Internet of Things: Data Collection and Analysis

John L. Manferdelli

johnmanferdelli@hotmail.com

May 6, 2023, 10:30

Data mining and analysis, part 1, probability

- Data mining is grounded in probability and statistics, you should know:
  - What is probability? How is probability related to information?
  - The properties of probabilities
  - How to calculate probabilities
  - How to compute expectation and variance
  - How to compute covariance
  - What a probability distribution is especially important distributions like the Gaussian distribution and the Binomial distribution.
  - How to compute the information content of the output of a distribution.
  - The law of large numbers
  - How do we measure confidence

Probability

- If we observe an experiment with n distinct outcomes, we call the union of these events a sample space and if we label each event E_i, i= 1,2,…,n. S = $\{E_1\} \cup \{E_2\} \cup \ldots \cup \{E_n\}$.
- A probability measure is a map from S to the interval [0, 1], P: S → [0,1], with the following property: $\sum_{i=1}^{n} P(E_i) = 1$.
- We can also talk about the outcome any member of a subset, T ⊆ S. For example, if T=$\{E_1, E_2\}$, P(T) = P(E_1) + P(E_2). For completeness, when we use subsets, we define P(∅)=0.
- Probability measures can have infinite domains as well, for example, the outcome of a temperature measurement may be any number in the non-negative real numbers (for Kelvin). Under specific experimental conditions, we can assign a probability to each possible temperature, P(x) and we have $\int_0^{\infty} P(x)dx$ = 1. Actually, P(x) is likely 0 and we usually talk about the probability of the temperature being in a subset of the temperature range, say, [t, t + **Δ**t], then P(t≤x≤t +**Δ**t)= $\int_t^{t+\Delta t} P(x)dx$.
- Our intuition is that if we perform the experiment many times (say, N times) and E_i is the outcome #(E_i) times that P(E_i)= $\frac{\#(E_i)}{N}$.

Probability

- If S is a sample space and T is a subset of S, P(T) + P(S\T) = 1.
- P is sometimes called a probability distribution
- To complete our intuition, if we have n events, E_1, E_2,… E_n with equally likely outcomes then $P(E_1)= P(E_2)= \ldots P(E_n)$, so $nP(E_i) =1$ and $P(E_i) = \frac{1}{n}$. An example is a fair coin toss where P(heads)= P(tails) = .5.
- One more bit of notation, a random variable, X, is a variable that takes a value in a sample space and we say $P(X=E_i)$ if the outcome of the experiment is E_i and $P(X \in T)= \sum_{X\in T} P(X)$.
- Given two random variables, X, Y from two distributions, we can write a joint distribution f(x,y) where $f(x,y)= P(X=x \wedge Y=y)$.
- Suppose we perform two experiments from two, possibly identical, sample spaces S_1 and S_2. Let X and Y be random variables representing the outcome. We say the experiments are *independent* if $\forall T_1,T_2$: $T_1 \subseteq S_1$ and $T_2 \subseteq S_2$, $P(X \in T_1 \wedge Y \in T_2)= P(X \in T_1) \times P(Y \in T_2)$. Sometimes we write this as $P(T_1 \cap T_2)$ and we can say $P(T_1 \cap T_2) = P(T_1) \times P(T_2)$. The joint distribution of two independent events satisfies f(x,y)= P(X=x) P(Y=y).

Conditional Probability and counting

- If we perform an experiment under known restrictions, we use conditional probabilities. P(A|B) is a probability distribution for A given B occurs. For example, suppose we perform an experiment and we select outcomes in a known subset T⊆S, we can ask about P(X=a|X∈T).
- Theorem: $P(A_{1,} A_{2,} A_3)= P(A_1) P(A_1|A_2) P(A_1|A_{2,}A_3)$.
- Theorem: If A_1 and A_2 are the results of *independent* experiments, $P(A_{1,} A_2)= P(A_1) P(A_2)$.
- Bayes rule (or the probability of causes): $P(B_k|A) = (P(B_k) P(A|B_k))/\sum_{k=1}^{n}(P(B_k)P(A|B_\mathrm{k}))$.
- Now we switch notation just a bit. Suppose S is a sample space and A, B ⊆ S then

  P(A∪B) = P(A) + P(B) – P(A∩B)
- To prove this, write A∪B = A\(A∩B) ∪ B\(A∩B) ∪ (A∩B), these are disjoint so P(A∪B) = P(A\(A∩B)) ∪ P(B\(A∩B)) ∪ P(A∩B) and note that P(A) = P(A\(A∩B)) + P(A∩B).
- From this we can already compute some interesting joint distributions. Suppose we make a series of n independent coin flips where P(heads) = p and P(tails) = q. The let S* be the sample space of the outcomes of the flips. P(X={T⊆S*: T has exactly k heads})= $\sum_{k=1}^{n} {}_{n}C_k\, p^k q^{n-k}$. This is the binomial distribution. Here, ${}_nC_k = \frac{n!}{(n-k)!k!}$, I should have already said n!= n×(n-1) ×…×1.

Expectation and variance

- Suppose P is a probability distribution over S, where each outcome is a single real number, we define the expectation, E(X) of a random variable X, representing the outcome of an experiment as E(X)= $\sum_{k=1}^{n} x_k P(X = x_k)$. We use the symbol μ_X = E(X) and call this the mean of the distribution.
- Suppose P is a probability distribution over S, where each outcome is a single real number, we define the *variance*, Var(X), of a random variable X, representing the outcome of an experiment as σ^2 = Var(X)= $\sum_{k=1}^{n} \left(x_k - E(X)\right)^2 P(X = x_k)$. σ is called the "standard deviation."
- Odds: If A is a subset of S, we say the odds of A is $\frac{P(A)}{P(S\setminus A)}$. For example, if the probability of an event is $\frac{1}{3}$ the odds of the event happening is $\frac{1}{2}$. Personally, I find this confusing and redundant, so I'll try to avoid it.

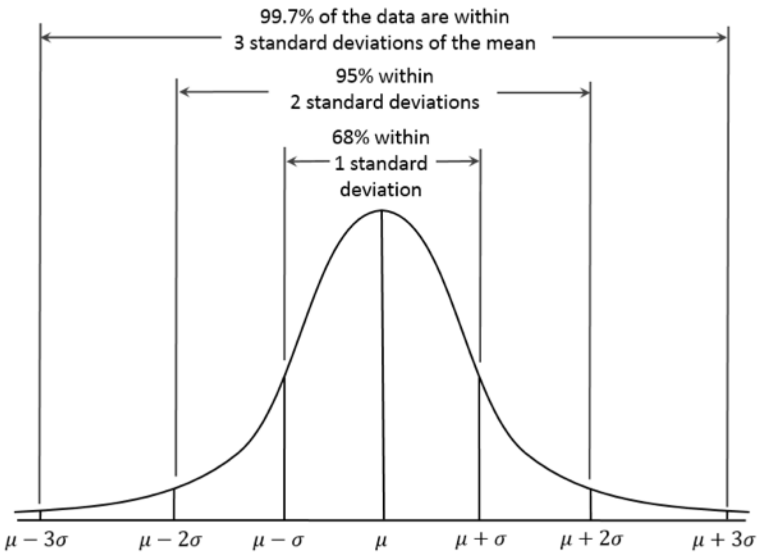
Covariance, correlation and inequalities

- *Covariance:* If X and Y are two random variables, and $f(x, y)$ is their joint distribution, we define Cov(X, Y)= $\sum_{i,j}^{n}(x_i - \mu_X)(y_j - \mu_Y)\,\mathrm{f}(x_i, y_j)$ = E((X-μ_X)(Y-μ_Y)). It is denoted σ_{XY}.
- *Correlation:* If X and Y are two random variables, we define their correlation as ρ_{XY}= $\frac{\sigma_{XY}}{\sigma_X \sigma_Y}$.
- If X and Y are independent, Cov(X, Y) = 0 = σ_{XY} and so ρ_{XY} = 0. If ρ_{XY} = 1, X and Y are perfectly correlated. Note that $\sigma_{X+Y}^2 = \sigma_X^2 + \sigma_Y^2$ + 2 Cov(X, Y).
- The above definitions carry over to continuous distributions with the obvious changes.
- **Chebyshev's inequality:** Suppose X is a random variable with mean μ and variance σ^2:
  $\forall \varepsilon$>0, P(|X- μ| ≥ ε) ≦ $\frac{\sigma^2}{\varepsilon^2}$.
- Markov's inequality: Let X be a random variable assuming non-negative values ∀t>0,
  P(X≥t) ≦ E(X)/t
- Jensen's inequality: If f(x) is convex [λf(a) + (1-λ) f(b) ≥ f(λa + (1-λ)b)], E(f(X)) ≥ f(E(X))

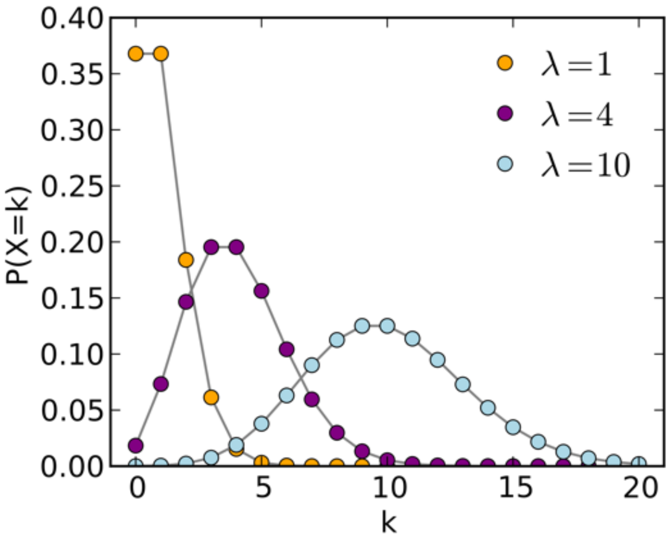
Distributions

- We’ve already seen one important distribution, namely the Binomial distribution
- The most important continuous distribution is the Gaussian or normal distribution. The sample space is all real numbers.

  $g_{\mu,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp(\frac{-(x-\mu)^2}{2\sigma^2})$ where $\exp(t) = e^t$.
- As you’d expect, $\int_{-\infty}^{\infty} x\, g_{\mu,\sigma}(x)dx = 1$, $E(X) = \mu$ and $E((X- \mu)^2) = \sigma^2$.
- Another important distribution is the Poisson distribution: If X is a discrete random variable, $p_\lambda(X=n) = (\lambda^n e^{-\lambda})/n!$.
  - The Poisson distribution models the arrival time of a service queue.



Normal distribution, Wikipedia



Poisson distribution, Wikipedia

Information

- Entropy: Let X be a random variable from a sample space $\{x_1, x_2, \ldots, x_n\}$ with distribution P. The entropy of X is

  $H(X) = \sum_{k=1}^{n} - P(X{=}x_k)\lg(P(X{=}x_k))$ where $\lg(x) = \log_2(x)$.

- Mutual information: Let X and Y be random variables, define the mutual information
  - I(X,Y) = H(X) - H(X|Y)
  - If X and Y are independent, H(X, Y)= H(X) + H(Y)
  - Suppose p(x) is an unknown distribution modelled by q(x). Define $K(p||q) = -\int p(x)\frac{q(x)}{p(x)}\,dx$. The mutual information is I(X,Y) = K(p(x,y)||p(x)p(y)).

Long term behavior

- Law of large numbers: Let X_1, X_2, …, X_n be independent, identically distributed random variables with mean μ and variance σ. Define $S_n= X_1+X_2+\ldots+X_n$.
    - $\text{Lim}_{x\rightarrow\infty} P\left(\left|\frac{S_n}{n} - \mu\right| \geq \epsilon\right) = 0)$.
- One of the most important theorems in probability is the central limit theorem which strengthens the law of large numbers.
- Central Limit Theorem: Let X_1, X_2, …, X_n be independent, identically distributed random variables with mean μ and variance σ. Define $S_n= X_1+X_2+\ldots+X_n$. Let $T_n= \frac{S_n - n\mu}{\sigma\sqrt{n}}$. Then

  $P(a \leq T_n \leq b)= \frac{1}{\sqrt{2\pi}} \int_a^b \exp\left(\frac{x^2}{2}\right) dx$.

Confidence and significance

- The Chi squared distribution: Let X_1, X_2, …, X_υ be independent, normally distributed random variables with mean 0 and variance 1. Define $\chi^2 = X_1^2 + X_2^2 + \ldots + X_\upsilon^2$. For x≥0, k= $\upsilon/2$,

  $P(\chi^2 \leq x) = \frac{1}{2^k \Gamma(k)} \int_0^x u^{(k-1)} \exp\left(-\frac{u}{2}\right) du$

  This is the chi distribution with υ-1 degrees of freedom.

- Suppose the observations X_1, X_2, …, X_n come from the same distribution. Let H_0 is a hypothesis that the observations X_1, X_2, …, X_n come from a normal distribution with mean μ and variance σ^2. Define $S^2 = ((X_1-\mu)^2 + \ldots + (X_n-\mu)^2))/n$. We “accept” H_0 with confidence .05 if $\chi(.025) \leq \frac{nS^2}{\sigma^2} \leq \chi(.975)$.

