Introduction to Machine Learning

Khoi Nguyen
ducminhkhoi@gmail.com,
http://www.nguyenducminhkhoi.com

December 16, 2015

This is the overview of basic and important machine learning algorithms, methods and concepts and theories. I acknowledge all information and knowledge including images, data... I took from those 2 machine learning courses from **Coursera**[1], **Stanford**[2] and **Oregon State University**[3].

Contents

[1] https://www.coursera.org/learn/machine-learning
[2] http://cs229.stanford.edu/
[3] http://classes.engr.oregonstate.edu/eecs/fall2015/cs534/

1 Introduction

First, when we talk about Machine Learning, we usually talk about models to fit the data. There are a lot of problems in Machine Learning, but we can list some here are **regression**, **classification** and **clustering**. On another aspect, we can classify Machine Learning method by **supervised**, **unsupervised** and **semi-supervised** learning. With supervised learning, we always have both $\{X, Y\}$ values in training examples while unsupervised we just have X value. Semi-supervised learning just has a small number of $\{X, Y\}$ and major is X only. So that is the differences between many machine learning methods depend on learning problem.

Besides, we can divide a typical machine learning process into 4 steps as follow: data pre-processing step, learning step, testing and report step. With data pre-processing step with the aim is that transform the raw data into standard data called **training set** input which have the form $\{X, Y\}$ where $X =< x_1, x_2, x_3, \ldots, x_n >$ are the **features** or attributes of the data. For example, when learning the parameter for predicting the weather tomorrow is

sunny or rainy. We have many features like today's temperature, today's humidity, today's weather and so on. And Y is the value tomorrow's is rainy or sunny. With the learning step, the learner (machine learning method) will try to learn some "hidden" parameters from these training examples to form a **model**. After that, the learner will apply this model to new data in **testing set** and try to predict new $\hat{Y}$ from X_{test} and compare values of $\hat{Y}$ with the value of Y_{test}. In some learning method, they also have hyper-parameter that is, cannot learn directly from the training example only, they have to use another set called **validation set** to learn these hyper-parameter. And finally report the result based on some **metrics** for each different learning problems. Now we come to the simple method for regression, that is linear regression.

2 Regression Algorithms

2.1 Linear Regression

Linear Regression defines the relation between X and Y in linear. That is

$$y = w_1x_1 + w_2x_2 + \ldots + w_nx_n$$

where $w_1, w_2, \ldots, w_n$ are the parameters of X to form a line or hyperplane. Linear Regression's goal is to learn these parameters, and when a new testing example, they will calculate the new $\hat{Y} = w_1x_1 + w_2x_2 + \ldots + w_nx_n$. In the testing step, we just calculate the accuracy of the linear regression model by using Sum of Square Error metric, that is:

$$SSE = \frac{1}{2}\sum_{i=1}^{N}(\hat{y_i} - y_i)^2$$

The smaller SSE, the better learner. But how we estimate these w for the linear regression model. We have several ways. First, we can use normal equation to find the closed form of the solution. In this case, we have $w = (X^TX)^{-1}X^TY$, however, this way is just applied to a small dataset.Algorithms

With larger data set, we cannot use this method, instead, we use **Gradient Descent** method to find w. We have to set up the Target function, in this case, is Loss function: SSE. We derive the gradient of SSE with the following formula:

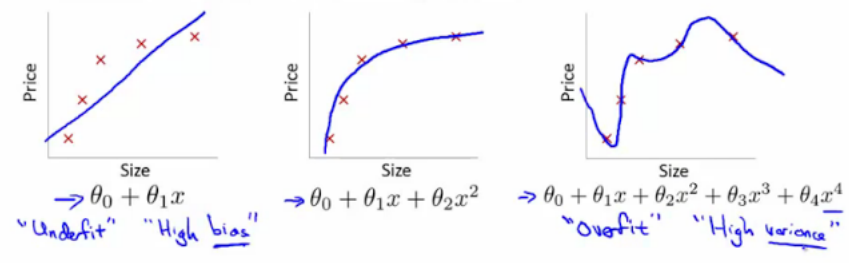
$$\nabla E(w) = \sum_{i=1}^{N}(w^Tx_i - y_i)x_i$$

And we update w with formula: $w = w - \lambda\nabla E(w)$ we repeat this step until $\nabla E(w) < \epsilon$, which ϵ is the converge criterion. λ here is learning rate. If λ is small, slower to converge but it avoids overstepping. Otherwise, λ is large, the learning algorithm will converge faster, but it will be oscillating in some cases. That is **Batch Gradient Descent**, it requires all training examples for each step to update w. In some cases, it takes a lot of times to train. However, we

can use **Stochastic Gradient Descent** to update w with just one learning example with $w = w - \lambda(w^T x_i - y_i)x_i$.

However, many problems in the real world are not linearly separable, so we have to modify the Linear Regression to work well with nonlinearly separable data as well. One of the solutions is to use a function with M-order polynomial. Instead of using different features, we use different orders of the same features or combine them in a different way.

For example, we can use $< 1, x, x^2, x^3, \ldots >$ to fit non-linear separable problems. With that idea, we can draw a conclusion is that the higher the order, the better the learner will perform because it is clear that the higher order will fit the data better than lower order polynomial. But wait, it is right for training data, with the testing data, it is another story. If the learner fit too much with the training data, it called **overfit**, then when it comes with a new example from testing data, the performance will decrease significantly. So we have a contradiction here is the lower polynomial will not match with the data both from training data and testing data, it called **underfit**, however, when the order of the polynomial is so high, it makes the learner become overfit with the training data. Therefore, there must be a particular order of polynomial that give the best performance in testing data. We can see the illustration in the following figure.



To achieve this, we have introduced the **regularization** term. We have the objective function as follow.

$$\sum_{i=1}^{N} (\hat{y}_i - y_i)^2 + \lambda \sum_{j=0}^{M} |w_j|^q$$

where λ in this scenario is **regularization coefficient**, M is the order of the polynomial and q is the type of regularizers (we commonly have $q = 1$ is called **Lasso regularization**, $q = 2$ is called **Ridge regularization**). We can also use Gradient Descent to estimate the w from the Regularized Objective.

With the regularization term, we have the closed form of linear regression:

$$w = (\lambda I + X^T X)^{-1} X^T Y,$$

If λ is large, it puts more penalty for regularization term or high order polynomial so it likely makes the lower order while small λ will allow the learner

fit more with the training data and it likely makes the higher order. To get the optimal λ we can use **cross-validation** with **k-fold**. In this case, k is the number of parts separated from the training data, with 1 part is used for testing and $k-1$ parts are used for training. The value of λ that give the best average performance on cross-validation is chosen.

That is all the basic things to know about the linear regression.

3 Classification Algorithms

3.1 Generative and Discriminative Models

With Linear Regression model, we can solve many regression problems, however, with classification problem we must have another model to handle. There are two different approaches the classification problem, they are **Generative** model and **Discriminative** model.

The Generative model in general will extract the probability distribution from the training example, and then with the new example, they will estimate the probability of the new example with different classes of classification problem, i.e. $P(y = k|x)$, which k is the class. The class with higher probability $P(y = k|x)$ will be chosen. Examples of this type of model is **Naive Bayes** and **Linear Discrminative Analysis** and One important characteristic that all Generative Models have is to use Bayes rule to calculate the $P(y = k|x)$:

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

Because P(x) are the same for all k classes so we have:

$$P(y = k|x) \propto P(x|y = k)P(y) \Leftrightarrow \arg\max_k P(y = k|x) = \arg\max_k P(x|y = k).P(y)$$

On the other hand, with the Discriminative model, there are two types of model. First is using Probability to compute the $P(y = k|x)$ directly from the data without using Bayes rule such as **Logistic Regression**. Other is to find the decision boundary between classes directly. We can list here some examples are **Perceptron**, **Support Vector Machine** (SVM) and **Decision Tree**.

One important point to mention here is that all the models, which use Probability to calculate the $P(y = k|x)$, form a group called **Probabilistic Graphical Model**. Besides, the important characteristic among all probabilistic models is likelihood. That is the relation between the given data D and the parameter w that we want to estimate from that data.

$$P(w|D) = \frac{P(D|w)P(w)}{P(D)}$$

Because $P(D)$ are the same with all w so we can omit P(D), so:

$$\propto P(D|w)P(w) = P(x, y|w)P(w) = P(y|x, w)P(x, w)P(w)$$

Because $P(x, w)$ can be dropped because it does not depend on w or w and x are independent, so:

$$\propto P(y|x, w)P(w)$$

So we have:

$$\arg\max_w P(w|D) = \arg\max_w P(y|x, w)P(w)$$

Notice that $P(w|D)$ is **Posterior**, $P(y|x, w)$ is **Likelihood**, we can think it is a function with parameter w, so we can write in form of $P(y|x)$; and $P(w)$ is **Prior**. With Generative Model, we have $P(y|x) = \frac{P(x,y)}{P(x)} \propto P(x, y)$, we calculate $P(y|x)$ by using Bayes rules as mentioned above. With Discriminative Model, we calculate $P(y|x)$ directly from the data.

3.2 Perceptron

The most basic model for classification problems is Perceptron. It is a linear classifier and just use a linear boundary to decide a point is belong what class. It is quite simple with the idea is that if the classification is wrong, it will change the decision boundary to fit with the new example.

With Perceptron model, we have:

$$y = sign(w_1x_1 + w_2x_2 + \ldots + w_nx_n) = sign(w^Tx)$$

We have to define the Loss function of Perceptron as follow:

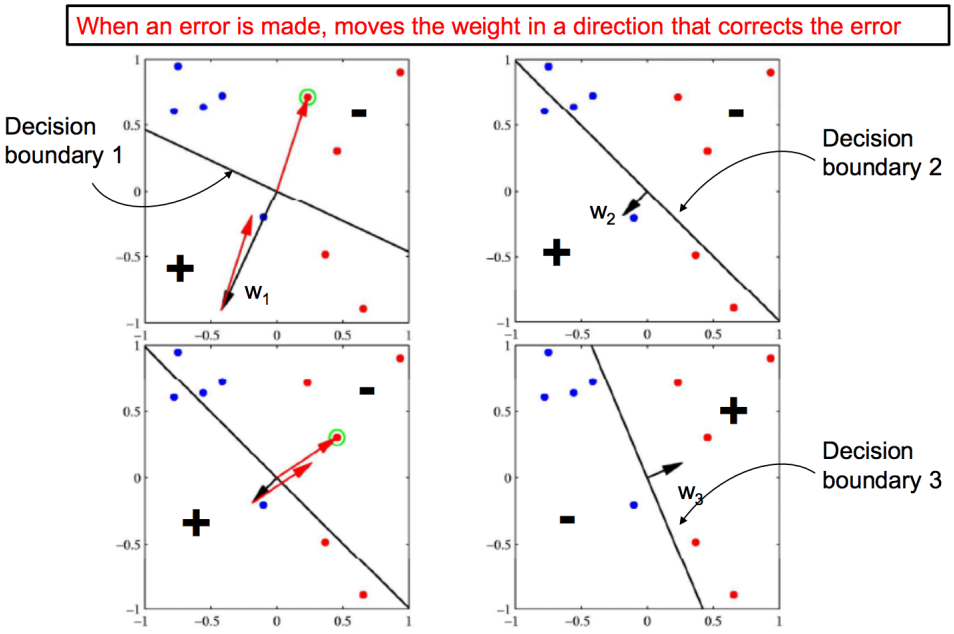
$$J(w) = \frac{1}{n}\sum_{m=1}^{n} max(0, -y_m w^T x_m)$$

$$J_m(w) = max(0, -y_m w^T x_m)$$

If we predict correctly, $-y_m w^T x_m < 0$, so there is no loss. Otherwise, when we predict wrong, $-y_m w^T x_m > 0$. With the same method as Linear Regression, we use Gradient Descent to estimate w to minimize the Loss Function above:

$$\nabla J(w) = \begin{cases} 0, & \text{if we predict correctly} \\ -y_m x_m, & \text{otherwise} \end{cases}$$

Notice that you can use both Batch Gradient Descent and Stochastic Gradient Descent for particular purposes. The boundary is characterized by normal vector $w =< w_1, w_2, w_3, \ldots, w_n >$. If the Perceptron model misclassifies any example, the new normal vector will be updated with the value: $w = w + y_m x_m$. We can see an example in the following figure



So we complete the Perceptron model, the simplest model for classification here. Next we discuss the Logistic Regression.

3.3 Logistic Regression

Logistic Regression is the most basic discriminative model that is simple but strong enough to build larger model like Neural Network. Logistic Regression is much like linear regression with minor exception we will discuss later. As we say above Logistic Regression is one of probabilistic graphical model so we have the likelihood as follow, remember we calculate $P(y|x)$ directly from the data:

$$h(x) = P(y = k|x) = \frac{1}{1 + e^{-w^T x}}$$

With the notice that function $h(x)$ above is called **Sigmoid** function which has value in range $(0, 1)$ from the value of $w^T x$ in range $(-\infty, \infty)$. Our job is to find w such that it maximize the term likelihood above. To do that, we have the reversed method of Gradient Descent called **Gradient Ascent**. That maximize the objective function, in this case is the likelihood above. Also, the process of maximize the likelihood is called **Maximum Likelihood Estimation**. With the same process with Gradient Descent, we first get the derivative of the objective function and then using iterative to update the w and after a number of iterative, the Algorithm will converge to get a optimal w.

In more detail, we have the likelihood function of Logistic regression can be written in compact form as $P(y_i|x_i) = \hat{y}_i^{y_i}(1-\hat{y}_i)^{(1-y_i)}$ where $\hat{y}_i = \frac{1}{1+e^{-w^T x_i}}$. Remember that each training example (x_i, y_i) is drawn Independent and identically distributed (**IID**) and we can use log of likelihood function, called **Log likelihood function** above for easy calculation: $l(w) = \sum_i \log P(y_i|x_i)$ We use Gradient Ascent and get:

$$\nabla l(w) = \sum_{i=1}^{N}(y_i - \hat{y}_i)x_i$$

With new example, we first compute the sigmoid function of $P(y = k|x_{new})$ we classify new example belong to class k if the value of sigmoid function is greater than the **Threshold** θ, otherwise it belongs to another. We can also use Logistic Regression for Multi-class Classification. We have 2 approaches: one vs. rest and pairwise. We set up the posterior probability using a so-called **soft-max** or **normalized exponential function**

$$P(y = k|x) = \hat{y}_k = \frac{e^{w_k^T x}}{\sum_{j=1}^{K} e^{w_j^T x}}$$

We will get the: $\nabla l(w) = \sum_{i=1}^{N}(y_k^i - \hat{y}_k^i)x^i$ where $y_k^i = 1$ if $y^i = k$, and 0 otherwise.

