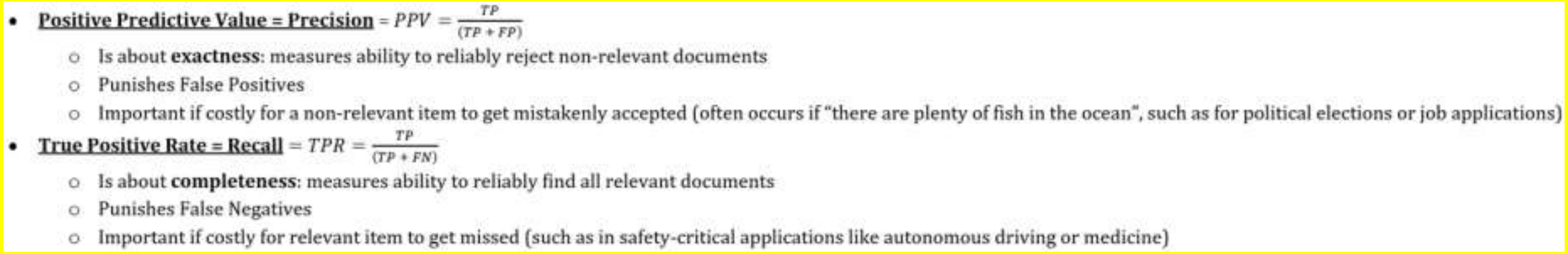
Loss Functions & Metrics

Intuitively, what is precision and recall? When should you pick one over the other?



- **Positive Predictive Value = Precision** = $PPV = \frac{TP}{(TP + FP)}$
  - Is about **exactness**: measures ability to reliably reject non-relevant documents
  - Punishes False Positives
  - Important if costly for a non-relevant item to get mistakenly accepted (often occurs if "there are plenty of fish in the ocean", such as for political elections or job applications)
- **True Positive Rate = Recall** = $TPR = \frac{TP}{(TP + FN)}$
  - Is about **completeness**: measures ability to reliably find all relevant documents
  - Punishes False Negatives
  - Important if costly for relevant item to get missed (such as in safety-critical applications like autonomous driving or medicine)

What is the harmonic mean, and why is it used in the F1 score?

- In general, for averages to be valid, you need the denominator values to be in the same scaled units
- This exactly what the harmonic mean does: it is the reciprocal of the arithmetic mean of the reciprocals of the given set of observations
  - Precision and recall both have true positives in the numerator, and different denominators. To average them it really only makes sense to average their reciprocals, thus the harmonic mean.

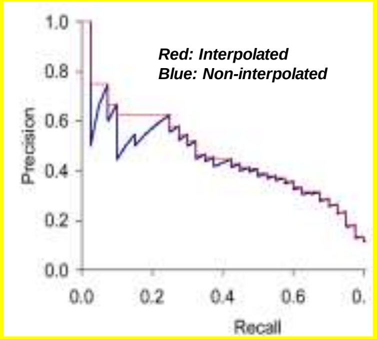
$$F_1 = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{tp}}{\mathrm{tp} + \frac{1}{2}(\mathrm{fp} + \mathrm{fn})}$$

$$\left(\frac{x_1^{-1} + \ldots + x_n^{-1}}{n}\right)^{-1}$$

$$\left(\frac{1^{-1} + 4^{-1} + 4^{-1}}{3}\right)^{-1} = \frac{3}{\frac{1}{1} + \frac{1}{4} + \frac{1}{4}} = \frac{3}{1.5} = 2$$

How do you compute precision/recall @ K? What is a precision-recall curve? What is the difference between average precision (AP) and mean average precision (mAP)?