Useful facts

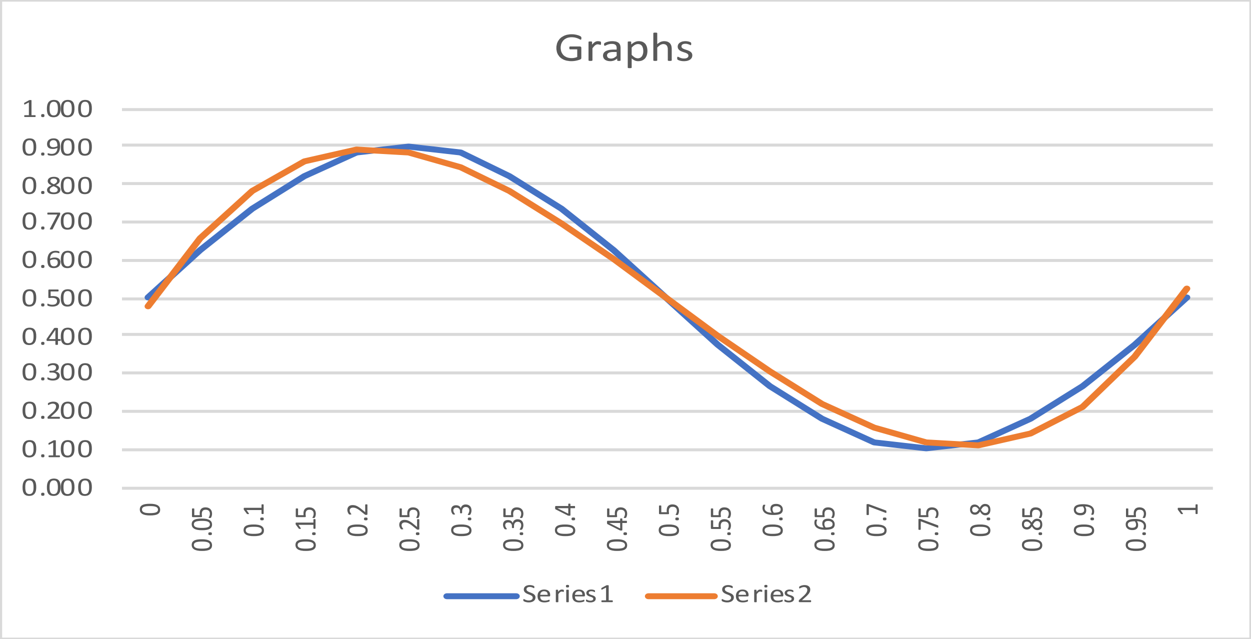
- Stirling: $n! \approx \sqrt{2\pi n}(\frac{n}{e})^{n}$.
- Newton approximations:
  - Suppose we want to find the roots of a differentiable function, f(x), Newton tells us to make an initial guess, x_0, and then make successive iterations $x_{i+1} = x_i$ - $\frac{f(x_i)}{f'(x_i))}$ until we're satisfied that we are "close enough" to a root.

Data mining and analysis, part 2

- The art of curve fitting
- Model for machine learning
  - Supervised and unsupervised learning
  - Features and collection
- Similarity and clustering
  - Feature cosine
  - K-means clustering

Curve fitting

- Approximate $f(x) = \frac{1}{2} + \frac{2}{5}\sin(2\pi x)$, $0 \le x \le 1$ by $g(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$
- "Training data" is $(k\Delta, f(k\Delta))$ for $k = 0, 1, \ldots, 10$, $\Delta = 1/10$.
- For n = 1 (ax+b), curve *underfits* data (It's just the line: y=.5)
- For n=3, the approximation is good (See diagram on below)
  $y=8.48x^3 - 12.72x^2 + 4.29x + .48$
- For n=10, curve *overfits* data. Try it! (No room in the margin here!)



What are we trying to learn?

- Often, we seek to use data to predict or detect behavior, characteristics or actions that are not directly observable from data.
- There are two common classes of machine learning tasks.
  - Regression: Predict a continuous variable (say, the selling price of a house) based on data that is available prior to sale with some assurance.
  - Classification: Determine which of a finite number of classes some process indicates. For example, whether an email is spam based on the text of the mail or determining whether a story is deceptive propaganda based on the way the story is written.
- In both cases, we're trying to learn the parameters (say, μ and σ) of an unknown distribution or discriminating whether an experimental sample comes from a distribution with parameters (μ_1, σ_1) or parameters (μ_2, σ_2).

How do we learn - 1

- First, we need to decide what data should be collected; we call each element of the data a feature.
  - For text, the features might be the presence or frequency of particular words.
  - For predicting the sale price of a house, the features might be total square footage, the past average selling price of houses in the neighborhood or the name (or traffic density) of facing or surrounding streets.
- Features can be real numbers, boolean values, or non-numeric characteristic (selected from a finite set, like color).
- Feature selection is one of the most important tasks in machine learning. Too many features can make practical collection impossible, mask characteristics, or lead to computationally impossible task for learning or operation (The curse of dimensionality”).

How do we learn - 2

- There are two kinds of learning:
    - **Supervised learning**
        - Here, we have examples of features and the corresponding, labeled, ground truth about the desired outcome. More precisely, we have a data set (**x(**k), y(k)), k = 1, 2,…, m where **x(**k) is a vector of feature data, one feature datum per vector coordinate. y(k) is a labelled outcome, that is, if we were asked to predict or classify based on **x(**k), we should say y(k). (**x(**k), y(k)), k = 1, 2,…, m is the training data.
    - **Unsupervised learning**
        - Does not have access to ground truth labelled data learning and prediction are continuous.
        - K-means clustering is an unsupervised technique.
    - For supervised learning:
        - In the learning phase, we “learn” a function of the features, based on the training data, which makes the prediction in which we are interested.
        - Sometimes training data is divided into “training” and “test” data. We train on the training data and use the test data to see how well we do.
        - In the operational phase, we use the self-same features in new data instances to predict the outcome.

Collection, provenance and features

- There are important practical questions in collection, learning and prediction.
    - How reliable is the data we’re using to learn from (where did we get it)?
    - Has someone introduced deceptive training data to adversely affect the learning.
    - How do we collect it?
    - How do we store and organize it?
    - How big is it?
    - What features should we focus on?

Let’s begin

- We’ll start with three simple techniques we use constantly
  - Similarity measurement
  - Clustering
  - Visualization

Similarity

- Suppose $\mathbf{v}^{(1)} = (x_1, x_2, \ldots, x_n)$ and $\mathbf{v}^{(2)} = (y_1, y_2, \ldots, y_n)$ are vectors representing the values of n (real) features.
- How "similar" are the observations $v^{(1)}$ and $v^{(2)}$? One measure is their "cosine".
  - $Sim_1\left(v^{(1)}, v^{(2)}\right) = \sum_{k=1}^{n} x_k y_k$
- We get a normalized value between -1 and 1 if we use
  - $Sim\left(\mathbf{v}^{(1)}, \mathbf{v}^{(2)}\right) = \dfrac{Sim_1(\mathbf{v}^{(1)}, \mathbf{v}^{(2)})}{\sqrt{Sim_1(\mathbf{v}^{(1)}, \mathbf{v}^{(1)})}\sqrt{Sim_1(\mathbf{v}^{(2)}, \mathbf{v}^{(2)})}}$
- Note $Sim(\mathbf{v}, \mathbf{v}) = 1$.

Clustering

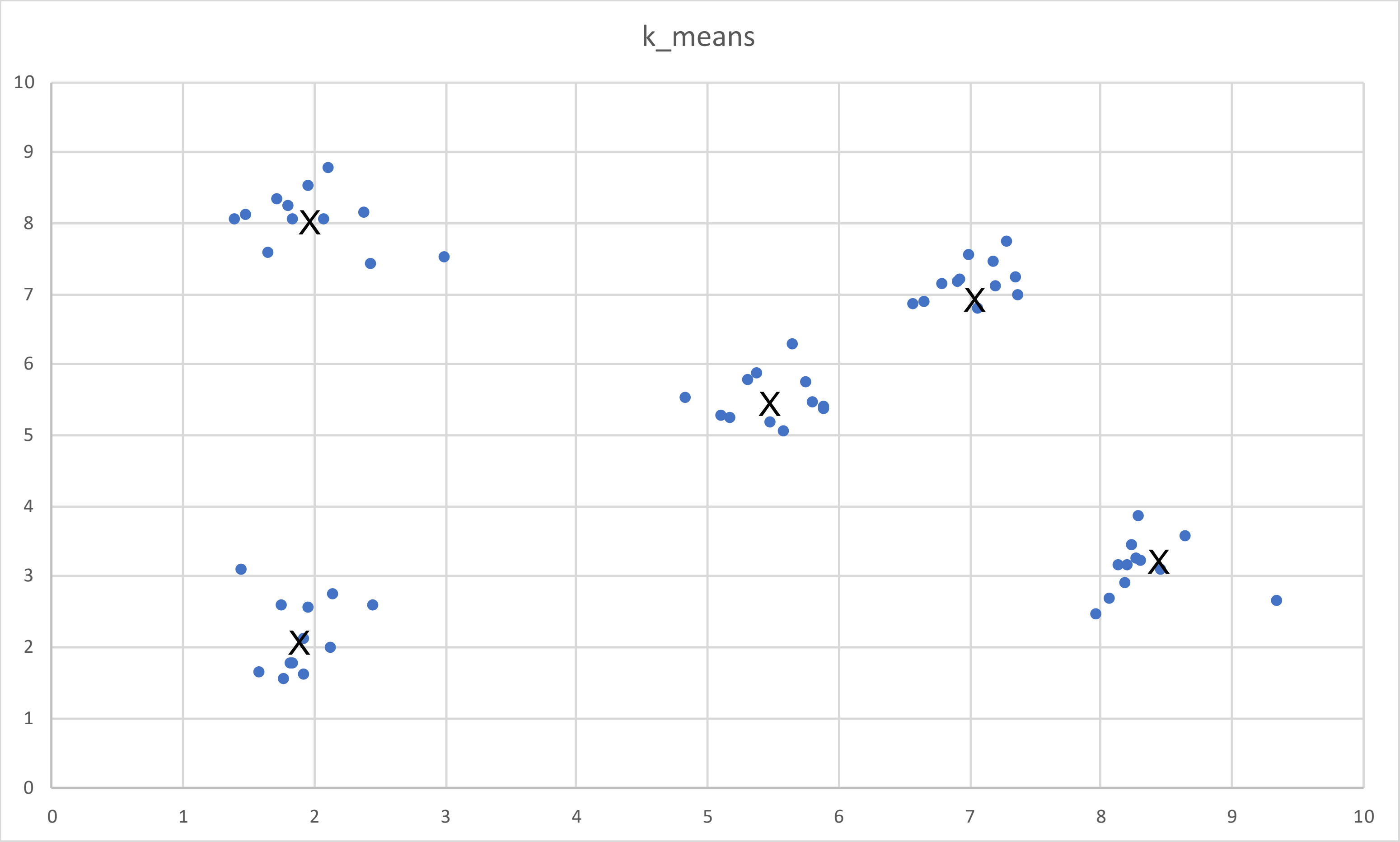
- A very simple and efficient way to take n observations in a real vector space of dimension p and distribute them into k clusters uses the “k-means” algorithm.
- In a cluster, each element of the cluster is closer to the cluster mean than the mean of another cluster.
- Here is the the k-means algorithm for the observations $\mathbf{x}_1$, $\mathbf{x}_2$, …, $\mathbf{x}_m$.

```
Let the k-cluster centers be $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_k$.
Initialization
    Pick k of the original m points at random (the first k are fine) and assign them as cluster centers
    Assign each each of the original points to a cluster based on the closest cluster center
    (old_cluster_assignment)
loop
    Compute a new cluster centers $\mathbf{y}_1', \mathbf{y}_2',, \ldots, \mathbf{y}_k'$ based on old_cluster_assignment to clusters
    Assign each original point to a cluster based on the new cluster (new_cluster_assignment)
    If old_cluster_assignment is the same as new_cluster_assignment return $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_k$.
    old_cluster_assignment ← new_cluster_assignment and $\mathbf{y}_1, \mathbf{y}_2,, \ldots, \mathbf{y}_k \leftarrow \mathbf{y}_1', \mathbf{y}_2',, \ldots, \mathbf{y}_k'$
    loop
```

- This converges quickly.

Clustering example

- Input was generated with gaussian distribution around 5 centers in $[0{,}10]^2$.
    - (1.998, 8.046)
    - (1.904, 2.150)
    - (5.496, 5.488)
    - (7.036, 7.148)
    - (8.351, 3.099)



Visualization

- Often, just plotting the data, or some derived version of the data is instructive.
- Among the most common graphing and plotting methods are:
  - Mathlab
  - Excel
  - Python graphing package like this and this.
- My daughter is an artist and could say a lot more.
- Not me.

Data mining and analysis, part 3, regression

- Regression and learning
  - Least squares
  - Gradient descent
  - Examples
- Overfitting, underfitting and regularization
  - Training and validation: training set and test set

Regression: learning functions

- The regression problem is to learn a function to predict the outcome of experiments based on observable features of a process (like words in a text).
- We will develop three types of regression learning:
  - Linear regression
  - Logistic regression
  - Deep learning (later)

Linear regression - least squares

- Let X and Y be random variables and (x_1, y_1), …, (x_m , y_m) be observations of (X, Y). Our goal is to find a linear function y= $h_{\boldsymbol{\theta}}(x) = \theta_1 x + \theta_0$ such that $\varepsilon(\boldsymbol{\theta}_{,}\ X, Y) = \sum_{k=1}^{m}(y_i - (\theta_1 x_i + \theta_0))^2$ is minimum (the "least squares minimum"). In other words, we want to "learn" (θ_1, θ_2).
- We can easily extend this to multi-variate feature sets **x** = $(1, x_1(k), x_2(k) \ldots x_p(k))$, a training set (**x**(k), y(k)), k = 1,2…,m and a set of parameters $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_p)$, where $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{i=1}^{p} \theta_i x_i$. Again, we minimize $\varepsilon(\boldsymbol{\theta}_{,}\ X, Y) = \sum_{k=1}^{m}\big(y_i - (h_{\boldsymbol{\theta}}(\boldsymbol{x})\big)^2$.
  - $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ is called the hypothesis and $\boldsymbol{\theta}$ is the set of parameters we seek to learn. We often employ the convention of making the first coordinate of **x** 1 rather than a measured feature to avoid treating the constant differently.
  - There is a closed form expression for obtaining the parameters, $\boldsymbol{\theta}$**.** We can minimize $\varepsilon(\boldsymbol{\theta}_{,}\ X, Y)$ by letting M_X= (-**x**(1)^T, -**x**(2)^T, …, -**x**(m)^T)^T , **y**= (y(1), y(2), …, y(m))^T and setting $\boldsymbol{\theta} = (M_X M_X^T)^{-1} M_X \mathbf{y}$, provided $M_X M_X^T$ is invertible.
  - We can also find $\boldsymbol{\theta}$ using gradient descent.

