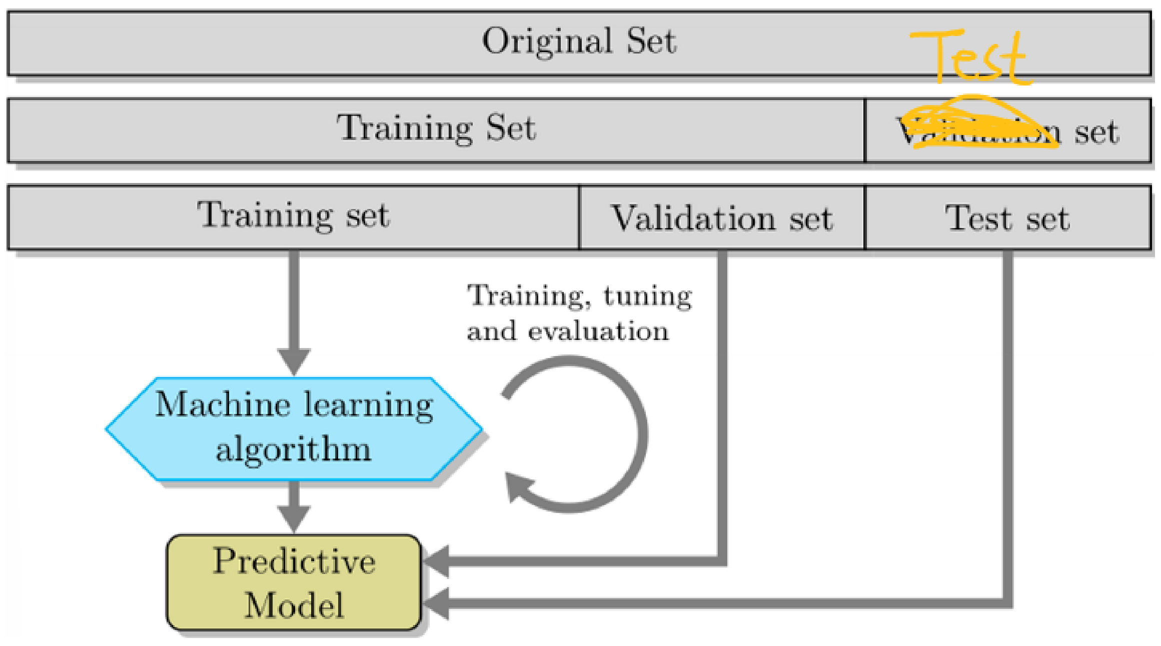
- we use some learning algorithm to train the machine how to predict.
- Construct a model (corresponding to a certain learning algorithm)
- Learn the model with observed samples in data sets.

1.2 Data split

The model is trained on training data set and process test data set to make predictions. These two sets are independent.

A larger test set gives a more accurate assessment of performance.

We sometimes also use a validation data set, which is a set of examples used to tune the hyperparameters (if any) of the model.



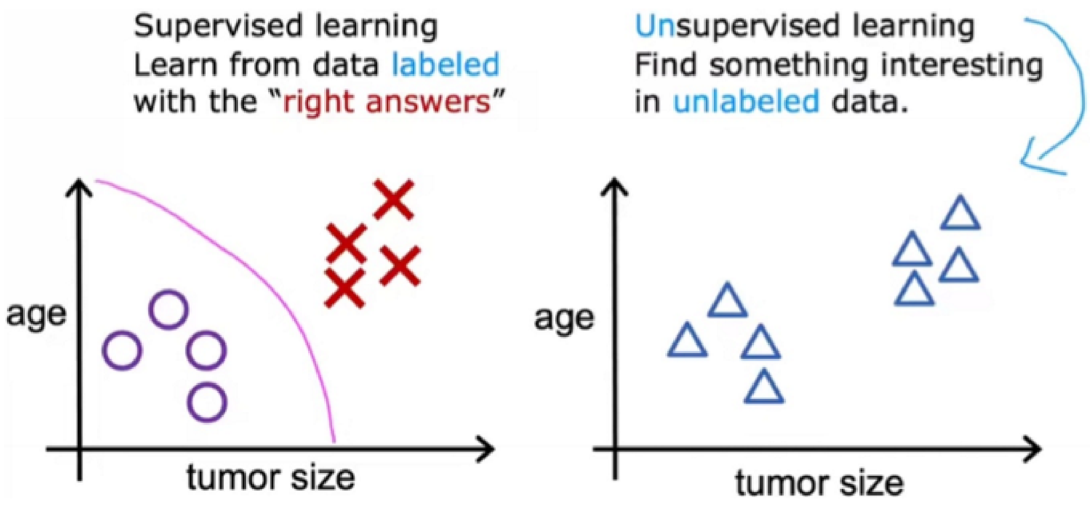
1.3 Supervised vs. Unsupervised vs. Reinforced Learing

- Supervised:
  - Given a set of (x,y), learn to predict y using x.
  - Trained on a labeled dataset, which means that each input to the algorithm is associated with the corresponding target or output.
  - The goal of supervised learning is to learn a mapping from variables to the corresponding output variables. The algorithm generalizes from the labeled training data to make predictions or classifications on new, unseen data.
  - Classification and regression problems fall under supervised learning. In classification, the algorithm predicts the category or class of input, while in regression, it predicts a continuous value.
- Unsupervised:
  - Given a set of x, the underlying structure of relationships of x.
  - The algorithm explores the data's inherent structure without explicit guidance on the output. (no labeled output for training)
  - The goal of Unsupervised learning is to find patterns, relationships, or structures in the data. It seeks to uncover the underlying distribution or representations within the input data.
  - Clustering and dimensionality reduction are common tasks in unsupervised learning. Clustering involves grouping similar data points, while dimensionality reduction aims to reduce the number of features while preserving important information.

- Reinforced:
  - Agent interacts with an environment and learns to make decisions by receiving feedback in the form of rewards or penalties.
  - The goal is to find a policy (a strategy or decision-making process) that maximizes the cumulative reward over time. The agent learns by exploring different actions and overcoming the consequences of those actions.
  - Game playing, robotic control, and autonomous systems are common applications of reinforcement learning. The agent learns to take actions that lead to desirable outcomes and avoid actions that lead to negative outcomes.

1.4 Classification vs Regression

- Classification: prediction of which class the sample belongs To
- Regression: Model the exact output given a sample.



2 Linear Regression

2.1 Basics

Machine learning usually has linear regression over manydimensions.

$$\hat{Y} = f_\theta(\mathsf{X}) = \mathsf{X}\Theta$$

where $\hat{\mathsf{Y}} = (\hat{y}^{(1)}, \ldots, \hat{y}^{(m)})^T, \quad \Theta = (\theta_0, \theta_1, \ldots, \theta_n)^T$, and

$$\mathsf{X} = \begin{pmatrix} X^{(1)^T} \\ \vdots \\ x^{(m)^T} \end{pmatrix} = \begin{pmatrix} 1 & x_1^{(1)} & \cdots & x_n^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n^{(m)} & \cdots & x_n^{(m)} \end{pmatrix}$$

Here we denote n variables $x_1, x_2, \ldots, x_n$ to represent n features

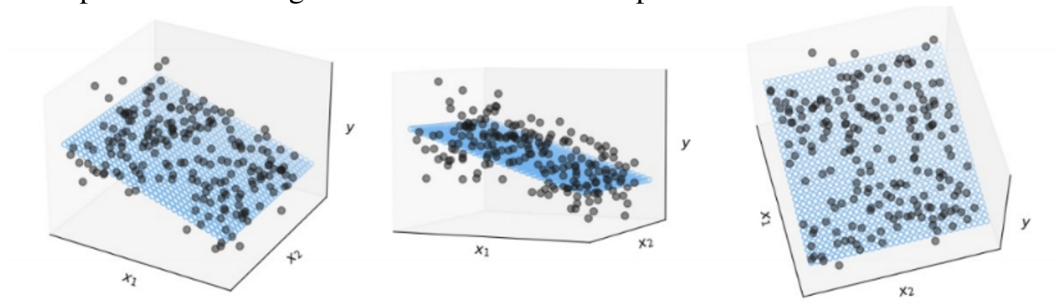
$$\hat{y} = f_{(\theta_0, \theta_1, \ldots, \theta_n)}(1, x_1, x_2, \ldots, x_n) = \theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n$$

$x_0 = 1$ to count for the constant θ_0.

2.2 Geometry

Data points $\{(x_1^{(1)}, \ldots, x_n^{(1)}, y^{(1)}), \ldots, (x_1^{(m)}, \ldots, x_n^{(m)}, y^{(m)})\}$ form a (n+1) - dimensional space.

Examples for linear regression on 2-dim feature space:



2.3 Optimizing fit

The best fit has the smallest distance between Y and $\hat{Y} = f_\theta(\mathsf{X})$

We denote $J_\theta(Y, \hat{Y})$ to be the objective function (a.k.a cost or loss function) to measure the distance between them.

Residual Sum of Squares (RSS):

$$J_\theta(Y, \hat{Y}) = \sum_{i=1}^{m} (f_\theta(X^i) - y^{(i)})^2 = ||\hat{Y} - Y||_2^2$$

It is a sum of the square of the difference between the data point and the linear regression model.

To minimize the convex objective function, find the θ that minimizes the loss function $J(\theta)$. It means:

$$J^{'}(\theta) = 0, J^{''}(\theta) > 0$$

2.4 Some vector and matrix calculus

To simplify a given calculation, it is often useful to write out the explicit formula for a single scalar element of the output in terms of nothing but scalar variables.

Example. Suppose we have a column vector $\vec{y}$ of length C that is calculated by forming the product of a matrix W that is C rows by D columns with a column vector $\vec{x}$ of length D:

$$\vec{y} = W\vec{x}$$

Now, with some simple matrix operation, we can easily calculate the partial derivatives of y against x. So first write out each one:

$$\vec{y}_i = \sum_{j=1}^{D} W_{i,j}\vec{x}_j$$

Write it out without the summation sign, we can overserve some tricks:

$$\vec{y}_i = W_{i,1}\vec{x}_1 + W_{i,2}\vec{x}_2 + W_{i,3}\vec{x}_3 + \cdots + W_{i,D}\vec{x}_D$$

Since we only need the term that we are differentiating for, the rest can be considered as constants. The derivative of a constant is zero!

$$\frac{\partial \vec{y}_i}{\partial \vec{x}_j} = W_{i,j}$$

Now we can get a formula of the Jacobian matrix of the desired C x D derivative matrix:

$$\begin{bmatrix} \frac{\partial \vec{y}_1}{\partial \vec{x}_1} & \frac{\partial \vec{y}_1}{\partial \vec{x}_2} & \cdots & \frac{\partial \vec{y}_1}{\partial \vec{x}_D} \\ \frac{\partial \vec{y}_2}{\partial \vec{x}_1} & \frac{\partial \vec{y}_2}{\partial \vec{x}_2} & \cdots & \frac{\partial \vec{y}_2}{\partial \vec{x}_D} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \vec{y}_C}{\partial \vec{x}_1} & \frac{\partial \vec{y}_C}{\partial \vec{x}_2} & \cdots & \frac{\partial \vec{y}_C}{\partial \vec{x}_D} \end{bmatrix} = \begin{bmatrix} W_{1,1} & W_{1,2} & \cdots & W_{1,D} \\ W_{2,1} & W_{2,2} & \cdots & W_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ W_{C,1} & W_{C,2} & \cdots & W_{C,D} \end{bmatrix}$$

So we can conclude that:

$$For \quad \vec{y} = W\vec{x}, \quad we \quad have \quad \frac{d\vec{y}}{d\vec{x}} = W$$

In the cases where x is a row vector:

$$\vec{y} = \vec{x}W \quad \Longrightarrow \frac{\partial \vec{y}_i}{\partial \vec{x}_j} = W_{j,i} \Longrightarrow \frac{d\vec{y}}{d\vec{x}} = W$$

Notice that W is the opposite of the previous example. But the idea is the same, Jacobian matrix is similar.

Now we expand to a higher dimension. Take three dimensions as an example We should observe that for:

$$\frac{d\vec{y}}{dW}, \text{Where W is a matrix}$$

Only the partial of $\vec{y}_j$ against the jth column of W will not be zero. The others would be zero. So we can write:

$$\frac{\partial \vec{y}_j}{\partial W_{i,j}} = \vec{x}_i$$

Let F represent the 3d array representing the derivative of $\vec{y}$ against W:

$$F_{i,j,k} = \frac{\partial \vec{y}_i}{\partial W_{j,k}}, \text{ then } F_{i,j,i} = \vec{x}_j, \text{ and all other entries are zero.}$$

So now we can define a new *two-dimensional* array G as:

$$G_{i,j} = F_{i,j,i}$$

Now onto multiple samples:

For example, let X be a two-dimensional array with N rows and D columns, and W the same as previous, D rows and C columns.

$$Y = XW \implies Y_{i,j} = \sum_{k=1}^{D} X_{i,k} W_{k,j} \text{ , and the partial is:} \frac{\partial Y_{a,b}}{\partial X_{c,d}}$$

Same idea as before: nonzero exists only when a = c. So:

$$\frac{\partial Y_{i,j}}{\partial X_{i,k}} = W_{k,j}$$

We have now generalized the matrix derivative.

The last topic is the chain rule:

$$\vec{y} = VW\vec{x}, \text{ define } \vec{m} = W\vec{x}, \text{ then } \vec{y} = V\vec{m}. \qquad \implies \frac{d\vec{y}}{d\vec{x}} = \frac{d\vec{y}}{d\vec{m}}\frac{d\vec{m}}{d\vec{x}}$$

In machine learning, the loss function that is generally optimized is a scalar function such as $x \to y \to$ dependencies of z. So we can write:

$$\frac{\partial z}{\partial x} = (\frac{\partial y}{\partial x})^T \frac{\partial z}{\partial y}$$

2.5 ChatGPT teaching time

Step 1: Prepare the Data

You should have a set of data points in the form of pairs $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where x_i is the independent variable, and y_i is the dependent variable.

Step 2: Create the Design Matrix X

The design matrix X is a matrix of size $n \times (d + 1)$, where n is the number of data points and d is the degree of the polynomial you want to fit (e.g., $d = 2$ for quadratic regression).

For quadratic regression ($d = 2$), X looks like this:

$$X = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{bmatrix}$$

Each row corresponds to a data point, and each column represents a term in the polynomial equation. The first column contains 1s for the intercept term, the second column contains the x values, and the third column contains the x^2 values.

Step 3: Create the Coefficient Vector b

The coefficient vector b contains the coefficients of the polynomial terms. For quadratic regression ($d = 2$), b is a vector of size $d + 1 = 3$:

$$b = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$

Step 4: Create the Observation Vector Y

The observation vector Y contains the dependent variable values for each data point. It's a vector of size n:

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Step 5: Solve for b Using Matrix Operations

The goal is to find the coefficients b that minimize the error in the polynomial regression equation. This can be done by solving the linear system of equations:

$$Xb = Y$$

This is a system of n equations (one for each data point) and $d+1$ unknowns (the coefficients $b_0, b_1, \ldots, b_d$).

To solve for b, you can use matrix operations. Specifically, you can use the equation:

$$b = (X^T X)^{-1} X^T Y$$

Here's what each part of the equation does:

- X^T is the transpose of matrix X. It converts the $n \times (d + 1)$ matrix into a $(d + 1) \times n$ matrix. - $X^T X$ is the matrix multiplication of the transposed matrix X^T and X, resulting in a $(d + 1) \times (d + 1)$ matrix. - $(X^T X)^{-1}$ is the inverse of the $(d + 1) \times (d + 1)$ matrix. - $X^T Y$ is the matrix multiplication of the transposed matrix X^T and vector Y, resulting in a $(d + 1) \times 1$ vector. - b is the coefficient vector you want to find.

Once you calculate b, it contains the coefficients of the polynomial equation that best fits your data.

Step 6: Use the Polynomial Equation

With the coefficients b obtained, you can construct the polynomial equation:

$$y = b_0 + b_1 x + b_2 x^2$$

This is the equation of the polynomial regression model.

This explanation covers quadratic regression ($d = 2$), but you can extend the same principles to higher-degree polynomial regressions by increasing the number of columns in the design matrix X and the size of the coefficient vector b.