In the Linear Regression, we use Regularization term called $\lambda \sum_{j=0}^{M} |w_j|^q$. So I will talk about it in more detail. First, from the Discriminative part in above section, I have discussed the Posterior $P(w|D)$ and Prior $P(w)$. If w is a constant, we have $P(w)$ is a constant to. However, in case of w is a random variable, we have $P(w)$ is a prior distribution, which can can get advantage of knowing a prior knowledge about the w for less complex calculation. For example, in Logistic Regression, we can assume that $P(w)$ is a **Normal Distribution** $N(0, \sigma^2)$. We called this method is **Maximum A Posterior** (MAP). Large weight for prior values correspond to more complex hypothesis, so this prior prefer simpler hypothesis. Thus we have new log likelihood function for Logistic Regression:

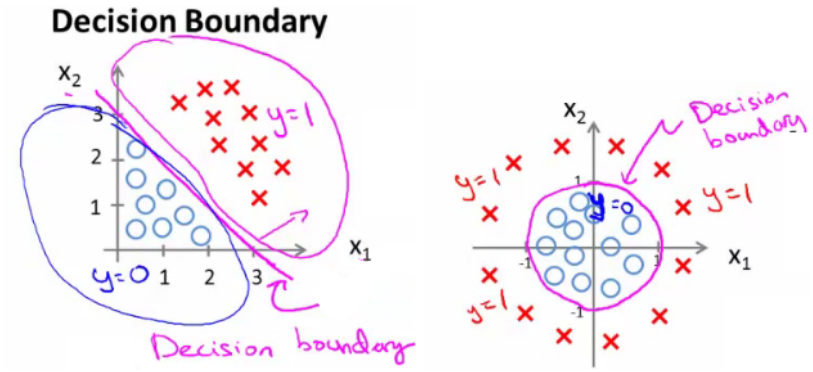
$$\arg\max_w \sum_j logP(y_j|x_j, w) - \frac{\lambda}{2}\sum_i w_i^2$$

The regularization term here is $-\frac{\lambda}{2}\sum_i w_i^2$ where $\lambda = \frac{1}{\sigma^2}$and the new Gradient is

$$\nabla L(w) = \sum_{i=1}^{N}(y_i - \hat{y}_i)x_i - \lambda w$$

We again, can use the value of λ to control the complexity of model. If λ is small, it is equivalent to MLE, we do not make use of prior knowledge, lead to over-fitting. If λ is large, it ignores training data, just based on the prior knowledge. We want to find the best value of λ to both make use of prior knowledge and learning data.

Last but not least, because Logistic regression is a discriminative model, we talk about its decision boundary. At first glance, we saw the sigmoid function is not linear, so we conclude that the decision boundary. It completely wrong, the term $w^T x$ in Sigmoid function is linear function so it also has linear decision boundary. We decide 1 if $P(y = 1|x) > P(y = 0|x) \Rightarrow \frac{P(y=1|x)}{P(y=0|x)} > 1 \Rightarrow \log \frac{P(y=1|x)}{P(y=0|x)>0} \Rightarrow w^T x > 0$. It separate the space into 2 parts which have $w^T x > 0$ and $w^T x < 0$ with a decision line is $w^T x = 0$. However, with non-linear sigmoid function, for example we have: $g(x) = w_0 + w_1 x_1 + w_2 x_1^2 + w_3 x_2^2$. x_1, x_2 have circular relation, so we have Decision boundary is a circle (center at $(0, 0)$ and a radius of 1) which divides the space into 2 parts, outside the circle ($g(x) \geq 0$) we predict 1 and inside the circle ($g(x) < 0$) we predict 0. The following figure will give more detail.



To this point, we complete the Logistic Regression here. Next, we move on a example of Generative Model, that is Naive Bayes model which is the simplest Generative Model.

3.4 Naive Bayes

With Naive Bayes, we should use the assumption of conditional independence (naive) between different features x_i given class y. Naive Bayes model is usually used in Text Retrieval and Text Classification problem, the problem of judging documents as belonging to one category or the other. As talking about the Likelihood of Generative model, in general form, Naive Bayes can be written as:

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

Because $P(x)$ is the same in all different classes, so we have: $P(y|x) \propto P(x|y)P(y)$. As we say earlier, we assumed that different features are independent given the class so we have

$$P(y|x) \propto \prod_{i=1}^{M} P(x_i|y)P(y)$$

where M is the number of features. To exactly compute $P(y|x)$, we have to compute the term $P(x)$ as follow:

$$P(x) = \sum_{k=1}^{K} \prod_{i=1}^{M} P(x_i|y=k)P(y=k)$$

where K is the number of classes. Again, we have the product function, instead of maximizing the likelihood directly, we can use log-likelihood for easier calculation. So we have:

$$\log P(y|x) \propto \sum_{i=1}^{M} \log P(x_i|y)P(y)$$

Thus we have:

$$\hat{y} = \arg\max_{y} P(y) \sum_{i=1}^{M} \log P(x_i|y)$$

With this conditional independent assumption, we can reduce the number of parameters for $P(x|y)$ from $k(2^d - 1)$ to kd parameters where k is the number of classes and d is the number of features. It removes the need for memorization and significantly reduces over-fitting. In addition to parameter for representing $P(y)$, we have total number of parameters for Naive Bayes model is $k(d-1) + (k-1)$. To this point, all the things we have to do is compute each $P(x_i|y)$ correctly. Notice that y is Discrete variable and we can easily compute $P(y)$ from the training set (i.e., (prior for a given class) = (number of samples in the class) / (total number of samples)). So depends on the value of x, we have different distribution of $P(x_i|y)$. With x is continuous, we have one popular distribution is **Gaussian Naive Bayes** (GNB), and when x is discrete, we have 2 popular distributions is **Bernoulli Naive Bayes** (BNB) and **Multinomial Naive Bayes**.

With Gaussian Naive Bayes, we assume that the continuous values associated with each class are distributed according to a Gaussian distribution. For example, suppose the training data contain a continuous attribute, x. We first segment the data by the class, and then compute the mean and variance of x in each class. Let μ_k be the mean of the values in x associated with class k, and let σ_k^2 be the variance of the values in x associated with class k. Then, the probability distribution of some value v given a class, $p(x_i = v|y = k)$, can be computed by plugging v into the equation for a Normal distribution parameterized by μ_c and σ_c^2. That is,

$$p(x_i = v|y = k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

Next, we discuss Bernoulli Naive Bayes. Instead of being continuous, in this case, x is binary value which only have value 0 and 1. $x_i = 1$ represents the occurrence of the word i^{th} in the vocabulary (for example in Document

Classification task), and vice versa $x_i = 0$ represents the absence of the word i^{th} in the vocabulary. So the likelihood of the document given the class k is:

$$P(x_i|y=k) = p_{ki}^{x_i}(1-p_{ki})^{(1-x_i)} \Leftrightarrow logP(x_i|y=k) = x_i logp_{ki} + (1-x_i)log(1-p_{ki})$$

where p_{ki} is the probability of class k generating the term w_i. We use MLE for Bernoulli to estimate p_{ki} as follow:

$$p_{ki} = \frac{\text{number of documents contain word } i \text{ in class k}}{\text{total number of documents in class k}}$$

Then, we talk about the Multinomial Naive Bayes. x is also discrete but x_i now represents for number of occurrence of word i^{th} in the document. so we have the likelihood of the document given the class k as follow:

$$P(x_i|y=k) = p_{ki}^{x_i} \Leftrightarrow logP(x_i|y=k) = x_i logp_{ki}$$

In this case, we have the likelihood

$$\begin{aligned}\log P(y=k|x) &\propto \log\left(P(y=k)\prod_{i=1}^{n} {p_{ki}}^{x_i}\right) \\ &= \log P(y=k) + \sum_{i=1}^{n} x_i \cdot \log p_{ki} \\ &= b + \mathbf{w}_k^\top \mathbf{x}\end{aligned}$$

where $b = \log p(y=k)$ and $w_{ki} = \log p_{ki}$. So we can see that Naive Bayes with Discrete value of x becomes a linear classifier like Logistic Regression in log-space. Again, we use MLE for Multinomial to estimate the value of p_{ki}, which lead to:

$$p_{ki} = \frac{\text{number of word } i \text{ in class k}}{\text{total number of words in class k}}$$

When we use MLE for estimating discrete data of Naive Bayes, we have to deal with the problem is that with some rare words say "Mahalanobis", when compute the $P(x_i|y=k)$, $p_{ki} = 0$, that leads to whole $P(y=k|x) = 0$, it seems like the class k, which does not have the word "Mahalanobis" in the training set, is the not the class of given test example although other words in test example all belongs to this k class. To deal with that problem, we have to use the MAP (Maximum a Posterior) as mentioned above. We introduce the **Conjugate Prior Distribution** of the base distribution. In particular, the conjugate prior distribution of Bernoulli distribution is Beta distribution and Conjugate prior distribution of Multinomial distribution is Dirichlet distribution. The definition of conjugate prior can be seen more in Wikipedia [4].

For Bernoulli NB, we estimate the optimize $w = \frac{n_i+\alpha-1}{n+\alpha+\beta-2}$ where n_i, n, α, β are the number of documents contain word i^{th} of class k, number of documents of class k and 2 parameters of Beta distribution respectively. We usually choose

[4] https://en.wikipedia.org/wiki/Conjugate_prior

$\alpha = \beta = 2$, and $w_{MAP} = \frac{n_i+1}{n+2}$ and we call this is Laplace Smoothing. For Multinomial NB, with K outcomes, a Dirichlet distribution has K parameters, each serves a similar purpose as Beta distribution's parameters. **Laplace Smoothing** in this case:

$$P_{ki} = \frac{n_k + 1}{n + K}$$

In case of document of classification, $p_{ki} = \frac{n_i+1}{n+V}$, where n_i is the number of word i in class k, n is the total number of words in class k and V is the size of vocabulary. Furthermore, we can use parameter α to control this smoothing as follow:

$$p_{ki} = \frac{n_i + \alpha}{n + \alpha V}$$

when $\alpha = 0$ this smoothing becomes MLE, when $\alpha = 1$, this is Laplace Smoothing. With different dataset, we have different optimal α and we have to use cross validation to choose the best *alpha* for each dataset.

That's all the things I want to discuss about the Naive Bayes. I move on to another Generative models called Linear Discriminant Analysis.

3.5 Gaussian Discrminant Analysis

Gaussian Discriminant Analysis or Linear Discriminant Analysis (LDA) is a generative model (it is confused with its name, huh?), in which we can learn the model's parameters by maximizing the joint likelihood $P(x, y) = P(x|y)P(y)$. Then we predict the class for giving testing example by:

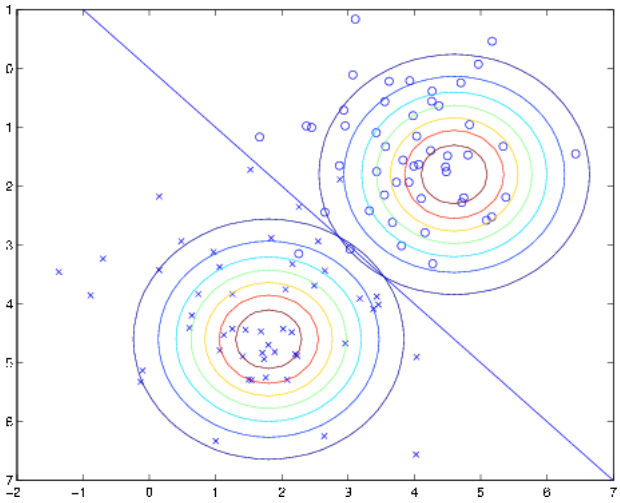
$$\arg\max_y P(y|x) = \arg\max_y \frac{P(x|y)P(y)}{P(x)} = \arg\max_y P(x|y)P(y)$$

In LDA, in basic set up, y has two values, which means 2 classes. We assume that each class draws from normal distribution:

$$P(x|y = 0) \sim N(\mu_0, \Sigma)$$

$$P(x|y = 1) \sim N(\mu_1, \Sigma)$$

Note that μ_0, μ_1 are 2 different means of 2 normal distributions, Σ is shared covariance matrix.



I will recall some knowledge about Multivariate Gaussian as follow:

$$\mathbf{x} \in R^d \text{and } \mathbf{x} \sim N(\mu, \mathbf{\Sigma})$$

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} \exp -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})$$

$\boldsymbol{\mu} = \mathbb{E}(\mathbf{x})$: the mean vector

$\mathbf{\Sigma} = \mathbb{E}[(x-\boldsymbol{\mu})(x-\boldsymbol{\mu})^T]$: the covariance matrix

With 2 classes, we have the following log-likelihood of LDA given a set of training data, with $y \sim Bernoulli(\psi)$

$$l(\psi, \mu_0, \mu_1, \Sigma) = \log \prod_{i=1}^{N} P(\mathbf{x_i}, y_i) = \sum_{i=1}^{N} \log P(x_i|y_i)P(y_i)$$
$$= \sum_{y_i=1} \log P(x_i|y_i)P(y_i) + \sum_{y_i=0} \log P(x_i|y_i)P(y_i)$$
$$= \sum_{y_i=1} \log \frac{\psi}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} \exp -\frac{1}{2}(x_i-\mu_1)^T\Sigma^{-1}(x_i-\mu_1)$$
$$+ \sum_{y_i=0} \log \frac{(1-\psi)}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} \exp -\frac{1}{2}(x_i-\mu_0)^T\Sigma^{-1}(x_i-\mu_1)$$

Like Naive Bayes, we also use MLE to estimate the parameters, and get:

$$\psi = \frac{\sum_{i=1}^{N} I(y_i == 1)}{N}$$

The portion of class-1 examples in the training data

N_1 and N_2: # of class-1 and class-0 examples in the training data

$$\mu_1 = \frac{1}{N_1} \sum_{y_i=1} \mathbf{x}_i$$

The mean of all class-1 examples

$$\mu_0 = \frac{1}{N_2} \sum_{y_i=0} \mathbf{x}_i$$

The mean of all class-0 examples

$$\Sigma = \frac{N_1}{N} \sum_{y_i=1} (\mathbf{x}_i - \mu_1)(\mathbf{x}_i - \mu_1)^T + \frac{N_2}{N} \sum_{y_i=0} (\mathbf{x}_i - \mu_0)(\mathbf{x}_i - \mu_0)^T$$

Next, we talk about the linear decision boundary of LDA. We will predict $y = 1$ if:

$$\frac{P(x|y=1)P(y=1)}{P(x|y=0)P(y=0)} > 1$$

Equivalently:

$$log \frac{P(x|y=1)P(y=1)}{P(x|y=0)P(y=0)} > 0 = w^T x + w_0 > 0$$

Besides, we have:

$$\log \frac{P(\mathbf{x}|y=1)P(y=1)}{P(\mathbf{x}|y=0)P(y=0)} = \underbrace{(\mu_1 - \mu_0)^T \Sigma^{-1} \mathrm{x}}_{\mathbf{w}^T} \underbrace{- \frac{1}{2} \mu_1{}^T \Sigma^{-1} \mu_1 + \frac{1}{2} \mu_0{}^T \Sigma^{-1} \mu_0 + \log \frac{\psi}{1-\psi}}_{w_0}$$

We can see that, it looks like decision boundary of logistic regression. In Logistic Regression, we conclude that the decision boundary of linear sigmoid function of logistic regression is a linear boundary. We also have the observation is that LDA makes stronger and more restrictive assumption. If we don't know if the data is followed Gaussian distribution, logistic regression will be more robust. Otherwise, LDA is preferred.

We complete LDA, and also complete the first section of series "Introduction to Machine Learning". In the next section, we talk about very famous Machine Learning models, they are K-Nearest Neighbors, Support Vector Machine (SVM), Decision Tree and Neural Network. All of them can be used for Regression problem as well. But I don't cover in this introduction.

