Machine Learning - Final Exam Review

Topics for the final:

1) Probability Theory
2) Random Variables
3) Parameter Estimation
   - (3.1) Maximum a posteriori (MAP)
   - (3.2) Maximum Likelihood (ML)
   - (3.3) Bayesian Estimation
4) Expectation-Maximization (EM) algorithm.
5) Prediction problems: Classification and Regression.
   - (5.1) Optimal classification and regression models.
6) Ordinary Least-Squares (OLS) Regression
   - (6.1) Maximum Likelihood Approach
   - (6.2) Algebraic view of OLS
   - (6.3) Non-linear regression using OLS regression
   - (6.4) Regularization
7) Newton-Rapshon method
8) Logistic Regression
9) Perceptron & Pocket Algorithm
10) Naive Bayes classifiers
11) Data-preprocessing and Performance Estimation.
12) Classification and Regression Trees
13) Neural Networks
14) Committe Machines. , 15) Support Vector Machines.

Machine Learning - Final Exam Review

1) Probability Theory:

Axioms of probability: sample space $\Omega \neq \Phi$ set of outcomes of experiment
event space $\mathcal{F} \neq \Phi$ set of set of subsets of Ω

$$\left.\begin{array}{l} 1.\ A \in \mathcal{F} \Rightarrow A^c \in \mathcal{F} \\ 2.\ A_1, A_2, \dots \in \mathcal{F} \Rightarrow \bigcup_{i=1}^{\infty} A_i \in \mathcal{F} \end{array}\right\}$$

$\mathcal{F}$ is a sigma field or sigma algebra.
$(\Omega, \mathcal{F})$ is a measurable space.

Let $(\Omega, \mathcal{F})$ be a measurable space. Any function $P: \mathcal{F} \rightarrow [0,1]$ such that

$$\left.\begin{array}{l} 1.\ P(\Omega) = 1 \\ 2.\ \forall A, B \in \mathcal{F} \text{ and } A \cap B = \emptyset \Rightarrow P(A \cup B) = P(A) + P(B) \end{array}\right\}$$

P is a probability measure. (or distribution).

$(\Omega, \mathcal{F}, P)$ is a probability space.

SAMPLE SPACES:
- discrete (countable), $\Omega = \{1,2,3,4,5,6\}$, Typically: $\mathcal{F} = \mathcal{P}(\Omega)$ Power set.
- Continuous (uncountable), $\Omega = [0,1]$, $\Omega = \mathbb{R}$. Typically: $\mathcal{F} = \mathcal{B}(\Omega)$ Borel field.

Probability mass function: Ω discrete (finite or countably infinite), $\mathcal{F} = \mathcal{P}(\Omega)$.
A function $p: \Omega \rightarrow [0,1]$ is a probability mass function if $\sum_{\omega \in \Omega} p(\omega) = 1$.
The probability of any event $A \in \mathcal{F}$ is defined as $P(A) = \sum_{\omega \in A} p(\omega)$.

Probability density function: Ω continuous, $\mathcal{F} = \mathcal{B}(\Omega)$.
A function $p: \Omega \rightarrow [0, \infty)$ is a probability density function if $\int_{\Omega} p(\omega) d\omega = 1$.
the probability of an event $A \in \mathcal{B}(\Omega)$ is defined as $P(A) = \int_A p(\omega) d\omega$

Conditional Probabilities: $P(A|B) = \frac{P(A \cap B)}{P(B)}$, where $P(B) > 0$. BAYES RULE $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$

Chain Rule: Given a collection of K events $\{A_i\}_{i=1}^{k}$

$$P(A_1 \cap A_2 \cap \cdots \cap A_k) = \underline{P(A_1)}\, P(A_2|A_1)\, P(A_3|A_1 \cap A_2) \cdots P(A_k|A_1 \cap A_2 \cap \dots \cap A_{k-1})$$

↳ Prior probability: likelihood of occurrence of A in absence of any other information or evidence

$P(A|B)$ is the posterior probability, quantifies likelihood about A in presence of B.

Independence of Events: Let $(\Omega, \mathcal{F}, P)$ be a probability space.
Two events A and B from $\mathcal{F}$ are independent if $P(A \cap B) = P(A) \cdot P(B)$.
Three events A, B and C from $\mathcal{F}$. Then A and B are conditionally independent given C if $P(A \cap B | C) = P(A|C) \cdot P(B|C)$.
Independence between events does not imply conditional independence and vice versa

2) Random Variables : a variable that, from the observer's point of view, takes values non-deterministically. Mathematically, is a function that maps one sample space into another.

Given a probability space $(\Omega, \mathcal{F}, P)$, a r.v. X is a function $X : \Omega \rightarrow \Omega_x$ such that $\forall A \in \mathcal{B}(\Omega_x) : \{\omega : X(\omega) \in A\} \in \mathcal{F}$. It follows:

$$P_X(A) = P(\{\omega : X(\omega) \in A\}) \text{ (relax notation } P(X = x)).$$

For discrete r.v. X defined on $(\Omega, \mathcal{F}, P) \Rightarrow \overset{\forall x \in \Omega_x}{P_X(x)} = P_X(\{\omega\}) = P(\{\omega : X(\omega) = x\})$ the probability of an event A : $P_X(A) = P(\{\omega : X(\omega) \in A\}) = \sum_{x \in A} P_X(x), \forall A \subseteq \Omega_x$

For continuous r.v. X define a cumulative distribution function (cdf)

$$F_X(t) = P_X(\{x : x \leq t\}) = P_X((-\infty, t]) = P(X \leq t) = P\{\omega : X(\omega) \leq t\})$$

If the cdf is differentiable, the probability density function (pdf) is defined as $P_X(x) = \frac{dF_X(t)}{dt}\Big|_{t=x}$ if P_X exists, $F_X(t) = \int_{-\infty}^{t} P_X(x)\,dx$.

Joint and Marginal Distributions : X, Y defined on same prob. space $(\Omega, \mathcal{F}, P)$. The joint distribution $P_{XY}(x,y) = P(X = x, Y = y) = P(\{\omega : X(\omega) = x\} \cap \{\omega : Y(\omega) = y\})$.
The marginal distribu. $P_X(x) = \sum_y P(X = x, Y = y) = \sum_y P(X = x | Y = y) \cdot P(Y = y)$.

Conditional distributions : $P_{Y|X}(y|x) = P(Y = y | X = x) = P_{XY}(X = x, Y = y) / P_X(X = x)$.

Independence : X, Y are independent iff $P_{XY}(X = x, Y = y) = P_X(X = x) \cdot P_Y(Y = y)$

For k random varicbles to be mutually indepondent, the joint distribution of any subset of varieblos can be expressed as a product of individual distributions.

Conditional Independence : $P_{XY|Z}(X, Y | Z) = P_{X|Z}(X|Z) \cdot P_{Y|Z}(Y|Z)$ C.I $\not\Leftrightarrow$ I.