2.6 Metrics and prediction errors

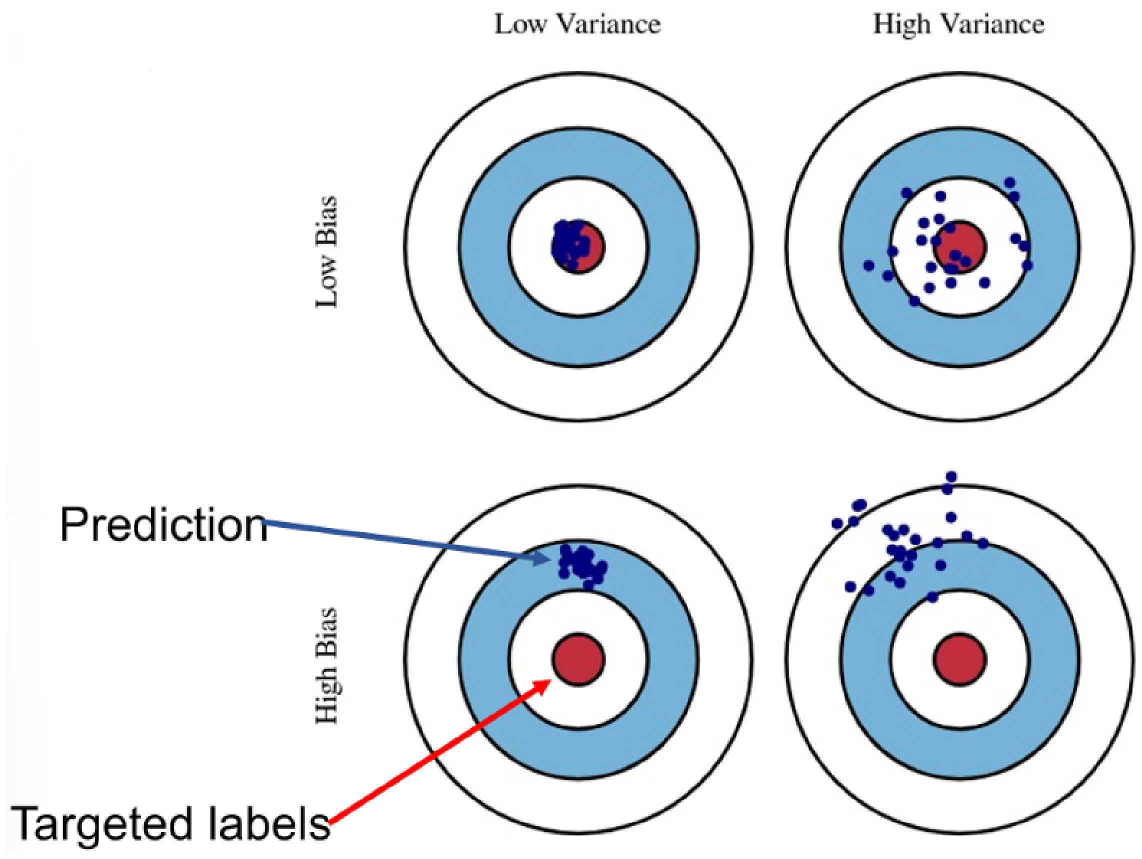
Denote total samples as N, true label as Y, prediction as $\hat{Y}$, the average of testing set as $\bar{Y}$

- Mean Absolute Error (MAE): MAE = $\frac{1}{N}\sum_n^N |Y_n - \hat{Y}_n|$
- Mean Square Error (MSE): MSE = $\frac{1}{N}\sum_n^N (Y_n - \hat{Y}_n)^2$
- Root Mean Square (RMSE): RMSE = $\sqrt{MSE}$
- R-square: $R^2 = 1 - \frac{\sum(Y-\hat{Y})^2}{\sum(Y-\bar{Y})^2}$

The higher the R-square, the more testing data points are from the prediction regression line.

3 Regression with Shrinkage and Logistic Regression

3.1 Bias and Variance



Bias = $\theta - \mathbb{E}\hat{\theta}$ and Variance = $\mathbb{E}(\hat{\theta} - \mathbb{E}\hat{\theta})^2$

- $\hat{\theta}$ is an estimation from the sample

- $\mathbb{E}$ is the expectation w.r.t.samples (i.e., avg estimator)
- We can only directly compute $\hat{\theta}$ from the observed samples. We do not know θ

Def: The expectation $\mathbb{E}$ is used to find the average or expected behavior of random variables.

3.2 Expected prediction error (Risk)

- The relation of input and output is modeled by the function f.
- Due to the noise from observation, y = $f(X) + \epsilon$, where $\mathbb{E}(\epsilon) = 0$ and Var$(\epsilon) = \epsilon^2$
- For any fixed input X and its label y, the expected prediction error (EPE) on X is:

$$EPE(X) = \mathbb{E}[(y - \hat{f}(X))^2] = Bias(\hat{f}(X))^2 + Var(\hat{f}(X)) + \epsilon^2$$

where

$$Bias(\hat{f}(X)) = f(X) - \mathbb{E}[\hat{f}(X)]$$

$$Var(\hat{f}(X)) = \mathbb{E}[(\hat{f}(X) - \mathbb{E}[\hat{f}(X)]^2)]$$

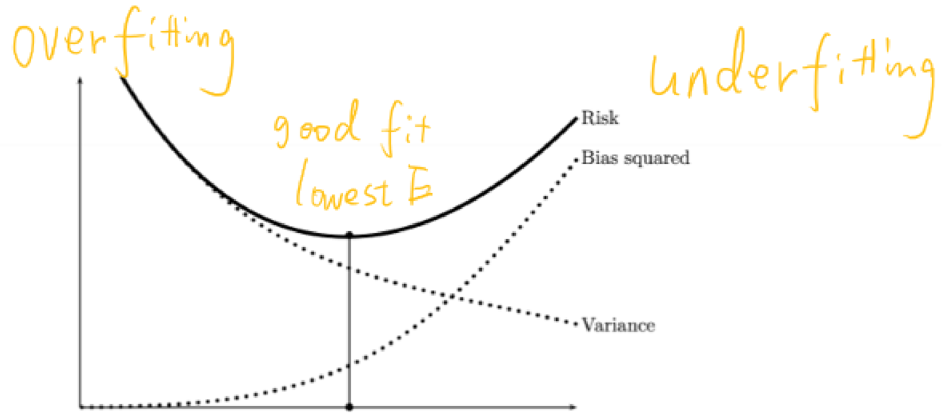
Becuase Risk = Bias^2+Variance:

$$\mathbb{E}[(y - \hat{f}(X))^2] = Bias(\hat{f}(X))^2 + Var(\hat{f}(X)) = (y - \mathbb{E}\hat{f}(X))$$

In general, low variance will cause high bias and vice versa. So we have to make trade-offs. For example: if we repeatedly train the sample data for too long.

- It may perfectly fit the training data. So low bias and high variance since the model will vary significantly among different training data.
- If the model is constant among different training data, the prediction variance is zero but definitely the prediction bias is very high.

3.3 Bias-variance trade-off



The complex model will lead to high bias, hence **Overfitting**
A simple model will lead to high variance, hence **Underfitting**
A good fit will have the lowest EPEX

3.4 Shrinkage methods

Ordinary least squares are prone to learn a low bias high variance model.

Tunning model complexity is a way to sacrifice some bias to reduce the variance to achieve lower EPE.

Coding examples of the methods used can be found here.

- Ridge regression adds a regularization term to the linear regression cost function to address the problem of multicollinearity and overfitting.

  Recall the linear function $\hat{Y} = \mathsf{X}\Theta + \theta_0$ Where we sperate the constant term from X and Θ.

  Now the ordinary least squares can be expressed as:

  $$\hat{\Theta} = \text{argmin}_{\Theta}||\mathsf{X}\Theta + \theta_0 - Y||_2^2$$

  Here argmin is the symbol to indicate the input that could minimize the smallest out, which is to minimize the L2 norm.

  Ridge regression aims to shrink the parameter by imposing a penalty on the square of the magnitude of the coefficients:

  $$\hat{\Theta}_{rigid} = \text{argmin}_{\Theta}||\mathsf{X}\Theta + \theta_0 - Y||_2^2 + \lambda||\Theta||_2^2$$

  The penalty term is $||\Theta||_2^2 = \sum_{i=1}^{n} = \theta_i^2$ (a.k.a. L2 Norm)

  Note that the bias term θ_0 is not included in the penalty term. λ is a hand-crafted hyper-parameter.

  Notes on the penalty term

  - When there are many correlated features in a linear regression model, their coefficients can become poorly determined and exhibit high variance.
  - A wildly large positive coefficient on one variable can be canceled by a similarly large negative coefficient on its correlated cousin.
  - The penalty term regularizes the coefficients such that if the coefficients take large values, the optimization function is penalized. Therefore, the above problem is alleviated.

- Lasso regression: Linear regression technique that combines the principles of linear regression with L1 regularization.

  $$\hat{\Theta}_{rigid} = argmin_{\Theta}||\text{X}\Theta + \theta_0 - Y||_2^2 + \lambda||\Theta||_1$$

  The penalty term is $\lambda||\Theta||_1 = \sum_{i=1}^{n}|\theta_i|$(a.k.a. L1 Norm), which is designed for measuring the sparsity of parameters.

  Sparsity implies the number of zeros in Θ. If many elements in it are around zero, fewer features will be used, namely, the model complexity is reduced.

- In comparison:

  Ridge regression's penalty term regularizes the coefficients such that the coefficients won't be large.

  Lasso regression's penalty term regularizes the coefficients such that coefficients are sparse (containing many zeros elements).

4 Classification

Classification is a process related to categorization, in which ideas and objects are recognized, differentiated and understood.

Binary classification:

- True or false about a statement
- For example: is it a cat? (0 for no and 1 for yes)

Multi-class classificaiton:

- More than two categories, usually use 0, 1, 2, 3, . . . to label categories
- For example, which animal is in the photo? Cat/dog/pig/duck/. . . .

We use **decision boundary** to partition the entire feature space into multiple parts. One for each category.

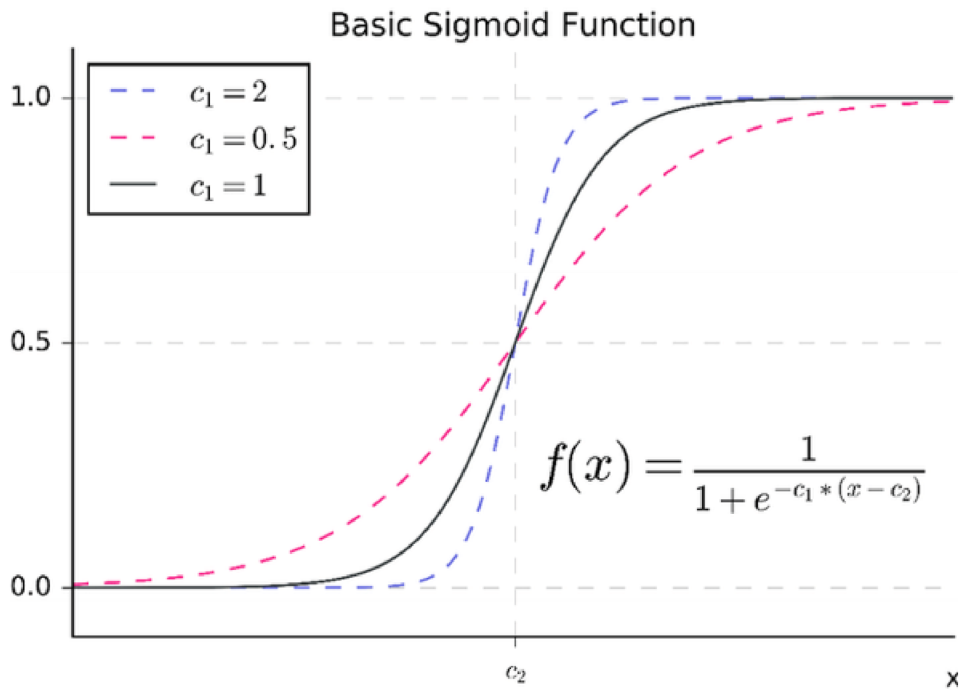
A linear classifier is a linear model we use to classify data based on a linear combination of input features. We usually learn some function to divide.

4.1 Logistic regression for classificaiton

Logistic regression models the probabilities for classification problems for an even occurring.

The cut-off rule is a transformation that maps linear functions' range $(-\infty, +\infty)$ to the range of probability (0,1). In the equation, it refers to z and p.

Then we see a dramatic rise in the boundary value (which is zero) as shown below:



logit:

$$z = \text{logit}(p) = \ln \frac{p}{1-p}$$

$$-z = \ln \frac{1-p}{p} \implies e^{-z} = \frac{1}{p} - 1$$

sigmoid:

$$\text{logit}^{-1}(z) = p = \frac{1}{1+e^{-z}}$$

The following is a probability view of logit:

- Event A and its probability to happen: $P(A) = p$, $P(A^C) = 1 - p$
- The odds of an event is the ratio of the probability of an event to the probability of its complement. So we can express the following:

$$\text{odds}(p) = \frac{p}{1-p}$$

- The range of odds is (0,+∞). Logit is defined as log-odds with range $(-\infty, +\infty)$.
- Recall the linear regression function:

$$f_\Theta = X^T\Theta \in (-\infty, +\infty)$$

- Recall our goal is to find $(P(y = 1|X))$. We should have

$$P(\hat{y} = 1|X) = \text{logit}^{-1}(X^T\Theta) = \frac{1}{1+e^{-(X^T\Theta)}} \in (0, 1)$$

- The equation above is the basic formulation of logistic regression.
- Classification rule:

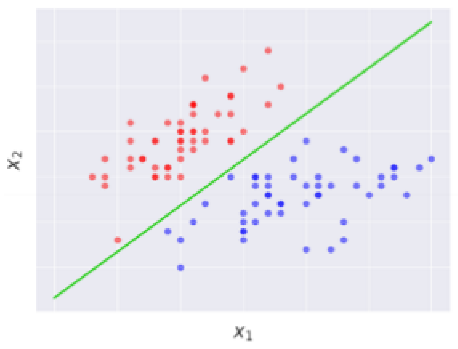
$$P(\hat{y}=1|X) \geq 0.5 \implies \hat{y}=1, \text{ otherwise } \hat{y}=0$$

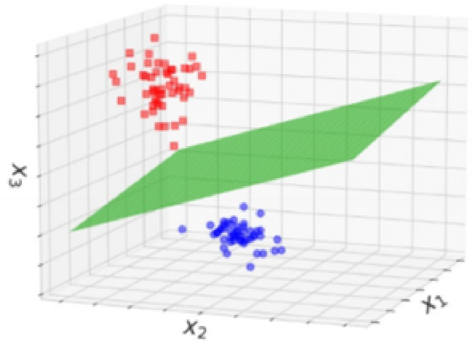
- Like linear regression, Θ is the learnable parameter.
- LR is more robust to outliers

Sometimes, we denote $\text{logit}^{-1}()$ as $\text{sigmoid}()$ or $\sigma()$.
As mentioned, it rises dramatically to 0. Also we know $\sigma(X^T\Theta) \geq 0.5 \iff X^T\Theta \geq 0$.

$$X^T\Theta \geq 0 \implies \hat{y}=1$$
$$X^T\Theta \leq 0 \implies \hat{y}=0$$

The decision boundary is the hyperplane $z = X^T\Theta$





4.2 Supervision criteria

For each sample $X^{(i)} = (x_1^{(i)}, \ldots, x_n^{(i)})$ we have:

$$P(\hat{y}^{(i)}=1|X^{(i)}) = \sigma(X^{(i)T}\Theta) = \frac{1}{1+e^{-(X^{(i)T}\Theta)}} \in (0,1)$$

which is a function with respect to Θ.

- if $y^{(i)}=1$, we expect $P(\hat{y}^i=1|X^{(i)})$ approaches 1 as close as possible.
- if $y^{(i)}=0$, we expect $P(\hat{y}^i=0|X^{(i)}) = 1-P(\hat{y}^{(i)}=1|X^{(i)})$ approaches 0 as close as possible.

To generalize, we can write the probability that the label is correctly predicted as

$$p_i \equiv P(\hat{y}^{(i)}=1|X^{(i)})^{y(i)}(1-P(\hat{y}^{(i)}=1|X^{(i)}))^{1-y(i)}$$

Namely:

- When $y^{(i)} = 0 \quad \Longrightarrow \quad P(\hat{y}^i = 0|X^{(i)})$
- When $y^{(i)} = 1 \quad \Longrightarrow \quad P(\hat{y}^i = 1|X^{(i)})$
- So no matter which category the sample belongs to, p_i can represent the probability that the prediction $\hat{y}^{(i)}$ matches the true label $y^{(i)}$

Next, we consider the joint probability for all the training samples, and maximize it to learn Θ.