4 Both Regression and Algorithms

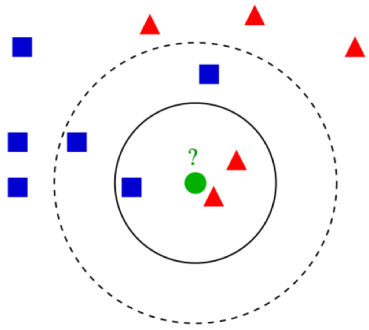
4.1 K Nearest Neighbors

After discuss 2 Generative models those are Naive Bayes and LDA, we come back to a famous Discriminative model kNN. It is a very simple Machine Learning

model used for classification and regression task. I extract some information about it on Wikipedia as follow. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression.

- In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If $k = 1$, then the object is simply assigned to the class of that single nearest neighbor.
- In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms. Both for classification and regression, it can be useful to assign weight to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of $1/d$, where d is the distance to the neighbor. A commonly used distance metric for continuous variables is **Euclidean distance**. For discrete variables, such as for text classification, another metric can be used, such as the overlap metric (or **Hamming distance**)



The above figure is an example of k-NN classification. The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles. If $k = 3$ (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle. If $k = 5$ (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).

k in k-NN is hyper-parameter so the best choice of k depends upon the data; generally, larger values of k reduce the effect of noise on the classification but make boundaries between classes less distinct. A good k can be selected by various heuristic techniques like Cross-validation.

We complete the k-NN model here. In the next section, I will talk about Support Vector Machine (SVM) - a very popular machine learning algorithm.

4.2 Support Vector Machine

Support Vector Machine likes Perceptron in the point, it does not assume the data followed any distributions or relation like Logistic Regression, Naive Bayes and LDA. So it is a very robust method when applying for unknown distribution data. When we talk about SVM, we talk about 2 very important features of SVM, those are Margin and Kernel.

To begin, we talk about its **margin**. In other previous machine learning models, we draw a separate line between many classes. However, given a linearly separable 2 classes data, we can draw many separate lines that satisfy the constraints that linearly separate 2 classes. So which line do you choose? As natural, you will choose the one that have the largest distance to the closest elements of 2 classes. That is the idea behind the SVM. The closest elements are called **Support Vectors**, that's why SVM got that name. The whole problem of SVM is a convex optimization.

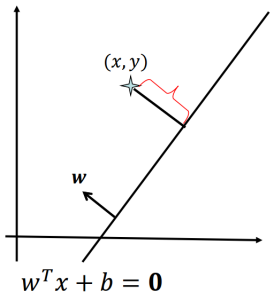
First, let's defined a **functional margin**. Let $d : w^T x + b = 0$ be a linear decision boundary. The functional margin for a point (x^i, y^i) is defined as:

$$y^i(w^T x^i + b)$$

For a fixed w and b, the larger functional value, the more confidence we have about the prediction. Next, we want to find w and b so we can relax to get a **possible goal**: find a set of w and b so that all training data points will have large (maximum) functional margin. However, we can arbitrarily change the functional margin without changing the boundary at all. So what we need here is geometric margin as follow:

$$\frac{y^i(w^T x^i + b)}{||w||}$$

It measures the geometric distance between the point and the decision boundary. It can be either positive or negative: Positive if the point is correctly classified or Negative if the point is misclassified.



We can represent the constrained optimization as follow:

$$\max_{w,b} \gamma$$

$$s.t : \frac{y^i(w^T x^i + b)}{||w||} \geq \gamma, i = 1, ., N$$

We can transform this optimization to new form as follow:

$$\max_{w,b} \frac{1}{2}||w||^2$$

$$s.t : y^i(w^T x^i + b) \geq 1, i = 1, ., N$$

From this form, we can use **Quadratic Programming** (QP) with Lagrangian and solve the dual problem instead of primal (the form above).

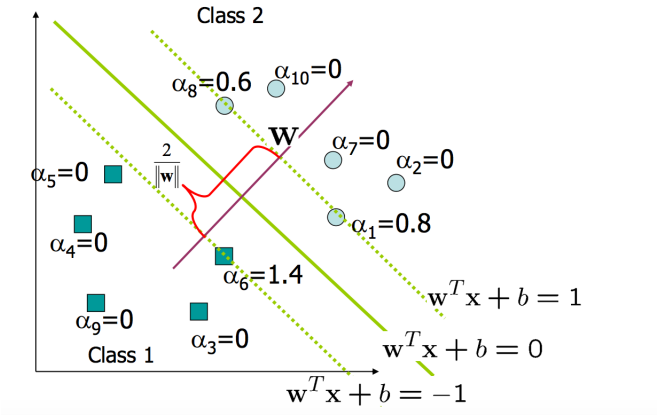
Let α be KTT multipliers, the previous constrained problem can be expressed as:

$$\arg\min_{\mathbf{w},b} \max_{\boldsymbol{\alpha}\geq 0} \left\{ \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i [y_i(\mathbf{w} \cdot \mathbf{x_i} - b) - 1] \right\}$$

And the solution can be expressed as a linear combination of the training vectors:

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

Only a few α_i will be greater than zero. The corresponding x_i are exactly the support vectors, which lie on the margin and satisfy $y_i(w.x_i - b) = 1$.



We can compute b:

$$b = \frac{1}{N_{SV}} \sum_{i=1}^{N_{SV}} w.x_i - y_i$$

We have the dual form: Maximize (in α_i)

$$\tilde{L}(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$$

subject to (for any $i = 1, \ldots, n$)

$$\alpha_i \geq 0$$

and to the constraint from the minimization in b

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

Here the kernel is defined by $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$.

W can be computed thanks to the α terms:

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

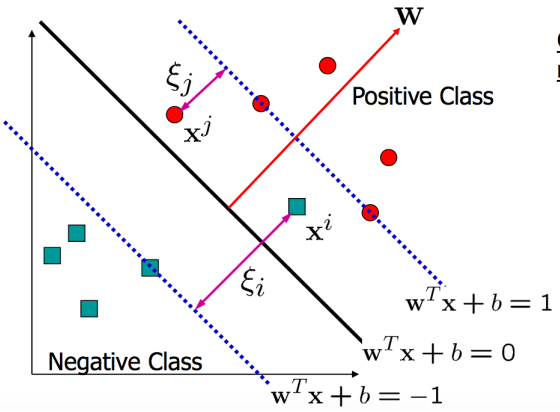
.

In practice, we can just regard the QP solver as a "black-box" without bothering how it works. For classifying a new input z, we compute:

$$A = w \cdot z + b = (\sum_{j=1}^{s} \alpha_{t_j} y^{t_j} x^{t_j}) \cdot z + b = \sum_{j=1}^{s} \alpha_{t_j} y^{t_j} (x^{t_j} \cdot z) + b$$

classify z as Positive if $A > 0$ and Negative otherwise. w need not be computed/stored explicitly, we can store the α_i's, and classify z by using the above calculation, which involves taking the dot product between training examples and the new example z.

Up to this point, we can let SVM solve the linearly separable data. However, how does SVM deal with noise? The real data must contain noise so using maximum margin SVM is not robust to noise. Thus, we come to the new type of SVM called **Soft Margin** SVM.



Originally functional margins need to satisfy:

$y^i(w \cdot x^i + b) \geq 1$

Now we allow it to be less than 1:

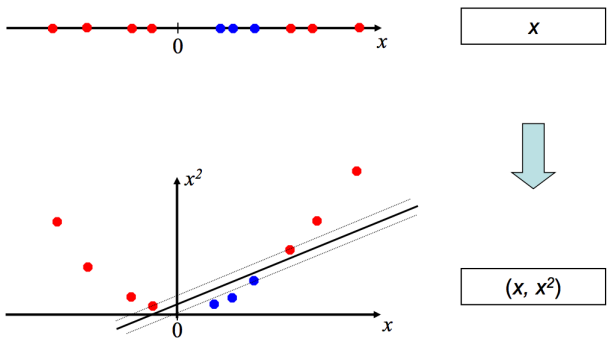
$y^i(w \cdot x^i + b) \geq 1 - \xi_i$ & $\xi_i \geq 0$

The objective changes to:

$$\min_{w,b} \|w\|^2 + c\sum_{i=1}^{N} \xi_i$$

We allow functional margins to be less than 1 (could even be ¡ 0). The ξ can be viewed as the "errors" of our fat decision boundary. We have a tradeoff between making the decision boundary fat and minimizing the error. Parameter c controls the tradeoff. Large c: ξ's incur a large penalty, so the optimal solution will try to avoid them, that means margin will be smaller than 1. Else, small c: small cost for ξ's, we can sacrifice some training examples to have a large classifier margin. c puts a box-constraints on α, weights of support vectors. It limits the influence of individual support vectors (maybe outliers). c is hyper-parameter to be set, so we can use cross-validation to do so.

And the next important features of SVM as I said above is **Kernel Function**. It helps SVM to deal with non-linearly separable data. I said this problem before in linear regression. With Linear Regression, we use high order polynomial to divide non-linearly separable data. However, with SVM, we use a different approach that call kernel trick. We map the input to higher dimensional space can solve the linearly inseparable cases.



A function $k(x_i, x_j)$ is called a kernel function if $k(x_i, x_j) =< \phi(x_i), \phi(x_j) >$ for some ϕ. For example, $K(a, b) = (a \cdot b + 1)^2$. This is equivalent to map to the quadratic space! In practice, we specify the kernel function without explicitly stating the transformation ϕ. Given a kernel function, finding its corresponding

transformation can be very cumbersome. A kernel function can be viewed as computing some similarity measure between objects. We can apply this kernel trick in SVM as follow:

Maximize (in α_i)

$$\tilde{L}(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$$

subject to (for any $i = 1, \ldots, n$)

$$0 \leq \alpha_i \leq C,$$

and

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

.

We have more common kernel functions:

- Linear kernel: $k(a, b) = a \cdot b$
- Polynomial kernel: $k(a, b) = (a \cdot b + 1)^d$
- Radial Basic Function kernel: $k(a, b) = \exp\left(\frac{(a-b)^2}{2\sigma^2}\right)$
- Closure probability of kernel: if K_1 and K_2 are kernel functions, then following are all kernel functions:
  - $K(x, y) = K_1(x, y) + K_2(x, y)$
  - $K(x, y) = aK_1(x, y)$
  - $K(x, y) = K_1(x, y)K_2(x, y)$

Here are some notes with kernel function:

- In practice, we often try different kernel functions and use cross-validation to choose
- Linear kernel, polynomial kernels (with low degrees) and RBF kernels are popular choices
- One can also construct a kernel using linear combinations of different kernels and learn the combination parameters (kernel learning)
- Selecting the kernel parameter and c is very strong impact on performance and often the optimal range is reasonably large

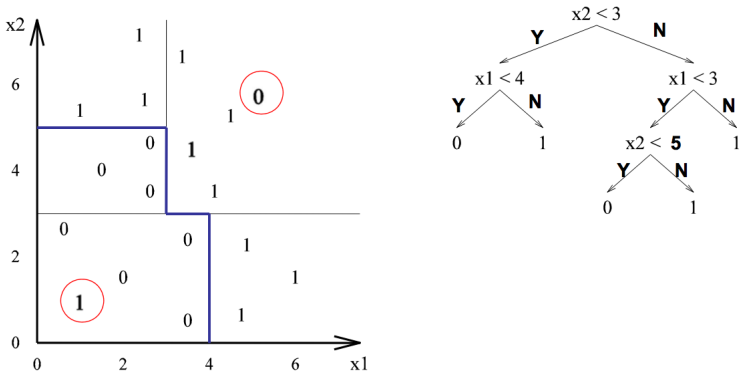
When comparing to Logistic Regression (another Discriminative Model), we have the following notices:

- If n (number of features) is large and m (number of training example) is small: we should use Logistic regression or SVM with linear kernel.
- If n is small and m is intermediate, we should use RBF.
- If n is small and m is large, SVM will be slow to run with RBF kernel. We could manually create or add more features and apply SVM with RBF kernel or use logistic regression of SVM with linear kernel.
- Logistic regression and SVM with a linear kernel are pretty similar. They do similar thing and get similar performance.

That completes all basic information about SVM. Next we move on another Discriminative model called Decision Tree.

4.3 Decision Tree

Decision trees have many appealing properties. They are similar to the human decision process, and easy to understand. They deal with both discrete and continuous features. With highly flexible hypothesis space, as the number of nodes (or depth) of the tree increase, decision tree can represent increasingly complex decision boundaries as the following figure.



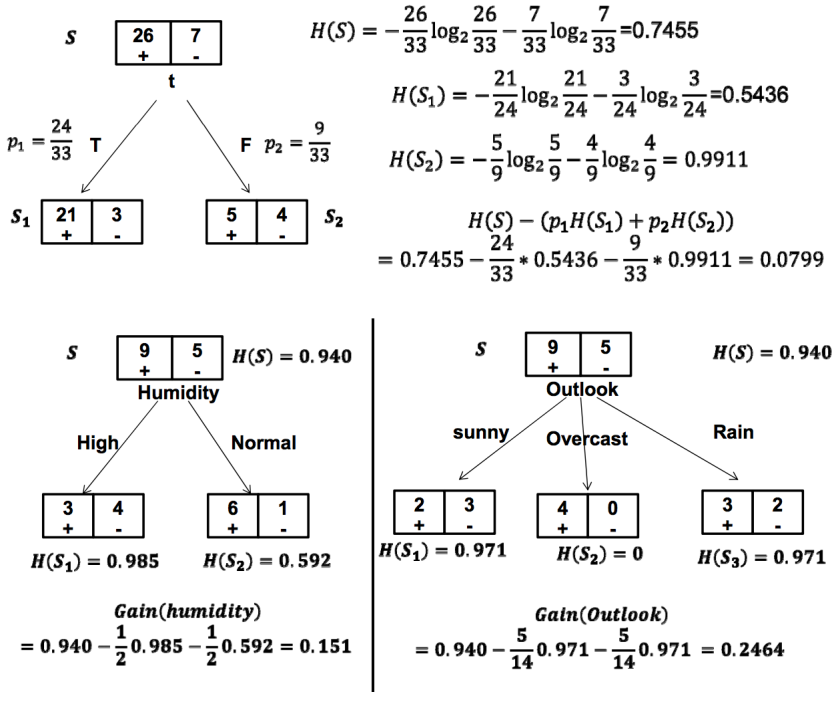
Like SVM, we have to set up a **possible goal** for Decision tree is: finding a decision tree h that achieves minimum error on training data. With that possible goal, we have the greedy algorithm for finding h. Instead of trying to optimize the whole tree together, we try to find one test at a time. We assume all features are discrete values. Remember that this algorithm is not guaranteed to find an optimal decision tree.

1. Choose the best attribute to test on at the root of the tree.
2. Create a descendant node for each possible outcome of the test
3. Training examples in training set S are sent to the appropriate descendent node

4. Recursively apply the algorithm at each descendant node to select the best attribute to test using its associated training examples. If all examples in a node belong to the same class, turn it into a leaf node, label with the majority class

The problem here is how to choose the best test to choose the best attribute. There are many different tests, at here, we choose **mutual information** (or **information gain** criterion) test to present because it is quite easy to understand and have good performance in general. The first time to know is **Entropy**. In information theory, entropy is the measure of uncertainty of a random variable. Given a set of training examples S and y denote the label of an example randomly draw from S. If all examples belong to one class, y has 0 entropy. If y takes positive and negative values with a 50

$$H(x) = \sum_{i=1}^{k} p_i \log_2 \frac{1}{p_i} = -\sum_{i=1}^{k} p_i \log_2 p_i$$



With the above example, we will choose to split the data with the feature "Outlook", which has higher information gain value. However, with this test, there is a problem that multi-nomial features (which has more than 2 possible values) will have higher information gain in general. This is called the bias, to

avoid this, we can rescale the information gain as follow:

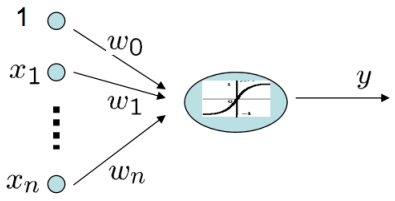
$$\arg\max_j \frac{H(y) - H(y|x_j)}{H(x_j)}$$

. Next, how we deal with continuous features. We test against a threshold θ_j for x_j. First we sort the examples according to x_j. Move the threshold θ from the smallest to the largest value. Select θ that gives the best information gain. Note that, we only need to compute information gain when class label changes. Decision tree has a very flexible hypothesis space. As the nodes increase, we can represent arbitrarily complex decision boundaries. This can lead to over-fitting (due to noise and outliers). To avoid Over-fitting, we can early stop, which means to stop growing the tree when data split does not offer large benefit. Or we can use post pruning. One thing to note here is Decision tree has a well-known implementation called **C4.5** by Ross Quinlan.