Gradient descent

- Again, we want to learn the parameters $\boldsymbol{\theta}$**,** $h_{\boldsymbol{\theta}}(x) = \boldsymbol{\theta}^{T}\mathbf{x}$. $\mathbf{X} = (1, x_1, x_2, \ldots, x_p)$, given m observations ($\mathbf{x}$(k), y(k)), k= 1, …, m.
- Put J($\boldsymbol{\theta}$) = $\frac{1}{2m}\sum_{k=1}^{m}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}(k)\right) - y(k)\right)^2$. To minimize J, from calculus, we learned to put $\frac{\partial J}{\partial \theta_i}$= 0, giving $\frac{\partial J}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_{\boldsymbol{\theta}}(\boldsymbol{x_i}) - y_i)(\boldsymbol{x}_i)_j$= 0. For $\theta = (\theta_0,\ \theta_1)$, the closed form solution is $\theta_1 = \frac{\sum_{i=1}^{n}(y_i - y_{av})(x_i - x_{av})}{\sum_{i=1}^{n}(x_i - x_{av})^2}$ and $\theta_0 = \ y_{av} - \theta_1 x_{av}$.
- We minimize J($\boldsymbol{\theta}$) by initializing the θ_i to some arbitrary (non-zero) value and repeatedly re-estimating the θ_i using the rule $\theta_i := \theta_i - \alpha\gamma\frac{\partial J(\theta)}{\partial \theta_i}$. α is “learning rate” and is between 0 and 1 and γ is a scaling factor.
- Least mean squares error by gradient descent:
  Loop until convergence
  $\theta_j := \theta_j - \alpha\gamma\sum_{k=1}^{m}(y(k) - h_{\boldsymbol{\theta}}(\mathbf{x}(k))\mathbf{x}(k)_j$
- We can morph gradient descent into gradient ascent (which maximizes J($\boldsymbol{\theta}$)) by changing the “–” to “+”: $\theta_j := \theta_j + \alpha\gamma\sum_{k=1}^{m}(y(k) - h_{\boldsymbol{\theta}}(\mathbf{x}(k))\mathbf{x}(k)_j$.

Gradient descent for non-linear functions

- Put J($\boldsymbol{\theta}$) = $\frac{1}{2}\sum_{k=1}^{m}(h_{\boldsymbol{\theta}}\big(\boldsymbol{x}(k) - y(k)\big)^2$.
- If $h_{\boldsymbol{\theta}}$ is linear, (i.e. $h_{\boldsymbol{\theta}}(x) = \theta^T x$), we can write our minimization rule using vector notation a little more compactly. Namely, $\frac{1}{m}\nabla_{\boldsymbol{\theta}}(h_{\boldsymbol{\theta}}(\boldsymbol{x}) - y)^2 = \frac{2}{m}\,(h_{\boldsymbol{\theta}}(\boldsymbol{x}) - y)\,\boldsymbol{x} = \mathbf{0}.$
- Even if $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ is not linear, we could use gradient (ascent) descent provided we could compute, by the chain rule, $\frac{2}{m}\nabla_{\boldsymbol{\theta}}(h_{\boldsymbol{\theta}}(\boldsymbol{x}) - y)^2 = \frac{2}{m}(h_{\boldsymbol{\theta}}(\boldsymbol{x}) - y)\nabla_{\boldsymbol{\theta}}(h_{\boldsymbol{\theta}}(\mathbf{x}))$.
- We'll see a special case of this when $h_{\boldsymbol{\theta}}(\mathrm{x}) = \mathrm{g}(\boldsymbol{\theta}^T\boldsymbol{x})$, where we can compute $\nabla_{\boldsymbol{\theta}}(h_{\boldsymbol{\theta}}(\mathbf{x}))$ using the chain rule as $\nabla_{\boldsymbol{\theta}}h_{\boldsymbol{\theta}}(\boldsymbol{x}) = g'\big(h_{\boldsymbol{\theta}}(\boldsymbol{x})\big)\,\boldsymbol{x}.$ A popular such g is g(z) = $\frac{1}{1+e^{-z}}$.
- The chain rule will come in handy for vector valued functions as well. Suppose **h(x)** = **f(y)** where **y** = **g(x)**. The Jacobian of **h(x)**, written as, $J_{\boldsymbol{x}}(\boldsymbol{h})(\mathrm{x})$ is and n x n matrix ($\frac{\partial h_i}{\partial x_i}$). By the chain rule, $J_{\boldsymbol{x}}(\boldsymbol{h})(\mathrm{x}) = J_{\boldsymbol{y}}(\boldsymbol{f})\big(\boldsymbol{g}(\boldsymbol{x})\big) \circ J_{\boldsymbol{x}}(\boldsymbol{g})(\boldsymbol{x}))$.

Gradient descent vs closed form

- Learning rate
    - If we pick $\boldsymbol{\alpha}$ too large, oscillations may be large and the process might not even converge.
    - If we pick $\boldsymbol{\alpha}$ too small convergence is slow.
- Gradient descent
    - Is iterative
    - Can be adapted to complex hypothesis
    - Works well when m is large
    - Supports incremental learning
    - Converges to local optimum (OK for quadratic functions.)
- Close form solution
    - Works well when m is small
    - No need to select learning rate

Ng’s example[1]: learning housing prices

size	bdrms	price
2104	3	399900
1600	3	329900
2400	3	369000
1416	2	232000
3000	4	539900
1985	4	299900
1534	3	314900
1427	3	198999
1380	3	212000
1494	3	242500
1940	4	239999
2000	3	347000
1890	3	329999
4478	5	699900

size	bdrms	price
1268	3	259900
2300	4	449900
1320	2	299900
1236	3	199900
2609	4	499998
3031	4	599000
1767	3	252900
1888	2	255000
1604	3	242900
1962	4	259900
3890	3	573900
1100	3	249900
1458	3	464500
2526	3	469000

size	bdrms	price
2200	3	475000
2637	3	299900
1839	2	349900
1000	1	169900
2040	4	314900
3137	3	579900
1811	4	285900
1437	3	249900
1239	3	229900
2132	4	345000
4215	4	549000
2162	4	287000
1664	2	368500
2238	3	329900

size	bdrms	price
2567	4	314000
1200	3	299000
852	2	179900
1852	4	299900
1203	3	239500

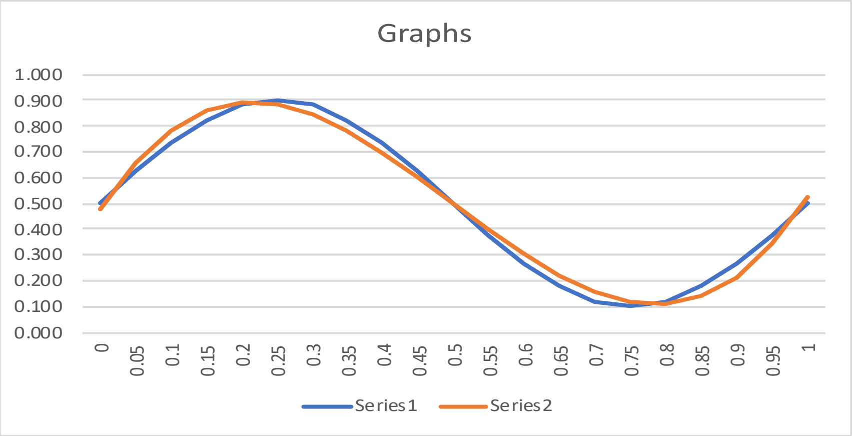
Direct solution
b: 92392.92, w: (139.30948, -9681.65919)
Prediction: 1500 square ft, 3 bedrooms, predicted price: 272312.16, rms: 9325.08

Gradient descent
num_steps: 78
b: 78680.71678, w: (128.67889, 2461.29430)
Prediction: 1500 square ft, 3 bedrooms, predicted price: 279082.93, rms: 11891.09

[1] From Andrew Ng’s Stanford class notes

Overfitting and underfitting revisited

- Earlier, we approximated the curve $f(x) = \frac{1}{2} + \frac{2}{5}\sin(2\pi x)$, for $0 \leq x \leq 1$, using a polynomial and invited you to think of this as a learning problem, namely, given the value of the true f(x) at m evenly spaced points, find a polynomial that “predicts” the values of f(x) at other points. We tried three polynomials: a linear function, a cubic and a polynomial of degree 10. We found the linear prediction “underfit” the data (and was hence a poor predicter), a cubic, which did a fine job and a 10th degree polynomial which “overfit” the data, oscillating wildly off the training set but doing perfectly on the training data.
- The cubic was $y = 8.48x^3 - 12.72x^2 + 4.29x +$ .48. But how did we find the parameters?.
  - The answer is we used linear regression with four features.
  - The features were $1, x_i, {x_i}^2$, and ${x_i}^3$.
- In fact, we did three things in the fitting exercise:
  - We selected a model with four parameters.
  - We applied linear regression.
  - We tested the quality of our prediction.



Motivation for this slide is from Bishop, Pattern Recognition and Machine Learning

Overfitting and underfitting revisited

- Now lets look at the question a little more broadly. Suppose we have training data (**x**(i), y(i)), i= 1, 2, …, m and validation data (**x**(i), y(i)), i= m+1, m+ 2, …, 2m
- Suppose, as in linear regression, we denote the parameters as a four-dimensional vector, **w** and denote the parameter values we settled on is denoted by **w***.
- By using the validation data, we can calculate the root mean square error $E_{RMS} = \frac{\sqrt{\epsilon(w^*)}}{m}$ where $\epsilon(w^*) = \sum_{i=m+1}^{2m}(y(i) - w^{*T} \cdot \boldsymbol{x}(i))^2$. This gives a good normalized metric for loss. Had we calculated it for out three proposed models, we would have found, as we noted by graphing, the cubic model was the best fit.
- This is an opportunity to introduce two topics in a simple setting:
    - Regularization
    - Cross validation

Motivation for this slide is from Bishop, Pattern Recognition and Machine Learning

Regularization

- We applied linear regression minimizing the “loss” function $J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_{\theta}(\boldsymbol{x}(i)) - y(i))^2$
- To avoid overfitting, which causes the parameters in our model to have large values we could instead *minimize* the loss function $J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_{\theta}(\boldsymbol{x}(i)) - y(i))^2 + \frac{\lambda}{2}\sum_{j=1}^{d}{\theta_j}^2$. λ ≥ 0 is the regularization parameter.
- This penalizes ”large coefficients” and reduces the risk of overfitting.

Motivation for this slide is from Bishop, Pattern Recognition and Machine Learning

Cross validation

- We had a very small training set of 11 values so splitting the known labelled data into a training and testing (or validation) set is risky.
- For small labelled sets, there is a technique, called cross validation, that helps.
- Suppose we have S_M labelled values. We "train" on different subsets of size S_{M-1} and then validate (for example, by calculating E_{RMS}) on the omitted subset of size S.

Motivation for this slide is from Bishop, Pattern Recognition and Machine Learning

Regularizing regression

- Let X and Y be random variables and ($\mathbf{x}_1$, y_1), …, ($\mathbf{x}_m$, y_m) be observations of (X, Y).
- Ridge regression
  - $J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(\boldsymbol{x}(i)) - y(i))^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j{}^2$
  - λ ≥ 0 is the regularization parameter.
  - $min_\theta \sum_{i=1}^{m}(h_\theta(\boldsymbol{x}(i)) - y(i))^2$, subject to $\sum_{j=1}^{d}\theta_j{}^2 \le \epsilon$
- Lasso regression
  - $J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(\boldsymbol{x}(i)) - y(i))^2 + \lambda\sum_{j=1}^{d}|\theta_j|$
  - $min_\theta \sum_{i=1}^{m}(h_\theta(\boldsymbol{x}(i)) - y(i))^2$, subject to $\sum_{j=1}^{d}|\theta_j| \le \epsilon$

Rescaling and data conditioning

- Given a distribution, X, we often scale it by putting $Y = \frac{X-\mu}{\sigma}$ so the distribution has mean 0 and variance 1.

Data mining and analysis, part 4, some mathematical methods

- Lagrange multipliers
- Likelihood
- Maximum likelihood estimation

Lagrange multipliers

- Let’s refresh our memory with respect to an old maximization (or minimization technique): Lagrange multipliers.
- For x=(x_1, x_2, …, x_p), suppose we wish to maximize f(**x**) subject to the constraints $\boldsymbol{\phi}_i$(x)=0.
- We form the function g(**x**)= f(**x**) + $\sum_{l=1}^{q} \lambda_l \phi_l(\boldsymbol{x})$, where q is the number of constraints.
- The λ_l are called Lagrange multipliers.
- Then we set the derivatives of g to 0: $\frac{\partial g}{\partial x_j} = 0$.
- Finally, we solve for the λ_l.