Since the samples are independent of each other, the joint probability can be written as:

$$\begin{aligned} p(\Theta) &= P\left(\hat{y}^{(1)} = y^{(1)}, \dots, \hat{y}^{(m)} = y^{(m)} \middle| X^{(1)}, \dots, X^{(m)}\right) \\ &= P\left(\hat{y}^{(1)} = y^{(1)} \middle| X^{(1)}\right) \cdots P\left(\hat{y}^{(m)} = y^{(m)} \middle| X^{(m)}\right) \\ &= \prod_{i=1}^{m} P\left(\hat{y}^{(i)} = 1 \middle| X^{(i)}\right)^{y^{(i)}} \left(1 - P\left(\hat{y}^{(i)} = 1 \middle| X^{(i)}\right)\right)^{1-y^{(i)}} \\ &= \prod_{i=1}^{m} p_i \end{aligned}$$

We want to make $p(\Theta)$ as large as possible. Maximizing a likelihood function is called *maximum likelihood estimation (MLE).*

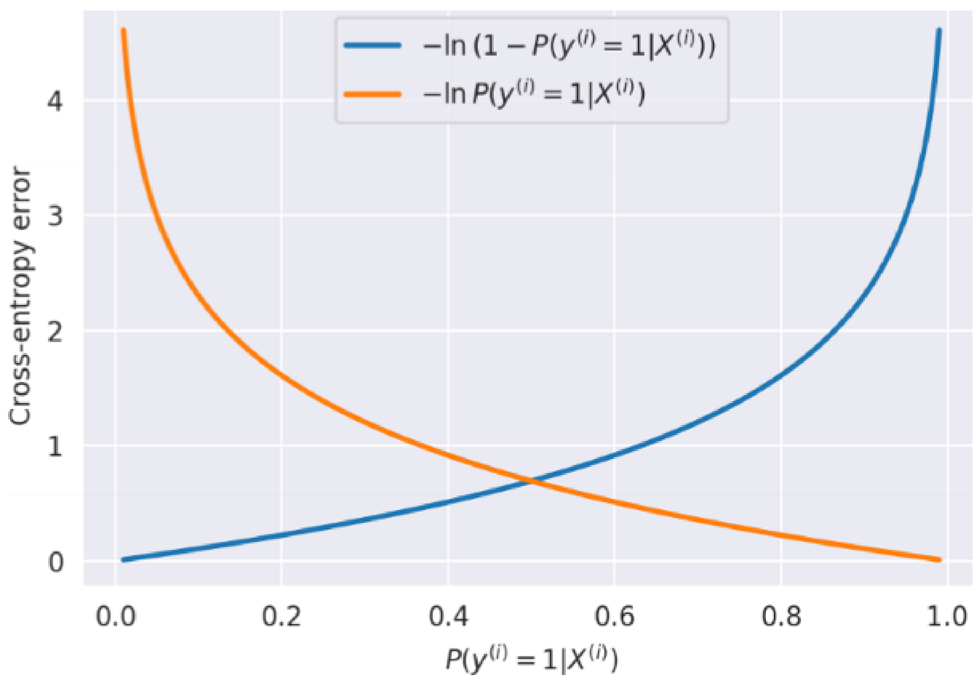
4.3 Cross-entropy

Binary cross-entropy error $\equiv -y\text{ln}P(y) - (1-y)\text{ln}(1-P(y)), y \in \{0,1\}$ It is a concept from information theory. $P(y)$ is the probability density function of taking value y. In our case:

$$P(y) = P(\hat{y}^{(i)} = 1|X^{(i)}) = \frac{1}{1 + e^{-X^{(i)T}\Theta}}$$

- when y = 1, error = -ln$(\frac{1}{1+e^{-\Theta X^{(i)T}}})$
- when y = 0, error = -ln$(1 - \frac{1}{1+e^{-\Theta X^{(i)T}}})$

Cross-entropy can be plotted as follows. To minimize it, we find the interception.



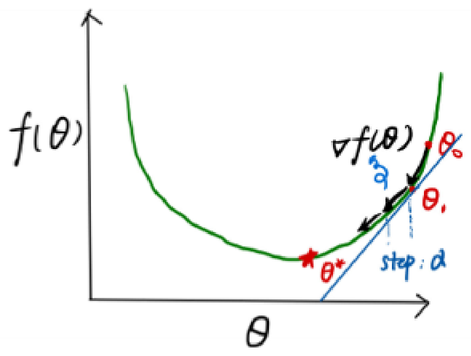
Similar to RSS, we could minimize cross-entropy through $\frac{\partial \mathbb{E}(\Theta)}{\partial \Theta}|\Theta^* = 0$ However, it will be too difficult. So we try **gradient descent**.

For a twice differentiable function (so that a second derivative exists) $f : \mathbb{R}^n \to \mathbb{R}$, the gradient $\nabla f(\Theta)$ points in the direction of the steepest ascent or descent at Θ.

$$\nabla f(\Theta) = \frac{\partial f(\Theta)}{\partial \Theta} = (\frac{\partial f(\Theta)}{\partial \theta_1}, \dots, \frac{\partial f(\Theta)}{\partial \theta_n}) \text{ , where } \Theta = \{\theta_1, \dots, \theta_n\}$$

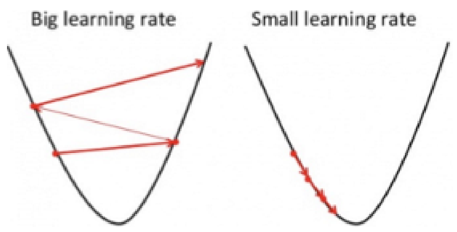
Gradient descent is designed to update parameters in the **opposite** the direction of the gradient. Through iterative updates, an optimal solution for our goal of $\min f(\Theta)$ can be attained. The simple algorithm can be illustrated below:

```
Gradient descent
Ensure: $\alpha > 0$
   Initialize $\Theta \leftarrow \Theta_0$ randomly
   while not converge do
       $\Theta \leftarrow \Theta - \alpha \nabla f(\Theta)$
   end while
```



- α is a small enough hyper-parameter called learning rate.
- Too small will lead to a very slow learning process.
- Too large leads to bad convergence. (Might diverge)

- You might be able to change it at every iteration



4.4 ChatGPT example of gradient descent

The linear model is defined as:

$$f_\Theta(x_1, x_2) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

The Mean Squared Error (MSE) loss is given by:

$$\mathcal{L}(\Theta) = \frac{1}{2N} \sum_{i=1}^{N} (f_\Theta(x_1^{(i)}, x_2^{(i)}) - y^{(i)})^2$$

The gradient descent update rule for the parameters is:

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} \mathcal{L}(\Theta)$$

Partial Derivatives:

$$\frac{\partial}{\partial \theta_0} \mathcal{L}(\Theta) = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}^{(i)} - y^{(i)})$$
$$\frac{\partial}{\partial \theta_1} \mathcal{L}(\Theta) = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}^{(i)} - y^{(i)}) \cdot x_1^{(i)}$$
$$\frac{\partial}{\partial \theta_2} \mathcal{L}(\Theta) = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}^{(i)} - y^{(i)}) \cdot x_2^{(i)}$$

Finally, the updated equations for each parameter are:

$$\theta_0 \leftarrow \theta_0 - \alpha \frac{\partial}{\partial \theta_0} \mathcal{L}(\Theta)$$
$$\theta_1 \leftarrow \theta_1 - \alpha \frac{\partial}{\partial \theta_1} \mathcal{L}(\Theta)$$
$$\theta_2 \leftarrow \theta_2 - \alpha \frac{\partial}{\partial \theta_2} \mathcal{L}(\Theta)$$

For complex and nonconvex functions, gradient descent usually falls into a local minimum instead of the global minimum.

5 Model Evaluation and Trainning

5.1 Multi-class Classification

Extend the previous binary regression methods to more than 2 classes.

Multinomial logistic regression extends the logistic regression model to K$\geq$ 2 using *one-versus-all*

$$\ln\left(\frac{P(Y=k|X)}{P(Y=0|X)}\right) = X^T\Theta_k, \qquad k = 1,2,\dots,K-1$$

$$P(Y=k|X) = \begin{cases} \dfrac{e^{X^T\Theta_k}}{1+\sum_{l=1}^{K-1} e^{X^T\Theta_l}}, k = 1,2,\dots,K-1 \\ \dfrac{1}{1+\sum_{l=1}^{K-1} e^{X^T\Theta_l}}, k = 0 \end{cases}$$

Each Θ_k represents a line that separates one type from the rest. So we need a total of k-1 Θ since the last one is redundant. k=0 means it does not belong to anything. So Θ_0 is all zeros by definition.

5.2 Discriminative and Generative Clssifier

- Generative models model both the input X and the output Y. ($P(y = k, X; \Theta)$) Bayes classifier is a good example. The theorem of minimizing risk R(h) is

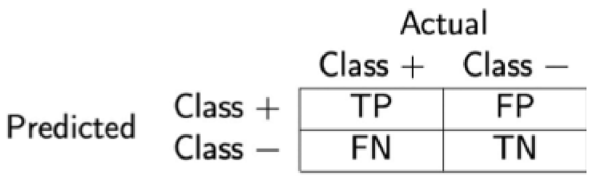
$$h^* = (P(Y=1|X) > 0.5) \quad 1:0$$

- Discriminative models model only the output Y given X. ($P(y = k|X; \Theta)$) Logistic regression is a discriminative model because we do not have a model for the input X. We only model the conditional probability $P(Y|X)$

5.3 Mertics for Classification

Metrics to evaluate model performance, we count the number of occurrences for the following.

- TP is "true positive": predict correctly as positive
- FP is "false positive": predict incorrectly as positive
- TN is "true negative": predict correctly as negative
- FN is "false negative": predict incorrectly as negative



		Actual	
		Class +	Class −
Predicted	Class +	TP	FP
	Class −	FN	TN

Accuracy = frcation of correct predictions = $\frac{TP+TN}{TP+FP+TN+FN}$
Precision = accuracy of the positive predictions = $\frac{TP}{TP+FP}$
Recall (a.k.a. sensitivity) = accuracy of positive class = $\frac{TP}{TP+FN}$

F1 score is calculated from precision and recall and is the **harmonic mean** of them. The highest possible value is 1.0

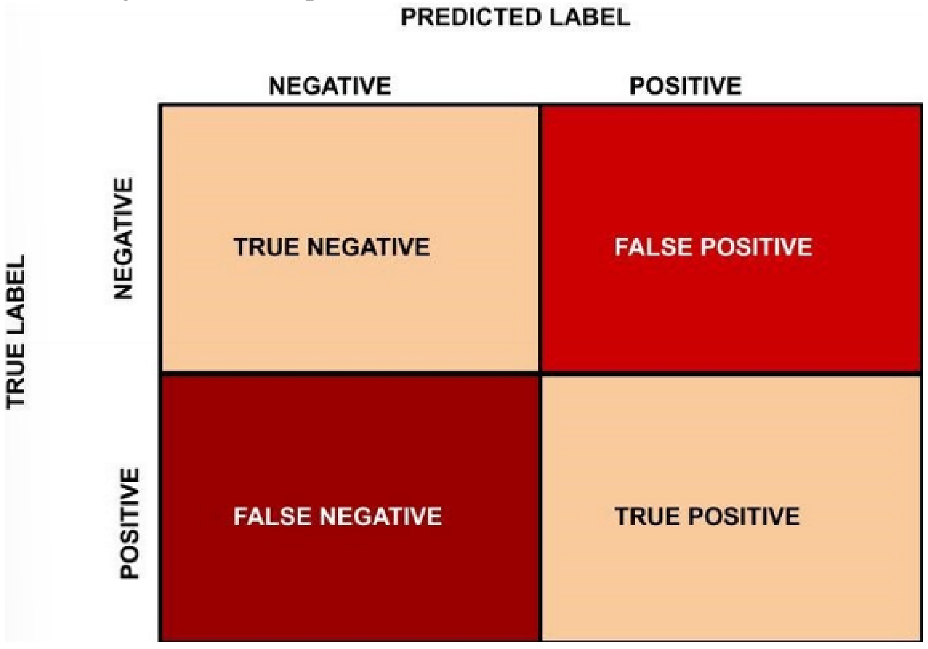
$$F_1 = 2\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + 0.5(FP + FN)}$$

5.4 Confusion matrix

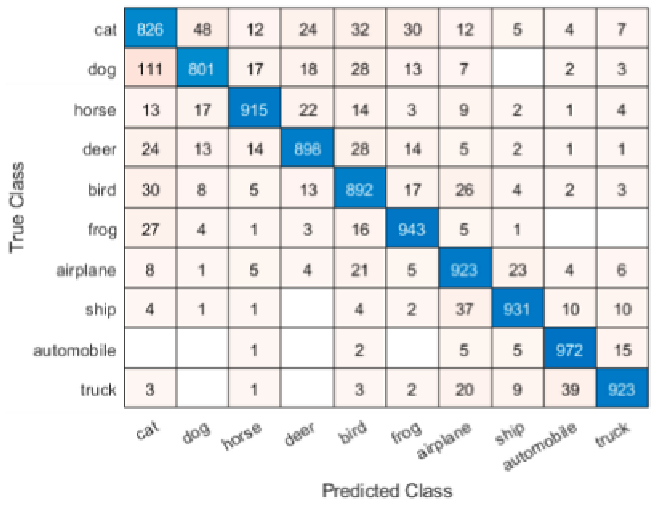
Confusion matrix is a specific table layout that allows visualization of the performance.

A matrix is a square with a side length equal to the total categories. One side is the actual class and one side is the prediction. The Diagonal represents the correct prediction whereas the other cells represent a false prediction to a different class.

Following are the examples of 2 classes and 10 classes:



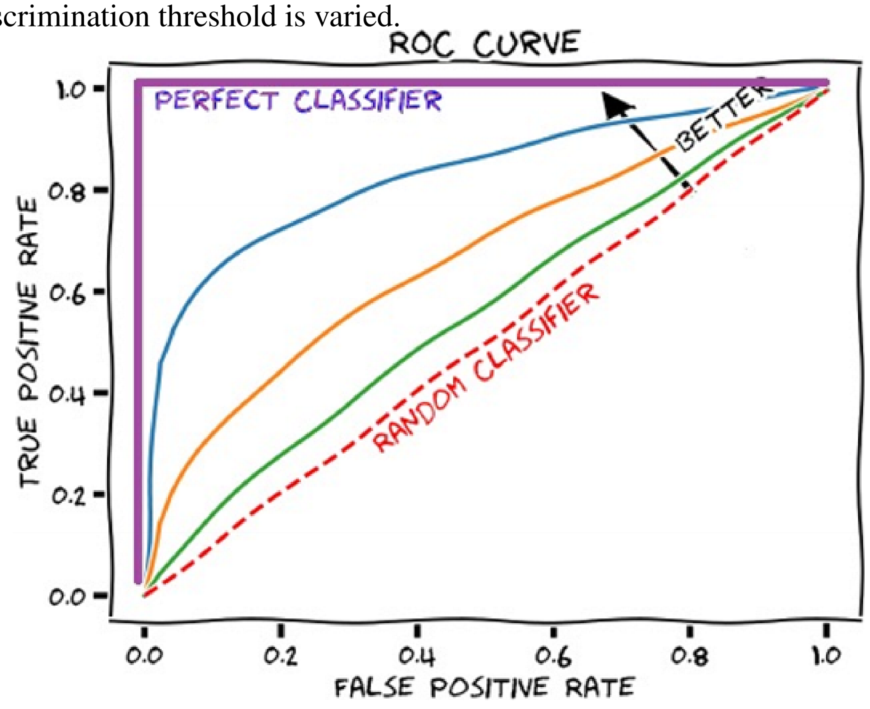
		PREDICTED LABEL	
		NEGATIVE	POSITIVE
TRUE LABEL	NEGATIVE	TRUE NEGATIVE	FALSE POSITIVE
	POSITIVE	FALSE NEGATIVE	TRUE POSITIVE



True Class \ Predicted Class	cat	dog	horse	deer	bird	frog	airplane	ship	automobile	truck
cat	826	48	12	24	32	30	12	5	4	7
dog	111	801	17	18	28	13	7		2	3
horse	13	17	915	22	14	3	9	2	1	4
deer	24	13	14	898	28	14	5	2	1	1
bird	30	8	5	13	892	17	26	4	2	3
frog	27	4	1	3	16	943	5	1		
airplane	8	1	5	4	21	5	923	23	4	6
ship	4	1	1		4	2	37	931	10	10
automobile			1		2		5	5	972	15
truck	3		1		3	2	20	9	39	923

5.5 Receiver operating characteristic curve (ROC)

ROC is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.



For random guess, true positive rate (TPR) = false positive rate (FPR). In contrast, perfect classifiers have only TPR and no FPR. So as long as the graph is above the random guess line (towards the perfect classifier), it is better than a random guess.

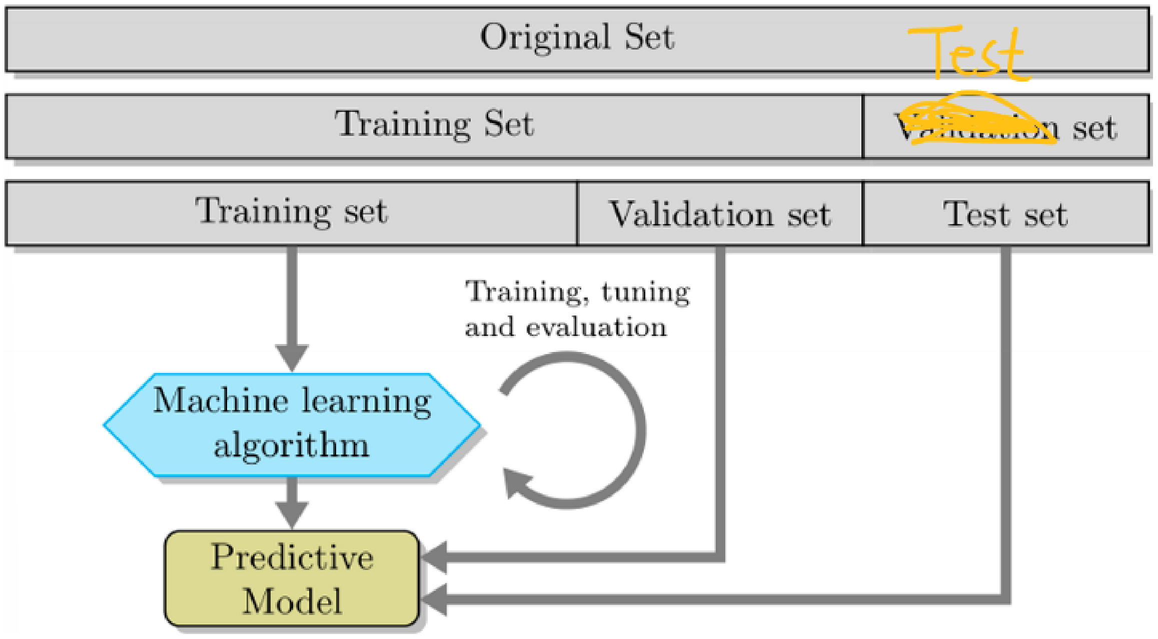
Comparing between the lines can be done by calculating the area under the curve if not easily visualized.