We complete all basic information about decision tree here. Next, we talk about the very famous model, and gain the current attention from many ML researchers around the world.

4.4 Neural Network

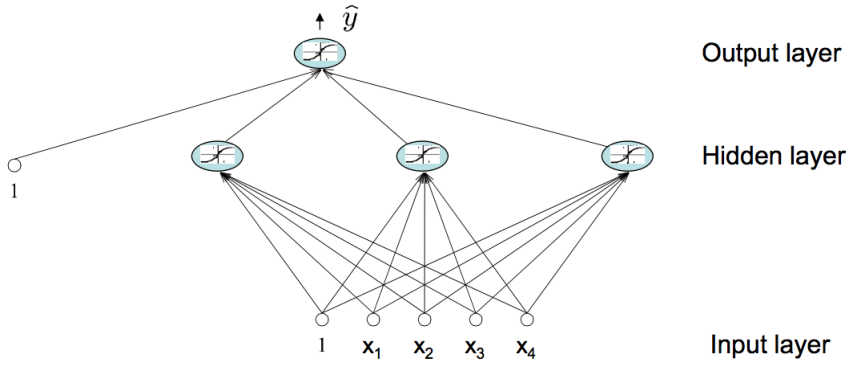
Neural Network is also known as **Artificial Neural Network** or ANN, it stimulates the activity of a neuron in biological neural networks. To study about ANN, we first talk about its smallest component called Neuron.



It receives n inputs (plus a bias term), then multiplies each input by its weight. Next it applies activation function to the sum of results and finally outputs result. **Activation function** controls whether a neuron is "active" or "inactive". There are several common activation functions. For example, **Threshold function** (outputs 1 when input is positive and 0 otherwise, similar to perceptron). Another activation function is **sigmoid function** that we use in Logistic Regression, this function has a good property for optimization is that it is differentiable.

$$\frac{1}{1+e^{-x}}$$

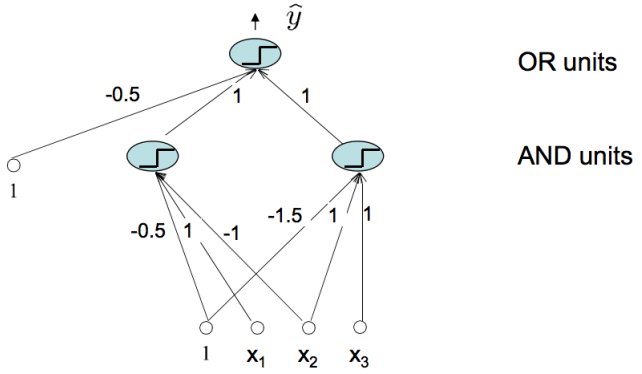
Next, we move on to basic multilayer Neural Network



In **Input layer**, the number of neurons comprising that layer is equal to the number of features (columns) in your data. Some NN configurations add one additional node for a bias term. Each **Hidden layer** receives its inputs from the previous layer and forwards its outputs to the next - feed forward structure. **Output layer**: sigmoid activation function for **classification**, and linear activation function for **regression**.

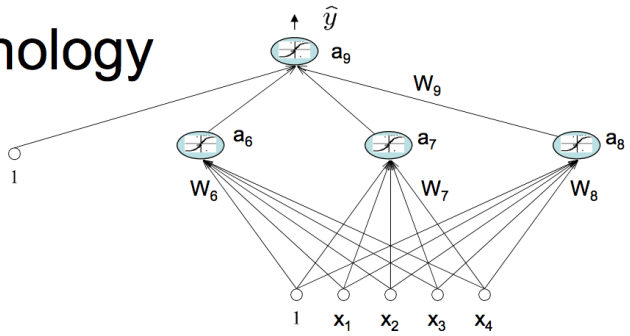
NN has a very powerful representational ability. With the combination of different weights of each input and sigmoid function, it can represent any Boolean Formula and arbitrary function.

$$(x_1 \wedge \neg x_2) \vee (x_2 \wedge x_3)$$



Before going to deeper the action of NN, I will discuss some terms used in NN:

Terminology



- $\mathbf{X} = [1, x_1, x_2, x_3, x_4]^T$ – the input vector with the bias term
- $A = [1, a_6, a_7, a_8]^T$ – the output of the hidden layer with the bias term
- $\boldsymbol{W}_i$ represents the weight vector leading to node i
- $w_{i,j}$ represents the weight connecting from the j-th node to the i-th node
  - $w_{9,6}$ is the weight connecting from a_6 to a_9
- We will use σ to represent the activation function, so

$$\hat{y} = \sigma(W_9 \cdot [1, a_6, a_7, a_8]^T) = \sigma(W_9 \cdot [1, \sigma(W_6 \cdot X), \sigma(W_7 \cdot X), \sigma(W_8 \cdot X)]^T)$$

NN will find the best $w_{i,j}$ for the whole network, we also use Gradient Descent as in Linear Regression. We have to minimize the mean squared error (MSE) on the training set.

$$J(W) = \frac{1}{2}\sum_{i=1}^{N}(\hat{y}^i - y^i)^2$$

$$J_i(W) = \frac{1}{2}(\hat{y}^i - y^i)^2$$

A useful fact: the derivative of the sigmoid activation function is

$$\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$$

After calculating all a_i from input layer to output layer to get the final $\hat{y}$. We next calculate the special term call "error" δ from output layer and push back to input layer. this process called **Back Propagation**. We calculate the "error" as follow. If current node is in output layer, for example node 9 in the above figure: $\delta_9^i = (\hat{y}^i - y^i)\hat{y}^i(1 - \hat{y}^i)$. If current node is in hidden layer, for example node 6 in the above figure, $\delta_6^i = \delta_9^i \cdot w_{9,6} \cdot a_6^i(1 - a_6^i)$. Or we can have general form:

$$\delta = \text{activation functon} \cdot \text{sum of signal with weights}$$

We have the following back propagation training:

- Initialize all the weights with small random values

- Repeat
  - For all training examples, do:
  - Begin Epoch
    - For each training example do
    - Compute the network output
    - Compute the error
    - Backpropagate this error from layer to layer and adjust weights to decrease this error
  - End Epoch

Put it altogether, we have the following back propagation:

- **Forward Pass** Given x, compute a_u and $\hat{y}_v$ for hidden units u and output units v.
- **Compute Errors** Compute $\epsilon_v = (\hat{y}_v - y_v)$ for each output unit v
- **Compute Output Deltas**

$$\delta_v = (\hat{y}_v - y_v)\,\hat{y}_v(1 - \hat{y}_v)$$

- **Compute Hidden Deltas**

$$\delta_u = a_u(1 - a_u)\Sigma_v w_{v,u}\delta_v$$

- **Compute Gradient**
  - Compute $\frac{\partial J_i}{\partial w_{v,u}} = \delta_v a_u^i$ for hidden-to-output weights.
  - Compute $\frac{\partial J_i}{\partial w_{u,j}} = \delta_u x_j^i$ for input-to-hidden weights.

  Backpropagate error from layer to layer

- **Take Gradient Descent Step**

$$w_{v,u} \leftarrow w_{v,u} - \eta \frac{\partial J_i}{\partial w_{v,u}} \qquad w_{u,j} \leftarrow w_{u,j} - \eta \frac{\partial J_i}{\partial w_{u,j}}$$

For Batch Gradient Descent, we get the $\sum_{i=1}^{N} \partial_W J_i(W)$ for each example i. Then take a gradient descent step. With online or stochastic Gradient Descent, we take a gradient descent step with $\partial_W J_i(W)$ as it is computed in above algorithm. Some important notice on training:

- No guarantee of convergence, may oscillate or reach local minima.
- In practice, many large networks can be adequately trained on large amounts of data for realistic problems
- Many epochs (thousands) may be needed for adequate training, large data sets may require hours or days of CPU time.
- Termination criteria can be: Fixed number of epochs, Threshold on training set error, or Increased error on a validation set.

Notes on Proper Initialization:

- Start in the "linear" regions
  - keep all weights near zeros, so tht all sigmoid unites are thir linear regions. this makes the whole net the equivalent of one linear threshold unit - a relatively simple function.
  - This will also avoid having very small graident
- Break symmetry
  - If we start with all weights equal, what would happen?
  - Ensure that each hidden unit has different input weights so that the hidden units move in different directions.
- Set each weight to a random number in the range

$$[-1, +1] \times \frac{1}{\sqrt{fan - in}}$$

  where the "fan-in" of weight $w_{v,u}$ is the number of inputs to unit v

Next, we talk about some problems with NN. Over-training prevention and over-fitting prevention

- Running to many epochs may overtrain the network and result in overfitting.
- Keep a validation set and test accuracy after every epoch. Maintain weights for best performing network ont he validation set and return it when performance decreases significantly beyound this.
- To avoid losing traiing data to validation:
  - Use 10-fold cross-validation to determine the average number of epochs that optimizes validation performance.
  - Train on the full data set using this many epochs to produce the final result.
- Also, too few hidden units prevent the system from adequately fitting the data and learning the concept.
- Too many hidden units leads to over-fitting.
- Similar cross-validation method can be used to decid an appropriate number of hidden units.
- Another approach to preventing over-fitting is weight decay, in which we multiply all weights by some fraction between 0 and 1 after each epoch.
  - Encourages smaller weights and less complex hypotheses.

– Equivalent to including an additive penalty in the error function proportional to the sum of squares of the weights of the network.

Special Notice on Input and Output:

- Appropriate coding of inputs/outputs can make learning easier and improve generalization.
- Best to encode discrete multi-category features using multiple input units and include one binary unit per value
- Continuous inputs can be handled by a single input unit, but scaling them between 0 and 1
- For classification problems, best to have one output unit per class. Continuous output values then represent certainty in various classes. Assign test instances to the class with the highest output.
- Use target values of 0.9 and 0.1 for binary problems rather than forcing weights to grow large enough to closely approximate 0/1 outputs.
- Continuous outputs (regression) can also be handled by scaling to the range between 0 and 1

With a basic neural network, we can build up another powerful network called deep learning that gain the recent attention of ML researchers. I will not discuss deep learning in these series. I will try my best to cover this knowledge later. Next, I will discuss about Ensemble methods to complete the Supervised Learning at here.

4.5 Ensemble Method

Ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms. We use different learning sets and/or learning algorithms to make better performance. There have been a wide range of methods developed. We will discuss some popular approaches: **bagging** (and **Random Forest**, a variant that builds de-correlated trees) and **boosting**. Both methods take a single (base) learning algorithm (learner) and generate ensembles.

4.5.1 Bagging

Given training set S, bagging works as follows:

1. Create T bootstrap samples $S_1, \ldots, S_T$ of S as follows:
    - For each S_i: Randomly drawing $|S|$ examples from S with **replacement**
    - Note: with large $|S|$, each S_i will contain $1 - \frac{1}{e} \approx 63.2\%$ unique examples

2. For each $i = 1, \ldots, T$ compute $h_i = Learn(S_i)$

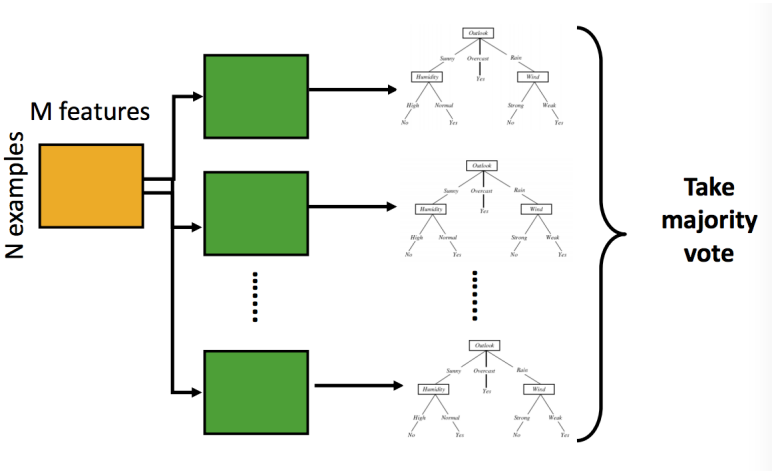
3. Output $H =< h_1, \ldots, h_T, \text{majority Vote} >$

Next we discuss the stability of Learner. A learning algorithm is **unstable** if small changes in the training data can produce large changes in the output hypothesis or **high variance**, otherwise stable. Bagging will have little benefit when used with stable learning algorithms (i.e, most ensemble members will be very similar). Bagging generally works best when used with unstable yet relatively accurate base learners (or high variance and low bias classifiers). We usually use Bagging with Decision Tree because it is a high variance algorithm, especially Decision Stump (a tree with a singular node). Another example of high variance learner is high order polynomial linear regression....

4.5.2 Random Forest

Random Forest is an extension of bagging. It builds an ensemble of de-correlated decision trees. It is one of the most successful classifiers in current practice because it is very fast, easy to train and there are many good implements available. Each bootstrapped sample is used to build a tree. When building the tree, each node only chooses from $m < M$ randomly sampled **features**. In other words, it combines "bagging" idea and random selection of features. **Gini index** is used to select the test just like in **C4.5**.

To read more about Random Forest, you can look at http://www.stat.stanford.edu/~hastie/Papers/ESLII.pdf. There is also available package at here: http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm



4.5.3 Boosting

With bagging: individual classifiers were independently learned. However, with Boosting, it looks at errors from previous classifiers to decide what to focus on for

the next iteration over data. A successive classifier depends on its predecessors. Thus, it puts more weights on 'hard' examples. One popular boosting algorithm that shows highly effectiveness (very often they outperform ensembles produced by bagging) is **AdaBoost**.

AdaBoost works by invoking Learn many times on different distributions over the training data set. So we need to modify base learner protocol to accept a training set distribution as an input. It indicates the base learner the importance of correctly classifying the i'th training instance. AdaBoost performs L boosting rounds, the operations in each boosting round l are:

1. Call Learn on data set S with distribution D_l to produce l^{th} ensemble member h_l, where D_l is the distribution of round l.
2. Compute the $(l+1)^{th}$ round distribution D_{l+1} by putting more weight on instances that h_l makes mistakes on.
3. Compute a voting weight α_l for h_l.
4. Output the ensemble hypothesis is: $H =< h_1, \ldots, h_L, \text{weighted Vote}(\alpha_1, \ldots, \alpha_L) >$

We have the following detailed AdaBoost algorithm:

AdaBoost algorithm:

Input: *Learn* - Base learning algorithm.
S - Set of N labeled training instances.