Expectations and Moments : Given a prob. space $(\Omega_x, \mathcal{B}(\Omega_x), P_X)$ and a function $f : \Omega_x \rightarrow \mathbb{R}$, define its expectation function as:

$$E_{(X)}[f(x)] = \begin{cases} \sum_{x \in \Omega_x} f(x) P_X(x) & \text{in case } X \text{ is discrete} \\ \int_{\Omega_x} f(x) P_X(x)\,dx & \text{in case } X \text{ is continuous} \end{cases}$$

↳ integrate or sum over x.

$\mathrm{Corr}(X,Y) = \dfrac{\mathrm{Cov}(X,Y)}{\sqrt{\mathrm{Cov}(X,X)\,\mathrm{Cov}(Y,Y)}} = \dfrac{\mathrm{Cov}(X,Y)}{\sqrt{\mathrm{Var}(X)\,\mathrm{Var}(Y)}}$ $\quad -1 \leq \rho \leq 1$

Expectation is linear: $E[cX] = cE[X]$, $E[X+Y] = E[X] + E[Y]$

Properties of Variance: $V[c] = 0$, $V[X] \geq 0$, $V[cX] = c^2 V[X]$

If X, Y are independent: $E[XY] = E[X] \cdot E[Y]$, $V[X+Y] = V[X] + V[Y]$

$\mathrm{cov}(X,Y) = 0$

Mixture of distributions: linear combinations of other prob. distributions.

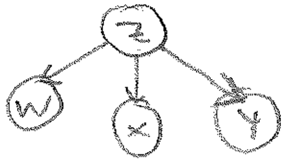
Given a set of m prob. dist. $\{p_i(x)\}_{i=1}^{m}$, a finite mixture distribution function, or mixture model, $p(x)$ is $p(x) = \sum_{i=1}^{m} w_i p_i(x)$, $w_i \geq 0$, $\sum_{i=1}^{m} w_i = 1$.

↳ mixing coefficients.

Graphical Representation of probability distributions

W, X, Y and Z. Suppose W, X and Y are conditionally indep. given Z.

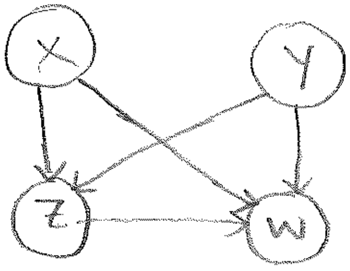
then,



$$P(W,X,Y,Z) = P(W,X,Y \mid Z) \cdot P(Z)$$
$$= P(W \mid Z) \cdot P(X,Y \mid Z) \cdot P(Z)$$
$$= P(W \mid Z) \cdot P(X \mid Z) \cdot P(Y \mid Z) \cdot P(Z)$$

W, X, Y and Z. We know $P(Y \mid X) = P(Y)$, so Y is independent of X. then,

$$P(W,X,Y,Z) = P(W \mid X,Y,Z) \cdot P(X,Y,Z)$$
$$= P(W \mid X,Y,Z) \cdot P(Z \mid X,Y) \cdot P(X,Y)$$
$$= P(W \mid X,Y,Z) \cdot P(Z \mid X,Y) \cdot P(Y \mid X) \cdot P(X)$$
$$= P(W \mid X,Y,Z) \cdot P(Z \mid X,Y) \cdot P(Y) \cdot P(X).$$



3) Parameter Estimation: presented with a set of observations and want to find a model, or function, $\hat{M}$ that shows good agreement with the data.

Ex: $\mathcal{D} = \{x_i\}_{i=1}^{n}$, $x_i \in \mathbb{R}$, $M \sim \text{Gaussian}(\mu, \sigma^2)$, $\mu \in \mathbb{R}$, $\sigma^2 \in \mathbb{R}^+$. In this case, Finding the best model means = finding the best parameters μ^*, σ^*. (Parameter estimation).

Main assumption: $\mathcal{D}$ is i.i.d.

1) Maximum a posteriori = MAP = $M_{MAP} = \underset{M \in \mathcal{M}}{\arg\max} \{p(M|\mathcal{D})\}$

Since $\underset{\text{posterior}}{P(M|\mathcal{D})} = \frac{P(\mathcal{D}|M) \cdot P(M)}{P(\mathcal{D})} \propto \underset{\text{Likelihood} \,\cdot\, \text{prior}}{P(\mathcal{D}|M) \cdot P(M)}$, we can instead optimize:

$$\boxed{M_{MAP} = \underset{M \in \mathcal{M}}{\arg\max} \{p(\mathcal{D}|M) \cdot P(M)\}}$$

we usually solve by using the log-likelihood.

↳ mode of posterior distribution.

2) Maximum likelihood = ML = $\boxed{M_{ML} = \underset{M \in \mathcal{M}}{\arg\max} \{p(\mathcal{D}|M)\}}$ → mode of likelihood function.

3) Bayesian Estimation. Addresses concerns of possibility of skewed dist. multimodal distributions or simply large regions with similar values of $P(M|\mathcal{D})$.

MAIN IDEA: minimize the posterior risk:

$$\boxed{R = \int_M \ell(M, \hat{M}) \cdot p(M|\mathcal{D}) \, dM},$$ where $\hat{M}$ is our estimate

and $\ell(M, \hat{M})$ is some loss function between two models. If $\boxed{\ell(M, \hat{M}) = (M - \hat{M})^2}$,

Then the minimizer of the posterior risk is the posterior mean

$$\boxed{M_B = E_M[M|\mathcal{D}]}$$ → Bayes Estimator.

4) Expectation - Maximization (EM) Algorithm.

This is parameter estimation for mixture models.

Given a set of i.i.d. observations $\mathcal{D} = \{x_i\}_{i=1}^{n}$, estimate the param. of $p(x|\theta) = \sum_{j=1}^{M} w_j \, p(x|\theta_j)$ where $\theta = (w_1, w_2, .., w_M, \theta_1, \theta_2, .., \theta_M)$ (all parameters).

this algorithm is used when calculus fail to provide a ML solution.

the intuition is to regard labels y_i as known, i.e., to suppose we know from which distribution each data point comes from and then maximize the respective expectation

In the EM algorithm we maximize the expected log-likelihood of both $\mathcal{D}$ and $\vec{y}$ (complete data):

$$E_{\vec{y}}[\log p(\mathcal{D}, \vec{y}|\theta) | \theta^{(t)}] = \sum_{\vec{y}} \log p(\mathcal{D}, \vec{y}|\theta) \cdot p(\vec{y}|\mathcal{D}, \theta^{(t)})$$

the procedure is then:

$$\boxed{\theta^{(t+1)} = \arg\max_{\theta} \{ E[\log p(\mathcal{D}, \vec{y}|\theta) | \theta^{(t)}] \}.}$$

5) Prediction Problems: Given $\mathcal{D} = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2), \dots, (\vec{x}_n, y_n)\}$, where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$ is the Target designation.