Refer to the slide for how it is calculated.

5.6 Cross validation

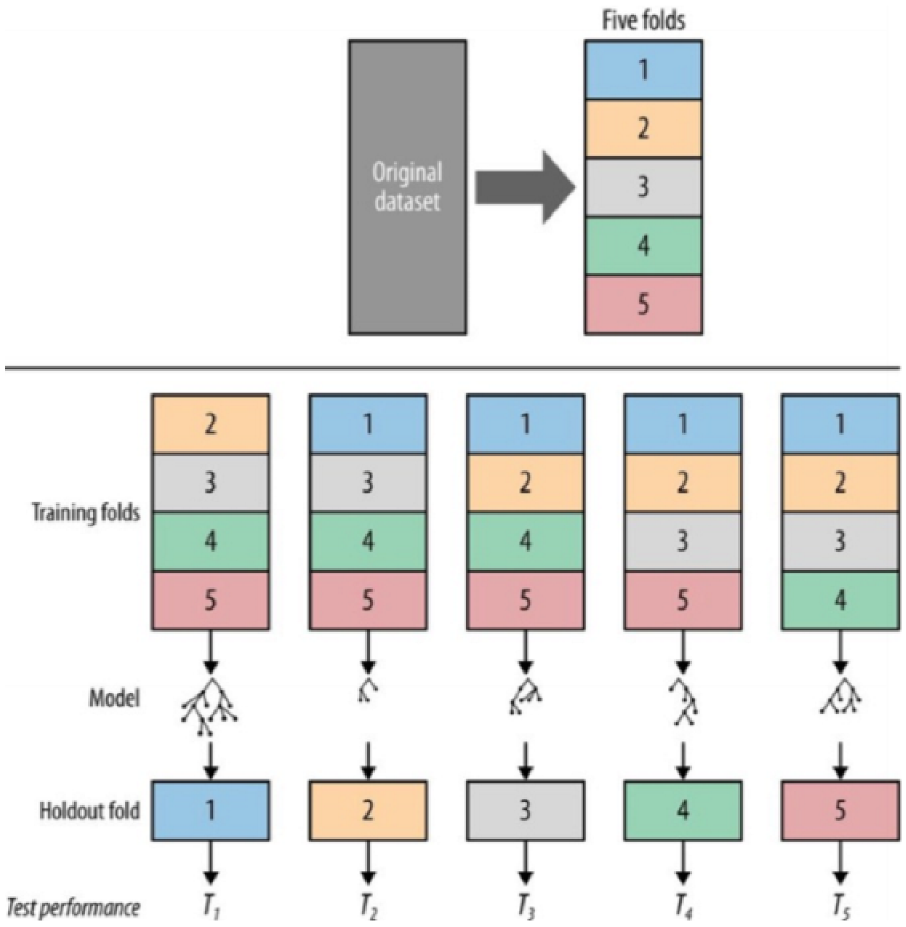
Performing machine learning involves creating a model, which is trained on a training data set and then process the test data set.

We sometimes also use a validation data set, which is a data set of examples used to tune the hyperparameters of the model.

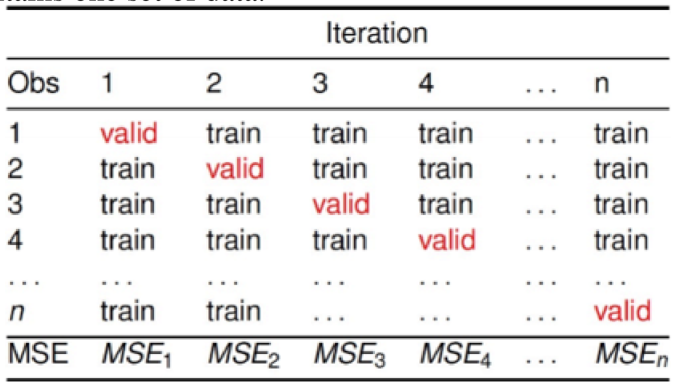


Cross-validation is used to evaluate AI models on a small-scale data set.

- Shuffle the data set randomly and split the data set into k groups.
- For each unique group:
  - Take the group as a holdout or test data set.
  - Take the remaining groups as a training data set.
- Summarize the skill of the model using all the model evaluation scores.



Leave-One-Out Cross-Validation: basically a special K group case where each group only contains one set of data.



	Iteration					
Obs	1	2	3	4	...	n
1	valid	train	train	train	...	train
2	train	valid	train	train	...	train
3	train	train	valid	train	...	train
4	train	train	train	valid	...	train
...	...	...	...	...	...	...
n	train	train	...	...	...	valid
MSE	MSE_1	MSE_2	MSE_3	MSE_4	...	MSE_n

LOOCV estimate (with n samples) of test error is given by

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} MSR_i$$

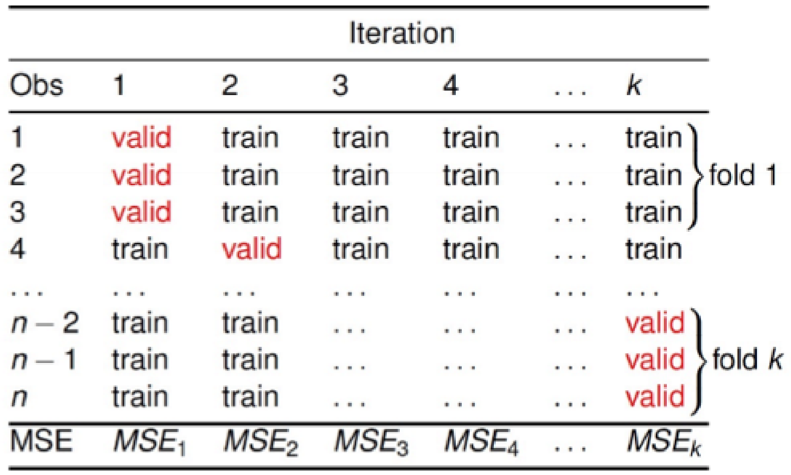
which is a single number, with no randomness. (K-fold has randomness since the the split group has different ways of formation.)

K-fold have each group formed randomly with a size larger than 1. Potentially faster approach.

- Randomly divide the data set into k folds.
- For $b = 1, \ldots, k$:
  - Use bth fold as a validation set.
  - Use the rest as the training set.
  - Compute validation error on bth fold.
- Estimate test error using:

$$CV_{(k)} = \sum_{b=1}^{k} \frac{n_b}{n} MSE_b$$

  where n_b is the total number obervations in bth fold and n is the total number of samples.



	Iteration						
Obs	1	2	3	4	...	k	
1	valid	train	train	train	...	train	fold 1
2	valid	train	train	train	...	train	
3	valid	train	train	train	...	train	
4	train	valid	train	train	...	train	
...	...	...	...	...	...	...	
$n-2$	train	train	...	...	...	valid	fold k
$n-1$	train	train	...	...	...	valid	
n	train	train	...	...	...	valid	
MSE	MSE_1	MSE_2	MSE_3	MSE_4	...	MSE_k	

6 Bayesian Classifier

6.1 Basic probabilities

Chain rule:

$$P(x, y, z) = p(x)p(y|x)p(z|x, y) = p(z)p(y|z)p(x|y, z)$$

Conditional Probability:

$$P(X = x|Y = y) = \frac{P(X = x) \cap P(Y = y)}{P(Y = y)}$$

$$P(x|y) = \frac{p(x, y)}{p(y)}$$

Marginalization:

$$p(x) = \sum_y P(x,y) = \sum_y P(y)P(x|y)$$

Bayes Rule:

$$P(x|y) = \frac{P(x)P(y|x)}{P(y)}$$

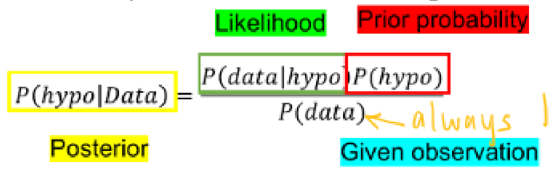
$$P(x|y,z) = \frac{P(x|z)P(y|x,z)}{P(y|z)}$$

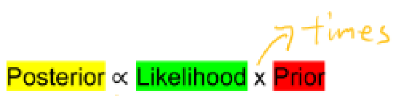
6.2 Defination:

Bayes' Original Theorem and Marginal Probability:

$$p(B|A) = \frac{p(A|B)p(B)}{p(A)} \quad p(A) = \sum_{B_i,\in S_B} p(A|B_i)p(B_i)$$

Probability of A is the sum of all probabilities of A given B_i for all B





Let $A_1, A_2, \ldots, A_k$ be the attributes with discrete values in class C. Given a test example, d with observed attribute values $a_1, a_2, \ldots, a_k$

If we assume that all attributes are conditionally independent, then we can say that:

$$Pr(A_1 = a_1, A_2 = a_2, \ldots, A_k = a_k)Pr(C = c_j) = \prod_{i=1}^{k} Pr(A_i = a_i|C = c_j)$$

Final Bayesian classifier:

$$Pr((C = c_j)|(A_1 = a_1, A_2 = a_2, \ldots, A_k = a_k)) = \frac{Pr(C = c_j)\prod_{i=1}^{k} Pr(A_i = a_i|C = c_j)}{\sum_{r=1}^{k}(Pr(C = c_r)\prod_{i=1}^{k} Pr(A_i = a_i|C = c_j))}$$

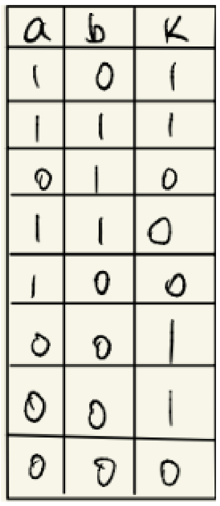
However, if we only need a decision on the most probable class for the test instance, we only need the numerator as its denominator is the same for every class.

Just need to find the c_j with the maximum value of the following:

$$c = \prod_{i=1}^{k} P(A_i = a_i|C = c_j)P(c_j)$$

- Bayesian learning is usually quite accurate with a certain degree of violation in practice.
- If the model is more mixed, it will tend to be more accurate.
- This approach is very efficient.

6.3 Example question:



a	b	K
1	0	1
1	1	1
0	1	0
1	1	0
1	0	0
0	0	1
0	0	1
0	0	0

Assume fearture a, b are independent, find the prediction probability for $P(k = 0|a = 1, b = 1)$

$$P(k = 0|a = 1, b = 1) = \frac{P(k = 0, a = 1, b = 1)}{P(a = 1, b = 1)}$$

Here $P(k = 0, a = 1, b = 1) = P(k = 0)P(a = 1|k = 0)P(b = 1|k = 0) = \frac{1}{2} \times \frac{1}{2} \times \frac{1}{2}$

Then calculate the denominator:

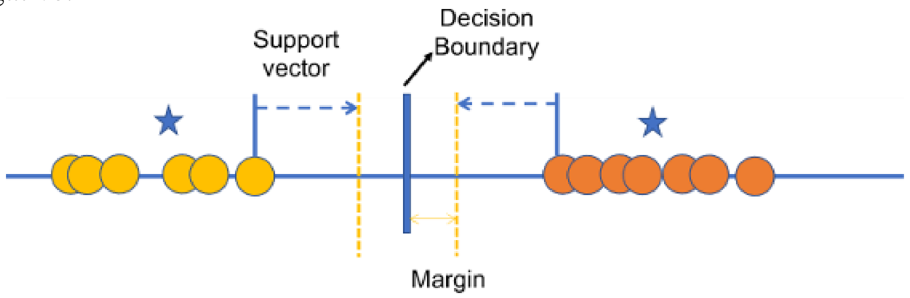
- Proper way:
  $P(a = 1, b = 1) = \sum_i^{\text{all possible k}} P(k = i|a = 1, b = 1)$
  $= P(k = 0|a = 1, b = 1) + P(k = 1|a = 1, b = 1)$
  where $P(k = 1|a = 1, b = 1) = P(k = 1)P(a = 1|k = 1)P(b = 1|k = 1) = \frac{1}{2} \times \frac{1}{2} \times \frac{1}{4}$
  So we have $P(a = 1, b = 1) = \frac{1}{8} + \frac{1}{16} = \frac{3}{16}$
- Sneaky way: Since a and ba are independent.

$$P(a = 1, b = 1) = P(a = 1 \wedge b = 1) = P(a = 1) \times P(b = 1) = \frac{4}{8} \times \frac{3}{8} = \frac{3}{16}$$

So the answer is $\frac{\frac{1}{8}}{\frac{3}{16}} = \frac{2}{3}$

7 Support vector machines (SVM)

SVMs are **linear classifiers** that find a hyperplane to separate two class of data, positive and negative.



Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane.

7.1 Basic concepts

Let the set of training examples D be represented by: $\{(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots, (\vec{x}_r, y_r)\}$, where $x_i = (x_1, x_2, x_3, x_4, \ldots, x_n)$.

So we have r samples each with n features. Then x_i is an input vector in a real-valued space $X \in R^n$ and y_i is its class label (output value), $y_i \in \{1, -1\}$. 1 for the positive class and -1 for the negative class.

SVM finds a linear function of the form (w: weight vector)

$$f(x) =< w \cdot x > +b$$

$$\begin{aligned} y_i &= 1 && \text{if } < w \cdot x_i > +b \geq 0 \\ y_i &= -1 && \text{if } < w \cdot x_i > +b < 0 \end{aligned}$$

Then the hyperplane that separates positive and negative training data is $< w \cdot x > +b = 0$. It is also called the **decision boundary (surface)**

A robust separating hyperplane has a “fat” margin (far from both sides of examples). So the goal is to find the fattest separating hyperplane.

7.2 linarly separable SVM

Assume that the data are linearly separable.

Consider a positive data point $(x^+, 1)$ and a negative $(x^-, -1)$ that are closest to the hyperplane.

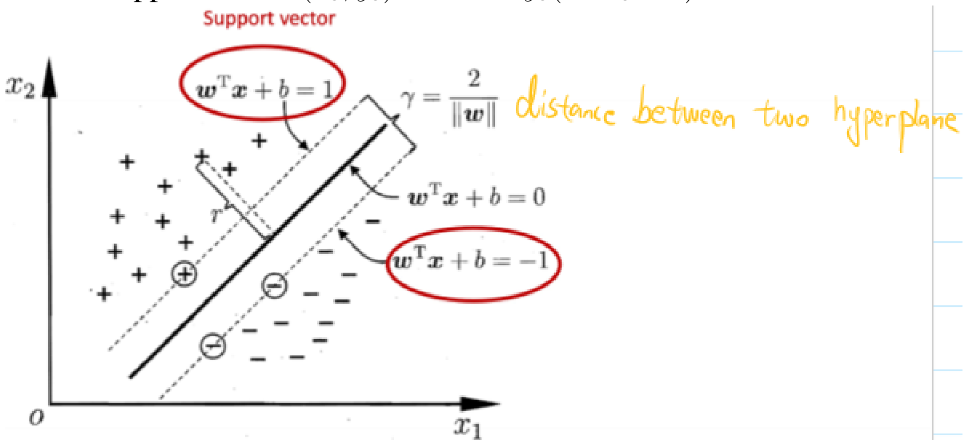
We define two parallel hyperplanes, H_+, H_- that pass through x^+, x^- respectively. The two hyperplanes should also be parallel to the decision boundary:

$$w^T x + b =< w \cdot x > +b = 0$$

We should be able to get:

$$\begin{aligned} H_+ : &\quad < w \cdot x > +b = 1 \\ H_- : &\quad < w \cdot x > +b = -1 \\ &\text{such that} \\ &< w \cdot x_i > +b \geq 1 \quad \text{if } y_i = 1 \\ &< w \cdot x_i > +b \leq 1 \quad \text{if } y_i = -1 \end{aligned}$$

Support vectors are the data points that lie closest to the decision boundary (hyperplane). So at a Support Vector (x_s, y_s), we have $y_s(w^T x_s + b) = \pm 1$



7.3 Compute the margin

To compute the distance between the two Marigin hyperplanes H_+ and H_-, recall that the (perpendicular) distance from a point x_i to the hyperplane $< w \cdot x > +b = 0$ is:

$$\frac{| < w \cdot x_i > +b|}{||w||}$$

Given $||w|| = \sqrt{< w \cdot w >} = \sqrt{w_1^2 + w_2^2 + \cdots + w_n^2}$

So for all support vectors on hyperplane, their distance is $\frac{y_i(w^T x_i + b)}{||w||} \rightarrow \frac{1}{||w||} \implies$ the summarized distance is called the margin $\gamma = \frac{2}{||w||}$

So now, our aim is basically to maximize the margin, which is to minimize the norm of w.