Lagrange multipliers: example

- Find the maximum and minimum of $f(x, y, z) = \ x^2 + y^2 + z^2$ subject to $\phi_1(x, y, z) = \frac{x^2}{4} + \frac{y^2}{5} + \frac{z^2}{25} - 1 = 0,$ and $\phi_2(x, y, z) = x + y\ - z = 0.$

- Put $g(x, y, z) = f(x, y, z) + \lambda_1 \phi_1(x, y, z) + \lambda_2 \phi_2(x, y, z).$

- $\frac{\partial g}{\partial x} = 2x + \lambda_1 \frac{x}{2} + \lambda_2, \frac{\partial g}{\partial y} = 2y + \lambda_1 \frac{2y}{5} + \lambda_2\ , \frac{\partial g}{\partial z} = 2z + \lambda_1 \frac{2z}{25} - \lambda_2$
- Solve for x, y, z in terms of λ_1 and λ_2 and substitute into second constraint to get:
  - $\frac{2\ \lambda_2}{\lambda_1 + 4} + \frac{5\ \lambda_2}{2\lambda_1 + 10}\ + \frac{25\ \lambda_2}{2\lambda_1 + 50}$ = 0.
- We get a quadratic equation in λ_1 whose solution is λ_1 = -10, λ_1 = $\frac{-75}{17}$ and λ_2 = ±6$\sqrt{\frac{5}{19}}$.
- λ_1 = -10 gives (x, y, z) = (2$\sqrt{\frac{5}{19}}$, 3$\sqrt{\frac{5}{19}}$, 5$\sqrt{\frac{5}{19}}$) and (x, y, z) = (12$\sqrt{\frac{5}{19}}$, -3$\sqrt{\frac{5}{19}}$, -5$\sqrt{\frac{5}{19}}$)
- λ_1 = $\frac{-75}{17}$ gives (x, y, z) = ($\frac{40}{\sqrt{646}}$, $\frac{-35}{\sqrt{646}}$, $\frac{5}{\sqrt{646}}$) and (x, y, z) = ($\frac{-40}{\sqrt{646}}$, $\frac{35}{\sqrt{646}}$, $\frac{-5}{\sqrt{646}}$).
- Substituting gives a maximum of 10 and a minimum of $\frac{75}{17}$.

Likelihood

- What is the likelihood of the output y, given **x** and the parameters $\boldsymbol{\theta}$? That is, given $\boldsymbol{\theta}$, how likely is it that we will observe the data **(x**(k), y(k)), k = 1, …, m.
- Put L($\boldsymbol{\theta}$)= p(y|X,$\boldsymbol{\theta}$). Often the log of this is easier to calculate: ℓ($\boldsymbol{\theta}$)= ln(L($\boldsymbol{\theta}$)).
- If we assume errors are normally distributed with mean 0 and variance σ^2, putting $\epsilon(k) = y(k) - h_\theta\big(\boldsymbol{x}(k)\big)$, we get $P\big(\epsilon(k)\big) = \frac{1}{\sqrt{2\pi}\sigma}\ \exp(-\frac{\epsilon(k)^2}{2\sigma^2})$.
- We often try to learn parameters by adjusting them to maximize the likelihood of the observed outcomes given the data and parameters. This is called maximum likelihood estimation (or "MLE").

Maximum Likelihood Estimation (MLE)

- Suppose we observe the sequence $\mathbf{x} =< x_1, x_2, \ldots, x_n >$ from a gaussian distribution and we want to estimate the parameters $\theta = (\mu, \sigma)$.
- Likelihood is $\mathcal{L}(\boldsymbol{x}|\boldsymbol{\theta}) = \prod_{i=1}^{n} P(x_i|\theta)$
- log likelihood is $\ell(\theta; \boldsymbol{x}) = \ln\big(\mathcal{L}(\theta; \boldsymbol{x})\big) = \sum_{i=1}^{n} \ln(P(x_i|\theta))$
- MLE maximizes $\mathcal{L}(\boldsymbol{x}|\boldsymbol{\theta})$, that is, the probability that **x** is observed given θ.
- $\theta' = \arg max_{\theta} (\ell(\theta; \boldsymbol{x}))$
- $\mu' = \frac{1}{n} \sum_{i=1}^{n} x_i$
- $\sigma^{2'} = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu')^2$
- $E(\sigma^{2'}) = \frac{n-1}{n} \sigma^2$

MLE example

- 400 samples from a normal distribution:
  - Real mu: 0.000, real sigma: 1.000, computed mu: -0.027, computed sigma: 0.878
  - Real mu: 4.000, real sigma: 1.000, computed mu: 3.995, computed sigma: 1.066
  - Real mu: 4.000, real sigma: 2.000, computed mu: 3.931, computed sigma: 1.096

Data mining and analysis, part 5, logistic regression

- Binary logistic regression
    - Sigmoid
    - Binary logistic regression
- General logistic regression
    - General logistic regression
    - LDA

The sigmoid function

- If we stick to linear functions as predictors of behavior, we will often be disappointed but linear functions provide efficient methods to help us improve parameters, as we’ve seen.
- Logistic regression employs a specific non-linear function called a sigmoid to expand the possible predictors for behavior.
- $g(z) = \frac{1}{1+e^{-z}}$
- $g'(z) = g(z)(1-g(z))$
- Our hypothesis will now take the form $h_{\boldsymbol{\beta}}(\mathbf{x}) = g(\boldsymbol{\beta}^T x)$
- One bit of notation, 1[condition] = 1 if the condition is true, 0 otherwise. Example: 1[x=x]=1, 1[2=3]=0.

Binary logistic regression

- Training data is ($\mathbf{x}$(k), y(k)), k= 1,2, …, m, y is binary (two classes).
- To predict y, we form hypothesis p(y=1|$\mathbf{x}$,$\boldsymbol{\theta}$) = $h_{\boldsymbol{\theta}}(\mathbf{x})$, so p(y=0|x,$\boldsymbol{\theta}$) = 1-$h_{\boldsymbol{\theta}}$(x).
- $h_{\boldsymbol{\theta}}(\mathbf{x})$= $g_{\text{sigmoid}}(\boldsymbol{\theta}^{T}\mathbf{x})$
- p(y|x,$\boldsymbol{\theta}$) = $h_{\boldsymbol{\theta}}(x)^{y}(1- h_{\boldsymbol{\theta}}(x))^{(1-y)}$, this is a Bernoulli distribution.
- $\ell(\boldsymbol{\theta})$= $\sum_{i=1}^{m} y(i)\ln(h_\theta\big(\boldsymbol{x}(i)\big) + \big(1-y(i)\big)\ln(1-h_\theta\big(\boldsymbol{x}(i)\big))$
- $\frac{\partial l}{\partial \theta_j}$ = (y- $h_{\boldsymbol{\theta}}(\mathbf{x})$)$\mathbf{x}_j$
- Solve this with gradient descent. The update rule is:
  - $\theta_j \;\coloneqq\; \theta_j - \;\alpha\sum_{i=1}^{m}\frac{\partial l}{\partial \theta_j}\;\boldsymbol{x}(i)_j = \theta_j + \alpha\sum_{i=1}^{m} y(i) - h_\theta\big(\boldsymbol{x}(i)\big)\,\boldsymbol{x}(i)_j$
  - $\theta_0 \;\coloneqq\; \theta_0 - \;\alpha\sum_{i=1}^{m}\frac{\partial l}{\partial \theta_0} \;= \theta_0 + \alpha\sum_{i=1}^{m} y(i) - h_\theta\big(\boldsymbol{x}(i)\big)$

Binary logistic regression - example

- Features are: grade in previous class, hours of study, number of lectures attended. Want to predict whether you pass the class.
- It’s important to normalize the data.
- 60 samples.
- Features normalized and constant weight set to 1
- Predicted parameters: -32.8078 (const), 37.1817 (previous grade), 25.0517 (hours studied), 26.4033 (lectures attended)
- Results

  features test 1: 1.0 0.77 (B+), 0.82 (33hrs), 0.90 (ratio of lectures) , prob pass: 1.0
  features test 2: 1.0 0.22 (C-) 0.32 0.57 , prob pass: 0.19

General logistic regression

- K possible outcomes (classes): $C_1, C_2, \dots, C_K$. Training data is $(\phi_n, t_n), n = 1,2,\dots, N$. ϕ_n is in an m dimensional feature space. We convert t_n into a vector of dimension K by setting $\boldsymbol{t}_{nk} = 1$ for $k = t_n$ and 0 otherwise; so it is a feature indicator.
- $p(y = k|\phi) = y_k(\phi) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$ (softmax), $a_k = {w_k}^T\phi$, the w_k are the m-dimensional parameters.
- $\boldsymbol{t}_n$ is a target vector for sample n, $t_{nk} = 1$, if sample ϕ_n belongs to class k.
- $T = (t_{nk})$ is a $N \times K$ matrix, so is (y_{nk}).
- $\frac{\partial \boldsymbol{y}_k}{\partial a_j} = \left(I_{kj} - \boldsymbol{y}_j\right)\boldsymbol{y}_k$, j= 1, …, m.
- $p(T|w_1, w_2, \dots, w_K) = \prod_{n=1}^{N} \prod_{k=1}^{K} p(C_k|\phi_n)^{t_{nk}} = \prod_{n=1}^{N} \prod_{k=1}^{K} {y_{nk}}^{t_{nk}}$.
- Log-likelihood expectation is $E(w_1, \dots, w_K) = -\ln(p(T|\, w_1, \dots, w_K) \;= -\sum_{n=1}^{N} \sum_{k=1}^{K} \ln(y_{nk}) t_{nk}$
- $\nabla_{w_j} E(w_1, \dots, w_K) = \sum_{n=1}^{N} (y_{nj} - t_{nj})\phi_n$
- Find MLE by gradient descent: $(w^{r+1})_k = (w^r)_k - \eta \nabla E_k$, η is the learning rate, and, E_k is kth row of $\nabla_{w_j} E(w_1, \dots, w_K)$.

General logistic regression example

- Same data as binary logistic example but now we want to predict final grade.
- Features normalized and constant weight set to 1. After run:

```
features test 1: 1.0000 0.7778 0.8250 0.9000
prediction:  0.0000(F), 0.0000(D), 0.0001(C-), 0.0004(C), 0.0069(C+),
             0.1152(B-), 0.2146(B), 0.2492(B+), 0.2271(A-), 0.1866(A)
features test 2:   1.0000   0.2222   0.3250   0.5667,
 prediction:  0.0821(F), 0.1745(D), 0.1730(C-), 0.1746(C), 0.0957(C+),
               0.1289(B-), 0.0133(B), 0.1578(B+), 0.0000(A-), 0.0000(A)
```

Linear Discrimination Analysis (LDA)

- Classify into one of k classes.
- LDA uses Bayes theorem to estimate: $P(Y = 1|x = k) = \frac{P(X = x|Y = k)P(y=k)}{P(X=x)}$
- $\Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}$
- $f_k(x) = \frac{1}{\sqrt{(2\pi)}\sigma_k} \exp(-\frac{1}{2{\sigma_k}^2}\ (x - \ \mu_k\ )^2)$
- $p_k(x) = \frac{\pi_k \frac{1}{\sqrt{(2\pi)}\sigma} \exp(-\frac{1}{2\sigma}(x - \ \mu_k\ )^2)}{\sum_{l=1}^{K} \pi_l \frac{1}{\sqrt{(2\pi)}\sigma} \exp(-\frac{1}{2\sigma}(x - \ \mu_k\ )^2)}$
- Estimate mean as $\pi_k' = \frac{1}{n_k} \sum_{i:y_i=k} x(i)$, ${\sigma_k}^2 = \frac{1}{n-K} \sum_{k=1}^{K} \sum_{i:y(i)=k} (x(i) - {\mu_k}')^2$, $\pi_k' = \frac{n_k}{n}$
- Given x, predict k that maximizes $\delta_k'(x) = x\,\frac{{\mu_k}'}{{{\sigma_k}'}^2} - \frac{{\mu_k}^{2\prime}}{2{\sigma_k}^{2}{\prime}} + \ln({\pi_k}'))$

LDA example

- Same data as logistic example but now we want to predict final grade.
- We are given an initial grade distribution.
- Todo

Data mining and analysis, part 6, classification

- Classification
- Naïve Bayes
  - Laplace smoothing
- KNN
- SVM
  - Optimization and margins
  - Duality
  - Regularization
  - Platt’s SMO
  - What happens if there’s not a separating hyperplane?

Classification: overview

- Classification seeks to predict the category of an observed data set. Medical diagnosis is an example. For example, given a set of symptoms, syndromes and medical tests, does a patient have the flu, meningitis, AIDS? There are several ways to judge classification efficacy. Let TP be the number of true positive predictions, TN be the number of true negative predictions, FP be the number of false positive predictions, and FN be the number of false negative predictions in a set of experiments.
  - Accuracy is defined as: (TP+TN)/(TP+TN+FP+FN)
  - Precision is defined as : TP/(TP+FP)
  - Recall is defined as: TP/(TP+FN)
- We'll study several classification methods including:
  - Naïve Bayes
  - Support Vector Machines (SVM)
  - Neural networks

Naïve naïve Bayes

- Suppose X= (X_1, X_2, …, X_r) and we wish to either predict an outcome or classify a characteristic based on a collection of training data (**x**(k), y(k)) like the data on the right. Here, we are trying to guess whether someone will evade taxes. But this method works when Y can take on more than two values.
- We assume conditional independence: $P(X_1,X_2,\ldots X_r|Y)= P(X_1|Y) P(X_2|Y)\ldots P(X_r|Y)$ --- this is the "naïve" part.
- P(Y|X)= P(X|Y)P(Y)/P(X) by Bayes rule. Using conditional independence, it's easy to see we have all the required information for the calculation of P(Y|X).
- For example, P(evade=Yes|Yes, S,125) = P(Yes|yes)P(Yes|S)P(Yes|125)/P(yes,S,125).
- This example uses very little training data and one of the features (income) is continuous. We can put continuous feature values into buckets to carry out the analysis or, model them as normal distributions with parameters.