$x_i = (x_{i1}, x_{i2}, \dots, x_{ik})$ is a k-dimensional vector called data point (or example). Each dimension of $\vec{x_i}$ is typically called a feature or an attribute.

Two types: classification and Regression. In both cases we try to construct a function that for a previously unseen data point $\vec{x}$ predicts its target y. In classification $|\mathcal{Y}|$ is usually small, we refer to y as label. In regression $\mathcal{Y} = \mathbb{R}$ (usually). this is sometime seen as fitting.

The main difference between multi-class and multi-label classification: multi-label means that a data point might belong to one or more label. For example, a news article might be on the sports and political section at the same time.

Multi-class means we can treat classes as being composed of one or more classes. For example, we can have classes $\{1,2,3\}$ and the multi-class classificator will clasify into one of $\mathcal{P}(\{1,2,3\})$. Hence, this is still classification where the number of classes grows like 2^n.

(5.1) Optimal Classification and regression models

Bayes risk classifier: $\boxed{f_{BR}(\vec{x}) = \underset{\hat{y} \in \mathcal{Y}}{\arg\min} \left\{ \sum_{y} c(\hat{y}, y)\, p(y|\vec{x}) \right\}}$

where c is a cost function. Choose the target with the minimum Cost.

If $c(\hat{y}, y) = \begin{cases} 0 & \text{when } y = \hat{y} \\ 1 & \text{when } y \neq \hat{y} \end{cases}$ then $f_{BR}(\vec{x}) = f_{MAP}(\vec{x}) = \underset{y \in \mathcal{Y}}{\arg\max} \{ p(y|\vec{x}) \}$

Learn the probability distribution $p(y|\vec{x})$, choose class with highest prob.

For Regression: Consider squared error loss $c(f(\vec{x}), y) = (f(\vec{x}) - y)^2$

Minimizing the expected cost in this case yields that the optimal model is $E[y|\vec{x}]$ (The mean of the posterior distribution).

Generative vs. discriminative classifiers

Generative classifiers learn a model of the joint probability $\overset{p(\vec{x}|y)\cdot p(y)}{\overset{\parallel}{p(\vec{x},y)}}$, and make their predictions by using Bayes rule to calculate $\boxed{p(y|\vec{x})}$, and then pick the most likely label y. Ex: Naive Bayes.

Discriminative classifiers model the posterior $p(y|\vec{x})$ directly, or learn a direct map from inputs x to class labels. Ex: Logistic Regression

6) <u>Ordinary Least-Squares (OLS) Regression</u>. data $\mathcal{D}=\{(\vec{x}_i, y_i)\}_{i=1}^{n}$

$f(\vec{x})=\sum_{j=0}^{k} w_j x_j$, where $\vec{x}=(x_0=1, x_1, x_2, \dots, x_k)$. Finding the best parameters $\vec{w}$ is the <u>linear regression problem</u>.

Choose the following performance measure: $E(\vec{w})=\sum_{i=1}^{n}(f(x_i)-y_i)^2$

So, $\boxed{E(\vec{w})=\sum_{i=1}^{n}\left(\sum_{j=0}^{k} w_j x_{ij}-y_i\right)^2}$. Now, after calculating the gradient $\nabla E(\vec{w})$ and the Hessian matrix $H_{E(w)}$, and optimize...

Another way is to set $E(\vec{w})=(X\vec{w}-\vec{y})^T(X\vec{w}-\vec{y})=\|X\vec{w}-\vec{y}\|^2$, where $\|\vec{v}\|=\sqrt{\vec{v}^T v}=\sqrt{v_1^2+\dots+v_n^2}$, norm of vector $\vec{v}$ (or L_2 norm).

In this framework, OLS is the solution: $\boxed{\vec{w}^*=\arg\min_{\vec{w}}\|X\vec{w}-\vec{y}\|}$

We can optimize this expression:

$$\nabla E(\vec{w})=\nabla_w\{(X\vec{w}-y)^T(X\vec{w}-y)\}=2(X^T(Xw-y))=2X^TXw-2X^Ty$$

$$\nabla E(\vec{w})=0 \Leftrightarrow 2X^TXw^*-2X^Ty=0 \Leftrightarrow \cancel{2}X^TXw^*=\cancel{2}X^Ty$$

$$\Rightarrow \boxed{w^*=(X^TX)^{-1}X^Ty}$$

The second derivative (hessian) shows this is the global minimum:

$$HE_{(w)}=\frac{\partial}{\partial w}\{2X^TXw-2X^Ty\}=2\left(\frac{\partial}{\partial w}X^TXw-\frac{\partial}{\partial w}X^Ty\right)=2(X^TX)^T=2X^TX$$

which is positive semi-definite.

The predicted target is $\boxed{\hat{\vec{y}}=Xw^*=X(X^TX)^{-1}X^T\vec{y}}$

matrix $X(X^TX)^{-1}X^T$ is called the <u>projection matrix</u> (Projects $\vec{y}$ to the column space of X).

(6.1) <u>ML Approach:</u> statistical approach of linear regression.

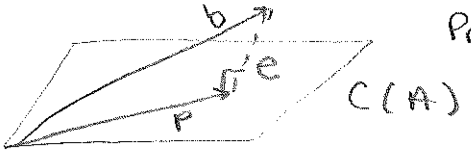
$\mathcal{D}$ is a data set of n points i.i.d. Assume also $Y=\sum_{j=0}^{k} w_j X_j+\varepsilon$, where X_j is the jth feature and $\varepsilon \sim N(0,\sigma^2)$ is an error term. $y \sim Y$.

$$Y \sim \text{Normal}(\vec{w}^T\vec{x}, \sigma^2)=p(y_i|\vec{x}_i, \vec{w})=\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left\{-\frac{(y-\sum w_j x_j)^2}{2\sigma^2}\right\}.$$

The solution to the ML problem is the same as before, i.e.,

$w^*_{ML}=\underset{w}{\arg\max}\{p(y|\underset{\text{matrix}}{X}\vec{w})\}$, $p(y|X,\vec{w})=\prod_{i=1}^{n}p(y_i|x_i,w)$, taking ll we get same solution as before.

(6.2) Algebraic view of OLS



Projection of vector b To the column space of A.

$\arg\min_x \quad \|Ax-b\|$

Goal: try to find a point in C(A) that is closest to b. Refer to it as p. then. $b = p + e$ and $p = Ax$, Since p and e are orthogonal: $p^T e = 0$. Now solve for x.

$$p^T e = 0 \Rightarrow (Ax)^T(b-p) = 0 \Rightarrow (Ax)^T(b-Ax) = 0 \Rightarrow x^T A^T (b-Ax) = 0$$

$$\Rightarrow x^T A^T b - x^T A^T A x = 0 \Rightarrow x^T(A^T b - A^T A x) = 0, \text{ but } x^T \neq 0, \text{ so}$$

$$\Rightarrow (A^T b - A^T A x) = 0 \Rightarrow A^T b = A^T A x \Rightarrow \boxed{x = (A^T A)^{-1} A^T b}$$

(6.3) Linear Regression for Non-Linear Problems.