7.4 Optimization of SVM

Transforming the objective:

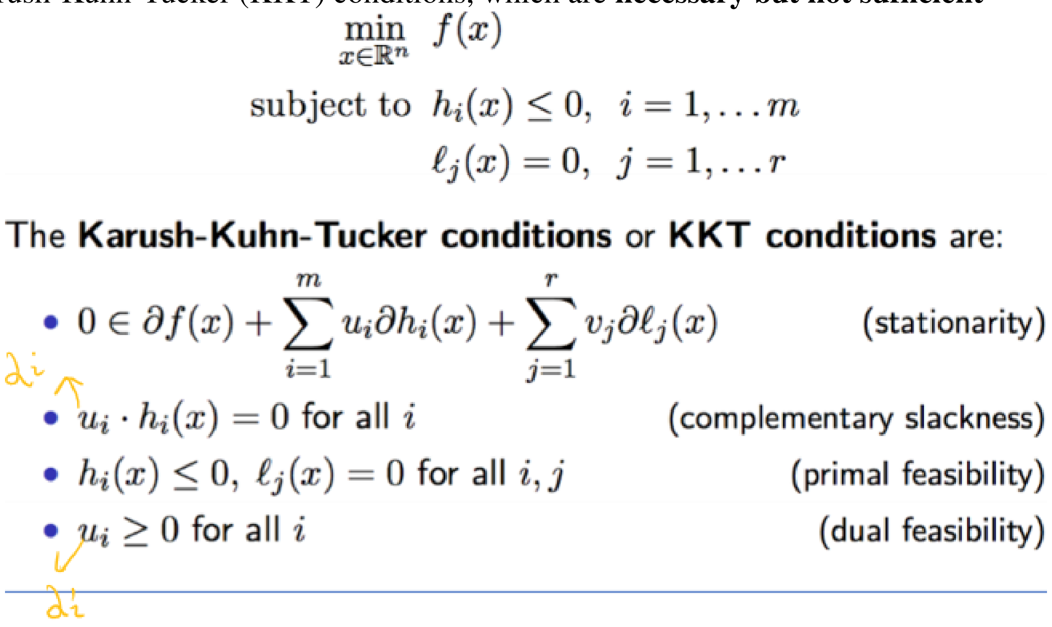
$$\max_w \frac{2}{||w||} \rightarrow \min_w \frac{1}{2}||w|| \rightarrow \min_w \frac{1}{2}||w||^2$$

So we need $\min_w \frac{1}{2}||w||^2$ s.t. $y_i(w^Tx_i + b) \geq 1$ or $g_i(w,b) = 1 - y_i(w^Rx_i + b) \leq 0; i = 1,2,\ldots,m$

Use Lagrange multipliers to solve the problem, write down the loss function as:

$$L(w,b,\alpha) = \frac{1}{2}||w||^2 + \sum_{i=1}^{m} \alpha_i(1 - y_i(w^Rx_i + b))$$
$$\alpha = (\alpha_1; \alpha_2; \ldots; \alpha_m)$$

Optimization theory says that an optimal solution to the Lagrangian method must satisfy Karush-Kuhn-Tucker (KKT) conditions, which are **necessary but not sufficient**



$$\min_{x\in\mathbb{R}^n} \; f(x)$$
$$\text{subject to } \; h_i(x) \leq 0, \;\; i = 1,\ldots m$$
$$\ell_j(x) = 0, \;\; j = 1,\ldots r$$

The **Karush-Kuhn-Tucker conditions** or **KKT conditions** are:

- $0 \in \partial f(x) + \sum_{i=1}^{m} u_i \partial h_i(x) + \sum_{j=1}^{r} v_j \partial \ell_j(x)$ (stationarity)
- $\overset{\lambda_i}{u_i} \cdot h_i(x) = 0$ for all i (complementary slackness)
- $h_i(x) \leq 0, \; \ell_j(x) = 0$ for all i, j (primal feasibility)
- $\underset{\lambda_i}{u_i} \geq 0$ for all i (dual feasibility)

Now take the derivative with respect to w and b, and set them equal to zero:

$$L(w,b,\alpha) = \frac{1}{2}||w||^2 + \sum_{i=1}^{m} \alpha_i(1 - y_i(w^Rx_i + b))$$
$$\frac{\partial L}{\partial w} = 0 \implies w = \sum_{i=1}^{m} \alpha_i y_i x_i$$
$$\frac{\partial L}{\partial \alpha} = 0 \implies 0 = \sum_{i=1}^{m} \alpha_i y_i$$

Now establish the KKT condition:

$$\alpha_i \geq 0$$
$$y_i(w^Tx_i + b) - 1 \geq 0$$
$$\alpha_i(y_i(w^Tx_i + b) - 1) = 0$$

Note that $\alpha_i = 0$ or $(w^Tx_i + b) - 1 = 0$ for the third condition.

- If $(w^T x_i + b) - 1 = 0$, it means that the sample point (x_i, y_i) is a support vector, and $\alpha_i > 0$
- if $(w^T x_i + b) - 1 > 0$, it means that the distance between the sample point (x_i, y) is farther than support vectors, and $\alpha_i = 0$

Now the optimization problem is a quadratic programming (QP) problem. There are some effective algorithm to solve the problem.

If we find the optimal α, we can find w using $w = \sum_{i=1}^m \alpha_i y_i x_i$.

For the optimal b, since we have optimal w and $y_s(w^T x_s + b) = 1$, we can use arbitrary support vector x_s, y_s to get $b = (\frac{1}{y_s} - w^T x_s)$, where $S = \{\alpha_i > 0, i = 1, 2, \ldots, m\}$ are the indices of support vector.

Practically, we suggest averaging over all eligible support vectors to alleviate numerical error:

$$b = \frac{1}{|S|} \sum_{s \in S} (\frac{1}{y_s} - w^T x_s)$$

7.5 Soft-margin SVM

Linear separable case is the ideal situation, real-life data may have noise or errors which would not satisfy the constraints, thus, no solution!

So the idea of soft-margin SVM is to allow some noisy samples not to obey the condition:

$$\begin{gathered}\text{goal: } \min \frac{1}{2}||w||^2 \\ \text{s.t. } y_i(w^T x_i + b) \geq 1 \text{ for clean } i \\ \text{and } y_i(w^T x_i + b) \geq -\infty \text{ for noisy } i\end{gathered}$$

7.5.1 Penalty function

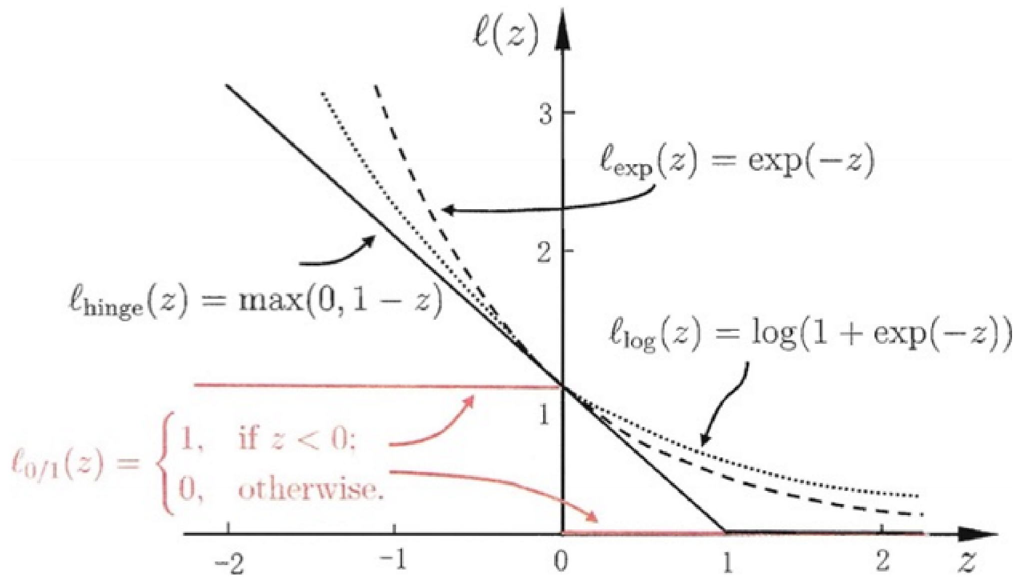
To minimize the number of noisy samples, we can use a new **penalty** item and our objective is transformed to:

$$\min_{w,b} \frac{1}{2}||W||^2 + C \sum_{i=1}^m l_{0/1}(y_i(w^T x_i + b) - 1)$$

Here, C is a constant and $l_{0/1}(z) = 1$ if $z < 0$, otherwise, it is 0. $z = (y_i(w^T x_i + b) - 1)$ in the equation.

So if the sample is in the constraint condition, there is no penalty. Otherwise, $l_{0/1}$ is 1 and it will be punished.

However, $l_{0/1}$ is a step function which is hard to optimize. Maybe we can use some other functions to replace it:



7.5.2 Relax the constraints

To allow errors in data, we relax the margin constraints by introducing **slack** variable, $\xi_i \geq 0$ as follows:

$$\begin{aligned} &< w \cdot x_i > +b \geq 1 - \xi_i && \text{for } y_i = 1 \\ &< w \cdot x_i > +b \leq -1 + \xi_i && \text{for } y_i = -1 \end{aligned}$$

Now we need to penalize the errors in the objective function. A natural way of doing it is to assign an extra cost for errors to change the objective function to minimize:

$$\frac{< w \cdot w >}{2} + C(\sum_{i=1}^{r} \xi_i)^k$$

k=1 is commonly used, which could make sure that neither ξ_i nor its Lagrangian multipliers appear in the dual formulation. Increasing k may allow more penalization for farther apart data points.

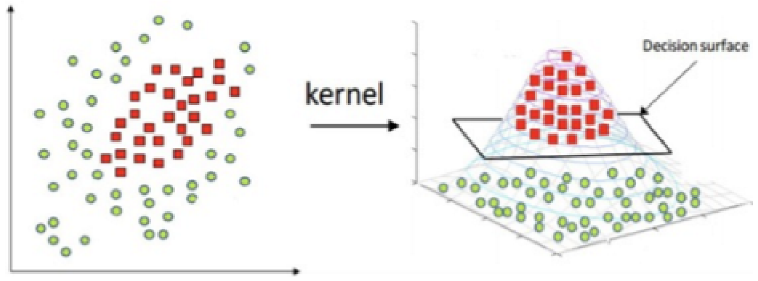
Summarize Lagrange multipliers:

$$L_p = \frac{< w \cdot w >}{2} + C(\sum_{i=1}^{r} \xi_i) - \sum_{i=1}^{r} \alpha_i [y_i(< w \cdot x > +b) - 1 + \xi_i] - \sum_{i=1}^{r} \mu_i \xi_i$$

7.6 Kernel SVM for non-linear datasets

To solve the cases where data sets are far from linearly separable, we increase the dimension, which makes them separable by planes.

For example, separate a 2D sample in 3D sapce:



Procedure:

- Map the original feature to the higher transformer space (feature mapping)
- Perform linear SVM in the higher space.
- Obtain a set of weights corresponding to the decision boundary hyperplane.
- Map this hyperplane back into the original 2D space to obtain a nonlinear decision boundary.

7.6.1 Polynomial kernel

It turns out that the above feature map corresponds to the well known **polynomial kernel** : $\kappa(x, x') = (x^T x')^d$. Let $d = 2$ and $\boldsymbol{x} = (x_1, x_2)^T$, we get

$\downarrow$ Scaler

$$k\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1' \\ x_2' \end{pmatrix}\right) = (x_1 x_1' + x_2 x_2')^2$$
$$= 2x_1 x_1' x_2 x_2' + (x_1 x_1')^2 + (x_2 x_2')^2$$
$$= (\sqrt{2}x_1 x_2,\ x_1^2,\ x_2^2) \begin{pmatrix} \sqrt{2}x_1' x_2' \\ x_1'^2 \\ x_2'^2 \end{pmatrix}$$

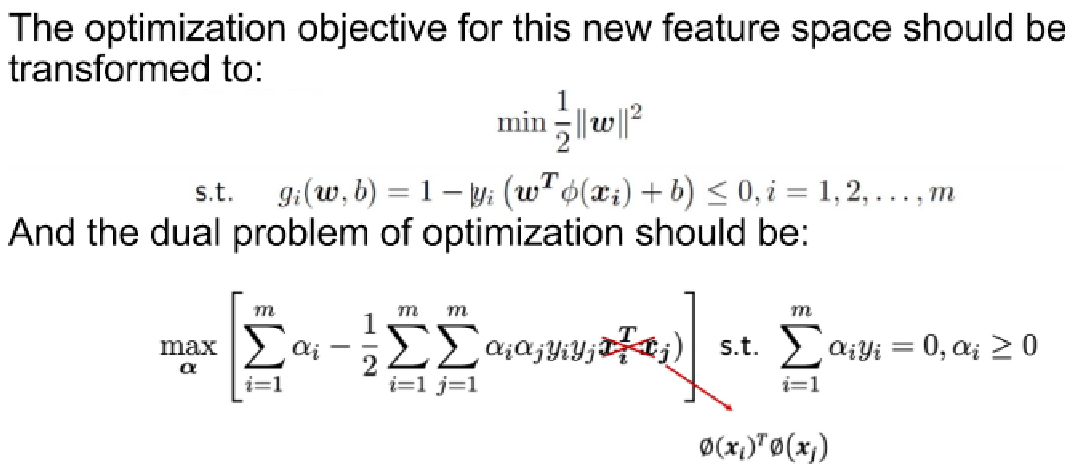
$$k\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1' \\ x_2' \end{pmatrix}\right) = \phi(\mathbf{x})^T \phi(\mathbf{x}')$$

$$\phi\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) = \begin{pmatrix} \sqrt{2}x_1 x_2 \\ x_1^2 \\ x_2^2 \end{pmatrix}$$

An example of transforming:
$X = (x^{(1)}, x^{(2)})^T \implies \phi(x) = ((x^{(1)})^2, (x^{(2)})^2, \sqrt{2x^{(1)}x^{(2)}})^T$

Now we can transform the two-dimensional data to threedimensional data by $x \to \phi(x)$, and the data can be linearly separatable in a higher-dimensional space:

$$f(x) = w^T \phi(x)$$


The optimization objective for this new feature space should be transformed to:

$$\min \frac{1}{2}\|\boldsymbol{w}\|^2$$

s.t. $g_i(\boldsymbol{w}, b) = 1 - y_i\left(\boldsymbol{w}^T \phi(\boldsymbol{x_i}) + b\right) \le 0, i = 1, 2, \ldots, m$

And the dual problem of optimization should be:

$$\max_{\boldsymbol{\alpha}} \left[\sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i\alpha_j y_i y_j \cancel{\boldsymbol{x_i^T x_j}})\right] \text{ s.t. } \sum_{i=1}^{m} \alpha_i y_i = 0, \alpha_i \ge 0$$

$\phi(\boldsymbol{x_i})^T\phi(\boldsymbol{x_j})$

7.6.2 Kernel trick for high dimensions

Very large feature spaces have potential issues of memory and computational cost. The kernel trick can help alleviate this issue. It allows operation in the original feature space without computing the coordinates of the data in the higher dimensional space.

Let's assume $k(x_i, x_j) = \Phi(x_i)^T\Phi(x_j)$

Do not understand this shit!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Refer to Soft-Margin-Kernel-SVM.ipynb in the same folder.

8 Decision tree

The decision tree splits the partition of the total possible space by multiple parameters. Each fork/node is split by 1 parameter. Prediction is made by the conjunction of all the provided parameters.

Prediction is made by tracing each test observation into a leaf R_j based on the sequence of conditions. Predict by R_j for all observation in R_j

Let $\hat{y}_{R_j}$ is a function of all training observations i in R_j

- Regression: $\hat{y}_{R_j} = \hat{y}_{i:i\in R_j}$
- Classification: $\hat{y}_{R_j}$ is the most frequently occurring class y_i for $i \in R_j$

Choice of tree building matters. We want to choose Rs that minimize RSS.

$$RSS = \sum_{j=1}^{j}\sum_{i\in R_j}(y_i - \bar{y}_{R_j})^2$$

Tree building takes a greedy approach. Grow the tree by **recursive binary splitting**, prune back the tree.

8.1 Algorithm

- Assume attributes are categorical now.
- Tree is constructed in a top-down recursive manner.
- Attributes are selected based on an impurity function (information gain)

8.1.1 Information theory

Information theory provides a mathematical basis for measuring information content.

If one already has a good guess about the answer, then providing the actual answer is less informative. So less we know, the more informative the truth is.

The objective is to reduce impurity or uncertainty in data as much as possible. The heuristic is to choose the attribute with the maximum **Information Gain or Gain Ratio** based on information theory.

A subset of data is pure if all instances belong to the same class.

8.1.2 Conditions for stopping partition

- all examples for a given node belong to the **same class**(So no need to split anymore)
- There are no remaining attributes for further partitioning
- There are no examples left

8.1.3 Overfitting

Overfitting trees get good accuracy on training data but poor on test data. It may have too deep and too many branches. Some may reflect anomalies due to noise or outliers.

Two approaches to avoid overfitting:

- Pre-pruning: Halt tree construction early. Difficult to decide because we do not know what may happen subsequently if we keep growing the tree.
- Post-pruning: Remove branches or sub-trees from a fully grown tree. This method is commonly used by estimating the errors at each node for pruning. A validation set may be used for pruning as well.

8.2 Calculate information gain

Information: reduction in uncertainty (amount of surprise in the outcome)

$$I(E) = \log_2 \frac{1}{p(x)} = -\log_2 p(x)$$

If the probability of this event happening is small and it happens, then the information is large! (surprise!)

Entropy: the expected amount of information when observing the output of a random variable X

$$H(X) = E(I(X)) = \sum_i p(x_i)I(x_i) = -\sum_i p(x_i)\log_2 p(x_i)$$

So the higher the disorder, the higher the entropy. But to get higher information gain, we prefer ordered, which means lower entropy. The lower the entropy, the purer the node is.