Output: $H = \langle \{h_1, \ldots, h_L\}, WeightedVote(\alpha_1, \ldots, \alpha_L) \rangle$

Initialize $D_1(i) = 1/N$, for all i from 1 to N. (uniform distribution)
FOR $l = 1, 2, \ldots, L$ **DO**
$h_l = Learn(S, D_l)$
$\varepsilon_l = error(h_l, S, D_l)$

$$\alpha_l = \frac{1}{2}\ln\left(\frac{1-\varepsilon_l}{\varepsilon_l}\right) \quad ;; \text{ if } \varepsilon_l < 0.5 \text{ implies } \alpha_l > 0$$

$$D_{l+1}(i) = D_l(i) \times \begin{cases} e^{\alpha_l}, & h_l(x_i) \neq y_i \\ e^{-\alpha_l}, & h_l(x_i) = y_i \end{cases} \quad \text{for } i \text{ from 1 to } N$$

Normalize D_{l+1} ;; can show that h_l has 0.5 error on D_{l+1}

Note that $\varepsilon_l < 0.5$ implies $\alpha_l > 0$ so weight is decreased for instances h_l predicts correctly and increases for incorrect instances

It is often straightforward to convert a base learner to take into account an input distribution D. When it's not straightforward, we can resample the training data according to D. Here are some interesting facts about Boosting. Training error goes to zero exponential fast. Boosting drives training error to zero, but it will not overfit because boosting is often robust to overfitting (but not always). Test error continues to decrease even after training error goes to zero because Adaboost adds more classifiers into our ensemble, the

training examples are getting larger margin (more confidence), thus improving the performance of the ensemble on the test data. In the order hand, boosting also has some pitfalls. It is sensitive to noise and outliers. If the number of outliers is small number, they can help to identify them. However, too many outliers can degrade classification performance (boosting might end up putting more and more weight on noise examples) and dramatically increase the time to converge. This phenomenon does not happen in Bagging because the noise does not get amplified and it could even be removed in boost-strap sampling procedure.

We conclude the Ensemble method with the comparing properties between Bagging and Boosting:

- Bagging
  - Resample data points
  - Weight of each classifier is the same
  - Only variance reduction
  - Robust to noise and outliers
- Boosting
  - Reweight data points (modify data distribution)
  - Weight of classifier vary depending on accuracy
  - Reduces both bias and variance
  - Can hurt performance with noise and outliers

To this point, we complete all discussions about ensemble method as well as classification problem and supervised learning. In the next section, I will talk about clustering problem.

5 Clustering

5.1 Introduction

In unsupervised learning, there are many tasks like grouping of clusters in the data, low dimensional... And the most important form in Unsupervised Learning is Clustering. Clustering is the process of grouping a set of objects into classes of similar objects with high intra-class similarity and low inter-class similarity. For example, find genes that are similar in their functions, group documents based on topics and so on. An important aspect in clustering is how we estimate the "similarity/distance" and what types of clustering.

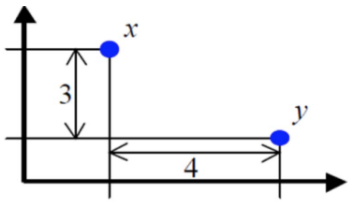
First, the similarity is a philosophical question, so it depends on representation and algorithm. For many algorithms, we usually use the term of distance (rather than similarity) between vectors. To measure distance, there are many ways, one way is using Minkowski Metric:

- Suppose we have two objects x and y y both have d features: $x = (x_1, \ldots, x_d), y = (y_1, \ldots, y_d)$

- The Minkowski metric of order r is defined by

$$d(x, y) = \sqrt[r]{(\sum_i |x_i - y_i|^r)}$$

- Common Minskowski metrics:
  - Euclidean ($r = 2$): $d(x, y) = \sqrt{(\sum_i (x_i - y_i)^2)}$
  - Manhattan distance ($r = 1$): $d(x, y) = \sum_i |x_i - y_i|$, also called L_1 distance
  - "Sup" distance ($r = \infty$): $d(x, y) = \max_i |x_i - y_i|$, also called L_∞ distance



1: Euclidean distance: $\sqrt[2]{4^2+3^2}=5$.
2: Manhattan distance: $4+3=7$.
3: "sup" distance: $\max\{4,3\}=4$.

Another way is using Hamming Distance and Mahalanobis distance as follow:

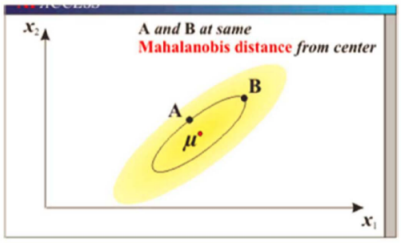
- Hamming distance (Manhattan distance on binary features)
  - # of features that differ
  - e.g.: distance of two sites based on their species composition

	sp1	sp2	sp3	sp4	sp5	sp6	sp7	sp8	sp9
Site A:	1	0	1	1	0	0	1	0	1
Site B:	0	0	1	0	1	1	1	0	1

D(A, B) = 4

- Mahalanobis distance (assuming $\mathbf{x}, \mathbf{y}$ follows a Gaussian distribution with covariance matrix Σ

$$D(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \Sigma^{-1} (\mathbf{x} - \mathbf{y})}$$



Or we can directly define similarity by using cosine similarities or kernels:

- Cosine similarities – commonly used to measure document similarity

$$cos(x, x') = \frac{< x \cdot x' >}{|x| \cdot |x'|}$$

- Kernels - e.g., RBF (Gassian) Kernel

$$S(X, X') = \exp \frac{-|X - X'|^2}{2\sigma^2}$$

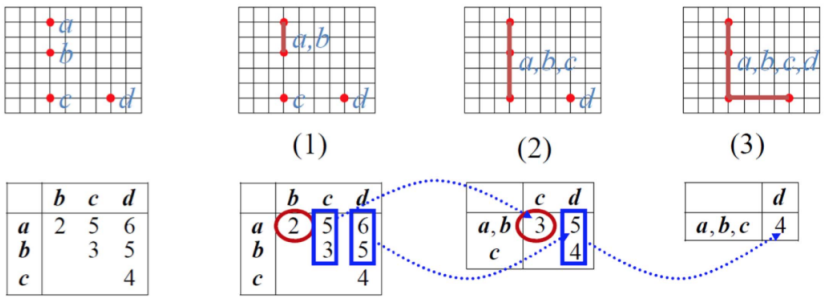
Next, we talk about types of clustering algorithms. There are 2 main types: **Hierarchical algorithms** and **Partition algorithms**. With Hierarchical algorithm, we have 2 approaches: Bottom up (agglomerative) and top down (divisive). We will discuss more bottom up approach. In Partition algorithms, we have 3 main algorithm, those are **K-means**, **Mixture of Gaussian** (GMM) and **Spectral Clustering**.

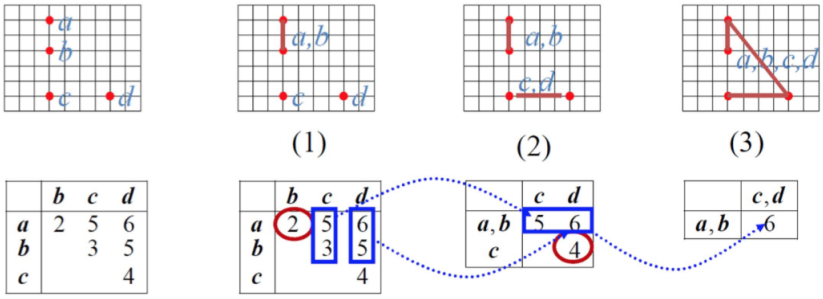
5.2 Hierarchical Agglomerative Clustering (HAC)

HAC starts with each object in a separate cluster and repeatedly joins the **closest pair of clusters** until there is only one cluster. So how we define the closest pair of clusters, there are 4 types: Single-link, complete-link, centroid and average-link as described follow (table is taken from Wikipedia:)

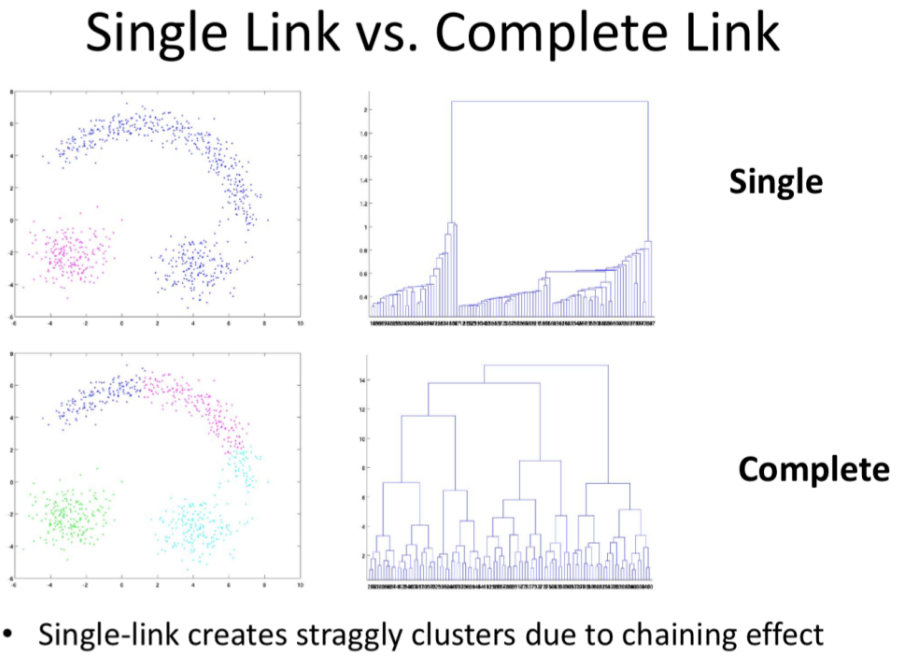
Names	Formula
Maximum or complete-linkage clustering	$\max \{ d(a,b) : a \in A, b \in B \}$.
Minimum or single-linkage clustering	$\min \{ d(a,b) : a \in A, b \in B \}$.
Average linkage clustering, or UPGMA	$\frac{1}{\|A\|\|B\|} \sum_{a \in A} \sum_{b \in B} d(a,b)$.
Centroid linkage clustering, or UPGMC	$\|\|c_s - c_t\|\|$

Here is an example of single-linkage clustering and complete-linkage clustering respectively:





To visualize the result of HAC, we use **Dendrogram**. The height of the joint = the distance between two merge clusters. The merge distance monotonically increases as we merge more and more for single, complete and average linkage methods, but not for the centroid method. We can use Dendrogram to identify the number of clusters in data and well-formed clusters. Bellow is Dendrogram of single and complete linkage method.



We complete discussion about HAC, next we move on to Partitional Clustering.

Given a data set of n points, we know that there are k clusters in the data. In general, there are $O(k^n)$ ways to partition the data, but how we decide which

is better? One intuition says that we want tight clusters. This leads to the following objective function

$$\sum_{i=1}^{k} \sum_{x \in C_i} |x - \mu_i|^2$$

It is the squared distance between data point x and its cluster center.

5.3 Kmeans

Below is the k-means algorithm. We will analyze the running time of this algorithm. At each iteration, we reassigning clusters: $O(kn)$ distance computations and we computing the centroids of each cluster: $O(n)$. We assume these two steps are each done once for l iterations: $O(lkn)$. It is a linear in all relevant factors, assuming a fixed number l of iterations and k number of clusters, it is more efficient than $O(n^2)$ of HAC.

Algorithm

Input – Desired number of clusters, *k*

Initialize – the *k* cluster centers (randomly if necessary)

Iterate –

1. Assigning each of the N data points to its nearest cluster centers
2. Re-estimate the cluster center by assuming that the current assignment is correct

$$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$$

Termination –

If none of the data points changed membership in the last iteration, exit. Otherwise, go to 1

Furthermore, k-means is **guaranteed to converge** because it takes an alternating optimization approach, each step is guaranteed to decrease the objective function. This is a very interesting characteristics of k-means. However, it is highly sensitive to the initial seeds. So it needs multiple random trials: choose the one with the best sum of squared loss. Besides, K-means is exhaustive, because it clusters every point, no notion of the outlier. So noise and outliers will cause problems because they will become singular clusters and bias the centroid estimation. K-means also has drawbacks that data point is deterministically assigned to one and only one cluster, but in reality, clusters may overlap. So we need another 'soft-clustering" algorithm that data points are assigned to clusters with certain probabilities called "Gaussian Mixture Model"

5.4 Gaussian Mixture Model

As we discussed in Gaussian Discriminative Analysis. Given a set of x's, estimate $\alpha_1, \ldots, \alpha_k, \theta_1, \ldots, \theta_k$. Once the model is identified, we can compute $p(y = i|x)$ for $i = 1, \ldots, k$.

However, when we use MLE to maximize the term: $\arg\max_\theta \prod_j P(x^j)$. We meet the log of sum, and it is difficult to optimize! We can use gradient ascent, but is very inefficient as follow:

$$\begin{aligned}\arg\max_\theta \prod_j P(\mathbf{x}^j) &= \arg\max_\theta \prod_j \sum_{i=1}^{k} P(\mathbf{x}^j, y^j = i) \\ &= \arg\max_\theta \sum_{j=1}^{n} \log \underbrace{\sum_{i=1}^{k} P(\mathbf{x}^j, y^j = i)}\end{aligned}$$

To deal with this problem, we introduce new optimize method called **Expectation Maximization** (EM). EM is a highly used approach for dealing with hidden (missing) data, here are the cluster labels. The much simpler than gradient methods. It is an iterative algorithm that starts with some initial guess of the model parameters. And it iteratively performs two linked steps:

1. **Expectation** (E-step): given current model parameter λ_t, compute the expectation for hidden (missing) data.

2. **Maximization** (M-step): re-estimate the parameters λ_{t+1}, assuming that the expected values computed in the E-step are the true values.

We have some interesting points about EM to discuss. Like K-means, it is guaranteed to converge because $P(x|\theta)$ must increase or remain the same between iterations. It bases on Optimization transfer for MLE with latent data. In practice, it may converge slowly, one can stop early if the change in log-likelihood is smaller than a threshold. However, it may converge to a local optimum as well. So it needs multiple restarts.