Refund	Status	Income	Evade
Yes	S	125	No
No	M	100	No
No	S	70	No
Yes	M	120	No
No	D	95	Yes
No	M	60	No
Yes	D	220	No
No	S	85	Yes
No	M	75	No
No	S	90	Yes

From Tan, Steinbach, et al

Naïve Bayes: fancier treatment

- We can apply maximum likelihood information estimation to a binary classification problem with two classes, 0 and 1. Again, we have training data ($\mathbf{x}$(k),y(k)), k= 1, 2,…, m, where $\mathbf{x}$(k) has dimension r.
- Assume conditional independence: P(X_1,X_2,…X_r|Y)= P(X_1|Y) P(X_2|Y)…P(X_r|Y).
- Let $h_{\boldsymbol{\theta}}$(X) be the hypothesis (predictor of classifier) where $h_{\boldsymbol{\theta}}(\mathbf{x}) = g_{\text{sigmoid}}(\boldsymbol{\theta}^T\mathbf{x})$. For binary classification, P(y=1|X, $\boldsymbol{\theta}$) = $h_{\boldsymbol{\theta}}$(X) and P(y=0|X, $\boldsymbol{\theta}$) = 1-$h_{\boldsymbol{\theta}}$(X). We can write this as P(y|X, $\boldsymbol{\theta}$) =$h_{\boldsymbol{\theta}}(X)^y(1-h_{\boldsymbol{\theta}}(X))^{1-y}$.
- The likelihood can be calculated as L($\boldsymbol{\theta}$)= P(y|X; $\boldsymbol{\theta}$) = $\prod_{k=1}^{m} p(y(k)|x(k),\boldsymbol{\theta})$. ℓ($\boldsymbol{\theta}$) = ln(L($\boldsymbol{\theta}$)).
- We can use gradient ascent to determine the parameters ($\boldsymbol{\theta}$). The maximization step is $\boldsymbol{\theta}$:= $\boldsymbol{\theta}$ + $\alpha\nabla\ell(\boldsymbol{\theta})$. Here is that calculation:
- $\frac{\partial l(\Theta)}{\partial \theta_j}$ = [y(k) 1/g($\boldsymbol{\theta}^T\mathbf{x}$(k)) – (1-y(k)) 1/(1- g($\boldsymbol{\theta}^T\mathbf{x}$(k)))] $\frac{\partial g(\Theta^T \boldsymbol{x}(k))}{\partial \theta_j}$
- This reduces as in the least squares case to θ_j := θ_j + α(y(k)- $h_{\boldsymbol{\theta}}$($\mathbf{x}$(k)) $\mathbf{x}(k)_j$ so the same algorithm works.

Naïve Bayes: Laplace smoothing

- One problem illustrated by the data on the right is P(yes|no,S,125) is 0 but likely we just don't have enough data to see a non-zero value.
- A larger source of data and reasonable "discretization" of income would help but generally having P(Y|X)=0 is problematic.
- One way to fix this is Laplace smoothing where we replace P(Y=a|X_1=b) = $\frac{n_a}{n_b}$ with $\frac{1+n_a}{n_a+n_c}$ where n_c is the number of classes for Y (for binary classification, n_c=2).

Refund	Status	Income	Evade
Yes	S	125	No
No	M	100	No
No	S	70	No
Yes	M	120	No
No	D	95	Yes
No	M	60	No
Yes	D	220	No
No	S	85	Yes
No	M	75	No
No	S	90	Yes

Data from Tan, Steinbach, et al

Example: Document classification using Naïve Bayes

- Set of labeled data consisting of documents (**x**(k), y(k)), determine what class a new document, d, belongs to. Represent the documents as vector of words: $\boldsymbol{x}_j = 1$ if word j is in document, d.
- Possible classifications: spam, source of news story, research paper, fake news.
- Models:
  - c classes. $\theta_c = \{\theta_{c1}, \theta_{c2}, \dots, \theta_{c|D|}\}$. θ_{cj} = P(word j occurs in document from class c).
  - $\sum_j \theta_{cj} = 1$, $P(d|\theta_c) = \frac{\left(\sum_j x_j\right)!}{\prod_j x_j!} \prod_j {\theta_{cj}}^{x_j}$
- Compute dictionary D over training set.
- Estimate class priors by counting: $\theta_{cj} = \frac{N_{cj}+1}{N_c+|D|}$
- $h(d) = \arg max_c \left(\ln(P(c) + \sum_j x_j {w_{cj}}'\right)$, where $w'_{cj} = \ln(\theta'_{cj})$
- Likelihood: $P(d|\theta_c) = \frac{\left(\sum_j x_j\right)!}{\prod_j x_j!} \prod_j (\theta_{cj})^{x_j}$, $\ln(\theta_c|d) = \ln(P_c) + \sum_{j=1}^{|D|} x_j \ln(\theta_{cj})$.

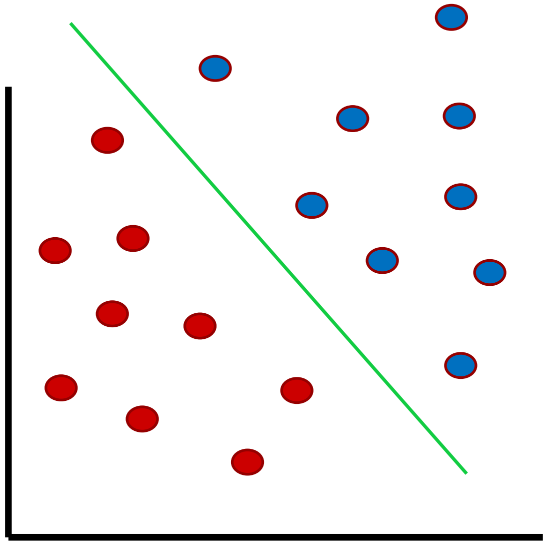
Credit: Alina Opera

kNN

- Find k nearest points to x in training set.
- Use majority vote.
- Somewhat computationally expensive during operational phase.

Classification: SVM

- Imagine we have two classes we wish to predict from data.
- In the picture on the right, data points are colored to indicate whether they represent a "red" characteristic or a "blue" characteristic. In this case, the red and blue class is separated by a line (more generally, a hyperplane) and the "learning" problem is to determine the hyperplane, separating the values for which the minimum distance from the points and the hyperplane and which side of the hyperplane determines which characteristics.
- The margin is the minimum distance from data points and the separating hyperplane. The margin could be measured vertically, horizontally or orthogonally to the hyperplane.
- Some background: A hyperspace in R^p is the set of points, **x**, satisfying $w_1x_1 + w_2x_2 + \ldots + w_px_p + w_0 = 0$. One side of the hyperplane consists of the points, **x**, such that $w_1x_1 + w_2x_2 + \ldots + w_px_p + w_0 > 0$ and the other side consists of the points, **x**, such that $w_1x_1 + w_2x_2 + \ldots + w_px_p + w_0 < 0$.



SVM: Optimization problem

- Again, we have m training points ($\mathbf{x}(k)$, $y(k)$), $y(k) = \pm 1$.
- $h_{w,b}(\mathbf{x}) = g(w^T\mathbf{x}+b)$ is the classifier where $g(z) = 1$ if $w^T\mathbf{x}+b>0$ and $g(z) = -1$ if $w^T\mathbf{x}+b<0$ (so the "bins" are 1, -1, rather than 1, 0)
- Put $\gamma(k) = y(k)(w^T\mathbf{x}(k) + b)$ and $\gamma = \min_k \gamma(k)$, k = 1,2,…,m, we want to find w to maximize γ.
- Problem1: Find $\max_{\gamma,w,b} \gamma$ such that $y(k)(w^T\mathbf{x}(k)+b) \geq \gamma$
  - This is the optimum margin classifier problem. We could form the Lagrangian and solve this but there is an efficient convex optimization algorithm we could use with a little more work.
- Problem 2: Find $\max_{\gamma',w,b} \gamma'/||w||$, such that $y(i)(w^T\mathbf{x}(i)+b) \geq \gamma'$, $\gamma = \gamma'/||w||$.
- Problem 3: Put $\gamma = 1$ find $\min_{w,b} \frac{1}{2}||w||^2$, such that $y(i)(w^T\mathbf{x}(i)+b) \geq 1$.

SVM: Optimization problem

- Problem 1 is the problem we want to solve but it has a non-convex constraint. It can be transformed to problem 2 but it has γ'/||w||, which is better but not ideal. By scaling, so that γ'/||w|| =1/||w||, we get problem 2.
- Problem 1 and problem 2 can be solved with Lagrange multipliers by putting L(w, β) = $f(w) + \sum_{j=1}^{m} \beta_j h_j(w)$. However, we can can transform problem 3. to problem 3. Problem 3 can be solved with an efficient linear programming technique.
- This final transformation uses a dual problems to enable the use of the Karush-Kuhn-Tucker formulation and a very efficient algorithm due to John Platt to finally solve the optimal margin classifier problem.

SVM: Lagrange Duality

- Primal optimality problem, $\boldsymbol{\Theta}_p$(w): $\max_{\boldsymbol{\alpha},\boldsymbol{\beta}}$f(x), $g_i(\boldsymbol{\alpha}) \leq 0$, I = 0,1,…k, $h_j(\boldsymbol{\beta})$=0, j= 1,2,…l
  - L(w, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$) = f(w) + $\sum_{i=1}^{k} \alpha_i g_i(w)$ + $\sum_{i=1}^{l} \beta_i h_i(w)$
  - $\boldsymbol{\Theta}_p$(w): $\max_{\boldsymbol{\alpha},\boldsymbol{\beta}}$ L(w, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$)
  - Then we solve $\min_w$ $\boldsymbol{\Theta}_P$(w).
- Dual optimality problem, $\boldsymbol{\Theta}_D$($\boldsymbol{\alpha}$, $\boldsymbol{\beta}$): $\min_w$f(x), $g_i(\boldsymbol{\alpha}) \leq 0$, I = 0,1,…k, $h_j(\boldsymbol{\beta})$=0, j= 1,2,…l
  - L(w, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$) = f(w) + $\sum_{i=1}^{k} \alpha_i g_i(w)$ + $\sum_{i=1}^{l} \beta_i h_i(w)$
  - $\boldsymbol{\Theta}_D$($\boldsymbol{\alpha}$, $\boldsymbol{\beta}$): $\min_w$ L(w, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$)
  - Then we solve $\max_{\boldsymbol{\alpha},\boldsymbol{\beta}}$ $\boldsymbol{\Theta}_D$(w).
- $\max_{\boldsymbol{\alpha},\boldsymbol{\beta}}$ $\boldsymbol{\Theta}_D$(w) ≤ $\min_w$ $\boldsymbol{\Theta}_P$(w). However, these problems have the same solution if the f and g_i are convex, the $g_i(w)$ constraints are strictly feasible, and the $h_j(\boldsymbol{\beta})$ are linear.
- The $g_i(w)$ are strictly feasible if ∃w: $g_{i(w)} < 0, \forall \mathrm{i}$.
- Under these circumstances, the solution can be found using a linear programming like algorithm (KKT).

Regularization

- Before we solve the optimization problem, using SMO, we generalize a little. We often need to introduce slack variables, ξ_{i} when the the data is not linearly separable. With slack variables, the dual optimization problem becomes:
  - $\min_{\gamma,\mathrm{w,b}}\left(\frac{1}{2}\,||w||^2 + C\sum_{i=1}^{m}\xi_i\right), such\ that\ y(i)(w^t\,\boldsymbol{x}(i)+b) \geq 1-\xi_i\,),\ \xi_i \geq 0,\ i=1,\ldots,m$
  - In the dual problem formulation of problem 3, $g_i(w) = 1 - y(i)(w^T\boldsymbol{x}(i)+b)$ and there are no h_i constraints
  - $\mathcal{L}(w,b,\xi,\alpha,r) = \frac{1}{2}\,w^Tw + \mathrm{C}\sum_{i=1}^{m}\xi_i\ -\ \sum_{i=1}^{m}\alpha_i[y(i)(x^Tw+b)-1+\xi_i]\ -\ \sum_{i=1}^{m}r_i\xi_i$, and
  - $\max_{\boldsymbol{\alpha}}$ W($\boldsymbol{\alpha}$) = $\sum_{i=1}^{m}\alpha_i\ -\ \frac{1}{2}\sum_{i,j}^{m}y(i)y(j)\alpha_i\alpha_j\left(\boldsymbol{x}(i)^T,\boldsymbol{x}(j)\right)$, with $0 \leq \alpha_i \leq C$, and $\sum_{i=1}^{m}\alpha_i y(i) = 0$
  - $w =\ \sum_{i=1}^{m}\alpha_{\mathrm{i}}y(i)\boldsymbol{x}(i)$
  - $\frac{\partial\mathcal{L}}{\partial b} = -\sum_{i=1}^{m}\alpha_i y(i) = 0$
  - C is a weighing between making$||w||^2$ large and ensuring that the functional margin is usually at least 1.
- The KKT complementarity conditions become:
  - $\alpha_i = 0\ \rightarrow y(i)(w^t\,\boldsymbol{x}(i)+b) \geq 1$
  - $\alpha_i = C\ \rightarrow y(i)(w^t\,\boldsymbol{x}(i)+b) \leq 1$
  - $0 \leq \alpha_i \leq C\ \rightarrow y(i)(w^t\,\boldsymbol{x}(i)+b) = 1$