Now, we might be interested in the subnodes. So given the previous attribute A_i and calculate the next partition D

$$\text{entropy}_{A_i}(D) = H(D|A_i) = \sum_j^{|A_i|} P(A_i = a_j)H(D|A_i = a_j)$$

Here it is the sum of probabilities of a_j times the entropy of D given $A_i = a_j$. This way we get the entropy after splitting over A_i. We considered all the probabilities of A_i here.

The information gained by selecting attribute A_i to brach or to partition the data instance

$$\text{gain}(D, A_i) = H(D) - H_{A_i}(D)$$

So information gain = (information before split)-(information after split) We choose the attribute with the highest gain to branch/split the current tree.

8.3 Classification Trees

Let $\hat{y}_i$ be the mpst commonly occurring class of training observations in $R_j, \forall i \in R_j$

$$\hat{y}_{R_i} = \arg\max_k \hat{p}_{jk}$$

where $\hat{p}_{jk}$ is the proportion of trainning obserbations in R_j.

Since we want to make sure that our splitting is as biased as possible (contains the largest majority), it aligns with increasing the probability p.

No longer want to minimize RSS but instead minimize Classification error rate:

$$E = \sum_{j=1}^{J} |R_j|(1 - \max_k(\hat{p}_{jk}))$$

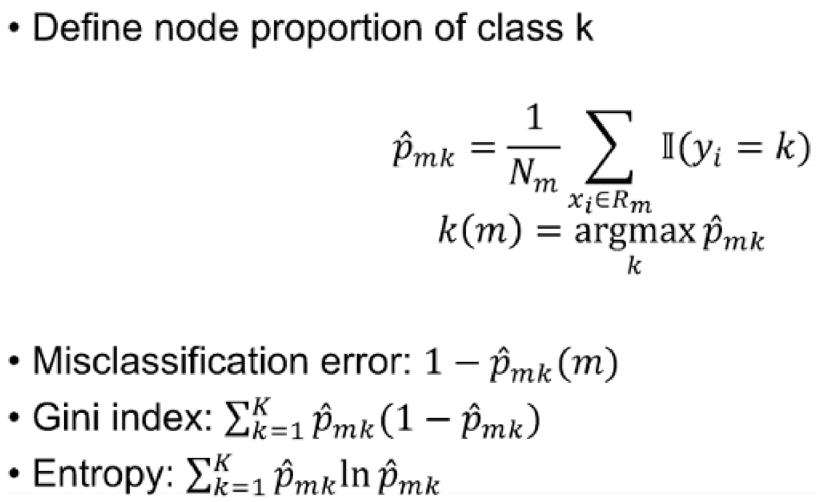
The Gini index does a similar job:

$$E = \sum_{j=1}^{J} |R_j| \sum_{k=1}^{K} \hat{p}_{jk}(1 - (\hat{p}_{jk}))$$

Note that the Gini index maximizes at $p = 0.5$ and minimizes at $p = 1, 0$, which makes sense since it wants one thing to dominate, and increase the purity.

8.4 Impurtiy measures

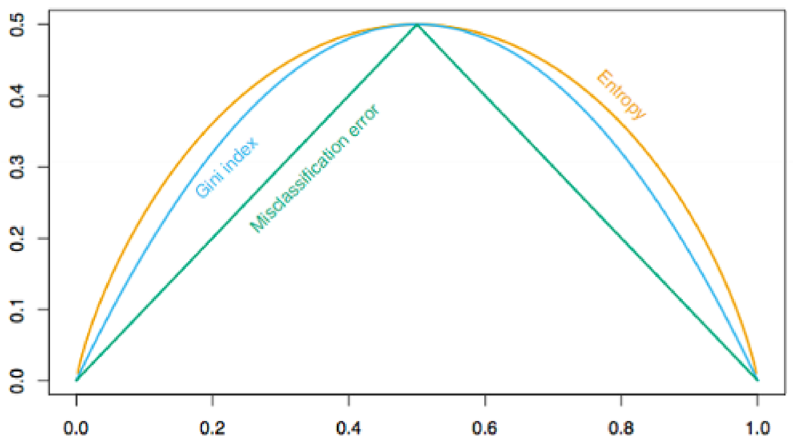
Define node proportion of class k:



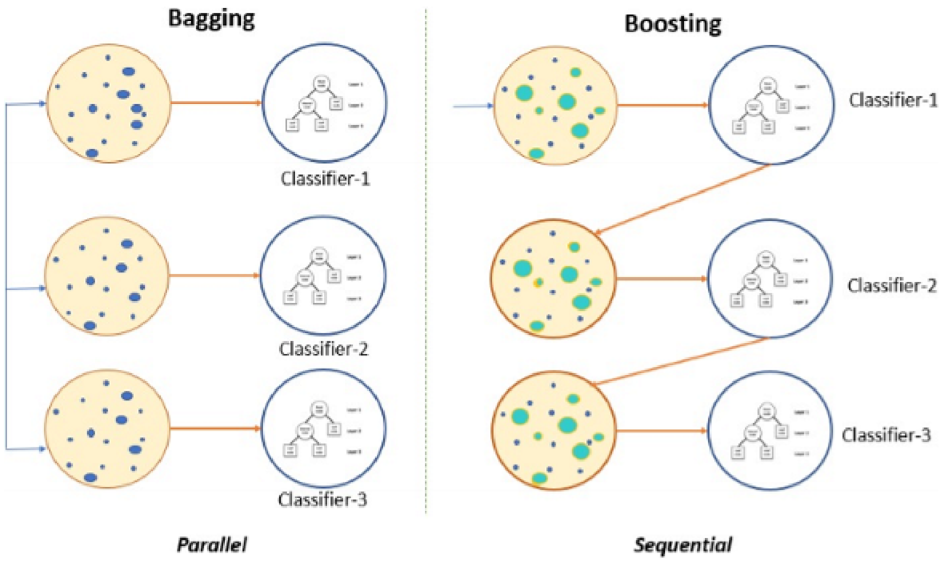
- Define node proportion of class k

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} \mathbb{I}(y_i = k)$$
$$k(m) = \underset{k}{\operatorname{argmax}}\, \hat{p}_{mk}$$

- Misclassification error: $1 - \hat{p}_{mk}(m)$
- Gini index: $\sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$
- Entropy: $\sum_{k=1}^{K} \hat{p}_{mk} \ln \hat{p}_{mk}$



8.5 Bagging and Boosting



8.5.1 Bagging (Bootstrap Aggregating)

In bagging, multiple subsets (bootstrap samples) of the training data are created by randomly selecting data points with replacements. Each bootstrap sample is used to train a separate decision tree.

The final prediction is made by aggregating (averaging or voting) the individual predictions of all the trees.

Bagging reduces the variance of the model and helps prevent overfitting. It is particularly useful when dealing with high-variance models, such as deep decision trees.

Key Features of Bagging:

- Parallel Execution: Each tree in the ensemble can be trained independently, making bagging suitable for parallel processing.
- Examples: Random Forest is a popular ensemble method that uses bagging with decision trees.
- Reduce variance, increase bias

8.5.2 Boosting

In boosting, an ensemble of weak learners is created sequentially. Each learner is trained using the data, and the focus is on examples that were misclassified by previous learners.

Weights are assigned to the data points, and misclassified points are given higher weights to encourage the next learner to focus on them.

The final prediction is made by combining the weighted predictions of all learners.

Boosting is effective at improving both bias and variance, and it can lead to highly accurate models. It is particularly useful when dealing with high-bias models.

Key Features of Boosting:

- Sequential Execution: Learners are trained sequentially, and each one tries to correct the mistakes made by the previous learners.
- Examples: AdaBoost, Gradient Boosting and XGBoost are popular boosting algorithms used with decision trees.
- Reduce bias, increase variance

8.5.3 Differences

1. **Sequential vs. Parallel:** Bagging learners are trained in parallel while boosting learners are trained sequentially.
2. **Data Importance:** Bagging treats all data points equally, whereas boosting assigns different weights to data points based on their classification difficulty.
3. **Combining Predictions:** Bagging combines predictions by averaging or voting while boosting combines predictions with weighted voting.

8.5.4 When to Use

- Use bagging when dealing with complex, high-variance models to reduce overfitting.
- Use boosting when you want to improve the accuracy of weak models, particularly when dealing with high-bias models.

It's worth noting that ensemble methods, such as Random Forest (a bagging technique) and Gradient Boosting (a boosting technique), have become popular and widely used for various machinelearning tasks due to their effectiveness in improving model performance.

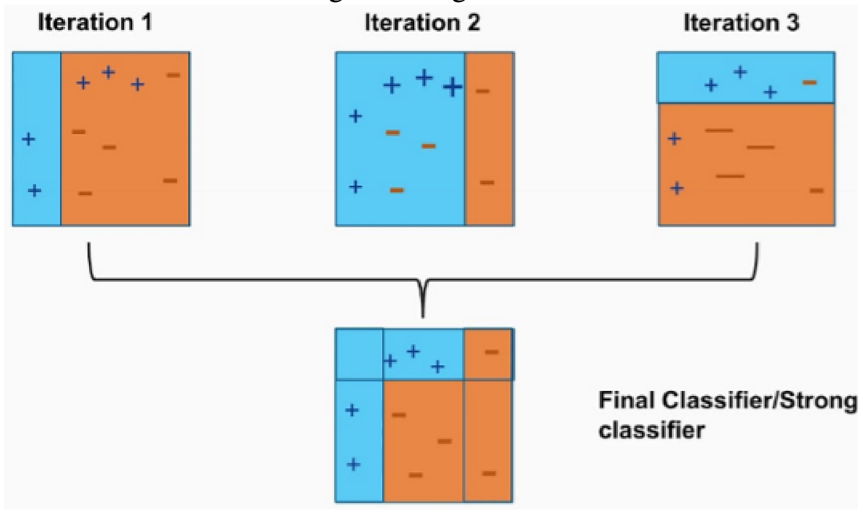
8.6 Boosting Algorithm

Boosting is a strategy to boost a weaker learner to a stronger learner.

- It starts with learning with the base learner M_1.
- Then based on the performace of M_1, we **reweight/resampling** the trainning samples. E.g., assigning larger weights to difficult samples.
- Based on the new distribution of samples, we train another learner M_2.
- Recursibely, we train $M1, \dots, M_T$.

- Finally, we aggregate the learners, e.g., $M = \sum_{i=1}^{T} \alpha_i M_i$

Here is an illustration of using Boosting:



To be added!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

8.7 Bagging

Bagging is short for **Bootstrap Aggregating**

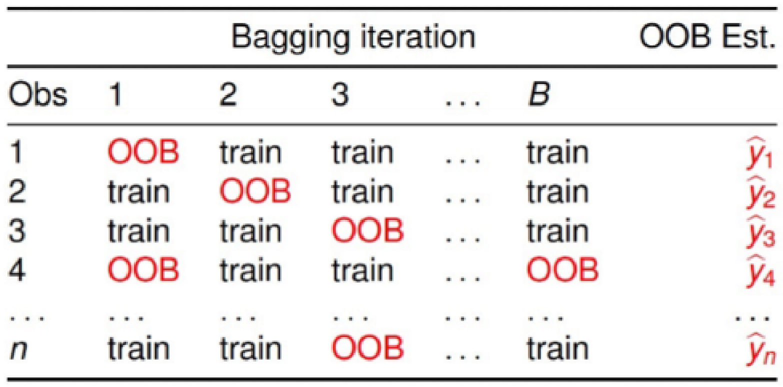
- Given a set D containing n training examples.
- Create a subset S of D by drawing n examples at **random with replacement** from D Chamce a particular example does not appear in the sample:

$$(1 - \frac{1}{n})^n \to \frac{1}{e} \approx 0.37$$

- S of size n: expected to leave out 0.37 of examples from D
- S contains duplicate samples

The idea of Bagging is to eliminate the influence of some very strong attributes so that the trees do not look similar to each other since there is a dominant attribute.

For the portion that we did not use, we call them out-of-bag samples(OOB), which we can use to calculate OOB MSE or classification error.



	Bagging iteration					OOB Est.
Obs	1	2	3	...	B	
1	OOB	train	train	...	train	$\widehat{y}_1$
2	train	OOB	train	...	train	$\widehat{y}_2$
3	train	train	OOB	...	train	$\widehat{y}_3$
4	OOB	train	train	...	OOB	$\widehat{y}_4$
...	...	...	...	...	...	...
n	train	train	OOB	...	train	$\widehat{y}_n$

By splitting the trees, we could ideally split an infinite amount of trees if we have infinite samples. The result is the sum of the majority vote.

If we split the data in random different ways, decision trees give different results and high variance. However, Bagging is a method that results in low variance.

If we had multiple realizations of the data (or multiple samples) we could calculate the predictions multiple times and take the average of the fact that averaging multiple onerous estimations produces less uncertain results.

Experimentally, bagging can help substantially for unstable learners, and may somewhat degrade results for stable learners.

The problem occurs when there is only one very strong predictor. It will always be selected, thus similar trees will be produced. This is due to the greedy tree algorithm.

8.7.1 Greedy tree

Assume we have data $D = \{A, B\}$ where $A = \{a_1, a_2\}, B = \{b_1, b2\}$.

Choose from information gain: $IG_A = H(D) - H(D|A), IG_B = H(D) - H(D|B)$

If $IG_A > IG_B$ split from A, else split from B.

So that is why one can dominate!

There is a way to skip from the dominance:

If we only consider a subset of the predictors at each split chosen randomly, then it is not a correlated tree. $\rightarrow$ **Random Forest**

8.8 Random Forest

Random forests (RF) are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest.

Using a random selection of features to split each node to save the issue of having similar trees.

The key difference between random forests and bagging is that random forests choose a sample of k predictors (attributes) at split out of a full set of d predictors. So $k < b$. But Bagging does not reduce it, so $k = b$.

8.8.1 Random Forests Algorithm

For b=1 to B:

1. Draw a bootstrap sample S of size d from the training data.
2. Grow a random forest tree to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree until the minimum node size $n_{\min}$ is reached.
    (a) Select k variables at random from the p variables.
    (b) Pick the best variable/split point among the k.
    (c) Split the node into two daughter nodes.
3. Output the ensemble of trees.

Note that, to predict a new point x, we need to do:

- Average the results for regression
- Majority vote for classification

RF increases the diversity of trees by perturbating samples (bootstrapping). RF adds another level of perturbation to attributes and usually achieves better generalization.

Like with Bagging, we can use OOB and therefore RF can be fit into one sequence, with cross-validation being performed along the way. Once the OOB stabilizes, the training can be terminated.

8.9 Comparing Bagging, Boosting with RF

- Bagging vs RF
    - It is obvious that RF makes small changes to Bagging
    - Different from Bagging which increases the diversity of trees by perturbating samples (bootstrapping)
    - RF adds another level of perturbating(randomness) on attributes. (Not only randomly select samples out of bags, but also features)
    - RF can achieve better generalization, and usually behaves better.
- Boosting vs RF
    - RF's accuracy is as good as Adaboost and sometimes better;
    - RF is relatively robust to outliers and noise;
    - RF is faster than bagging or boosting;
    - RF gives useful internal estimates of error, strength, correlation and variable importance;

  - RF is simple and easily parallelized

The issue of RF is that: When the number of variables is large, but the fraction of relevant variables is small, random forests are likely to perform poorly when k is small. It is because, at each split, the chance can be small that the relevant variables will be selected since we are randomly selecting variables.

8.10 Variable importance

While bagging improves upon the predictive ability of trees, it kills off the interpretability of the model. Now we need a tool to measure the **variable importance**

Variable importance can be measured by the amount that the Residual Sum of Squares (RSS) or Gini index is reduced due to splits over a given predictor, averaged over all B trees.

8.10.1 Gini importance

Mean Gini gain produced by X_j over all trees (m indicates region(leaf), k indicates class) for variables of different types: biased in favor of variables with many categories (or continuous variables)

Gini index: $\sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$, here $\hat{p}_{mk}$ is the frequency of sample in some class, max at $\hat{p}_{mk} = 0.5$. So lower the Gini, the more is reduced and the more information gained.

8.10.2 Permutation importance

Another smart way of detecting the importance of the feature is to change the order of the data representations of one feature while keeping the other features the same. Then we use the gini index gain by splitting at that node to compare the value. If the feature is more important, then the error is higher.

9 K-Nearest Neighbor Classification (KNN)

Unlike all the previous learning methods Logistic Regression, **KNN does not build a model from the training data**

In essence, KNN performs a voting mechanism to determine the class of an unseen observation. This means that the class with the majority vote will become the class of the data point in question.

So no training is needed. Classification time is linear in the training set size for each test case.

KNN can do both regression and classification. For regression, we just average it, for classification, we group it.

9.1 KNN intuition

To classify a test instance d, define $K-$neighborhood P as k nearest neighbors of d.

The predicted label for d is the majority class in P

If $K = 1$, then we will use only the nearest neighbor to determine the class of a data point. Same thing as K increases.