Finally, we can apply EM in GMM in simple case and general Gaussian:

EM – simple case

- A simple case:
  - We have unlabeled data $x^1, \cdots, x^m$
  - We know there are k classes
  - We know $\alpha_1 = P(y = 1), \ldots \alpha_k = P(y = k)$
  - We don't know $\mu_1 \cdots \mu_k$, but know the common variance σ^2

Start with an initial guess for $\mu_1, \ldots, \mu_k$,

1. If we know $\mu_1, \ldots, \mu_k$, we can easily compute probability that a point x^j belongs to class i:

$$p(y = i|x^j) \propto \exp\left(-\frac{1}{2\sigma^2}|x^j - \mu_i|^2\right) p(y = i)$$

Simply evaluate this, then normalize **E-step**

2. If we know *the* probability that each point belongs to each class, we can estimate the $\mu_1, \ldots, \mu_k$

$$\mu_i = \frac{\sum_{j=1}^m p(y=i|x^j) x^j}{\sum_{j=1}^m p(y=i|x^j)}$$ **M-step**

EM – General Gaussian

Start with an initial guess for $\mu_1, \ldots, \mu_k, \Sigma_1, \cdots, \Sigma_k, \alpha_1, \cdots, \alpha_k$,

1. If we know the parameters, we can easily compute probability that a point x^j belongs to class i:

$$p(y = i|x^j) \propto p(x^j|\mu_i, \Sigma_i)\, p(y = i)$$ **E-step**

Simply evaluate this, then normalize

2. If we know *the* probability that each point belongs to each class, we can estimate the $\mu_1, \ldots, \mu_k, \Sigma_1, \cdots, \Sigma_k, \alpha_1, \cdots, \alpha_k$,

$$\mu_i = \frac{\sum_{j=1}^m p(y=i|x^j) x^j}{\sum_{j=1}^m p(y=i|x^j)} \qquad \alpha_i = \frac{\sum_{j=1}^m p(y=i|x^j)}{m}$$

$$\Sigma_i = \frac{\sum_{j=1}^m p(y=1|x^j)(x^j-\mu_i)(x^j-\mu_i)^T}{\sum_{j=1}^m p(y=j|x^j)}$$

M-step

5.5 Spectral Clustering

Back to K-means, we have discussed that K-means is highly sensitive to initial starts. So how we choose the best initial for k-means. One approach is Spectral Clustering. We can represent data points as the vertices V of a graph G. Vertices are connected by edges E. Edges have weights described by matrix W. Large weight $W(i,j)$ means that the points i and j are very similar; otherwise, small weights imply dissimilarity. To calculate the similarity between objects, we use a Gaussian Kernel:

$$W(i,j) = exp\frac{-|x_i - x_j|^2}{\sigma^2}$$

We define some necessary terms in graph:

- Degree of nodes: $d_i = \sum_j w_{i,j}$
- Volume of a set: $vol(A) = \sum_{j \in A} d_i, A \subseteq V$
- Cut(A,B): sum of the weights of the set of edges that connect the two groups: $cut(A,b) = \sum_{i \in A, j \in B} w_{ij}$.
- Mincut: minimize weight of connections between group: $\min_{A \cap B = \emptyset, A \cup B = V} Cut(A,B)$. However we prefer more balance partitions, so we need normalized Cut
- Normalized Cut: $Ncut(A,B) = \frac{cut(A,B)}{Vol(A)} + \frac{cut(A,B)}{Vol(B)} = cut(A,B)\frac{Vol(A)+Vol(B)}{Vol(A)Vol(B)}$. We get maximized when Vol(A) and Vol(B) are equal thus it encourages balanced cut.
- Diagonal Matrix $D(i,i) = d_i$

With necessary terms, we now have the spectral clustering algorithm by Ng, Jordan, and Weiss 2001):

- Form the affinity matrix W

$$W(i,j) = \exp(-\frac{|x_i - x_j|^2}{2\sigma}), W(i,i) = 0$$

- Compute the degree matrix $D = diag(W \cdot \mathbf{1})$
- Compute the normalized graph Laplacian

$$L = D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$$

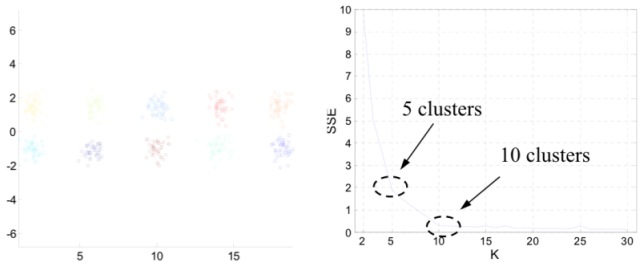
- Find the k largest eigenvectors, for new data matrix $X'_{n \times k}$
- Normalize each row(each example) to have unit length
  - $x_i \leftarrow \frac{x_i}{|x_i|}$
- Treating each row as a data point in k-d space and cluster the data into k clusters via kmeans

5.6 Model Selection

As we have discussed so far about Clustering, one important point that all algorithms have to deal with is finding the best k clusters to give the best results. It is called Model Selection in Unsupervised Learning. So we will talk about this problem in this section.

As you know, each choice of k corresponds to a different statistical model for the data. Model selection searches for a model (a choice of k) that gives us the best fit of the training data. We have many approaches: heuristic, penalty, cross-validation and stability based methods.

With the heuristic method, we plot the sum of squared error for different k values. SSE will monotonically decrease as we increase k. We pay attention to knee points because it suggests possible candidates for k.



With penalty method, we usually use Bayesian Information Criterion (BIC) or AIC as measures.

- Based on Bayesian Model Selection
  - Determine the range of k values to consider $1 \leq k \leq K_{max}$
  - Apply EM to learn a maximum likelihood fitting of the Gaussian mixture model for each possible value of *k*
  - Choose *k* that maximizes BIC

# of data points

$$2l_{\mathcal{M}}(x, \hat{\theta}) - m_{\mathcal{M}} \log(n) \equiv \text{BIC}$$

Loglikelihood of the resulting Gaussian Mixture Model

# of parameters to be estimated in *M*

  - Given two estimated models, the model with higher BIC is preferred
  - Larger k increases the likelihood, but will also cause the second term to increase
  - Often observed to be biased toward less complex model
  - Similar method: AIC = $2l_m - 2m_M$, which penalize complex model less severely

With Cross-validation method, the likelihood of the training data will always increase as we increase k. so we use cross-validation as follow:

- For each fold, learn the GMM model using the training data
- Compute the log-likelihood of the learned model on the remaining fold as test data.

With Stability Based methods, Stability is defined as repeatedly produce similar clustering on data originating from the same source. So high level of agreement among a set of clusterings =¿ the clustering model k is appropriate for the data. We evaluate multiple models and select the model resulting in the highest level of stability. We have 2 main algorithms in stability method.

- Based on resampling (Levine & Domany, 2001)
- For each k
  1. Generate clusterings on random samplings of the original data
  2. Compute pairwise similarity between each pair of clusterings
  3. Stability(k) = mean pairwise similarity.
- Select k that maximize stability

- Based on prediction accuracy (Tibshirani et al., 2001)
- For each k
  1. Randomly split data into training and testing
  2. For each split
     - cluster the training data using k
     - Predict assignment for test set and compare it to the clustering result on test set
  3. Stability(k) = mean Prediction strength
- Select k that maximize stability

5.7 Model Evaluation

Unlike supervised learning, we have class label, and we can directly compute the accuracy of testing data, in unsupervised learning, we don't have these label so we have different measurements called **Internal Criterion** and **External Criterion**. With Internal criterion, a good clustering will produce high quality clusters if it has high intra-cluster similarity and low inter-cluster similarity. But good scores on an internal criterion do not necessarily translate into good effectiveness in an application. An alternative to internal criteria is direct evaluation in the application of interest. So we need an external criterion. Rand Index, Normalized Rand Index are used when we know the ground truth, and Purity and Normalized Mutual Information are used when we do not know the ground truth.

Suppose we have the true class labels (ground truth) are known, the validity of clustering can be verified by comparing the class labels and clustering labels.

$$\begin{array}{|c|c|}\hline N & \cdot \\ \hline \cdot & n_{..} \\ \hline \end{array} = \begin{array}{|cccc|c|}\hline n_{11} & n_{12} & \cdots & n_{1l} & n_{1.} \\ n_{21} & n_{22} & \cdots & n_{2l} & n_{2.} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ n_{k1} & n_{k2} & \cdots & n_{kl} & n_{k.} \\ \hline n_{.1} & n_{.2} & \cdots & n_{.l} & n_{..} \\ \hline \end{array}$$

Rand Index and Normalized Rand Index

- Given partition (*P*) and ground truth (*G*), measure the number of vector pairs that are:
  - *a*: in the same class both in *P* and *G*.
  - *b*: in the same class in *P*, but different classes in *G*.
  - *c*: in different classes in *P*, but in the same class in *G*.
  - *d*: in different classes both in *P* and *G*.

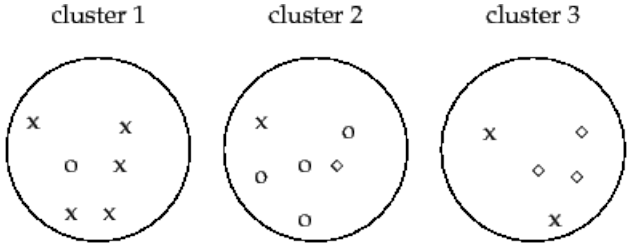
$$R = \frac{a + d}{a + b + c + d}$$

- Adjusted rand index: corrected-for-chance version of rand index
  - Compare to the expectation of the index assuming a random partition of the same cluster sizes

$$ARI = \frac{Index - ExpectedR}{MaxIndex - ExpectedR} = \frac{\sum_{i,j}\binom{n_{ij}}{2} - \left[\sum_i \binom{n_{i\cdot}}{2} \sum_j \binom{n_{\cdot j}}{2}\right] / \binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{n_{i\cdot}}{2} + \sum_j \binom{n_{\cdot j}}{2}\right] - \left[\sum_i \binom{n_{i\cdot}}{2} \sum_j \binom{n_{\cdot j}}{2}\right] / \binom{n}{2}}$$

To compute purity, each cluster is assigned to the class which is most frequent in the cluster, and then the accuracy of this assignment is measured by counting the number of correctly assigned documents and dividing by N. Formally:

$$purity(\Omega, \mathbb{C}) = \frac{1}{N} \sum_k \omega_k \cap c_j$$



► **Figure 16.1** Purity as an external evaluation criterion for cluster quality. Majority class and number of members of the majority class for the three clusters are: x, 5 (cluster 1); o, 4 (cluster 2); and ◇, 3 (cluster 3). Purity is $(1/17) \times (5 + 4 + 3) \approx 0.71$.

Where $\Omega = \omega_1, \omega_2, \ldots, \omega_K$ is the set of clusters and $\mathbb{C} = c_1, c_2, \ldots, c_J$ is the set of classes. High purity is easy to achieve when the number of clusters is large - in particular, purity is 1 if each document gets its own cluster. Thus, we cannot use purity to trade off the quality of the clustering against the number of clusters. A measure that allows us to make this tradeoff is normalized mutual information or NMI. The value of NMI is always between 0 and 1.

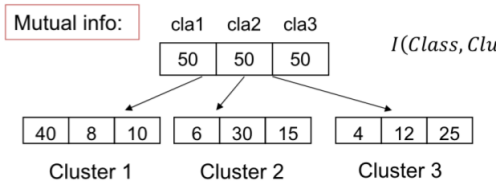
$$NMI(\Omega, \mathbb{C}) = \frac{I(\Omega; \mathbb{C})}{[H(\Omega) + H(\mathbb{C})]/2}$$

where:

$$H(\Omega) = -\sum_k P(\omega_k) log P(\omega_k)) = -\sum_k \frac{|\omega_k|}{N} log \frac{|\omega_k|}{N}$$

$$I(\Omega; \mathbb{C}) = \sum_k \sum_j P(\omega_k \cap c_j) \log \frac{P(\omega_k \cap c_j)}{P(\omega_k)P(c_j)} = \sum_k \sum_j \frac{|\omega_k \cap c_j|}{N} \log \frac{N|\omega_k \cap c_j|}{|\omega_k||c_j|}$$

where $P(\omega_k), P(c_j), P(\omega_k \cap c_j)$ are the probabilities of a document being in cluster ω_k, class c_j, and in the intersection of ω_k and c_j, respectively.



For more information and example about above measurements, you can look at here: http://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-clustering-1.html. We complete the clustering problem at here. In the next and the last section, we will discuss the Dimension reduction and some important theories in Machine Learning.

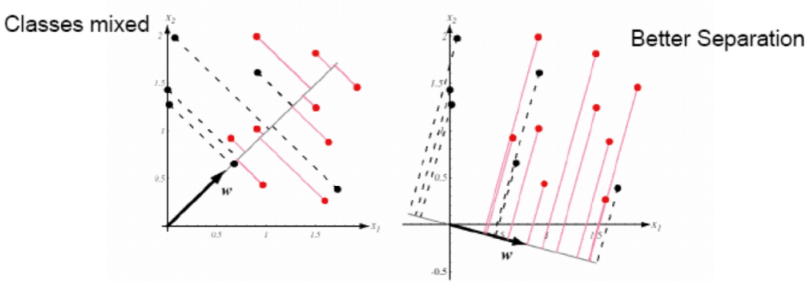
6 Dimension Reduction

Why we need dimension reduction? In high dimensional data or a large number of features, for example, documents represented by thousands of words, or million of bigrams or Images represented by thousands of pixels. There are a lot of redundant and irrelevant features like not all words are relevant for classifying/cluster documents). Also, it is difficult to interpret and visualize the high dimensional data. And one problem is that when we compute the distances to nearest and furthest neighbors will be similar.

With **Linear Features**, we can linearly project n dimension data onto a k dimension data. If supervised learning, we would like to maximize the separation among classes, we use **Linear Discriminant Analysis** or LDA. If unsupervised learning, we would like to retain as much data variance as possible, we use **Principle Component Analysis** or PCA.

6.1 Linear Discriminant Analysis (LDA)

LDA is also named Fisher Discriminant Analysis. It can be viewed as a dimension reduction method with a generative classifier $p(x|y)$: Gaussian with distant μ for each class but shared Σ. So we would find a projection direction so that the separation between classes is maximized. In other words, we are looking for a projection that best discriminates different classes.



One way to measure separation is to look at the class means μ_1 and μ_2. we want the distance between the projected means to be as large as possible. We also want the data points from the same class to be close as possible. This can be measured by the within-class scatter (variance within the class). Combining the two sides, we have the following objective:

$$J(\mathbf{w}) = \frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w}^T S_w \mathbf{w}}$$

$$S_i = \sum_{x \in \mathbf{C}_i} (x - m_i)(x - m_i)^T$$

$S_B = (m_1 - m_2)(m_1 - m_2)^T$
the between class scatter matrix

$S_w = S_1 + S_2$
the total within class scatter matrix, where

$$S_i = \sum_{x \in C_i} (x - m_i)(x - m_i)^T$$

- The above objective is known as generalized Reyleigh quotient, and it's easy to show a w that maximizes $J(w)$ must satisfy $S_B w = \lambda S_w w$
- Noticing that $S_B w = (m_1 - m_2)(m_1 - m_2)^T w$ always take the direction of $m_1 - m_2$ (Scalar)
- Ignoring the scalars, this leads to:

$$(m_1 - m_2) = S_w w$$

$$w = S_w^{-1}(m_1 - m_2)$$

That is LDA for 2 classes. How about LDA for Multi-Classes. We can use:

- Many variants exist. This is one of the commonly used ones:

$$J(w) = \frac{w^T S_B w}{w^T S_w w}$$

- Objective remains the same, with slightly different definition for between-class scatter:

$$S_B = \frac{1}{k}\sum_{i=1}^{k}(m_i - m)(m_i - m)^T$$

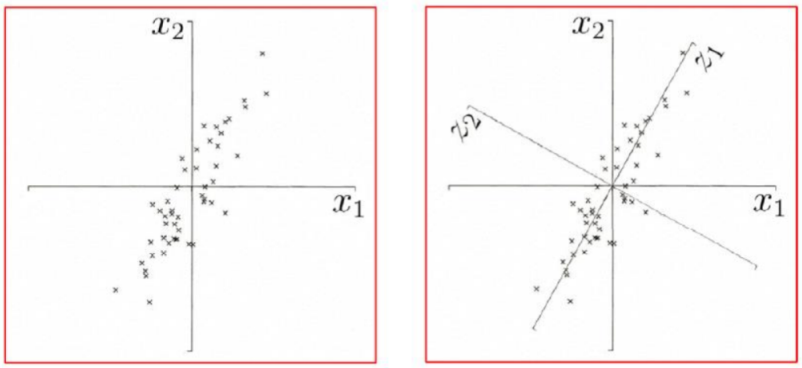
  where m is the overall mean

- Solution: $k-1$ eigenvectors of $S_w^{-1}S_B$

We have some comments about LDA. The result of LDA is optimal if and only if the classes are Gaussian and have equal covariance. It is better than PCA (discuss later), but not necessarily good enough.