MAin idea: apply a non-linear transformation to the data matrix X prior to the fitting step, which then enables a non-linear fit.

Polynomial Curve fitting: $f(x) = \sum_{j=0}^{p} w_j \phi_j(x) = w^T \phi$, where $\phi_j(x) = x^j$, and $\phi = (\phi_0(x), \phi_1(x), \ldots, \phi_p(x))$. Apply this transformation to every data point in X to get a new data matrix Φ. then $\boxed{w^* = (\Phi^T \Phi)^{-1} \Phi^T y}$

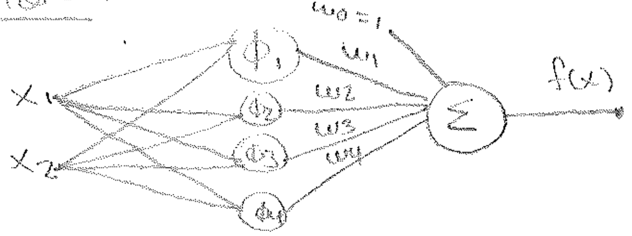
Note: One signature of overfitting is an increase in the magnitude of coefficaots.

Sigmoid function: $\phi_j(x) = 1/1+e^{-\frac{x-\mu_j}{s_j}}$ or Gaussian $\phi_j(x) = e^{-\frac{(x-\mu_j)^2}{2\sigma_j^2}}$

these only work for a one-dimensional input. Higher dimension we use:

Radial BASIS Function: A RBF network with 2-dimensional inputs and 4 basis functions.

these are also refered to as kernel functions $\phi_j(\vec{x}) = K_j(\vec{x}, \vec{c_j})$



Basis function ϕ_i
weights w_i
inputs x_i
Bias $w_0 = 1$.

(6.4) Regularization

Regularization is needed to avoid overfitting, i.e., to allow room for error in training a learning algorithm hoping that the predictor will work better on test data.

Two kinds of regularization: lasso and ridge.

Lasso: $E = \sum_{i=1}^{n} (f(\vec{x_i}) - y_i)^2 + \lambda \sum_{i=1}^{n} |w_i|$. The last term bounds the coefficients

7) Newton-Rapshon Method

A function $f(x)$ in the neighborhood of point x_0, can be approx. using Taylor series

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!} (x-x_0)^n$$

the first three terms approximation is:

$$f(x) \simeq x_0 + f'(x_0)(x-x_0) + \frac{f''(x_0)}{2}(x-x_0)^2$$

Optimize this function by finding the first derivative and set to zero.

$$f'(x) \simeq f'(x_0) + f''(x_0)(x-x_0) = 0 \quad \text{(solve for } x\text{)}$$

$$\Rightarrow \quad x = x_0 - \frac{f'(x_0)}{f''(x_0)}$$

We assume that a good enough solution x_0 already exists.

From this an iterative procedure follows:

$$\boxed{x^{(i+1)} = x^{(i)} - \frac{f'(x^{(i)})}{f''(x^{(i)})}}$$

Newton-Raphson method of optimization.

If $\vec{x}^{(0)}$ is sufficiently close to optimum $\vec{x}^*$ and H is positive definite, the N-R is well-defined and converges at second order.

For multivariate functions:

$$\boxed{\vec{x}^{(i+1)} = \vec{x}^{(i)} - \eta\left(H_{f(\vec{x}^{(i)})}\right)^{-1} \cdot \nabla f(\vec{x}^{(i)})}$$

If we don't use the hession, This is called Gradient descent

$$\boxed{\vec{x}^{(i+1)} = \vec{x}^{(i)} - \nabla f(\vec{x}^{(i)})}$$

We also introdue a parameter η, in context of machine learning called a learning rate, s.t. $\eta > 0$, to set

$$\boxed{\vec{x}^{(i+1)} = \vec{x}^{(i)} - \eta \nabla f(\vec{x}^{(i)})}$$

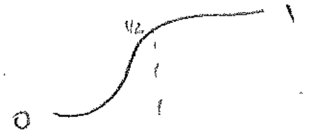
Typically $0 < \eta < 1$

8) <u>Logistic Regression</u>: binary classification in $\mathbb{R}^k$, $\mathcal{X} = \mathbb{R}^{k+1}$; $\mathcal{Y} = \{0,1\}$. add a column of one

<u>BAsic idea</u>: hypothesize a closed-form representation for the posterior probability that The class label is positive and learn parameters $\vec{w}$ from data.

$$P(Y=1|\vec{x},\vec{w}) = \frac{1}{1+e^{-\vec{w}^T\vec{x}}} \Rightarrow P(Y=0|\vec{x},\vec{w}) = 1 - \frac{1}{1+e^{-\vec{w}^T\vec{x}}}$$

$f(t) = \frac{1}{1+e^{-t}}$ is the logistic function.



8.1) <u>Maximum Conditional likelihood estimation</u>

$\mathcal{D} = \{(\vec{x}_i, y_i)\}_{i=1}^{n}$, i.i.d sample from fixed but unknown Prob. dist. $p(\vec{x}, y)$.

Assume The data generation draws $\vec{x}$ from $(X_0=1, X_1, \dots, X_k)$ according to $p(\vec{x})$ and then sets its class label Y according to Bernoulli distribution:

$$p(y|x,w) = \begin{cases} \left(\frac{1}{1+e^{-w^Tx}}\right)^{y} & \text{for } y=1 \\ \left(1-\frac{1}{1+e^{-w^Tx}}\right)^{1-y} & \text{for } y=0 \end{cases}$$

Maximize the conditional likelihood of observed class labels $\vec{y} = (y_1, y_2, \dots, y_n)$ given the inputs $X = (x_1^T, x_2^T, \dots, x_n^T)$

$$w_{ML} = \arg\max_{\vec{w}} \{\ell(\vec{w})\} = \arg\max_{\vec{w}} \left\{\prod_{i=1}^{n} p(y_i|\vec{x}_i, \vec{w})\right\}, \text{ where}$$

$$\ell(\vec{w}) = \prod_{i=1}^{n} \left(\frac{1}{1+e^{-w^Tx_i}}\right)^{y_i} \cdot \left(1-\frac{1}{1+e^{-w^Tx_i}}\right)^{1-y_i}.$$ Maximize log-likelihood instead.

$$\ell\ell(\vec{w}) = \sum_{i=1}^{n} y_i \log\left(\frac{1}{1+e^{-w^Tx_i}}\right) + (1-y_i)\log\left(1-\frac{1}{1+e^{-w^Tx_i}}\right).$$ Optimize This to get solution:

$$w^{(t+1)} = w^{(t)} + \Big(\overbrace{X^T P^{(t)}(I-P^{(t)})X}^{\text{Hessian}}\Big)^{-1} \overbrace{X^T(y-p^{(t)})}^{\text{gradient}}$$

8.2) <u>Minimization of Euclidean distance</u>

between a vector of class labels $\vec{y}$ and a vector of model outputs $\vec{p} = (p_1, p_2, \dots, p_n)$, where $p_i = P(Y_i=1|x_i,w)$. This is equivalent to minimizing The squared error function $E(\mathcal{D}, w)$ or $E(w)$

$$w^* = \arg\min_{\vec{w}} \{E(\vec{w})\} = \arg\min_{\vec{w}} \left\{\sum_{i=1}^{n} (y_i - p_i)^2\right\}$$

In this optimization the Hessian is not guaranteed to be pos. semi-definite. $E(\vec{w})$ not convex $\Rightarrow$ multiple minima with different values for objective function.