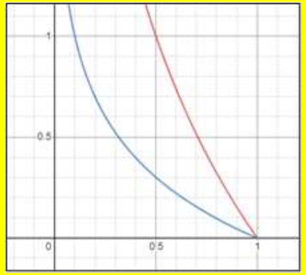
- Suppose we have some **ranked ordering/retrieval** of the positive predictions. Then:
  - **Precision @ K** = (# relevant out of top K) / K = (TP in K) / K
    - Can go up or down as you increase K
  - **Recall @ K** = (# relevant out of top K) / (# relevant in total) = (TP in K) / TP
    - Monotonically increasing; cannot be 1 until K >= # relevant
- Note that true/false negatives are not taken into account; we just want the positives to be ranked highly
- **Precision-Recall (PR) Curves** plots the precision against the recall at all values of K.
  - In general they tend to be inversely proportional (e.g. higher recall -> lower precision)
  - Also, PR curves are more useful when the positive class is rare/interesting. They ignore True Negatives.
  - When plotting these curves, we can **interpolate** and use the best precision for a level of recall R or greater.
    - In some literatures, this is standard (the justification is that almost anyone would be prepared to look at a few more documents if it would increase the percentage of the viewed set that were relevant).
    - However, some view this as being too optimistic, and do not perform the interpolation.
- **Average Precision (AP)** is calculated by finding the area under the precision-recall curve (can be either interpolated or not)
  - For example, scikit-learn only gives the non-interpolated version (see equation, where there are n thresholds)
- **Mean Average Precision (mAP)** is the AP averaged across a dataset (eg, an AP for each class, AP for each IoU thresholds, etc)



$$\mathrm{AP} = \sum_n (R_n - R_{n-1}) P_n$$

$$\mathrm{mAP} = \frac{1}{N} \sum_{i=1}^{N} \mathrm{AP}_i$$

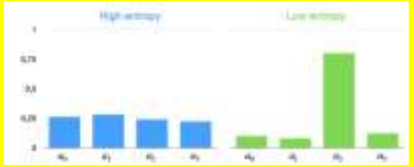
Define Shannon information, entropy, cross entropy, KL/JS Divergence. What are their relationships?



Shannon information: red is b = 2, blue is b = e

Note: the following applies to discrete random variables, though there are extensions to the continuous case.

- **Shannon Information I(X=x)**: Intuitively quantifies the level of “surprise” of a probabilistic realization, by taking the negative log of its probability
  - It’s an inverse relationship; the higher the likelihood/probability of that realization, the lower the shannon information
  - As p approaches 0, the entropy approaches infinity; if p = 1, entropy = 0.
- **Shannon Entropy H(X)**: The average amount of uncertainty of a R.V., i.e. the expected value of the shannon information
  - Higher (lower) entropy means that the R.V. on average has more (less) uncertainty/randomness.
  - Maximized when the distribution is multinoulli uniform; minimized when an event has probability 1 while all others is 0
  - If b=2, we can interpret H(X) as the lower bound for the average number of bits needed to encode a piece of data (X=x). Therefore, besides a notion of uncertainty entropy also represents a way to **quantify the “amount of information” in the data**.
    - For example, if H(X)=0, it’s the same value over and over again; no need to encode.
    - If H(X) is moderate, there are some underlying patterns and tendencies that allow for some optimizations.
    - If H(X) is high, there are many possibilities; need many bits to fully capture data.
    - Suppose we have a fair n-sided die. Then, the entropy becomes H(X)=-log_2(1/n)=log_2(n), which intuitively is the number of bits needed to record outcomes of a dice roll.



- **Cross Entropy H(P,Q)**: Intuitively, average number of **total** bits required to encode data coming from a source with “true” distribution P when we use model Q.
  - If P is a fixed reference distribution, then the cross entropy reaches its minimum H(P) when Q = P (from Gibbs’ inequality).
  - This is how the cross entropy loss is formulated, usually where P is a one-hot ground truth.
  - It is the same as the **negative log-likehood**; their only difference is the way they’re interpreted
  - Also known as the **log loss**
- **KL Divergence D(P||Q)**: average number of **extra** bits required to encode data coming from a source with “true” distribution P when we use model Q.
  - Interpretation 2: expected log likelihood ratio of P and Q which characterizes how much more likely x is drawn from P vs Q. If P = Q, then it is 0. Specifically, E_p [log(P/Q)].
  - The cross entropy of P and Q is equal to the KL divergence of P and Q plus the entropy of P. If P is fixed (e.g. doesn’t change with time), then minimizing the cross entropy is the same as minimizing the KL divergence up to a constant.
- **JS Divergence JSD(P||Q)**: Symmetric version of the KL divergence using a mixture distribution of P and Q.