6.2 Principle Component Analysis (PCA)

When the label of data is unknown, we have another solution for dimension reduction called PCA. The goal of PCA is to account for the variation in the data in as few dimension as possible. We have the following figure describing the Geometric picture of Principle components (PCs)



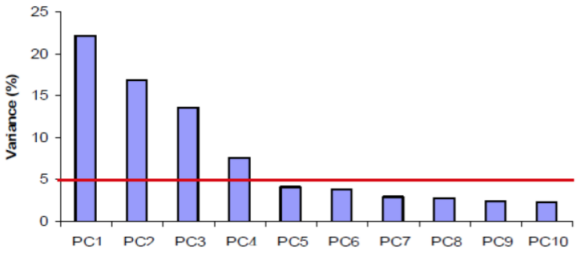
The first and second PC is the projection direction that maximize the variance of the projected data (note: $z_1 \perp z_2$).

- Given n data points: $x_1, \ldots, x_n$
- Consider a linear projection specified by v
- The projection of x onto v is $z = v^T x$
- The variance of the projected data is: $var(z) = var(v^T x) = v^T Cov(x) v = v^T v$
- The 1^{st} PC maximizes the variance subject to the constraint $v^T v = 1$

$$S = Cov(x) = \frac{1}{n}\sum_{i=1}^{n}(x_i = \bar{x})(x_i - \bar{x}^T)$$

Altogether, we have the following steps to find PCA:

- Calculate the covariance matrix of the data S
- Calculate the eigen-vectors/eigen-values of S
- Rank the eigen-values in decreasing order
- Select a fixed number of eigen-vectors, or just enough to retain a fixed percentage of the variance, for example 75%, the smallest d such that $\frac{\sum_{i=1}^{d} \lambda_i}{\sum_i \lambda_i} \geq 75\%$
- Note: You might loose some info. But the eigen-values are small, the lost is not much.



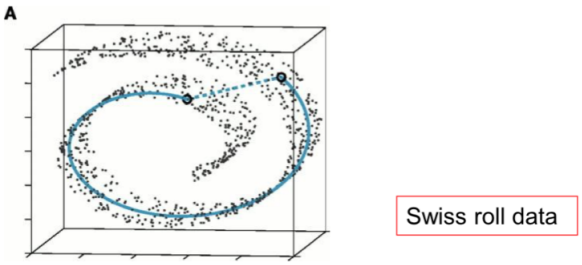
So we have some conclusions about PCA:

- PCA helps to reduce the computational complexity
- It also helps in supervised learning too because it reduces dimension (simpler hypothesis space) and smaller VC dimension (less over-fitting).
- PCA can also be seen as noise reduction because some noise is diminished when projected onto PC
- Can use PCA to visualize high dimension data for getting the first notion of data, usually $d = 2 or d = 3$
- However, you may lose important information when the small variance directions contain useful information, for example, classification follow:

PCA and LDA algorithm above is just worked in Linear Dimension. We next talk about nonlinear dimension reduction.

6.3 Nonlinear Dimension Reduction

Data often lies on or near a nonlinear low-dimensional curve. We call such low dimension structure manifolds.



We have ISOMAP (Isometric Feature Mapping). It preserves the global, nonlinear geometry of the data by preserving the geodesic distances. There are two steps:

1. Approximate the geodesic distance between every pair of points in the data
    - The manifold is locally linear
    - Euclidean distance works well for points that are close enough (connecting i and j if $d(i,j) < \epsilon$ or i is one of j's kNN). $d(i,j)$ is Euclidean distance
    - For the points that are far apart, their geodesic distance can be approximated by summing up local Euclidean distance. (can be computed as shortest path distance between 2 points)
2. Find a Euclidean mapping of the data that preserves the geodesic distance.

We have the some notices about ISOMAP. It preserves global nonlinear structure by approximating **geodesic distance**. It is sensitive to the parameters used in the graph construction (k in k-isomap and ϵ in ϵ-isomap). If data is overly sparse, the shortest path approximation to the geodesic distance can be poor because we may not have enough data to construct the manifold.

Up to this point, we complete whole basic idea about dimension reduction. In the next section, we will discuss some important theories in Machine Learning.

7 Learning Theory

This part, I think is the most boring part of our Machine Learning series, but it is quite important to build stronger intuition and develop the rule of thumb about how to best apply learning algorithms in different settings, so we have to mention in here. So let's get started!

7.1 Bias and Variance

First, we will look the analysis of **Bias and Variance**. We can write the expected loss $E(L)$ as follow:

$$E(L)$$

$$= \int (t(\mathbf{x}) - E_D[\hat{y}(\mathbf{x}; D)])^2 p(\mathbf{x}) d\mathbf{x} \qquad \text{Bias}$$

$$+ \int E_D\left[\left(E_D[\hat{y}(\mathbf{x}; D)] - \hat{y}(\mathbf{x}; D)\right)^2\right] p(\mathbf{x}) d\mathbf{x} \qquad \text{Variance}$$

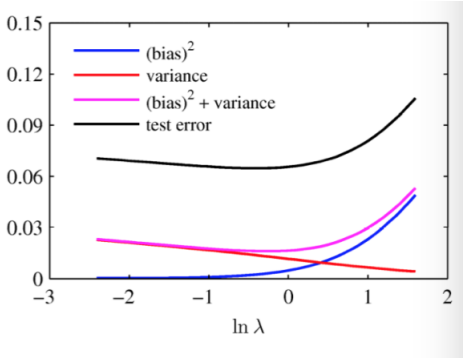
$$+ \iint (y - E[y|\mathbf{x}])^2 p(\mathbf{x}, y) d\mathbf{x} dy \qquad \text{noise}$$

$$\text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise}$$

Where:

- Bias: how well on average can our learning algorithm capture the target function
- Variance: how significantly does our learning algorithm fluctuate depending on the training set
- Noise: inherent to the data generation process itself, not controlled by the choice of learning algorithm

Next we talk about the Bias-Variance Trade-off. Taking an example from an over-regularized model (large λ or simple model) will have the high bias but low variance while an under-regularized model (small λ or complex models) will have a high variance but low bias.



Then, we talk about **computational learning theory**. It provides a theoretical analysis of learning and can show us when to expect a learning algorithm to succeed and shows when learning may be impossible. There are typically 3 areas:

- **Sample Complexity**: How many examples we need to find a good hypothesis?

- **Computational Complexity**: How much computational power we need to find a good hypothesis?
- **Mistake Bound**: How many mistakes we will make before finding a good hypothesis?

We have the following framework for Noise Free Learning:

- Assumptions:
  - Data generated according to an unknow distribution $D(x)$
  - Data labeled according to an unknown function $f : y = f(x)$
  - Hypothesis space H contains the target function f
- a simple learning framework
  - looks for a hypothesis $h \in H$ consistent with training data
    Consistent-Learn
    Input: access to a training example generator, sample size m, hypothesis space H
    1. Draw a set E of m training examples (drawn from the unknown distribution and labeled by the unknown target)
    2. Find an $h \in H$ that agress with all training examples in E

    Output: h

We define the generalization Error of a hypothesis h is the probability that h will make a mistake on a new example randomly drawn from D

$$error(h, f) = P(h(x) \neq f(x))$$

Realistic Expectation of Learning

- **Generalization Error:**
  - Many possible target functions and small set of training examples
  - Can't expect algorithm to achieve zero generalization error
  - Satisfied with an **approximately correct** hypothesis: an h with a small generalization error ϵ (that we specify)
- **Reliability:**
  - Non-zero probability to get a bad training set (e.g. non-zero probability that the training set contains a single repeated example).
  - Can't expect the algorithm to always return an ϵ-good hypothesis.
  - Will be satisfied if it returns an ϵ-good hypothesis with high probability (probably).

- How many training examples are required such that the algorithm is **probably, approximately correct** (PAC)?
  - That is , with probability at least $1-\delta$, the algorithm returns a hypothesis with generalization error less than ϵ (ϵ-good)?
  - e.g. return a hypothesis with accuracy at least 95% ($\epsilon = 0.05$) at least 99% ($\delta = 0.01$) of the time.

Base on types of H, we have different cases:

7.2 Computational Learning Theory

7.2.1 Case 1: Finite Hypothesis Space

We have **Blummer Bound** for consistent hypothesis and **Hoeffding Bound** for no consistent hypothesis

- Consistent hypothesis
  - Sample complexity required to ensure $(1-\delta)$ probability of returning a ϵ-good hypothesis is

$$m \geq \frac{1}{\epsilon}(\ln|H| + \ln\frac{1}{\delta})$$

  - Given m samples, with at least $1-\delta$ prob., the learned hypothesis will have generalization error

$$\epsilon \leq \frac{1}{m}(\ln|H| + \ln\frac{1}{\delta})$$

- No consistent hypothesis, learner finds the hypothesis h that minimizes training error

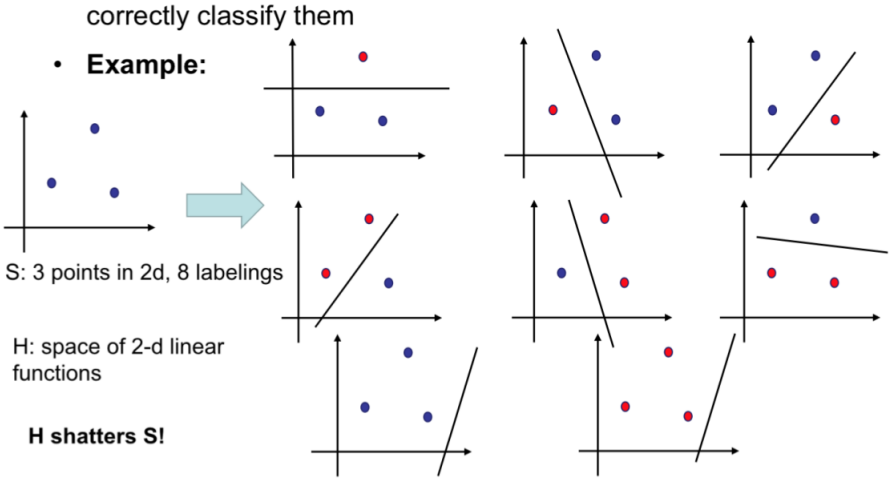
$$\epsilon(h_L) < \epsilon(h^*) + 2\sqrt{\frac{1}{2m}log\frac{|H|}{\delta}}$$

7.2.2 Case 2: Infinite Hypothesis Space

For finite spaces, the complexity of a hypothesis space was characterized roughly by $|H|$. For infinite spaces, we will introduce a concept called **VC-dimension**. We have the following definition:

Definition: Shatter

- **Definition:** Consider S, a set of m points in the input space, a hypothesis space H is said to shatter S if for every possible way of labeling the points in S, there exists an $h \in H$ that correctly classify them
- **Example:**



- **Definition of VC-dimension**: The Vapnik-Chervonenkis dimension (VC-dimension) of an hypothesis space H is the size of the largest set S than can be shattered by H
  - As long as we can find one set of size m that can be shattered by H, then $VC(H) \geq m$
  - It does not matter if some other set of size m cannot be shattered by H
- So to prove that the VC dimension of H is m, we need to:
  - Show there exists a set of size m that can be shattered by H
  - Show that no set of size $m + 1$ can be shattered by H

In general, the VC-dimension for **linear separators** in n-dimensional space is $n + 1$. A good heuristic is that VC-dimension is equal to the number of tunable parameters in the model (unless the parameters are redundant). For finite space H, we have $VC(H) \leq log_2|H|$. VC dimension measures the complexity of $|H|$

So we have bounds for Consistent Hypotheses:

- The following bound is anologous to the Blumer bound.

- if $h \in H$ is an hypothesis consistent with a training set of size m, and $VC(H) = d$, then with probability at least $1 - \delta$, h will have an error rate less than ϵ if
$$m \geq \frac{1}{\epsilon}(4 \log_2 \frac{2}{\delta} + 8d \log_2 \frac{13}{\epsilon})$$
- Compared to previous bound based on H:
$$m \geq \frac{1}{\epsilon}(\ln |H| + \ln \frac{1}{\delta})$$
since $VC(H) \leq \log_2(H)$, VC dimension generally gives a tighter upper bound on the number of examples required for PAC learning

And Bounds for Inconsistent Hypotheses:

- **Theorem** Suppose $VC(H) = d$ and a learning algorithm finds $h \in H$ with error rate ϵ_T on a traiing set of size m. Then with probability $1 - \delta$, the true error rate ϵ of h is
$$\epsilon \leq \epsilon_T + \sqrt{d(\log \frac{2m}{d} + 1) + \log \frac{4}{\delta}}$$
- Empirical Risk Minimization Principle: If you have a fixed hypothesis space H, the your learning algorithm should minimize ϵ_T: the error on the training data. (ϵ_T is also called the "empirical risk")

8 Related Topics

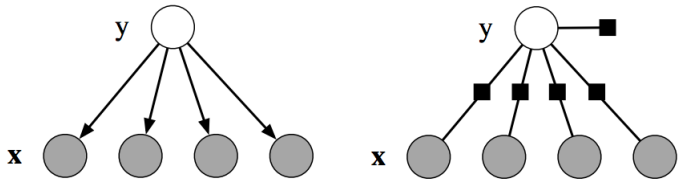
8.1 Major Problems in Machine Learning

Besides 3 basic and important task in Machine Learning comprises of Regression, Classification and Clustering as discussed above, we also have many interesting problems as follow:

- **Anomaly Detection**: in the discussion above, we talk about outliers and noise are greatly affect the performance of many learning algorithms. Therefore, if we can detect and discards these data in our process, it will boost our learning algorithm so much. The process of detect these data called Anomaly Detection. You can read for an overview at this Wikipedia page: https://en.wikipedia.org/wiki/Anomaly_detection You can read more detail in the survey: http://cucis.ece.northwestern.edu/projects/DMS/publications/AnomalyDetection.pdf
- **Reinforcement learning**: according to Andrew Ng and his lecture notes in course cs229 (http://cs229.stanford.edu/). He introduces reinforcement learning as follow: "In supervised learning, we saw algorithms that tried to make their outputs mimic the labels y given in the training set. In that setting, the labels gave an unambiguous"right answer" for each of the

inputs x. In contrast, for many sequential decision making and control problems, it is very difficult to provide this type of explicit supervision to a learning algorithm... In the reinforcement learning framework, we will instead provide our algorithms only a reward function, which indicates to the learning agent when it is doing well, and when it is doing poorly... It will then be the learning algorithm's job to figure out how to choose actions over time so as to obtain large rewards." You can read more about problems and methods in reinforcement learning at his lecture notes: http://cs229.stanford.edu/notes/cs229-notes12.pdf

- **Structured Learning**: In the introduction to Generative and Discriminative Model, I had discussed Probabilistic Graphical Model, with the two example models that we discussed, i.e. Logistic Regression and Naive Bayes. In those two models, you can express the relation between many features (variables) and output results. The left figure describes Naive Bayes graphical models. Each features x is independent with each other given the class label y. And the right figure describes the Logistic Regression graphical models. The arrows go from the x nodes to the y node. Note this is exactly the opposite of Naive Bayes models. This is two most simplest cases of structured learning. Graphical Models also have more powerful model general data called Bayesian Networks (BN) and Markov Random Fields (MN) and for sequence data such as Hidden Markov Model (HMM) and Conditional Random Field (CRF).