So, The minimization of The Euclidean distance between The predictions and the class labels on The training set is <u>not</u> guaranteed To find a global minimum.

(8.3)

<u>incremental or stochastic gradient descent</u>: A method where weights are updated after seeing each individual data point.

Alternatively,

<u>batch mode of training</u>: the training algorithm Utilizes all data points

<u>In both cases</u>: the influence of each data point on The weight update depends on how close ~~the~~ data point is to the separation hyperplane and whether it lies on the correct side of it.

(8.4) <u>Predicting class labels</u>: For a previously unseen data point $\vec{x}$ and a set of coefficients $\vec{w}^*$ found from logistic regression, we simply calculate the posterior probability as:

$$P(Y=1 \mid \vec{x}, \vec{w}^*) = \frac{1}{1+e^{-w^{*T}\vec{x}}}.$$

$\hat{y}=1$.

If $P(Y=1 \mid \vec{x}, \vec{w}^*) \geq 0.5$ then data point $\vec{x}$ should be label as positive

O/w $P(Y=1 \mid \vec{x}, \vec{w}^*) < 0.5$, then $\hat{y}=0$.

Note $P(Y=1 \mid x, w^*) \geq 0.5 \iff w^T x \geq 0$. thus, $w^T x = 0$ is The eq. of a hyperplane that separates positive and negative examples.

thus, <u>logistic regression model is a linear classifier</u>.

9) Perceptron & Pocket Algorithm:

Consider linear binary classification in $\mathbb{R}^k$, $\mathcal{X}=\{1\}\times\mathbb{R}^k$, $\mathcal{Y}=\{-1,+1\}$

Directly find a linear decision surface $w^Tx=0$ that separates + from −

the perceptron is a simple machine of function $f:\mathcal{X}\to\mathcal{Y}$ defined as:

$$f(\vec{x})=\begin{cases}+1 & \text{if } w^Tx\geq 0.\\ -1 & \text{if } w^Tx<0.\end{cases}$$

Because $f(\vec{x})$ is non-differentiable, we cannot use differential calculus. Instead, we use the following algorithm to train (perceptron update rule)

```
if data point $x_i$ is missclassified
    if $w^Tx_i \leq 0$ but $y_i=+1$ (underclassified)
        $w^{(t+1)} = w^{(t)} + x_i$
    else if $w^Tx_i > 0$ but $y_i=-1$ (overclassified)
        $w^{(t+1)} = w^{(t)} - x_i$
```

MAIN Property: if $\mathcal{D}=\{(\vec{x_i},y_i)\}_{i=1}^{n}$ are linearly separable, Then the perceptron training algorithm converges and finds a separation hyperplane in a finite number of steps.

Upper limit on the number of updates of perceptron: (assuming D is L.S.)

$\boxed{l_{MAX} < \frac{M^2}{\varepsilon^2},}$ where $M=\max_{x_i\in D^+}\|x_i\|$, and

$w_0^Tx>\varepsilon$; $\forall x\in\mathcal{D}^+$. ($\varepsilon$ is the minimum separation between data points and line)

MAIN Problem: if data is not linearly separable, perceptron does not converge.

Solution: Pocket Algorithm. (Minimizes the # of missclassified points).

```
set $\vec{w}$ to 0
Select $\vec{x}$ at random
if correctly classified by perceptron
    if the current run with w is longer than with $w^{(Pocket)}$
        $w^{(Pocket)} \longleftarrow w$
else
    $w \leftarrow w + yx$.
```

10) Naive Bayes Classifier

posterior class conditional

Recall: $$P(y|\vec{x}) = \frac{P(\vec{x}|y)\cdot P(y)}{P(\vec{x})} = \frac{P(\vec{x}|y)\cdot P(y)}{\sum_{y} P(\vec{x}|y)\cdot P(y)}$$

discriminative Generative

Instead of a discriminative learner where we learn the posterior $p(y|\vec{x})$, we can try to learn a generative learner where we learn $p(\vec{x},y)$ or equivalently, $p(\vec{x}|y)\cdot p(y)$. But how much training data would be needed?

To learn $p(y)$: A 100 i.i.d drawn training examples will suffice to obtain a maximum likelihood estimate of $P(y)$ (within a few percent of correct value).

to learn $P(X|y)$: we require many, many more!

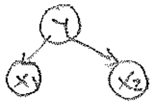
Suppose Y is boolean and $\vec{X} = (x_1, x_2, \dots, x_n)$, where x_i is boolean. We need to learn a set of parameters $\theta_{ij} \equiv P(\vec{X} = x_i | Y = y_j)$. x_i is 2^n possible values and y_i is 2. So a total of $2^n \cdot 2 = 2^{n+1}$ approx. the actual number is: for fixed y_i (0 or 1), we have to learn $2^n - 1$ (since the prob. must add to one). So a total of $\boxed{2(2^n-1) \text{ parameter } \theta_{ij}} = \Theta(2^n)$

Moreover, to obtain reliable estimates we will need to observe each of these instances multiple times!

This is not feasible: if $|\vec{X}| = 30 \Rightarrow$ need to estimate more than 3 billion param.

this is where the Naive Bayes Assumption kicks in:

Assume that the attributes $X_1, \dots, X_n$ are all conditionally independent of one another given Y.



For example, for 2 features: $P(\vec{X}|Y) = P(X_1, X_2|Y) = P(X_1|X_2, Y)\cdot P(X_2|Y)$
$= P(X_1|Y)\cdot P(X_2|Y)$.

In general $\boxed{P(\vec{X}|y) = \prod_{i=1}^{n} P(X_i|y)}$ Notice that when Y and X_i are boolean, we need only $2n$ or $\Theta(n)$ parameters

The reduction is from $\Theta(2^n)$ to $\Theta(n)$ if we make the naive Bayes assumption.

(9.1) Naive For Discrete-Valued Inputs : $\hat{\pi}_k = \hat{P}(Y=y_k) = \frac{\# D\{Y=y_k\}}{|D|}$ → # in Training set.

$$\hat{\theta}_{ijk} = \hat{P}(X_i = x_{ij} | Y = y_k) = \frac{\# D\{X_i = x_{ij} \wedge Y = y_k\} + l}{\# D\{Y = y_k\} + lJ},$$

where J = # of distinct values X_i can take on and

l determines the strength of the smoothing. (# of fictitious examples)

$l = 1$ is called Laplace Smoothing.