Platt’s sequential minimal optimization (SMO) algorithm

- SMO algorithm

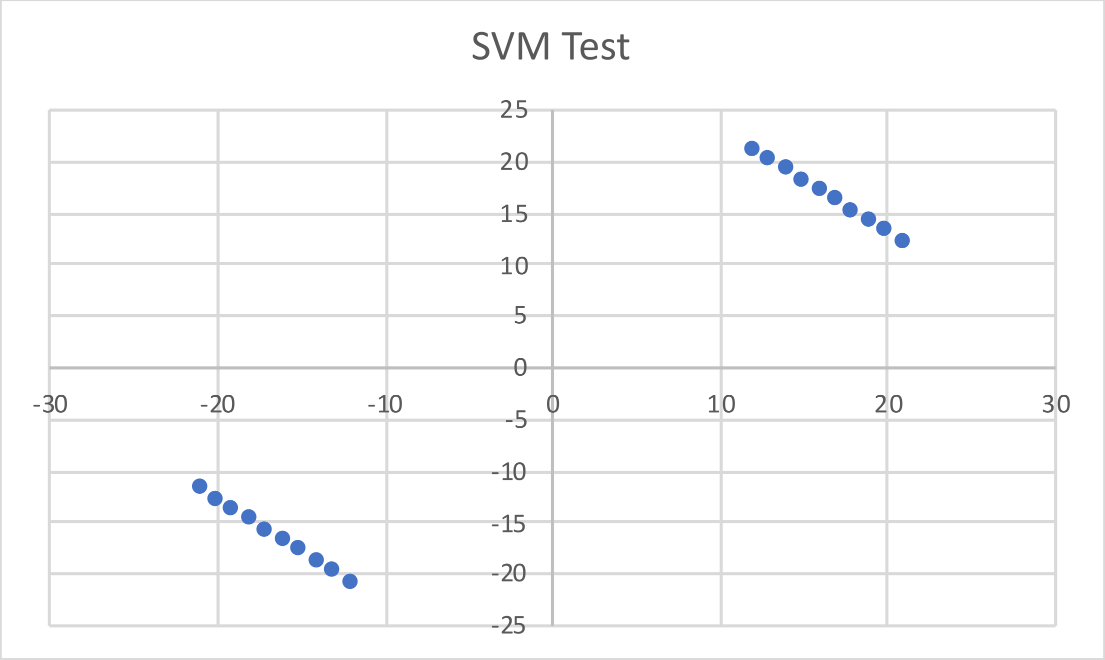
```
Repeat until convergence
    Pick $i, j$: $y(i) - \sum_{i=1}^{m} \alpha_i (w(\boldsymbol{x}(i), \boldsymbol{x}(j)) + b)$ is largest
    Reoptimize W($\boldsymbol{\alpha}$) changing only $\alpha_\mathrm{i}$, $\alpha_\mathrm{j}$.
    At optimization step, $\zeta$ = y(i)$\alpha_\mathrm{i}$ + y(j)$\alpha_\mathrm{j}$ is constant and L ≤ $\alpha_\mathrm{j}$ ≤ H, where $\zeta = -\sum_{i=3}^{m} \alpha_i y(i)$.
    Since $\alpha_i = \frac{\zeta - y(j)\alpha_j}{y(j)}$, $W(\alpha)$ becomes a quadratic function in $\alpha_j$, which is easy to optimize,
    we call the solution $\alpha_\mathrm{j}$’.
    Setting L = $\alpha_i - \alpha_j$ and H= min(C,  C + $\alpha_i - \alpha_j$),
        if L ≤ $\alpha_\mathrm{j}$’ ≤ H , put $\alpha_\mathrm{j}$= $\alpha_\mathrm{j}$’
        if $\alpha_\mathrm{j}$’ ≤ L, put $\alpha_\mathrm{j}$= L
        $\alpha_\mathrm{j}$’ ≥ H. and H if $\alpha_\mathrm{j}$’ ≥ H.
```

What happens when there is not a separating hyperplane?

- Introduce slack variables to deal with noisy data (see regularization)
  - y(i)= 1 if $w^T x(i)+b \geq 1-\xi_I$
  - y(i)= -1 if $w^T x(i)+b \geq -1+\xi_i$
- Transform into higher dimensional problem
  - $\boldsymbol{\Phi}(x_1, x_2) = (x_1{}^2, x_2{}^2, \sqrt{2x_1x_2})$, or
  - $x_1{}^2 + x_2{}^2 - x_1 - x_2$, $\boldsymbol{\Phi}$: (x_1, x_2) → $(x_1{}^2, x_2{}^2, \sqrt{2}x_1, \sqrt{2}x_2, 1)$
  - Sometimes problem is separable in higher dimension
- $max_\alpha\, W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^m y(i)y(j)\alpha_i\alpha_j\, (\Phi(\boldsymbol{x}(i))^T \Phi(\boldsymbol{x}(j)))$, $\alpha_i \geq 0$, $\sum_{i=1}^m \alpha_i y(i) = 0$
  - Use Kernel trick:$(\Phi(\boldsymbol{x}(i))^T \Phi(\boldsymbol{x}(j))$= K$(\boldsymbol{x}(i), \boldsymbol{x}(j))$

SVM example

- $\mathbf{w}^T\mathbf{x} + b = 0$, $\mathbf{x} = (x, y)$
  - **w**: (0.4961, 0.8682), b: 0.0000
- test point: (8.0000, 8.0000), **w**: (0.4961, 0.8682), b: 0
  - Prediction 1: 1.00
- test point: (-8.0000, -8.0000), **w**: (0.4961, 0.8682), b: 0
  - Prediction 2 : -1.00



Another SVM example

- Todo

Data mining and analysis, part 7, Likelihood maximization

- EM algorithm
- Gaussian mixtures
- Markov chains

Gaussian mixture and the EM Algorithm

- Denote the multi-variate gaussian by $\mathcal{N}(\boldsymbol{x}|\ \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})$, where $\boldsymbol{\mu_k}$ is the mean and $\boldsymbol{\Sigma_k}$ is the variance
- Suppose there are K distributions and we want to find out which one was the source. We introduce the K-dimensional variable z, where $z_k = 1$, if the source distribution is $\mathcal{N}(\boldsymbol{x}|\ \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})$, and 0 otherwise. Let π_k be the probability that the source distribution consists solely of $\mathcal{N}(\boldsymbol{x}|\ \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})$ *so* $\mathrm{P}(z_k = 1) = \pi_k$. With all this, $P(\boldsymbol{z}) = \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{x}|\ \boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})^{z_k}$. For HMM, $\boldsymbol{z}$ represents the latent variables.
- $P(\boldsymbol{x}|\boldsymbol{\pi}, \boldsymbol{z}) = \sum_{k=1}^{K} \pi_k\, \mathcal{N}(\boldsymbol{x}|\ \mu_k, \Sigma_k)$, and $\sum_{k=1}^{K} \pi_k = 1$.
- Finally, $\mathrm{p}(\boldsymbol{x}) = \sum_z p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z}) = \sum_{k=1}^{K} \pi_k\, \mathcal{N}(\boldsymbol{x}|\ \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$
- Put $\gamma_k = p(z_k = 1|\boldsymbol{x})$, then $\gamma_k = p(z_k = 1|\boldsymbol{x}) = \frac{p(z_k=1)p(\boldsymbol{x}\ |z_k=1)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}|\ \boldsymbol{\mu_j}, \boldsymbol{\Sigma_j})}$ by Bayes

Gaussian mixture and the EM Algorithm

- Now let $\boldsymbol{X}$ the random variable for an experiment $\text{consisting of}\ N$ observations $(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N)$ from the source
- $\mathcal{L}(\boldsymbol{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x_n}|\ \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$; and,
- $\ell((\boldsymbol{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \ln\left(\mathcal{L}(\boldsymbol{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})\right) = -\ \sum_{n=1}^{N} \ln\left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x_n}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\right)$
- Set $\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\boldsymbol{x_n}|\ \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x_n}|\ \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$
- Setting $\frac{\partial \ell}{\partial \mu_k} = 0$, we get $-\sum_{n=1}^{N} \gamma(z_{nk}) \Sigma_k (\boldsymbol{x}_n - \mu_n) = 0$
- So $\mu_k{}' = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})\, \boldsymbol{x}_n$, where $N_k = \sum_{n=1}^{N} \gamma(z_{nk})$
- Setting $\frac{\partial \ell}{\partial \Sigma_k} = 0$, $\Sigma_k{}' = \frac{1}{N_k} \sum_{i=1}^{N} \gamma(z_{ik})(\boldsymbol{x}_i - \mu_k)(\boldsymbol{x}_i - \mu_k)^T$
- Finally, we get $\pi_k' = \frac{N_k}{N}$ by maximizing $\ln(p(\boldsymbol{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi})$ with respect to $\boldsymbol{\pi}$ subject to $\sum_{k=1}^{K} \pi_k = 1$, using the usual Lagrange multiplier machinery. $\lambda = -N$.

Summary EM for Gaussian Mixing

- Initialize $(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi})$
- E step
  - Put $\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\boldsymbol{x_n} |\, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x_n} |\, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$
- M step
  - $\mu_k{}^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})\, x_n$
  - $\Sigma_k{}^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})\, (\boldsymbol{x}_n - \mu_k{}^{new})(\boldsymbol{x}_n - \mu_k{}^{new})^T$
  - $\pi_k{}^{new} = \frac{N_k}{N}$
- Check convergence by evaluating
  - $\mathrm{p}(\boldsymbol{X} | \pi, \mu, \Sigma) = \sum_{n=1}^{N} \ln\left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x_n} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\right)$

Generalized EM Algorithm

- EM algorithm

1. Initialize π, μ, Σ
2. E step

   Calculate $p(z|x, \theta^{old})$
3. M step

   Calculate $\theta^{new} = \arg max_{\theta} \mathcal{Q}(\theta, \theta^{new})$

   Repeat until convergence

EM example

Generative model

```
pi[0]:  0.1500,  mu[0]:  0.0000,  sigma[0]:  0.2000
pi[1]:  0.2500,  mu[1]:  2.0000,  sigma[1]:  0.2000
pi[2]:  0.1500,   mu[2]: 4.0000,  sigma[2]:  0.2000
pi[3]:  0.2500,   mu[3]: 6.0000,   sigma[3]: 0.2000
pi[4]:  0.2000,    mu[4]: 8.0000,  sigma[4]: 0.2000
```

Guess of parameters

```
pi_in[0]: 0.2000,  mu_in[0]: 0.0000,  sigma_in[0]: 0.2000
pi_in[1]: 0.2000,  mu_in[1]: 1.5000,  sigma_in[1]: 0.2000
pi_in[2]: 0.2000,  mu_in[2]: 3.0000,  sigma_in[2]: 0.2000
pi_in[3]: 0.2000,  mu_in[3]: 4.5000,  sigma_in[3]: 0.2000
pi_in[4]: 0.2000,  mu_in[4]: 6.0000,  sigma_in[4]: 0.2000
28 iterations, old_likelihood: -24682.7084,  new_likelihood: -24682.7083
```

EM estimate

```
pi_out[0]: 0.1500, mu_out[0]: 0.5991, sigma_out[0]: 0.2000
pi_out[1]: 0.2500, mu_out[1]: 2.3512, sigma_out[1]: 0.2000
pi_out[2]: 0.1500, mu_out[2]: 4.1084, sigma_out[2]: 0.2000
pi_out[3]: 0.2500, mu_out[3]: 6.4742, sigma_out[3]: 0.2000
pi_out[4]: 0.2000, mu_out[4]: 8.6784, sigma_out[4]: 0.2000
```

Markov chains

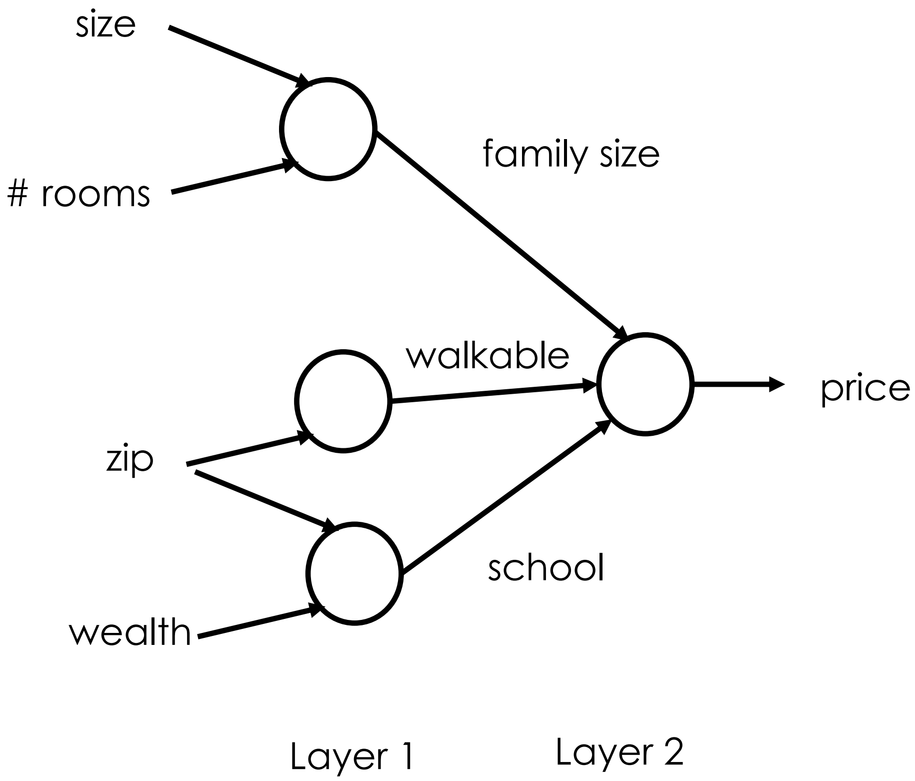
- A Markov chain is a stochastic process involving a sequence of states and transitions from state to state where transitions to next state depend only on current state. Simplest model of dependent time sequences
- Parameters
  - states S={1,2,…, s}
  - initial probabilities π(i), i∈S
  - transition probabilities p_{ij}= probability state i transitions to state j, i, j ∈S
- EM also solves the maximum likelihood estimation of Markov parameters.