$$I_X(x) := -\log_b[\mathrm{p}_X(x)]$$

$$H(X) = E[I(X)] = -\sum_i p_X(x_i)\log_b p_X(x_i)$$

$$H(P,Q) = -\sum_{x\in\mathcal{X}} p(x)\log(q(x))$$

$$D_{KL}(P \parallel Q) = \sum_{x\in\mathcal{X}} P(x)\log\left(\frac{P(x)}{Q(x)}\right)$$

$$H(P,Q) = H(P) + D_{KL}(P \parallel Q)$$

$$JSD(P\|Q) = \frac{1}{2}D_{KL}(P\|M) + \frac{1}{2}D_{KL}(Q\|M)$$

$$M = \frac{1}{2}(P+Q)$$

$$H(P,P) = H(P)$$

$$D_{KL}(P\|P) = 0$$

★

What are the differences between the regular and reverse KL divergence?

- In general, the KL divergence is $KL(P||Q) = E_{x \sim P}[\log \frac{P(x)}{Q(x)}]$, where P is the reference/GT/true distribution and Q is the model distribution. However, there's also a notion of a forward and reverse KL divergence:
- **Forward KL**: $KL(p_{data}||p_{model}) = E_{x \sim p_{data}}[\log \frac{p_{data}(x)}{p_{model}(x)}]$
  - Intuitively, mode covering behavior – tries to cover all data region and avoid data modes
  - If the model places mass in places with no data, that's not penalized as much
  - Kind of like incentivizing recall
- **Reverse KL**: $KL(p_{model}||p_{data}) = E_{x \sim p_{model}}[\log \frac{p_{model}(x)}{p_{data}(x)}]$
  - Intuitively, mode-seeking behavior
  - Focuses on making samples look realistic, instead of covering all possible data modes
  - May be OK with mode collapse
  - Kind of like incentivizing precision

Why do we use KL divergence for knowledge distillation rather than cross entropy, as we normally use for classification?

- Usually for classification, the target (label) distribution is one-hot, so the entropy part of the cross-entropy loss is 0
    - So **in this case cross-entropy is equivalent to KL divergence**; it would also be correct to say we're using the KL divergence as a loss function
    - However, the cross-entropy formulation is a bit faster to compute usually in many libraries

- For distillation, we want to compare the student's predicted label distribution to the teacher's, which is not one-hot.
    - This means the entropy portion will be non-zero; it is an additive constant
    - Since the entropy portion does not depend on the network parameters, this doesn't affect things from an optimization standpoint
    - However, it leads to the "optimal" loss being non-zero (i.e. only contributions from entropy portion) and constantly fluctuate depending on the batch, which is not as intuitive or useful.

$$H(P,Q) = -\sum_{x\in\mathcal{X}} p(x)\log(q(x))$$

$$H(P,Q) = H(P) + D_{KL}(P \parallel Q)$$

Cross Entropy

$$H(X) = E[I(X)] = -\sum_i p_X(x_i)\log_b p_X(x_i)$$

Entropy (0 in the one-hot case)

$$D_{KL}(P \parallel Q) = \sum_{x\in\mathcal{X}} P(x)\log\left(\frac{P(x)}{Q(x)}\right)$$

KL Divergence

$$-\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ic}\log\left(f(x_i)_c\right)$$

What is the MSE loss, and how does it compare to the cross entropy loss?

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

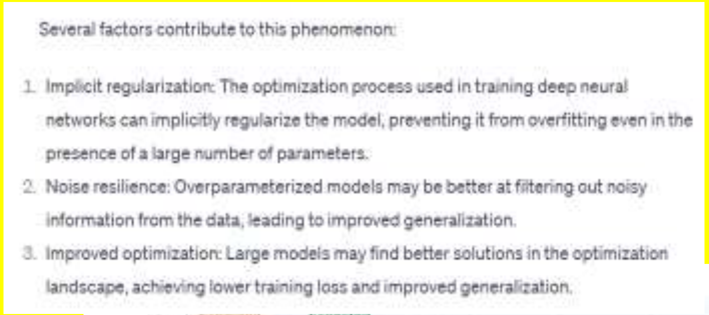
- MSE is typically used for regression, while cross entropy is usually used for classification
    - Although it's possible to use MSE for classification, MSE + softmax is not convex.
    - Also, cross entropy has a better probabilistic interpretation & heavily penalizes wrong predictions with high confidence, encouraging better class separation.