(9.2) Naive Bayes for Continuous Inputs :

In this case we must choose a way to represent $P(X_i|Y)$.

A common approach is to assume that for each feature i and each value from y, $P(X_i|Y) \sim Normal(\mu_{im}, \delta_{im})$.

We must estimate : $\mu_{im} = E[X_i | Y = y_m]$, $\sigma^2_{im} = E[(X_i - \mu_{im})^2 | Y = y_m]$.

Note that, for continuous features, Gauss $\mu_{j0} \neq \mu_{j1}$ and $\delta_{j0} = \delta_{j1}$

⇒ NB is equivalent to logistic Regression.

Also, N.B. is generally non-Linear but linear in previous case.

BAYES OPTIMAL CLASSFIER

$$y_{MAP} = \arg\max_{y} \{P(y|x)\}$$

Machine Learning - Final Exam Review

11) DATA - Preprocessing and Performance Estimation

containing errors

DATA Preprocessing: DATA in the real world is dirty: incomplete, noisy, inconsistent, duplicate records. No quality data => No good results.

stdev.

Measures of Central tendency: mean, media, mode.

Measures of Data Dispersion: Quartiles, Inter-quartile range, Boxplot, Outliers, Variance

or tree

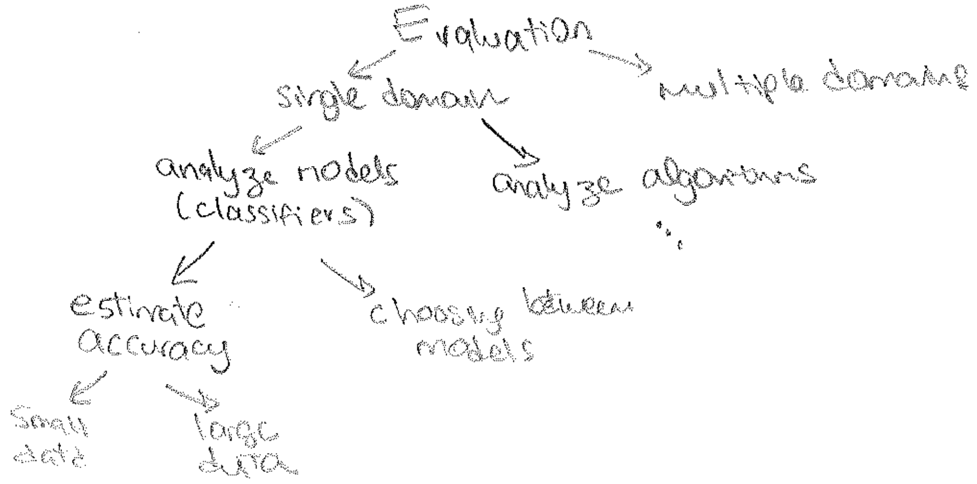
How to handle missing data?: most probable value: inference based such as Bayesian formula.

How to handle Noisy data?: sort data and partition into (equal-frequency) bins. then one can smooth by bin means, smooth by bins median, etc.

DATA transformation: Min-Max normalization: $v' = \frac{v - min}{max - min}$, Z-score normalization: $v' = \frac{v - \mu}{\sigma}$

Performance Estimation: We have a model $f: \mathcal{X} \rightarrow \mathcal{Y}$, an algoritm $a: (\mathcal{X} \times \mathcal{Y})^n \longrightarrow \mathcal{F}$, where $f \in \mathcal{F} \rightarrow$ hypothesis spaces (e.g. set of all linear functions)

How to evaluate f? Many ways:



Performance measures: (1) Classification: accuracy

(2) Regression: mean square error and R^2

n-fold cross validation : Partition data into n partitions.

in the ith step : partition i is Test, all others are train.

So, in the ith step you will have a trained model from which you can get predictions for points in the ith partition.

You will evaluate every point, but point on n^{th} partition are evaluated using the n^{th} parameters. Here The model is fixed but parameter might change from partition to partition.

From this you get the Confusion Matrix (Binary class)

leave one out is cross validation where $N = |D|$

True Class

Predicted Class	0	1
0	N_{00} ✓	N_{01} ×
1	N_{10} ×	N_{11} ✓

$$\boxed{\text{Accuracy} = \frac{N_{00} + N_{11}}{N_{00} + N_{01} + N_{10} + N_{11}}}$$

$$\boxed{\text{Error} = 1 - \text{Accuracy}}$$

N_{00} = # of True negatives; N_{01} = # of false negatives; N_{10} = # of false positives

N_{11} = # of true positives.

Sensitivity = Recall = true positive rate = $\frac{N_{11}}{N_{01} + N_{11}}$ ⇒ $1 - S_n$ = False negative rate

Specificity = True negative rate = $\frac{N_{00}}{N_{10} + N_{00}}$ ⇒ $1 - S_p$ = False positive rate

Precision = $\frac{N_{11}}{N_{10} + N_{11}}$ (accuracy on data points predicted as 1) = positive predictive value.

⇒ 1 - precision = False discovery rate.

For Regression : $MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$

$$R^2 \in (-\infty, 1] \Leftrightarrow R^2 = 1 - \frac{\sum (y_i - f(x_i))^2}{\sum (y_i - \bar{y})^2}, \quad \bar{y} = \frac{1}{n} \sum y_i$$

Ideal Predictor $R^2 = 1$.

If the predictor is such that $R^2 \leq 0$, it is useless, use mean.

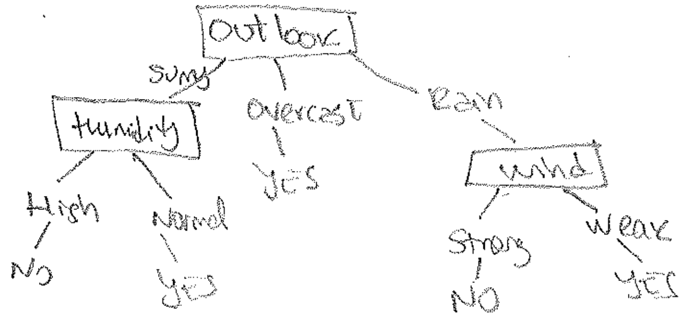
Trivial classifier : pick Majority class.

Random classifier : pick class according to distribution of data

$$\left.\begin{array}{l} \text{Pick } + \text{ w/ } p(+) \\ \text{pick } - \text{ w/ } p(-) \end{array}\right\} \Rightarrow \text{Accuracy} = p^2(+) + p^2(-)$$

12) Classification and Regression Trees.

IDEA: recursive partitioning of the data set $\mathcal{D}$, based on features.



Each node in the tree specifies a test of some attribute of the instance and each branch descending from that node corresponds to one of the possible values for this attribute.

Trees are a method for approximating discrete-valued functions that is robust to noisy data and capable of learning disjunctive expressions.