Data mining and analysis, part 8, neural nets

- Neural Networks
  - Backpropagation
  - Deep learning

Neural networks

- f($\mathbf{x}$) outputs house price. For NN simplest is f($\mathbf{x}$)= max($a^T\mathbf{x}$+b,0)
- Input to NN are features $x_1, x_2, \ldots, x_{n,}$ outputs are called activations, $a_1, \ldots, a_n$. For single neuron, $z = W^T\mathbf{x} + b$**,** activation function is $a = g_{sigmoid}(z)$ or ReLu(z)= max(z, 0) or tanh(z).
- Notation:
  - Training set ($\mathbf{x}$(k), y(k)), i= 1, 2,…, m, L layer neural net.
  - Input to layer l is $x_1^{[l]}, \ldots, x_n^{[l]}$. $z^{[l]}_i = W^{[l]T}\,\mathbf{x}_i + b^{[l]}_i$. Activation at layer l is $a^{[l]}_i = g(z^{[l]}_i)$.
  - $z^{[1]}_1 = W^{[1]T}{}_1\mathbf{x} + b^{[1]}_1,\ a^{[1]}_1 = g(z^{[1]}_1)$
  - $z^{[1]}_2 = W^{[1]T}{}_2\mathbf{x} + b^{[1]}_2,\ a^{[1]}_2 = g(z^{[1]}_2)$
  - $z^{[1]}_3 = W^{[1]T}{}_3\mathbf{x} + b^{[1]}_3,\ a^{[1]}_3 = g(z^{[1]}_3)$
  - $z^{[2]}_1 = W^{[2]T}{}_1\mathbf{a^{[1]}} + b^{[2]}_1,\ a^{[2]}_1 = g(z^{[2]}_1)$
  - For N layers, we put W = matrix of the $W^{[l]}$. Same for b, a and z.



From Ng

Neural networks

- From last slide, $\mathbf{z}^{[2]}$ = $W^{[2]}$ $\mathbf{g}(\mathbf{a}^{[1]})+\mathbf{b}^{[2]}$ = $W^{[2]}\mathbf{g}(W^{[1]}\mathbf{x}+\mathbf{b}^{[1]}) + \mathbf{b}^{[2]}$ for a two layer NN.
- We could write a more general equation for an L-layer NN.
- The optimization problem for an L layer NN is minimize J where:
  - $\mathcal{L}_i(\boldsymbol{x}(i), y(i)) = -\left[\left(1 - y(i)\right)\ln\left(1 - a^{[L]}(i)\right) + y(i)ln\left(a^{[L]}(i)\right)\right]$, and,
  - $J = \sum_{i=1}^{m} \mathcal{L}_i(\boldsymbol{x}(i), y(i))$
  - We often use the notation $y'(i) = a^{[L]}(i)$.
- The update rule for this optimization problem at each layer is:
  - $W^{[l]}$:= $W^{[l]}$ - $\alpha\frac{\partial J}{\partial W^{[l]}}$, $b^{[l]}$:= $b^{[l]}$ - $\alpha\frac{\partial J}{\partial b^{[l]}}$
  - We can use the chain rule to calculate, say, $\frac{\partial J}{\partial W^{[2]}}$, as
  - $\frac{\partial J}{\partial W^{[2]}}$ = $\frac{\partial J}{\partial a^{[3]}}\frac{\partial a^{[3]}}{\partial z^{[3]}}\frac{\partial z^{[3]}}{\partial a^{[2]}}\frac{\partial a^{[2]}}{\partial Wz^{[2]}}\frac{\partial Wz^{[2]}}{\partial W^{[2]}}$ (Note, we need to align the dimensions of the vectors)

Neural networks

- As we can see, given a multi-layer NN, a modest number of input features and significant data the computational size of the NN optimization problem is enormous compared to other problems we've seen. However, conceptually, the optimization employs the same stochastic gradient descent we've seen several times.
- Procedure

  Initialize $W^{[1]}$, $b^{[1]}$, $W^{[2]}$, $b^{[2]}$, … to small non-zero random values

  Apply Update rules

  Repeat until convergence

Propagation

- Forward propagation
  - $z^{[l]}= W^{[l]}a^{[l-1]}+b^{[l]}$
  - $a^{[l]}= g^{[l]}(z^{[l]})$
- Backward propagation
  - $L(y',y) = \sum_{j=1}^{k} 1[y = j] \ln(y_j j')$

1. $\delta[l] = \nabla_{z[l]} L(y',y)$

   $\nabla_{z[l]} L(y',y)= \nabla_{y'} L(y',y) \cdot g[N]'z[N]$

2. $J(W,b)= L(a^{[N]}, y)= L(y', y)$

   $\delta^{[l]} = (W^{[L+1]} \delta^{[l+1]})\cdot g'[z^{[l]}]$

3. Gradients
   - $\nabla_{W[l]} J(W,b) = \delta^{[l]} a^{[l-1]T}$
   - $\nabla_{b[l]} J(W,b) = \delta^{[l]}$

Regularization

- Suppose the optimized model performs well on training data but significantly worse on test data.
- J$_{L2}$ = $J + \frac{\lambda}{2} + ||W||^2$, the larger λ, the larger the regularization.
- Update rule becomes W := $W - \alpha \frac{\partial L}{\partial W} - \alpha \frac{\lambda}{2} \frac{\partial W^T W}{\partial W} = (1 - \alpha\lambda)W - \alpha \frac{\partial J}{\partial W}$

Deep Learning

- Multi-layer neural networks
- Must pick input features, number of layers, activation function, NN topology
- A multi-layer NN learns an “arbitrary” non-linear function
  - How many layers?
- Advantages
  - Better asymptotic learning with large data
- Disadvantages
  - Huge computational and training data requirements

Neural networks: Small example

- Todo

Other neural network topology

- Feed forward
- Convolutional (feed forward)
- Recurrent
- Todo

Data mining and analysis, part 9 , ensemble methods

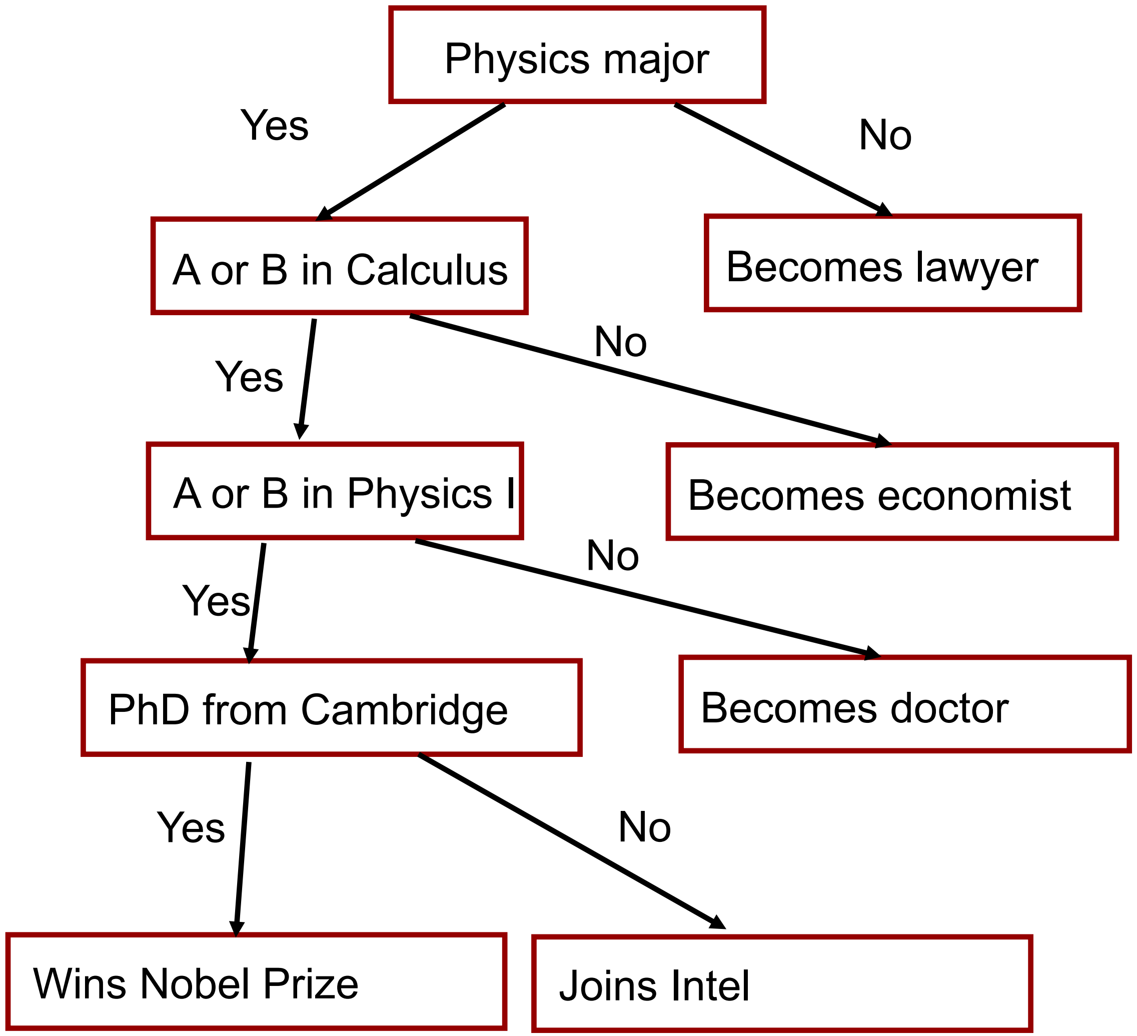
- Measuring how good features are
- Decision Trees
- Principal Component analysis
- Ensemble learning: Adaboost

Measuring how good features are

- The first method is simple but expensive:
  - Collect all the features you can and collect labelled data.
  - Divide the labelled set into two subsets: a training set to train your selected classified and a test set.
  - Train your classifier on every subset of the features and evaluate each of them on the test set.
  - Pick the subset that does best.
- There are heuristics you can use. One is to choose features with maximally independent information.
  - For example, in our ongoing housing example, “number of bedrooms” and “size of house” are very strongly correlated so discarding one is probably a good idea.
  - You can estimate cross feature information using entropy: $I(X|Y)= H(X|Y) – H(Y)$.

Decision trees

- The silly diagram on the right illustrates a decision tree with binary features determining branches. (Apologies for the inaccurate and elitist underlying model.)
- Building a tree requires:
  - Determining features (They do not have to be binary)
  - Knowing what outcomes to predict
  - Determining branching criteria
- Branching criteria is often determined by "predictive power" of decision and discrimination. We want branches that are highly predictive and which efficiently "split" the incoming data equally to shorten the tree. Entropy is a good measure.



Decision trees

- One way to determine efficient splitting criteria is impurity or how dissimilar are class labels for a given node.
- For example, if there are two classes and a node, N, contains 6 observations, using entropy as a measure of impurity (or information).
  - I(N)= -.5 lg(.5) - .5 lg(.5) =1 if there are 3 elements of each class in the node.
  - I(N)= -1/6 lg(1/6)-5/6 lg(5/6) = .65 if there is one element of one class and 5 elements of the other class.
- Suppose we split node N with n elements into c child nodes, N_j, each with n_c elements based on some splitting criteria. Define W(children) = $\sum_{j=1}^{c} \frac{n_j}{n} I(N_j)$ which is a weighted average of the children's impurity. One way to select splitting criteria is to maximize $\Delta_{split} = I(N) - I(children)$.
- For example, in the (3,3) case above if the split results in two nodes with all members of class 1 in N_1 and 3 members in N_2, $\Delta_{split} =$ 1. If the split results in two nodes with N_1 having population (1,2) and N_2 having population (2,1) $\Delta_{split} =$ .09. The first split is far better.

Principal component analysis

- Orthogonal projection of data to lower dimension
    - Maximize variance of projected data
    - Minimize mean squared distance between data points and projections
- First principal component is largest eigenvector of covariance. Same as maximizing binary form subject to ||w|| = 1.
- Let $X = (x_i \ - \mu_i)$. M = XX^T.
- Second principal component is orthogonal to first and second eigenvectors

Credit: Alina Opera

PCA example

- PCA data

```
    1.0000    1.0000    1.0000
    2.0000    1.0000    0.5000
    3.0000    1.0000    0.3333
    4.0000    1.0000    0.2500
    5.0000    1.0000    0.2000
    6.0000    1.0000    0.1667
    7.0000    1.0000    0.1429
    8.0000    1.0000    0.1250
    9.0000    1.0000    0.1111
   10.0000    1.0000    0.1000
```

- Direction with greatest variation: 0.9993 0.0000 -0.0373

Ensemble learning

- Given classifiers h_1, …, h_L, construct classifier H(x) that combines them.
- Requires diversity: different classifiers should make different mistakes
- Adaboost (adaptive bosting) is an example.

Credit: Alina Opera

AdaBoost

- Turn T base learners into high performance classifier by weighting them.

```
Input: Training data, (**x**(i), y(i)), I = 1,…,m
Initialize: $w_1 = (\frac{1}{m}, \frac{1}{m}, \dots, \frac{1}{m})$
for(t=1 to T) {
    Train $h_t$ on training data with weight $\mathbf{w}_t$.
    Compute $\epsilon_t = \sum_{i:y(i) \neq h_t(\boldsymbol{x}(i))} w_{t,i}$, choose $\beta_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$
    Update $w_{t+1,i} = w_{t,i} \exp(-\beta_t \, y(i) h_t(x(i))$
    Normalize $w_{t+1,i} = \frac{w_{t+1,i}}{\sum_{j=1}^{m} w_{t+1,j}}$
}
Return $H(x) = sign(\sum_{t=1}^{T} \beta_t h_t(x))$
```

Credit: Alina Opera

AdaBoost example

- Use logistic regression to determine classifiers based on parameter 1 and parameter 2.
- Combine them using AdaBoost.