★

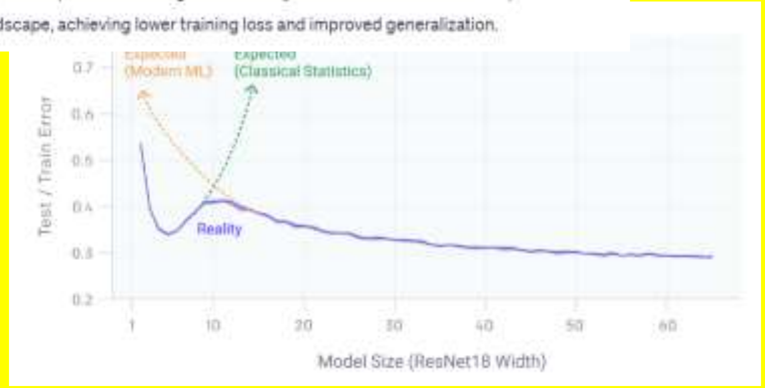
Experimental Designs and Paradigms

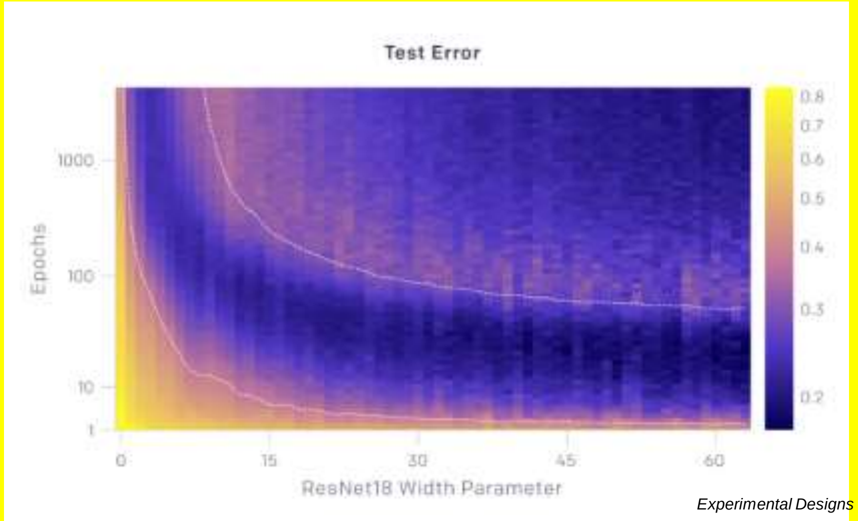
What is the double descent phenomenon?

- Challenges the traditional understanding of the bias-variance trade-off: traditionally, a simple model underfits and an overly complex model overfits
- Suggests that for very overparameterized models, the performance can be not only better than underparameterized models but also better than well parameterized or moderately overparameterized models
- Happens once a model reaches the interpolation threshold (where it can perfectly fit the training data; when there are the same amount of params as data)
- Some theories:
  - Overparameterization allows model to find simpler solutions that generalize better, instead of fitting noise; can isolate signal from noise more effectively, since it has more "room" to express the true function and isolate noise
  - The simpler solution is incentivized due to biases (eg favoring low-complexity solutions via smaller weights) encoded in the loss function


Several factors contribute to this phenomenon:

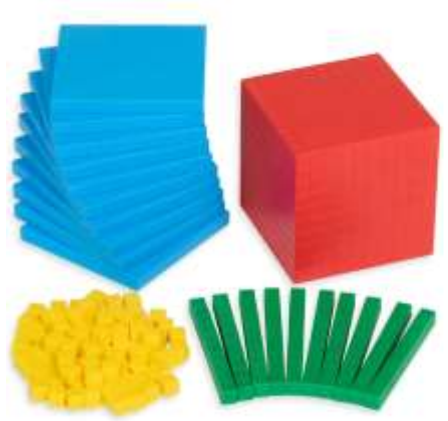
1. Implicit regularization: The optimization process used in training deep neural networks can implicitly regularize the model, preventing it from overfitting even in the presence of a large number of parameters.
2. Noise resilience: Overparameterized models may be better at filtering out noisy information from the data, leading to improved generalization.
3. Improved optimization: Large models may find better solutions in the optimization landscape, achieving lower training loss and improved generalization.