Trees, in general represent a disjunction of conjuctions:

$$(Outlook = Sunny \wedge Humidity = Normal) \vee (Outlook = Overcast) \vee (Outlook = Rain \wedge Wind = Weak)$$

To partiotion based on features we usully split based on Information GAIN

First define entropy on a set of examples S:

$$\boxed{Entropy(S) \equiv \sum_{i=1}^{c} -p_i \log_2 p_i ,}$$

where c is the number of classes (c = 2 in binary classification)

Now define GAIN:

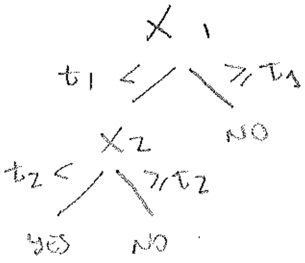
$$\boxed{GAIN(S,A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v).}$$

where Values(A) is the set of all possible values for attribute A, and S_v is the subset of S for which attribute A has value v ($S_v = \{s \in S | A(s) = v\}$).

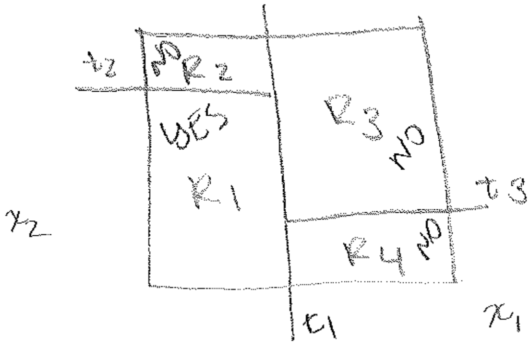
- Infer decision trees by growing from from the root downward, greedily selecting the next best attribute for each new decision branch added to the tree.
- Search is over a complete hypothesis space (All space of binary functions) compare this to perceptron that only explores linear functions. This avoids a major drawback: that the concept to be learned might not be in the search space.
- Inductive bias: preference for smaller trees. Grow trees only as large as needed in order to classify the available training examples.

Continuous Features: Sort features and pick tresholds with changes of class. For example:

T:	40	48		60	72	80		90
y:	NO	NO	↑ treshold	YES	YES	YES	↑ treshold.	NO



Note that this method is linear locally, but non-linear globally!



Regression trees (trees with continuous output).

optimal: $f_0(x) = E[y|\vec{x}]$, $f(x) = \sum_{i=1}^{m} c_m \cdot I(x \in R_m)$, where

$c_m = \text{average}(y|R_m)$.

Take average of values y of a region R_m where the example $\vec{x}$ belongs.

How to pick feature j and threshold t_j?

$$\min_{j,t_j} \left\{ \sum_{x_i \in R_{\leq t_j}} (y_i - c_1)^2 + \sum_{x_i \in R_{> t_j}} (y_i - c_2)^2 \right\}$$

Overfitting: strategies:

- early stopping
- overfit, then post prune
- use validation data, stop when error increases
- use statistical tests
- use complexity measures
  ↳ the bigger the tree the worst
  Occam's RAZOR

Trivial classifier:

pick majority class

Random classifier:

pick + with $p(+)$

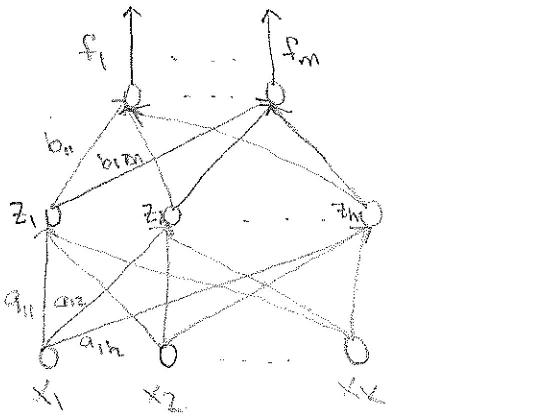
pick − with $p(-)$

accuracy $= p^2(+) + p^2(-)$

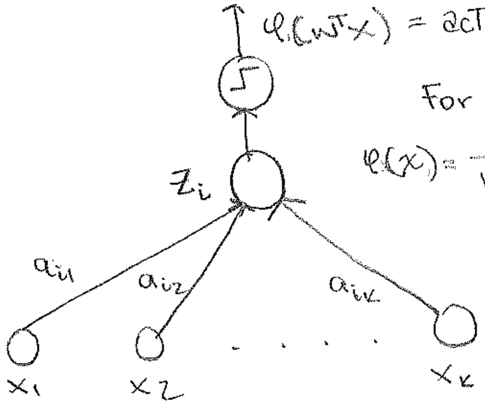
Random > trivial

13) Neural Networks:

i) A simple 2-layer Network



ii) A hidden node in a neural network



$\varphi(w^T x)$ = activation function

For feedforward NN,

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

$$\boxed{\text{theorem: 3 layer NNs are universal approximators.}}$$

We want to minimize SSE

$$\boxed{E = \sum_{i=1}^{n} \sum_{j=1}^{m} (f_j(\vec{x}_i) - y_{ij})^2}$$

Choosing $\varphi(x) = \frac{1}{1+e^{-x}}$, and deriving gradient descent for above error E, we obtain the backpropagation update rules:

(a) point-wise:

$$b_{uv}^{(t+1)} = b_{uv}^{(t)} - \eta \cdot \sum_{i=1}^{n} \beta_{iu} \cdot z_{iv}$$

$$a_{uv}^{(t+1)} = a_{uv}^{(t)} - \eta \cdot \sum_{i=1}^{n} \alpha_{iu} x_{iv}$$

$\beta_{iu} = -2(y_{iu} - f_{iu}) f_{iu}(1 - f_{iu})$,

$f_{iu} = \varphi(b_u^T z_i)$

$z_u = \varphi(a_u^T x)$

$\alpha_{iu} = \beta_{iu} \cdot b_{ju} \varphi'(a_u^T x_i)$

(b) in matrix form:

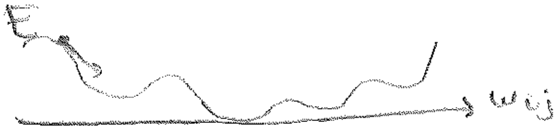
$$B^{(t+1)} = B^{(t)} - \eta \beta^T \cdot Z$$

$$A^{(t+1)} = A^{(t)} - \eta \alpha^T \cdot X$$

$\beta = [\beta_{ij}]$, $Z = [z_{ij}]$, $\alpha = [\alpha_{ij}]$

BACK propagation is gradient descent on neural networks. Error is backpropagated to a weight proportional to how strong a that weight contributed

Practical Issues:



- error function is not convex, hence there are multiple local minima + to avoid local minima, we use certain heuristics such as

  MOMENTUM : $\Delta w_i^{(t)} = \eta \delta z_i + \mu \Delta w_i^{(t-1)}$, $0 \le \mu < 1$

- Preventing overfitting. $\sum_{i=1}^{n} \sum_{j=1}^{m} (f_{ij} - y_{ij})^2 + \lambda \sum_{i,j} w_{ij}^2$
- Network or data may have to be preconditioned. (data Normalized) initialization $-t \le w \le t$ where $t \le \frac{1}{\sqrt{|\text{fan-in}|}}$.