- **Feature Learning**: according to Wikipedia (https://en.wikipedia.org/wiki/Feature_learning) "Feature Learning is a set of techniques that learn a feature: a transformation of raw data input to a representation that can be effectively exploited in machine learning tasks... Traditional hand-crafted features often require expensive human labor and often rely on expert knowledge. Also, they normally do not generalize well. This motivates the design of efficient feature learning techniques, to automate and generalize this." Feature learning comprises of Feature Selection (choose a subset of the original set of features, read more at: read more at here: http://www.jmlr.org/papers/volume3/guyon03a/guyon03a.pdf) and Feature Extraction (build a new set of features from the original feature set , read more at: http://www.pca.narod.ru/DimensionReductionBrifReview.pdf).

- **Online Learning**: All kinds of machine learning algorithm we have discussed so far based on the assumption that the training set are available at the time we train. However, in some cases, we do not have all training data at the same time like the value of stock, or weather data... In these cases, we use another type of approach called online learning. In online learning, the mapping is updated after the arrival of every new data point in a scale fashion. We also discussed one type of Gradient Descent called Stochastic Gradient Descent is aimed with boosting the time processing for Gradient Descent, but also used for this scenario. You can read more at this Wikipedia page https://en.wikipedia.org/wiki/Online_machine_learning

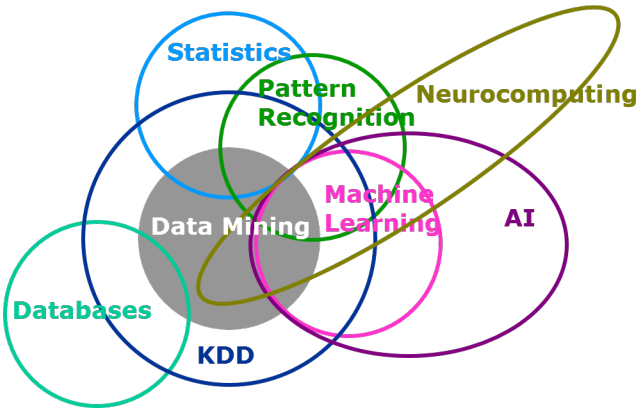
- **Semi-supervised Learning**: according to this Wikipedia page: (https://en.wikipedia.org/wiki/Semi-supervised_learning) "Semi-supervised learning falls between unsupervised learning (without any labeled training data) and supervised learning (with completely labeled training data). Many machine-learning researchers have found that unlabeled data, when used in conjunction with a small amount of labeled data, can produce considerable improvement in learning accuracy. The acquisition of labeled data for a learning problem often requires a skilled human agent (e.g. to transcribe an audio segment) or a physical experiment (e.g. determining the 3D structure of a protein or determining whether there is oil at a particular location). The cost associated with the labeling process thus may render a fully labeled training set infeasible, whereas acquisition of unlabeled data is relatively inexpensive. In such situations, semi-supervised learning can be of great practical value. Semi-supervised learning is also of theoretical interest in machine learning and as a model for human learning." Other problems: Association Rules (https://en.wikipedia.org/wiki/Association_rule_learning), Learn to rank (https://en.wikipedia.org/wiki/Learning_to_rank), Grammar Induction (https://en.wikipedia.org/wiki/Grammar_induction)

8.2 Relations between Machine Learning and other fields

Machine Learning is a subfields of Computer Science, it explores the study and construction of algorithms that can learn from and make predictions on data. Also, it is always related to other fields. In this part, I will list some relations between machine learning and:

- **Artificial Intelligence**: Machine Learning can be considered a subfield of Artificial Intelligence. Its algorithms operate by building a model from example inputs in order to make data-driven predictions or decisions, rather than following strictly static program instructions. (according to Wikipedia: https://en.wikipedia.org/wiki/Machine_learning). Nowadays, Machine Learning is the most active research field in Artificial Intelligence.

- **Data mining, Statistics, Pattern Recognition**: can be expressed in the following diagram. You can read more in this post: http://machinelearningmastery.com/where-does-machine-learning-fit-in/:



- **Probabilistic Graphical Model** (PGM): as you can see in the previous discussion, machine learning uses a lot of models inspired from PGM like Logistic Regression, Naive Bayes, Hidden Markov Model and Conditional Random Field.
- **Deep Learning**: Deep Learning is a new area of Machine Learning research, which has been introduced with the objective of moving Machine Learning closer to one of its original goals: Artificial Intelligence. (according to http://deeplearning.net/)
- **Computer Vision and Natural Language Processing**: they are both very active research fields which apply machine learning methods. Computer Vision processes vision data like images or videos while Natural Language Processing processes language data like speech, text... Machine Learning methods can be applied in 2 steps: pre-processing data and classification step. For example, Computer Vision uses GMM model to extract foreground from background or NLP use HMM model to POS task.

8.3 Machine Learning Libraries

After discussing a lot of "theoretical" concept and definition used in machine learning, in this part, I will introduce some machine learning libraries in 3 major machine learning languages: R, Python and Matlab. You can use this section as a reference when you are looking for a library.

Algorithm	R	Python (sklearn)	Matlab
Linear Regression	lm()	LinearRegression	fitlm()
Perceptron	–	Perceptron	perceptron()
Logistic Regression	glm()	LogisticRegression	fitglm()
Naive Bayes	naiveBayes()	GaussianNB or MultinomialNB or BernoulliNB	fitcnb()
LDA	lda()	LinearDiscriminantAnalysis	fitcdiscr()
SVM	svm()	SVC or SVR	fitcsvm()
Decision Tree	rpart() or ctree()	DecisionTreeClassifier or DecisionTreeRegressor	fitctree()
Neural Network	neuralnet()	MLPClassifier or MLPRegressor	fitnet()
k-NN	knn()	KNeighborsClassifier or KNeighborsRegressor	fitcknn()
Bagging	ipred()	BaggingClassifier or BaggingRegressor	fitensemble() or TreeBagger()
Random Forest	randomForest	RandomForestClassifier or RandomForestRegressor	–
Boosting	adabag()	AdaBoostClassifier or AdaBoostRegressor	fitensemble()
HAC	hclust()	AgglomerativeClustering	clusterdata()
K-means	kmeans()	KMeans	kmeans()
GMM	bgmm()	GMM	fitgmdist()
Spectral Clustering	kernelab()	SpectralClustering	–
PCA	princomp()	PCA	pca()
LDA	lda()	transform	fitcdiscr()
ISOMAP	isomap()	Isomap	Isomap()
HMM	HMM()	hmmlearn	hmmtrain()
CRF	CRF()	–	crfChain()

9 Summary

I have presented to you some Machine Learning algorithms and models. Of course, it is not enough, but it is the most basic and fundamental machine learning algorithms, any other algorithms derived from these algorithms so you should not worry about other topics in machine learning. Just make sure you understand the underlined idea behinds these algorithms, you can understand other algorithms easily.

You can find more machine learning algorithms at here:

- http://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/
- https://en.wikipedia.org/wiki/List_of_machine_learning_concepts

Each algorithm presented here has its own pros and cons, no algorithm is perfect, so you should know the characteristics of each algorithm to apply appropriately with different situations is the most important thing!

To understand more thoroughly about Machine Learning algorithms, you can read these documentations:

- **sklearn** - a popular machine learning library of python programming language: User guides all API
- **Matlab** - a popular scientific and engineering programming language: Overview
- Wikipedia page about Machine Learning: https://en.wikipedia.org/wiki/Machine_learning

To sum up, there are 3 main tasks in machine learning: regression, classification and clustering. Depending on different requirements, you can choose the appropriate machine learning algorithms to belong to these 3 tasks.

- With Regression task, you have some solutions: Linear Regression, K-Nearest Neighbor Regression, Neural Network Regression, Support Vector Regression, Decision Tree Regression, Ensemble regression (Bagging, Random Forest, Boosting)...
- With Classification task, you have some following solutions: Perceptron (it is the simplest), Logistic Regression, Naive Bayes, Discriminant Analysis, Support Vector Machine, Decision Tree, Neural Network, K-Nearest Neighbors, Ensemble method (Bagging, Random Forest, Boosting)...
- With Clustering task, you have some following solutions: HAC, K-means, GMM, Spectral Clustering, Neural Network clustering...
- Other tasks: Dimension Reduction: LDA, PCA, Isomaps,... Online Learning: K-Nearest Neighbors, Perceptron...

After deciding which group of algorithm to use, you may decide which is the most suitable algorithm for your project, that depends on first:

- Characteristics of the data: if your data is followed any distribution of probabilistic graphical model assume, you should use them. For example, Gaussian Mixture Model assume your data is generated by many Gaussian distributions, you should apply this model over other such as Logistic Regression. Else, you do not know any information about the data, you can use Discriminative models over any Generative models.
- Number of example data: if you have a lot of relevant training examples, you should you Neural Network over other methods because Neural Network is designed for this situation. Or if you just have adequate data, you can use other methods over Neural Network. In case, you have few number of example, you should use decision tree instead or simpler model like Perceptron or Nearest Neighbors.

- Dimension of data: if your data has too many features, you first job may be reducing the data into smaller dimension space by some dimension reduction algorithms like PCA (unknown label) or LDA (known label)
- Other advice: you can read more at here: http://cs229.stanford.edu/materials/ML-advice.pdf and here: http://www.holehouse.org/mlclass/10_Advice_for_applying_machine_learning.html

Nowadays, what you can do in machine learning. Research in machine learning can be divided into 3 categories:

- Application research: Apply one or combination of machine learning algorithms to an application in computer vision, natural language processing, speech recognition, bioinformatics...
- Algorithmic research: If you are genius and talented, you can completely invent new schools of machine learning algorithm like one of discussed algorithms. Or you can enhance any existing algorithm to work more robust or have better performance in specific domains like in text classification or object recognition in image processing... One of the school of machine learning algorithm you can make great contribution is Graphical Model or currently Deep Learning.
- Theoretical research: this research may focus on proving some interesting (non-trivial) properties of a new or existing learning algorithm. This fields is used for one who are good or very good at math and reasoning skills.

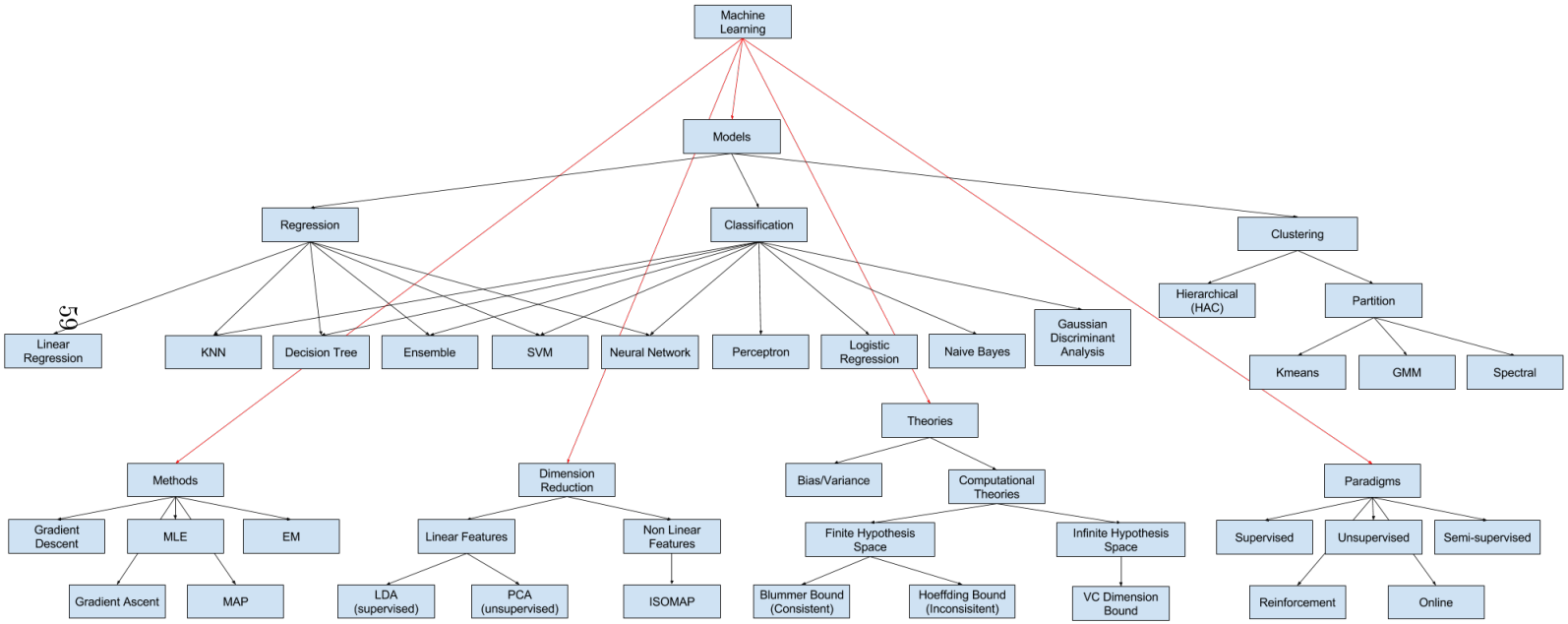
To read more about some current hot topics in Machine Learning, you can read this discussion in: Quora

I have summarized all important concepts and terms of this series in the last figure. You can go to this link for larger figure. To this point, I have completed discussion about some basic Machine Learning models, concepts and terms. I hope that throughout this series, you can obtain much of interesting knowledge. Again, I acknowledge all information including figures, information and so on on those courses: https://www.coursera.org/learn/machine-learning, http://cs229.stanford.edu/ and http://classes.engr.oregonstate.edu/eecs/fall2015/cs534/ If you have any comments, advice or questions, feel free to send me an email. Thanks for you reading!!!

References

1. https://www.coursera.org/learn/machine-learning
2. http://classes.engr.oregonstate.edu/eecs/fall2015/cs534/
3. http://cs229.stanford.edu/
4. https://en.wikipedia.org/wiki/Machine_learning
5. http://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/

6. https://en.wikipedia.org/wiki/List_of_machine_learning_concepts
7. http://scikit-learn.org/dev/user_guide.html#
8. http://scikit-learn.org/dev/modules/classes.html
9. http://www.mathworks.com/solutions/machine-learning/


Machine Learning
Models
Regression
Classification
Clustering
Linear Regression
KNN
Decision Tree
Ensemble
SVM
Neural Network
Perceptron
Logistic Regression
Naive Bayes
Gaussian Discriminant Analysis
Hierarchical (HAC)
Partition
Kmeans
GMM
Spectral
Theories
Methods
Dimension Reduction
Bias/Variance
Computational Theories
Paradigms
Gradient Descent
MLE
EM
Linear Features
Non Linear Features
Finite Hypothesis Space
Infinite Hypothesis Space
Supervised
Unsupervised
Semi-supervised
Gradient Ascent
MAP
LDA (supervised)
PCA (unsupervised)
ISOMAP
Blummer Bound (Consistent)
Hoeffding Bound (Inconsisitent)
VC Dimension Bound
Reinforcement
Online