9.2 KNN Algorithm

For each data point in the test data:

1. Find the distance to all training data samples
2. Store the distance on an ordered list and sort It
3. Choose the top K entries from the sorted list
4. Label the test point based on the majority of classes present in the selected points.
5. If it is a regression task, just take the average of the K neighbors

9.2.1 Measure of similarity/distance

Imagine for the two points $\vec{a} = (a_1, a_2, \ldots, a_n), \vec{b} = (b_1, b_2, \ldots, b_n)$

- Euclidean distance (L2-norm). The square root of the sum of squares.

$$\text{dist}(\vec{a}, \vec{b}) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$$

- Manhattan distance (L1-norm). The sum of the absolute difference.

$$\text{dist}(\vec{a}, \vec{b}) = \sum_{i=1}^{n} |a_i - b_i|$$

- Cosine similarity. The angle between two points.

$$\text{dist}(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{||\vec{a}|| \cdot ||\vec{b}||} = \frac{\sum_{i=1}^{n}(a_i \cdot b_i)}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2}}$$

In most cases, Euclidean Distance is the default.

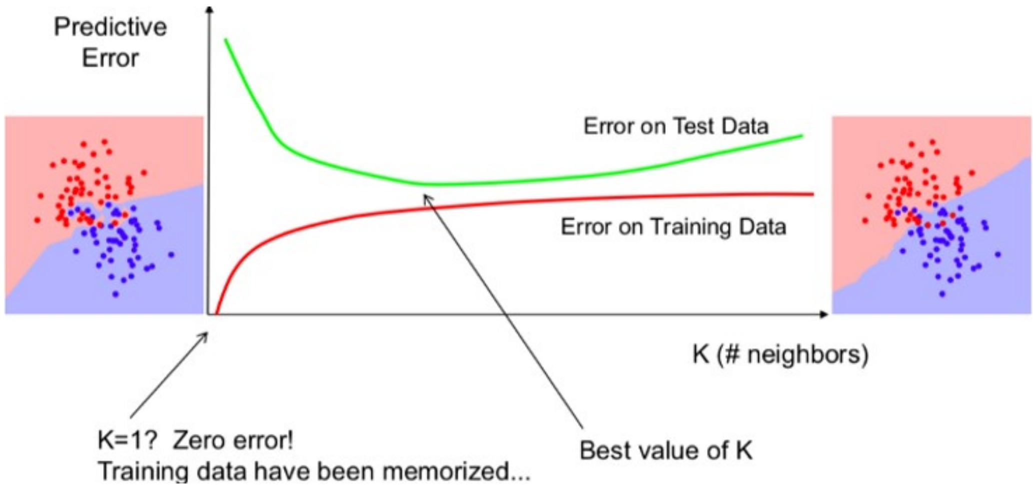
9.3 Discussion

KNN can deal with complex and arbitrary decision boundaries with reasonable accuracy. But KNN is slow at the classification time.

9.3.1 Voting

We are giving equal weight to voting for the k nearest neighbors. However, one optimization might be that voting can be weighted according to the distance to the destination.

9.3.2 Bias and Variance on K



When K = 1, the nearest neighbor is just itself, which is basically memorization. So very high variance due to overfitting. On the other hand, if K = number of samples, then that is basically underfitting, thus high bias. So Low K leads to High variance and High K leads to High bias.

10 Clustering

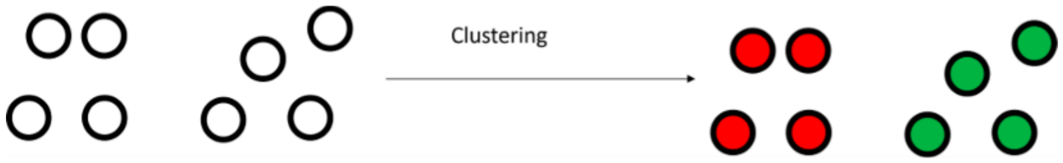
Now we moved to the region of unsupervised learning. Given a set of x, underlying structure or relationships of x.

10.1 Clustering basis

Samples represent a set of individuals or objects collected or selected from a statistical population by a defined procedure. Each sample can be called a Data Point.

Label is a category or class applied to an object or a thing.

A clustering group is a set of objects so that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters).



There is a high intracluster similarity and low intercluster similarity.

- Intracluster cohesion (compactness): Cohesion measures how near the data points in a cluster are to the cluster centroid.

- Intercluster separation (isolation): Separation means that different cluster centroids should be far away from one another.

11 K-Means

The K-Means algorithm partitions the given data into K clusters:

- Each cluster has a cluster center (a.k.a centroid)
- K is manually specified by the user

11.1 convergence (stopping) criteria

- no (or minimum) reassignment of data points to different clusters, or
- no (or minimum) change of centroids or
- minimum decrease in the sum of squared error (SSE)

Let C_i be the ith cluster, and μ_i be the centroid of C_i (the mean vector of all data points in C_i). If x is the data point in the cluster and we are considering Euclidean distance:

$$\text{SSE} = \sum_{i=1}^{K} \sum_{x \in C_i} \text{disntace}(x, \mu_i)^2$$

11.2 algorithm

1. Select K points as the initial centroids.
2. repeat:
   (a) Form K clusters by assigning all points to the closest centroid.
   (b) Recompute the centroid of each cluster.
3. until fulfill the stopping condition

11.3 complexity

No understanding for now. Check the slides

11.4 Advantages

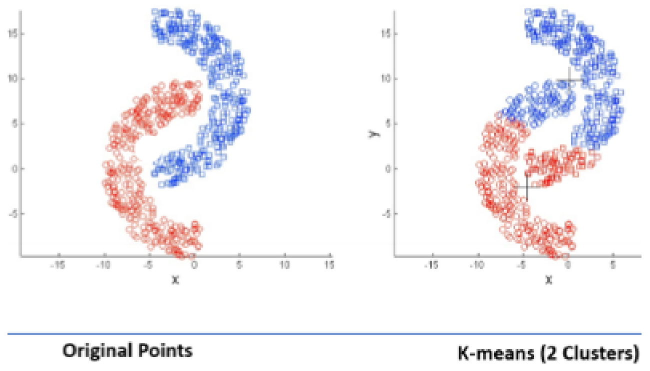
- Easy to understand and implement
- Efficient, if K and I are small, considered as a linear algorithm
- Scale to large data sets
- **Guarantee convergence**
- Clusters can be easily interpreted and even visualized.
- May produce tighter clusters than hierarchical clustering

11.5 Disadvantages

1. KMeans needs to **manually** set hyperparameter K.
    - Sensitive to the hyperparameter of preassigned number of clusters.
    - Different K would lead to different clustering results.

   So if K is too small, it will lead to insufficient partitioning. K too large will lead to over-partitioning.
2. KMeans algorithm is sensitive to outliers.
    - Outliers are data points that are very far away from other data points.
    - Outliers have a high influence since they can make SSE large, which will pull the center towards the outliers a lot.
3. KMeans may not work well for data points with nonglobular shapes

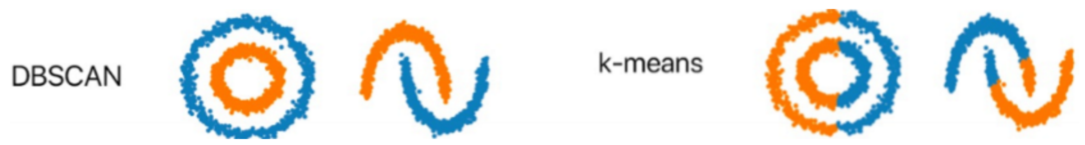
   Since the cluster is based on Euclidean distance, the circular cluster is preferred and will be dominant:



12 Density-based clustering

Different from KMenas:

- Clusters are dense regions in the data space
- A cluster is defined as a maximal set of density-connected points
- Noise points lie in regions of low-density
- Discover clusters of arbitrary shape and size.



12.1 DBSCAN

DBSCAN stands for Densitybased spatial clustering of applications with noise.

Three parameters:

- ϵ: maximum radius of the neighborhood.
- $\epsilon-$Neighbor: data points within a radius of ϵ from a data point (including the point itself)
- *MinPts*: minimum number of points required in a $\epsilon-$Neighbor.

Definition:

- density is the number of points within a specified radius
- "high density" means data point's $\epsilon-$Neighbor contains at least MinPts data.

Given ϵ and MinPts, categorize the data points into three:

1. Core points: has more than or equal to MinPts within ϵ. Should be at the interior of a cluster.
2. Border point: has fewer than MinPts within ϵ, but is in the neighborhood of a core point.
3. Noise point or Outlier: any point that is neither a core point nor a border point.

As the name suggests, core points are in the center of a cluster, and border points form the border of the cluster and then enclose each cluster.

12.1.1 Density-reachable

An object q is directly density-reachable from object p if p is a core object and q is in p's $\epsilon-$neighborhood.

Note that it is not symmetric, reflexive or transitive.

12.2 algorithm

1. Label data points in core, border and noise
2. Eliminate noise points
3. For every **core points** p that has not been assigned to a cluster: Create a new cluster with the point p and all the points that are densityconnected to p. Then extend this to all core points in the same cluster till all no more directly reachable core points are available in the cluster.
4. Assign border points to the cluster of the closest core point.

12.3 principles

DBSCAN relies on a densitybased notion of the cluster: A cluster is defined as a maximal set of densityconnected points.

1. Each cluster contains at least one core point.
2. Given any two core points p and q, if N(p) contains q or N(q) contains p, then p and q are in the same cluster.
3. A border point p is assigned arbitrarily to one of the clusters containing these core points. (Maybe choose the one with closer distance?)
4. All noise points do not belong to any clusters.

12.4 Advantages

- DBSCAN can handle clusters of arbitrary shapes and sizes.
- Number of clusters is determined automatically, which is different from KMenas.
- Can separate clusters from noise, Robust to outliers.
- Also a very commonly used clustering algorithm.

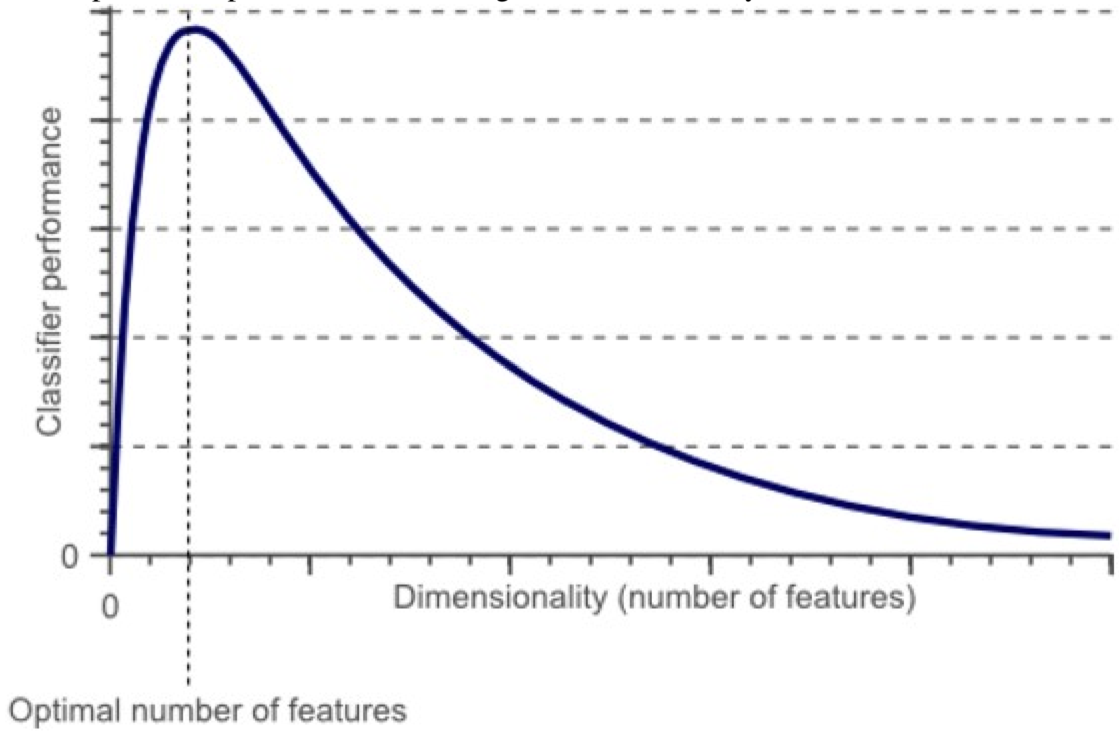
12.5 Disadvantages

- Sensitive to parameters: hard to determine the optimal parameters.
- Has problem identifying clusters of varying densities.
- DBSCAN may not work well in highdimensional data. (one dimension might dominate)
- Density estimation is kind of simplistic (e.g., does not create a real density function, but rather a graph of density-connected points)

13 Principal Component Analysis (PCA)

13.1 The need for dimension reduction

Samples in data sets might contain thousands of features. However, too many features not only cause computational problems but also might cause inaccuracy.



13.1.1 Concept of dimensionality reduction

Dimensionality reduction is the process in which we transform $X \in \mathbb{R}^D$ to $X' \in \mathbb{R}^d$. Here X' represents X when $d << D$.

- X' can be regarded as an approximation or abstraction of X
- it is expected to preserve the information of X as much as possible while reducing the dimension. So it keeps only the feature with high variance. So other words, drop all the identical/repeated/common/similar features.

Besides common patterns, correlated features can also be reduced.

- Consider some 2dimensional features X_1, X_2 for each dimension are two random variables.
- If the two random variables are **highly positive correlated**, we can observe these feature vectors are around a line spanned by vector S.
- In this way each feature vector can be **approximated** by kS.

- So we can use a 1dimensional vector with corresponding coefficient k to represent a 2-dimensional feature.

13.1.2 Projection

Although feature spaces of raw data are always high dimensional. In many cases, feature vectors do not occupy the entire highdimensional feature space and most of them may lie around a low-dimensional subspace of the ambient highdimensional space.

Thus, we can find a subspace of it to preserve the best representation of the original data. To represent, we orthogonally project a feature vector $X \in \mathbb{R}^D$ onto the subspace.

For example, the feature vector $\vec{X}$ is a $D \times 1$ vector. Then to represent with a lower dimension $d << D$. We can use:

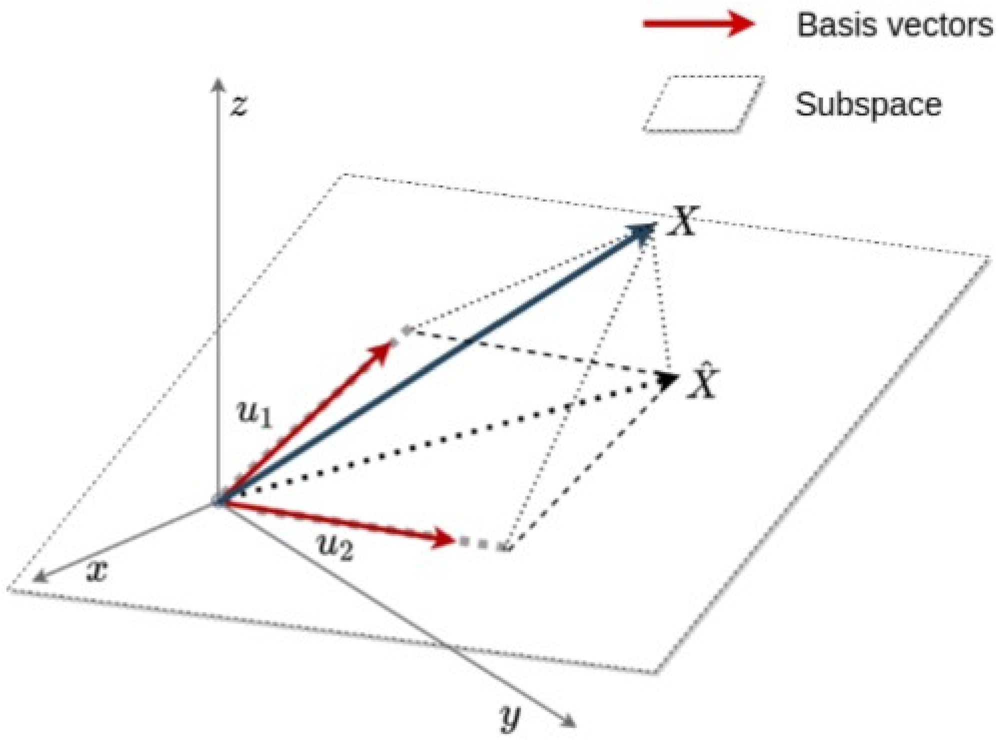
$$\vec{X}' = U\vec{X}$$

So U is a $d \times D$ matrix that could lower the dimensionality and we need to find it. The same use can be extended to multiple data matrices.

Note that:

- U is an orthogonal matrix, meaning that elements $u_1, u_2, \ldots$ are all orthogonal to each other.
- $UU^T X \neq X$ because projection will lose some information.

The figure below illustrates an orthogonal projection of X on the subspace spanned by u_1, u_2.



13.2 PCA algorithm

One thing to note is that to correctly do PCA. It is important to normalize data. Otherwise, it is not going to produce the correct result.

13.2.1 Covariance

Variance and Covariance measure the "spread" of a set of points around their center of mass (mean).

- Variance measures the deviation from the mean for points in one dimension
- Covariance measures how much each of the dimensions varies from the mean with respect to each other.
  - Covariance is measured between two dimensions
  - Covariance sees if there is a relation between two dimensions
  - Covariance between one dimension is the variance

We get positive Covariance when two dimensions are positively related, meaning increase and decrease at the same time. Negative Covariance would require the opposite behavior.

We use it to find the relationships between two dimensions in highdimensional data sets:

$$q_{jk} = \frac{1}{N}(X_{ij} - E(X_j))(X_{ik} - E(X_k))$$

Note that the expected value is just the sample mean.