Good Properties:

MASSIVE parallelism; graceful degradation; expressiveness: universal approx.
Computational complexity $O(n \cdot W)$ per epoch;
Good Generalization and tolerance to noise.
The price you pay is that NN's are not transparent.

(RPROP) Resilient Propagation:

MAIN IDEA : instead of using the full gradient as the step in GRADIENT DESCENT, USE only the sign of THE GRADIENT.
If we overshoot, then bactrack and try an smaller step.
If we don't overshoot, then accelerate.

Committe Machines:

Optimal Bayes classifier

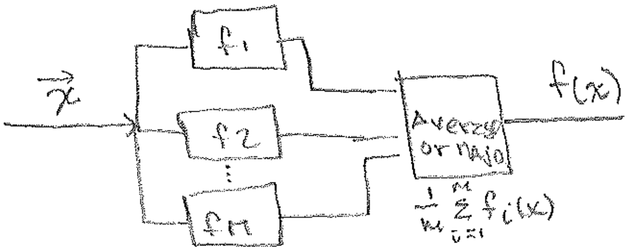
given f, $\mathcal{D}$ is not necessary.

$$p(y|\vec{x},\mathcal{D}) = \sum_{f\in\mathcal{F}} p(y|x,f,\cancel{\mathcal{D}}) \cdot p(f|x,\mathcal{D})$$

Two major strategies in combining classifiers:

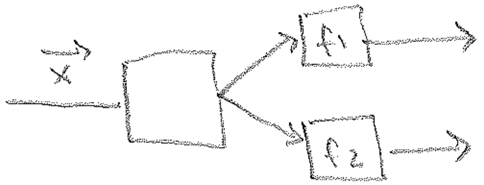
1) Static Models:

- pre-trained models, ex: bagging, boosting, random forests



2) dynamic models:

- models applied based on $\vec{x}$, ex: mixture of experts



gating Network.

Why might average be good? Consider the following argument:
Let $\mathcal{D}$ be a dataset and let $f_A(\vec{x})$ be the average predictor. then,

$$f_A(\vec{x}) = E_D[f(x,D)].$$

Note that: $E_D[(y-f(x,D))^2] = y^2 - 2y\,E_D[f(x,D)] + E_D[f^2(x,D)]$

Now, by Jense's inequality and $()^2$ being convex, we get

$$E_D[(y-\underbrace{f(x,D)}_{\text{individual classifier}})^2] \geq E_D[(y-f(x,D))]^2 = (y-\underbrace{f_A(x)}_{\text{Average classifier}})^2$$

So the error is expected to be lower with average classifier.

Ways to Aggregate Models:

1) Bootstrap Aggregating (Bagging): $f_{BAGG}(\vec{x}) = \frac{1}{B}\sum_{b=1}^{B} f_b(\vec{x})$

the idea is:

(*) Bootstrap (sample with replacement) data set $\mathcal{D}^{(b)}$ from $\mathcal{D}$

(**) Construct f_b from $\mathcal{D}^{(b)}$

(***) Aggregate (Average or majority class) outup from al $f_b(\vec{x})$.

2) Boosting (comes from PAC learning - probably approx. correct).

Strong learning is equivalent to weak learning., where
↳ almost correct. ↳ barely better than random.

↙ (Ada Boost)
Two types of boosting: (i) by filtering, (ii) by sampling/re-weighting

(i) Boosting by filtering. Given a -large- dataset: buid 3 models:

↱ first expert.
- f_1 (just better than 50%), trained on n_1 points randomly selected.
- f_2: select n_2 points as follow: flip a coin:

if Heads, then pass examples through f_1
until the first missclassified point x_m

include x_m for training data set

o/w tails, then pass examples through f_1
until first correct example x_m

include x_n for traing data set

So, training for f_2 is 50% correct on f_1 and 50% incorrect on f_1.

- f_3. Train with data size n_3, consisting of points where f_1, f_2 disagree

Main idea: f_1, f_2, f_3 combined should increase accuracy.

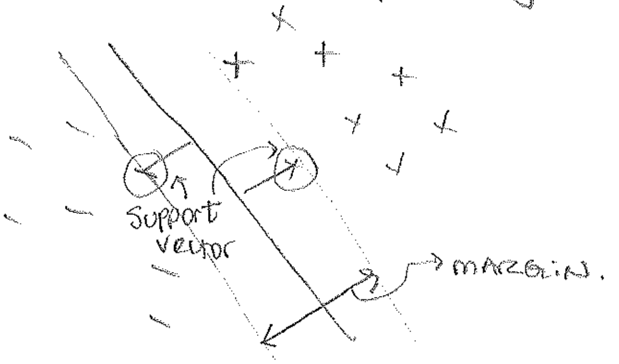
It can be shown that the error of the combined classifier is bounded by $g(\varepsilon) = 3\varepsilon^2 - 2\varepsilon^3$

Theoretically: set $f_1^{new} = f_1 + f_2 + f_3$. Train f_2^{new} and f_3^{new} as before keep iterating. Error $\to 0$ as $n \to \infty$, but impractical (need many data points)

(ii) Sampling based on distribution of weights ⟺ Ada Boost.

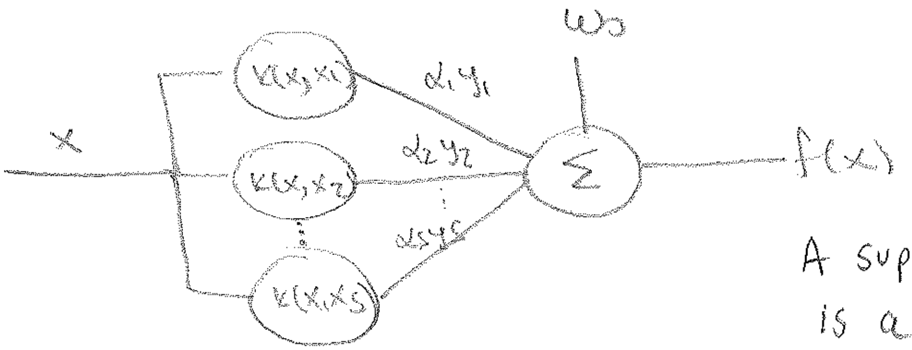
Support Vector Machines:

Binary Classification via a hyperplane, were we wont to: MAximize the MARGIN, i.e., MAximize perpendicular distance between data points and separating hyperplane:



$$(w^*, w_0^*) = \underset{w,\, w_0}{\arg\max} \left\{ \frac{1}{\|w\|} \min_{i} (y_i (w^T x_i + w_0) \right\}$$

we solve this optimization problem using Lagrangian multipliers.



A support vector machine is a kind of neural network.