Param 1	Param 2	Decision
2	3	true
2.1	2	true
4.5	6	true
4	3.5	false
3.5	1	false
5	7	true
5	3	false
6	5.5	true
8	6	false
8	2	false

From: Sefik Ilkin Serengil

Algorithm comparison

Algorithm	Problem	Interpretable	Explainable	Accuracy	Training speed	Prediction speed
kNN	either	yes	yes	low	fast	Depends on n
Linear regression	regression	yes	yes	low	fast	fast
Logistic regression	classification	sometimes	sometimes	low	fast	fast
Naïve Bayes	classification	often	sometimes	low	fast	fast
Decision trees	either	often	sometimes	low	fast	fast
Random Forest	either	A little	no	high	slow	moderate
AdaBoost	either	A little	no	high	slow	fast
Neural Networks	either	no	no	high	slow	fast

From Alina Opera

But wait, there’s more

- We have not discussed:
  - How we pick features (At least not in depth)
  - Generating training data
  - Confidence measures
  - Detecting data that has been tampered with
  - How fast can we learn?
  - How fast can we predict?

Exercises

1. Classifying images
2. Classifying syndromes
3. Predicting car sales prices
4. Diffusion
5. What data would you collect from IoT devices to predict maintenance issues
6. What can you discover about net revenue from an IoT connected store?
7. Flight patterns
8. IFF problems

References

1. UC, Berkeley, Amplab Tutorials
2. Andrew Ng, Stanford Machine Learning class notes and videos, cs229. Highly recommended.
3. Bishop, Pattern recognition and machine learning.
4. Tan, Steinbach and Kumar, Introduction to Data Mining.
5. Spiegal, Probability and Statistics.
6. Alina Opera, Class notes, Northeastern University.
7. Kevin Murphy, **Machine Learning: a Probabilistic Perspective**

License

-
-

Data analytics

- Collection, provenance and data formatting
- Probability, randomness, information, distributions and significance
- What are we trying to learn?
    - Features, supervised and unsupervised learning
- Regression, similarity, curve fitting and loss functions
- Clustering
- Bayes theorem, maximum likelihood re-estimation, EM and latent variables
- Classification
    - Naïve Bayes
    - SVM
    - Neural networks
- Principal Component analysis and the curse of dimensionality
- Deep learning

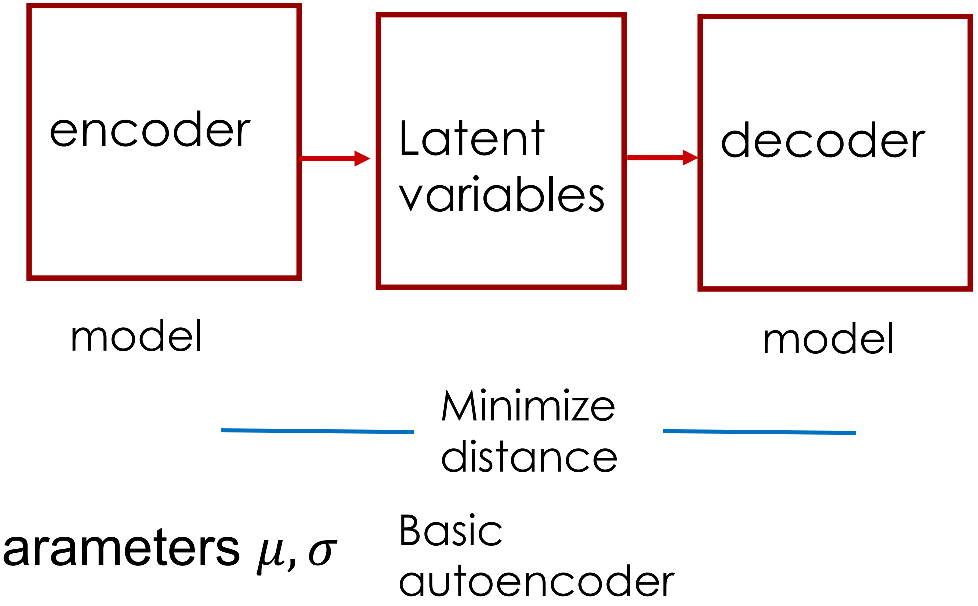
Towards Large Language Models

- Incomplete intro (mostly references) to Large Language Models

Variational autoencoders

- Used for
- Dimensionality reduction
- Compression
- De-noising
- Anomoly detection
- Feature extraction

- For variational, add noise from normal distribution with parameters μ, σ

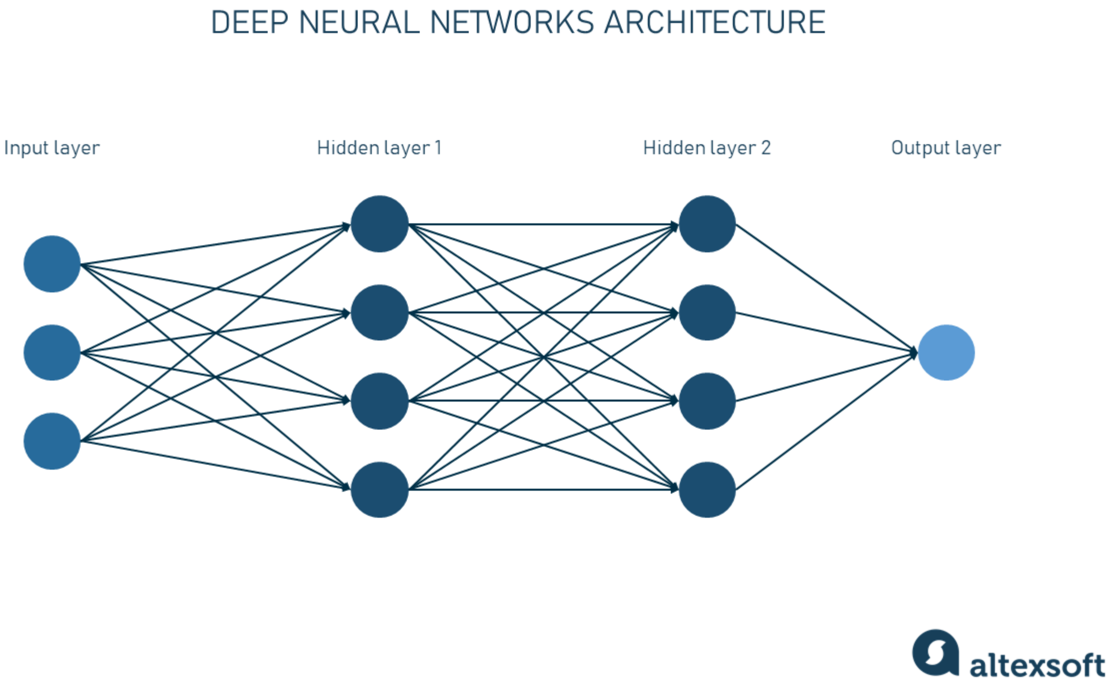


- Reference
  - https://maurocamaraescudero.netlify.app/post/assessing-a-variational-autoencoder-on-mnist-using-pytorch/

RNN

- Language models
- text generation
- Translation
- speech recognition
- text summarization

- References
  - https://medium.com/dejunhuang
  - leewayhertz.com/artificial-intelligence-applica
  - https://www.analyticsvidhya.com/

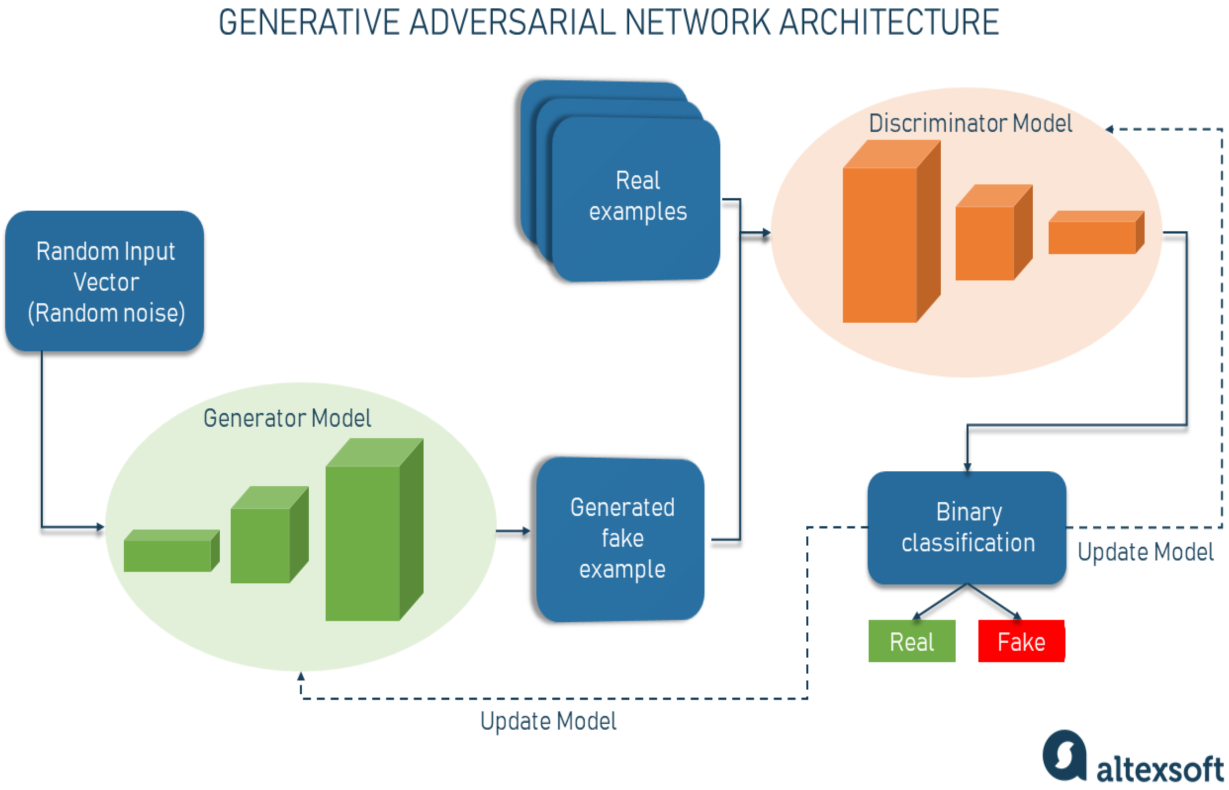


Generative AI

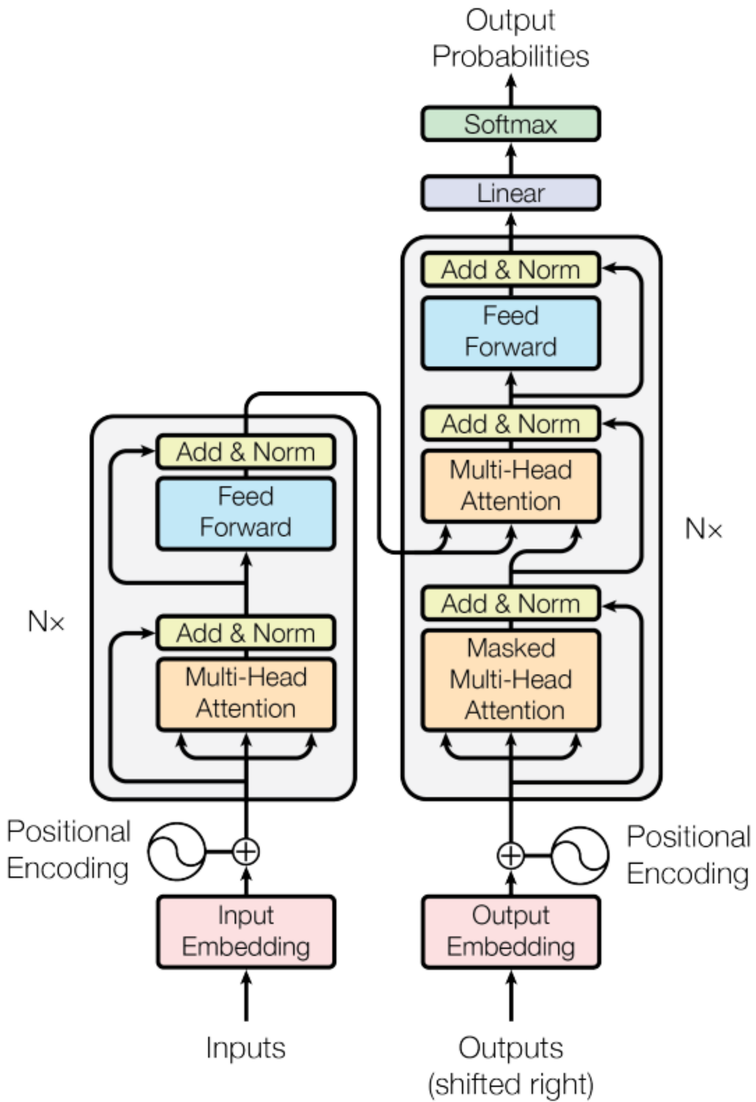
- Reference
  - https://www.altexsoft.com/blog/generative-ai/
- Generative Adversarial Networks
  - Generator and Discriminator are both NN
  - Train them both and add noise

Transformers: Attention Is All You Need

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jak
Gomez, Lukasz Kaiser, Illia Polosukhin



Transformers



Papers

1. Ashish Vaswani et al, Attention Is All You Need (Archiv)
2. Yang et all, XLNet: Generalized Autoregressive Pretraining for Language Understanding (Archiv)
3. Zhu, Label Propagation
4. Lepkhin, Gshard (Archiv)
5. Kingma, Autoencoding Variational Bayes
6. Devlin ey al, BERT