13.2.2 Spectral theory

PCA is related to eigenvalues. Review eigenvalue:

$$Av = \lambda v, \forall A \in \mathbb{R}^{m \times n}$$

Any λ, v pairs that satisfy the above relationship and are not zero are the eigenvalue and eigenvector.

For symmetric $A, (m = n, A^T = A)$, it must have n real eigenvalues and vectors. Here we replace A with the data covariance matrix in PCA:

Give an example below:

Want $X = \begin{pmatrix} -1 & -1 & 0 & 2 & 0 \\ -2 & 0 & 0 & 1 & 1 \end{pmatrix}$ to be reduced to 1 dimension where the first row is feature 0 and the second row is feature 1.

1. Normalize it so that each row has a mean of 0 (which is done already in the matrix)

2. Find the Covariance matrix: (since the mean is 0, there is no subtraction)

$$C = \frac{1}{N}XX^T = \frac{1}{5}\begin{pmatrix} -1 & -1 & 0 & 2 & 0 \\ -2 & 0 & 0 & 1 & 1 \end{pmatrix}\begin{pmatrix} -1 & -2 \\ -1 & 0 \\ 0 & 0 \\ 2 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{6}{5} & \frac{4}{5} \\ \frac{4}{5} & \frac{6}{5} \end{pmatrix}$$

With that we can find eigenvalue and vector:

$$\lambda_1 = 2, c_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \lambda_2 = \frac{2}{5}, c_2 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

3. Now we standardize eigenvectors:

$$\begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \implies P = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$$

4. Now we choose 1 eigenvector to lower it since we want a 1 D representation.

$$Y = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}\begin{pmatrix} -1 & -1 & 0 & 2 & 0 \\ -2 & 0 & 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} \frac{-3}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 & \frac{3}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{pmatrix}$$

Summarize the algorithm in words:

1. Find the mean of each feature
2. Compute a $D \times D$ sample covariance.
3. Find the first d eigen vector of covariance. Note that $d << D$ since it is the lower dimension you are aiming for
4. Lastly, project it by timing the d eigenvector matrix to the original data.

13.2.3 Optimization

Choosing the larger eigenvalue would cause less loss of information. It can be found using the Variance percentage:

$$p_i = \frac{\lambda_i}{\sum \lambda_i}$$

So $\lambda_i > \lambda_j \implies i^{th}$ eigenvector explains more variance.

14 Nural Network basis

Linear models need *non-linearities* to be able to extend work to greater areas. Thus call for neural networks:

14.1 Nonlinearities

For linear regression, our loss function, for example, (x, y) is:

$$L = \frac{1}{2}(y - \beta^T x - \beta_0)^2 = \frac{1}{2}(y - f)^2$$

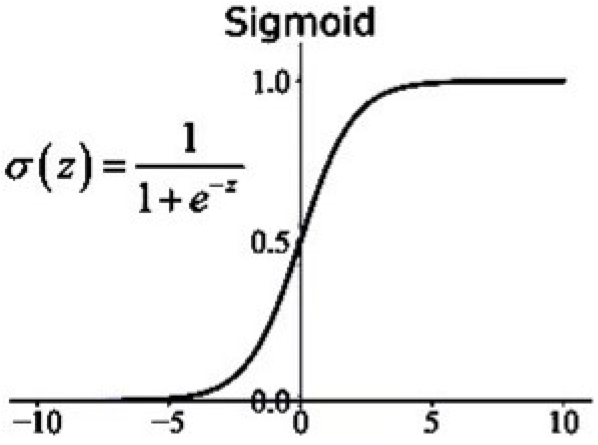
where $f = \beta^T x + \beta_0$. Now consider $f = \beta^T h + \beta_0$:

- If $h = Wx + b$, then it is still a linear model.
- We have to add a non-linear function Φ to it. Let $h = \Phi(Wx + b)$. But note that the last layer $f = \beta^T h + \beta_0$ is still linear.

This way, we can add as much information as we want while keeping the output linear.

14.1.1 Activation (nonlinear) functions

- Sigmoid squashes real number to range between 0 and 1

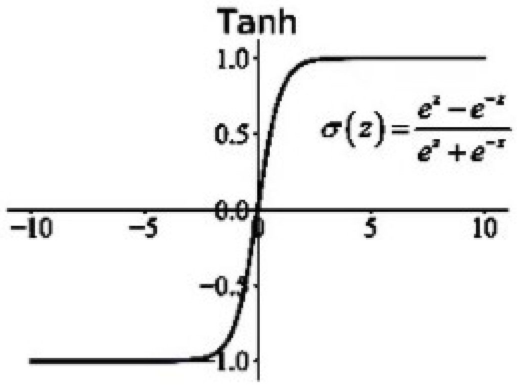
  It is a nonlinear funciton:

  - Countinuous which is easy to derivate.
  - Limited output range, easy to optimize, can be used as output layer.

  When $x \to \pm\infty$, the gradient approaches 0, A.K.A **gradient vanishing**. It might terminate gradient descent undesirably.

  Also, it is not centered at 0, making it slow in convergence.

- Tanh squashes real numbers to range between (-1,1)

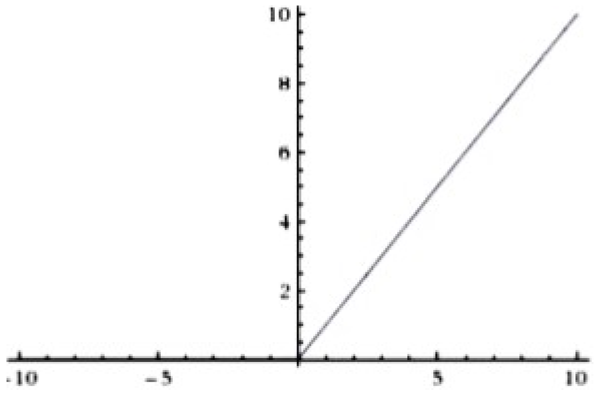


$$\text{Tanh} = 2 \times \text{Signmoid} - 1 = 2\sigma(x) - 1$$

It is centered at 0, making it converge fast. However, it still experiences **gradient vanishing**, when $x \to \pm\infty$.

Also, the computational complexity is high since it involves too many exponentials.

- ReLU (Rectified Linear Unit) is a step function $f(x) = \max(0, x)$.

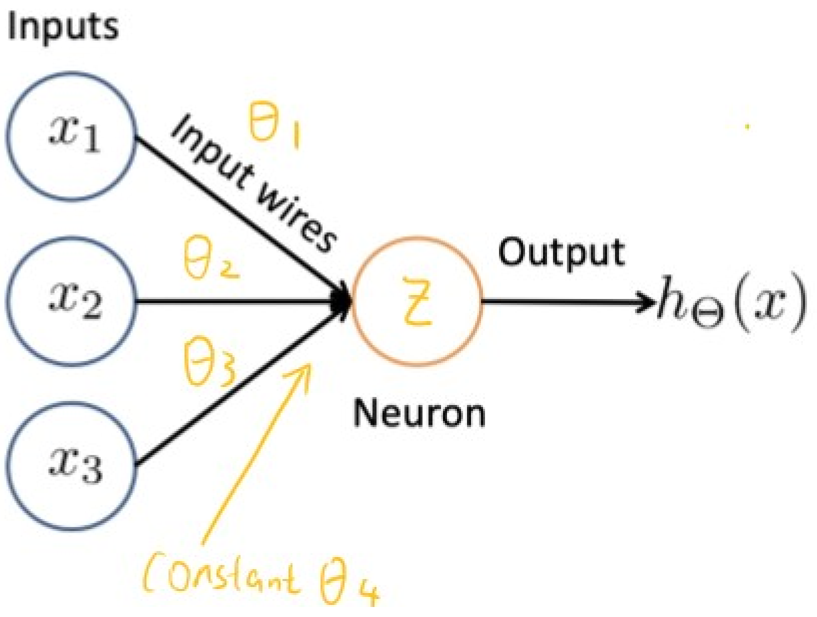


Compared to the previous two, it never experiences gradient disappearance since the gradient is never saturated and the convergence speed is fast. Also, no exponentials, so the calculation is fast and complexity is low.

However, when $x < 0$, the gradient is 0. Thus, the parameter is not updated, A.K.A **neuron death**. At the same time, the output mean value is greater than 0, which affects the convergence of the network.

14.2 Neruon meodel

Here we model a neuron as a logistic unit:



$$Z = f_\theta(\vec{x}) = \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4$$

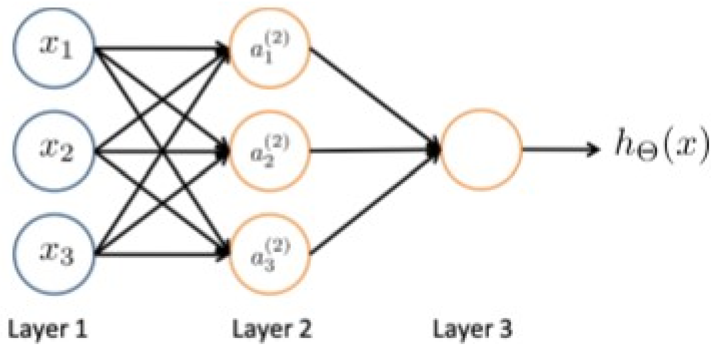
Here θ_4 could be some constants introduced here.

Now output is an activation function where $\vec{x} = [x_1, x_2, x_3]^T$, $\Theta = [\theta_1, \theta_2, \theta_3]^T$:

$$g_\Theta = \frac{1}{1 + e^{-\Theta^T \vec{x}}}$$

The activation function exists so that we can get an answer of either 1 or 0.

14.3 Feed Forward



The first layer is called the input, final layer is the output. Everything between the input and output are hidden layers as they cannot be seen.

Hidden layers are the ones to keep the training information by tuning the parameters.

Here we can get value for layer 2 by:

- $a_1^{(2)} = g(\Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$

- $a_2^{(2)} = g(\Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$
- $a_3^{(2)} = g(\Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$

Here, $\Theta^{(1)}$ is the coefficient/weight from layer 1 to layer 2. g is the activation function. Note that g(x) is not linear!!!

Note that each calculation in a neural network only involves the data layer and the next layer. So a relationship can be extended to numerous layers without increasing the complexity of each calculation.

14.3.1 model complex function with nonlinearity

Each neuron calculation can be seen as a step function with a corrected coefficient/weight that could be used to represent a piece of function.

Now if we cut a complex function infinitely small, then each small piece can be approximated using a step function with weights. This way, all functions can be modeled by a neural network.

14.4 Backpropagation

A neural network consists of two components:

- The network architecture, which defines layer numbers, neuron numbers and connections.
- The parameters, values of the connections and model weights.

14.4.1 Before starting

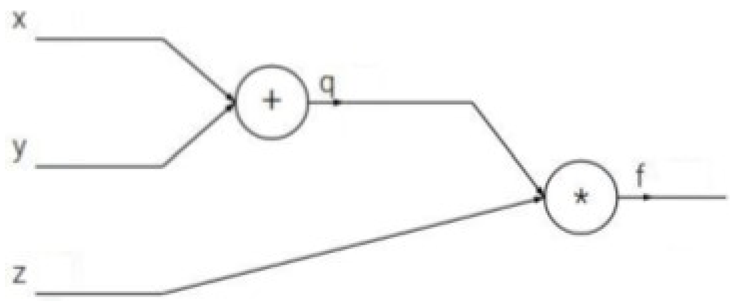
Before starting, we need to initialize the parameters:

In Practice, randomly initialize the parameters to small values, usually normally distributed around 0; $\mathcal{N}(0, 0.1)$

14.4.2 backpropagation algorithm

1. Input sample to the network and calculate the output (forward pass)
2. Compare the output with the correct output to get the loss term.
3. For all layers (backward pass from output layer, to hidden, to input)
    - propagates the loss term back to the previous layer
    - updates the weight between the two layers until the earliest hidden layer is reached

Give a simple example:



It is a simple illustration of $f(x, y, z) = (x + y)z$.
Note that:

$$\frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial x} = 1$$
$$\frac{\partial f}{\partial q} = z \qquad \frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = z$$
$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y} = z$$

Because of the Chain Rule, we can build a gradient connection between the output and input via propagation.

14.5 Training and Validation

General formulation:

- Given an input dataset $X = \{x_1, x_2, \ldots, x_n\}$ with corresponding label $Y = \{y_1, y_2, \ldots, y_m\}$
- Build a neural network with parameter Θ
- Get prediction $\hat{Y} = \{\hat{y_1}, \hat{y_2}, \ldots, \hat{y_m}\}$, where $\hat{y_i} = f(x_i, \Theta)$
- Total loss $\mathcal{L} = \sum_{i=1}^{n} l(y_i, \hat{y_i})$

14.5.1 MSE

We can use the mean square error as the loss:

- Mean square error (MSE) is the average squared difference between the estimated values and the actual value. MSE can be expressed by

$$l_{MSE} = (y_i - \hat{y_i})^2 = (y_i - f(c_i, \Theta))^2$$

- Further, we can compute the gradient of MSE by

$$\frac{\partial l_{MSE}}{\partial \Theta} = -2(y_i - f(x_i, \Theta))\frac{\partial f(x_i, \Theta)}{\partial \Theta}$$

See the slides for an example

14.6 Cross entropy error

Cross entropy error (CE) is the crossentropy between two probability distributions over the same underlying set of events.

- CE can be expressed by

$$l_{CE} = -y_i \log(\hat{y}_i) = -y_i \log(f(x_i, \Theta))$$

- Further we can compute the gradient of CE by

$$\frac{\partial l_{CE}}{\partial \Theta} = -\frac{y_i}{f(x_i, \Theta)} \frac{\partial (x_i, \Theta)}{\partial \Theta}$$

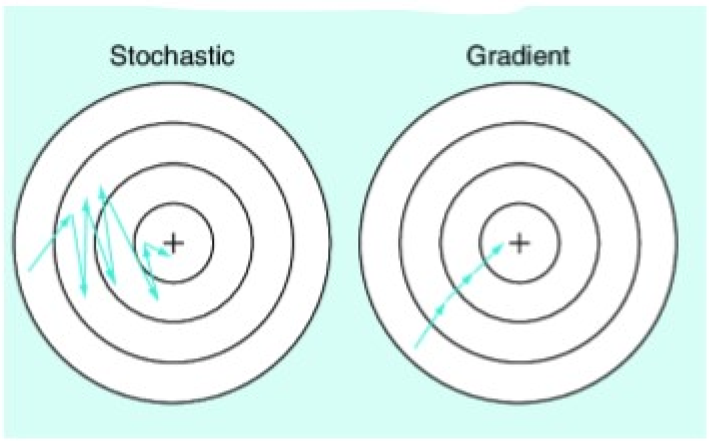
See the slides for an example

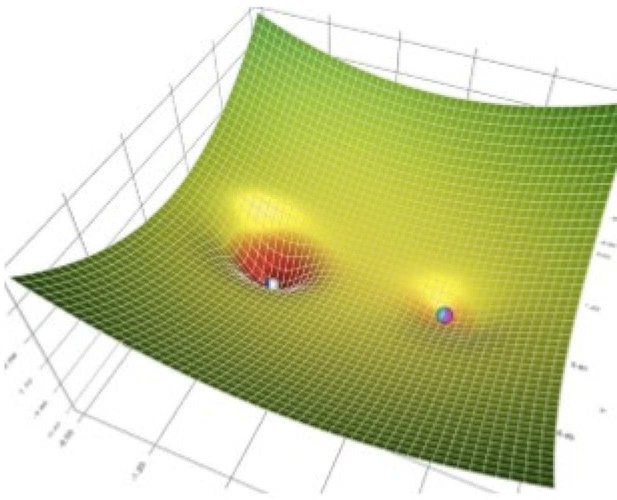
14.6.1 Stochastic Gradient Descent (SGD)

We use SGD be there is more than 1 optimal point or local minimum. We would aim for the global minimum:

$$\theta_{t=1} = \theta_t - \eta g(\theta_t) \qquad \eta \text{ is the learning rate}$$

$$g(\theta_t) = \frac{1}{b} \sum_{i \in B_t} \nabla_\theta l(x_i; \theta_t) \qquad b \text{ is the Batch size}$$





14.6.2 Training

- One Epoch is completed when a complete data set is cycled forward and backward through the neural network.
- Batch size is a parameter that defines the number of samples to work through before updating the neural network weights.
- Iterations are the number of batches needed to complete on epoch

14.6.3 Learning rate decay

We should reduce the learning rate as we complete more Epoch. Since we do not want oscillations when getting to the correct gradient. There are lots of ways to decay:

1. Step decay: reduce the rate at a few fixed points, making it as stairs.
2. Linear decay: linearly decay proportional to the number of Epoch
3. Cosine decay
4. Inverse square root decay

15 NN advance

There are lots of variants if neural network:

- Convolutional Neural Networks (CNN): Popular for Image Data, directly applicable for 2D representations
- Recurrent Neural Networks (RNN): Contains self-loop, Good for Sequential data.
- Generative Adversarial Network (GAN):
- Transformer:
- Graph Neural Networks (GNN): deal with graph inputs.

15.1 AutoEncoder(AE)

Here are some important aspects for AE:

- Unsupervised: No labels used, discovers useful features of the input.
- Compression: Code reduces the dimension of data
- Lossy: input won't be reconstructed exactly
- Trained: The compression algorithm is learned for specific data

16 Ok! Neural Network is too hard, comeback and fill it when my math improves