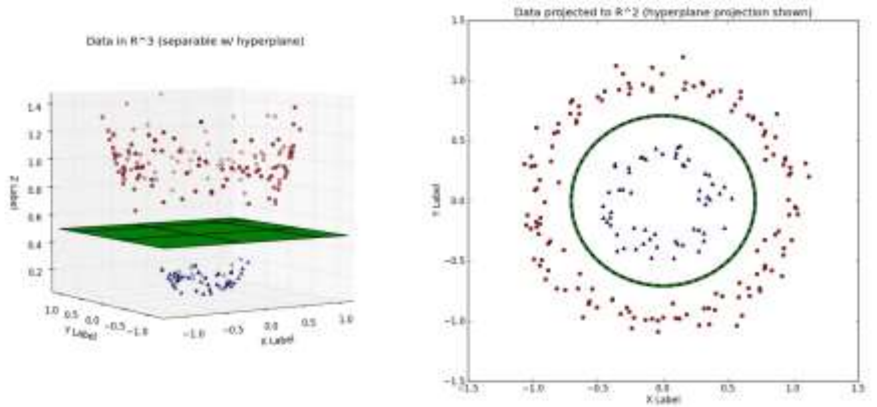
What is the “curse/blessing of dimensionality”?

- The **curse of dimensionality** refers the problem that, when the dimensionality increases, the **volume of the space increases so fast** that the available data become **sparse**.
  - This sparsity is problematic for any method that requires **statistical significance**.
  - In order to obtain a statistically sound and reliable result, the amount of data needed to support the result often grows **exponentially** with the dimensionality.
- The blessing of dimensionality states that in some cases, there are some advantages to working with high dimensional data
  - Data becomes easier to linearly separate (eg, what kernels do)



Example of the curse of dimensionality.

- ***To capture 50% of 1D 10x1 grid, need 5 data points***
- ***To capture 50% of 2D 10x10 grid, need 50 data points***
- ***To capture 50% of 3D 10x10x10 grid, need 500 data points***



Example of the blessing of dimensionality; points become separable if you add a third dimension representing the radius from the middle.

At a high level, what is empirical risk minimization (ERM)? What's its connection to the law of large numbers?

- **Risk** is the theoretical expected loss over all possible x, y, for a given model
- **Empirical risk** is the expected loss over an empirical dataset of {x_i, y_i}
- **ERM** is a core principle of ML, and assuming that your dataset is sampled from the true underlying distribution, states that minimizing error on that dataset will empirically minimize the true risk (what you actually want). This comes from the law of large numbers (LLN).
- **LLN** states: As the number of trials or observations increases, the sample average of the results will get closer to the expected (theoretical) average.

$$R(h) = \mathbf{E}[L(h(x), y)] = \int L(h(x), y)\, dP(x, y)$$

$$R_{\text{emp}}(h) = \frac{1}{n} \sum_{i=1}^{n} L(h(x_i), y_i)$$

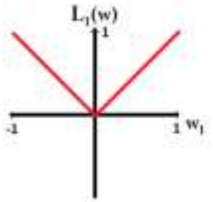
Compare and contrast L1/L2 regularization. What other names are they known by?

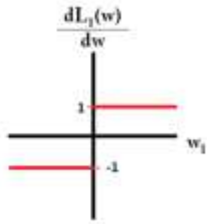
Suppose we have weights w1, w2, … wn. Then:

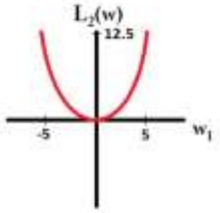
- L1 Regularization (i.e. Lasso regression):
  - Derivative/gradient is either 1, -1, or 0
  - Leads to sparsity (more weights being 0)
- L2 Regularization (i.e. Ridge regression):
  - Derivative/gradient is reduced linearly as the weight goes towards 0, so smaller gradient steps are taken (the motivation is diminished as you get closer)
  - Pushes weights closer to zero, but not sparsity

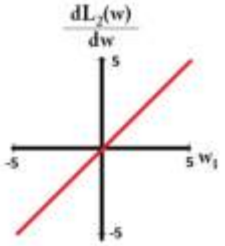
$$L_1(w) = \Sigma_i |w_i|$$

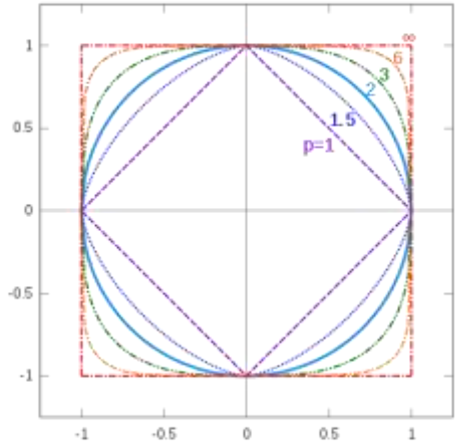
$$L_2(w) = \frac{1}{2}\Sigma_i w_i^2$$











Unit circles for different Lp norms.

Statistical Data Processing

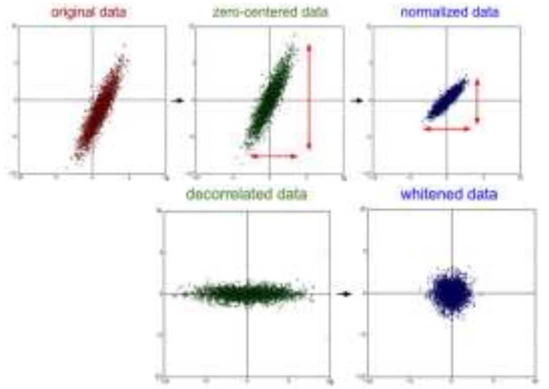
Define the following types of data processing: centering, normalization, standardization, decorrelation, whitening. When is it necessary to standardize/normalize, and when is it not necessary?

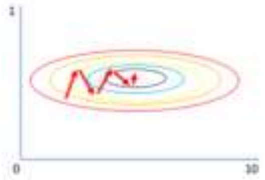
$$X' = a + \frac{(X - X_{\min})(b - a)}{X_{\max} - X_{\min}}$$

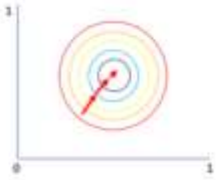
$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

Feature Scaling for [a,b] and [0,1]

- **Centering**: Subtract mean to each feature
- **Normalization**: Restrict all values in a dimension to be within a range [a,b].
  - useful when we use features that represent different characteristics and are on very different scales but of equal importance (eg number of rooms in a house and house price)
  - Doesn't make any assumptions about underlying distribution
- **Standardization**: Centers and sets the variances in the data to be 1 (diagonal of cov. Matrix becomes 1)
  - Can be done by dividing each dimension by its standard deviation
  - Unlike normalization values remain unbounded
  - Assumes gaussian distribution
  - **Batchnorm is doing this**
- **Decorrelation**: Removes the linear correlations in the data (cov. Matrix becomes diagonal)
  - Visually, rotates the data so the directions of most variation are aligned with the axes
  - PCA transformation does this
  - For regression, done only on the independent variables, not the dependent variables (otherwise, there would be nothing to regress)
- **Whitening**: Decorrelation + Standardization, data becomes uncorrelated and variances are 1 (identity cov. Matrix)
  - PCA, then dividing by the square root of the (now diagonal) covariance matrix does this
  - Warning, this step can greatly exaggerate the noise in the data, since it stretches all dimensions (including the irrelevant dimensions of tiny variance that are mostly noise) to be of equal size in the input
  - Decorrelated Bach Norm (CVPR 18 paper) explored this, but is mostly unused
- Cases when scaling is **necessary**:
  - Helps gradient descent optimization **converge faster**
    - **Prevents cost function contours from being too elliptical** and more spherical
    - Needed since gradient descent is only 1st order, and does not use 2nd derivative based curvature information; descent directions are perpendicular to cost contour
  - PCA measures variance, so it's is sensitive to the scale, if in different units
- Cases when scaling is **unnecessary**
  - Decision trees

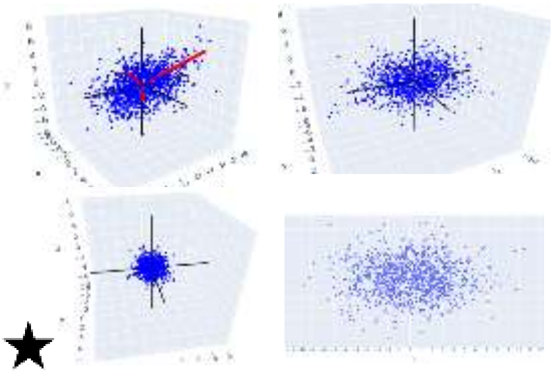



Gradient of larger parameter dominates the update

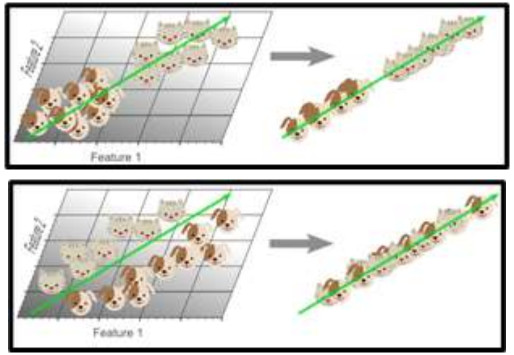

Both parameters can be updated in equal proportions

Explain what PCA does, its steps and their first principles, what it’s used for, and some pitfalls/relevant issues.

- PCA on a dataset gives an set of **orthogonal vectors** defining an orthogonal transformation (ie, “**rotation**”) that **transforms the data** such that scalar projecting the data onto the first axis of the transformation yields the greatest variance of any direction, projection onto the second axis yields the second greatest variance, etc.
- Using the set of axes (ie, “principal components”) that PCA provides:
  - The transformed dataset undergoes a change of basis, which renders the points **uncorrelated**
    - It can further be **whitened** by by dividing each data point’s dimension by that dimension’s standard deviation. Then, the result is not only uncorrelated, but has **identity covariance**.
  - You can choose to only project the data to a few of the principal components, which allows for dimensionality reduction
- At a high level, the steps are:
  - Center the dataset
  - If features are of different units, standardize the data since PCA is sensitive to different ranges and variance
    - Gets insights of non-axis aligned covariance, beyond an ordering of the variables by their dimension-based variance
    - Intuitively, results should be same if you change from inches to cm and pounds to kg, etc
  - Compute the covariance matrix, and eigendecompose it so we have its eigenvectors (directions of greatest variance) and eigenvalues (magnitude of the variance)
  - Perform a change of basis using the top k eigenvectors (resulting in a nxk eigenvector matrix), to “rotate” the data points to be axes-alined and uncorrelated
    - The dimensions not in the top k are “dropped”/ignored
  - You can also whiten it by (ie, multiplying with the covariance matrix to the -½ power)
- It’s nontrivial to decide how many principal components to keep; a heuristic is to plot r_k and pick a k such that sufficient variability is explained
- One assumption is that the most discriminative information is captured by the largest variance in the feature space. However, there are cases where the discriminative information actually resides in the directions of the smallest variance, such that PCA could greatly hurt classification performance. LDA avoids this.
- Theoretically, PCA is derived from trying to find directions which maximize the variance after projection. It turns out that eigenvectors solve this optimization problem.



Clockwise from upper left: points with eigenvectors plotted, decorrelated points, whitened points, projects projected to 2D



Top: Direction with most variance is discriminative

Bottom: Dimension with least variance is discriminative

$$r_k = \frac{\sum_{i=1}^{k} \sigma_i^2}{\sum_{i=1}^{n} \sigma_i^2}$$

Misc. Classical ML Models

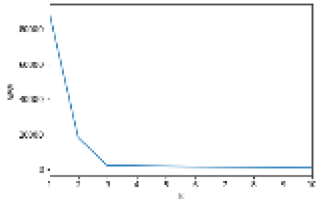
Explain the K-Means Clustering problem, an algorithm for it, and its applications.

Given a set of observations (x1, x2, ..., xn), where each observation is a d-dimensional real vector, k-means clustering aims to partition the n observations into k (≤ n) sets S = {S1, S2, ..., Sk} such that the within-cluster variances are minimized

$$\underset{\mathbf{S}}{\arg\min} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \left\| \mathbf{x} - \boldsymbol{\mu}_i \right\|^2 = \underset{\mathbf{S}}{\arg\min} \sum_{i=1}^{k} |S_i| \operatorname{Var} S_i$$

In practice, this is an NP-Hard problem and **Lloyd's algorithm** (which is not guaranteed to find the optimum) is usually used. Given an initial set of k means/centroids (eg, by randomly choosing from the observations), we repeat the following:

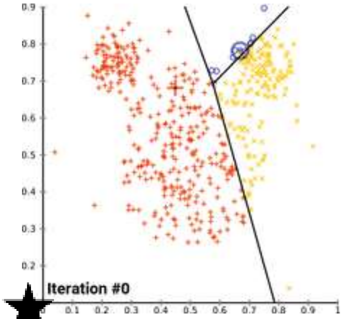
- **Assignment Step**: Assign each observation to the cluster with the nearest centroid (based on euclidean distance)
- **Update Step**: Recalculate the centroid by taking the mean of the points in each cluster

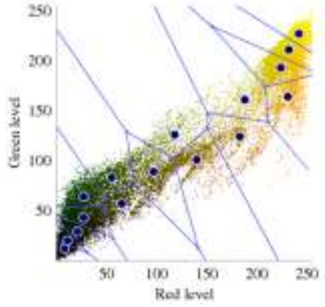


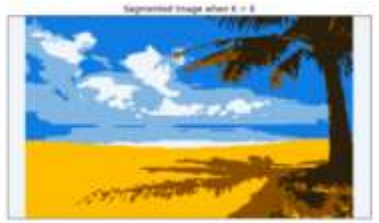
The value of k can be chosen based on "**the elbow method**", which checks for when the **within-cluster-variance** suddenly drops for values of k.

Applications to k-means:

- The resulting clusters can be used to discover/visualize groups within your dataset
  - Eg, discover types of customers, genres of books, fraud detection groups, groups of colors (for image segmentation)
- Voronoi diagram can be used to quantize/discretize the dataset's vectors into k bins
- Centroids can be used for k-NN classifier

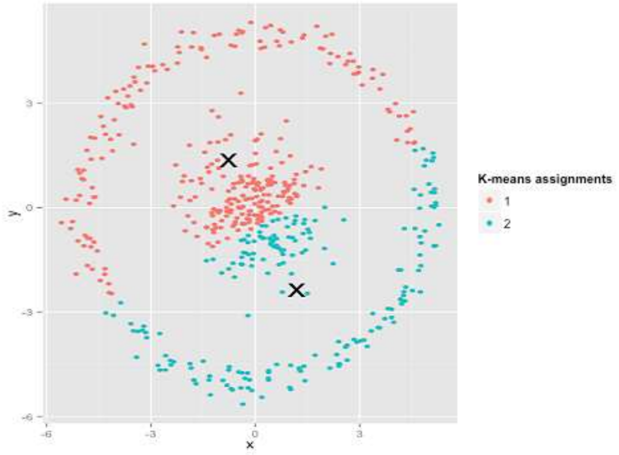


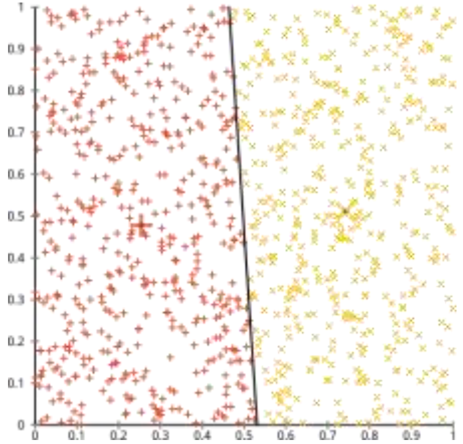


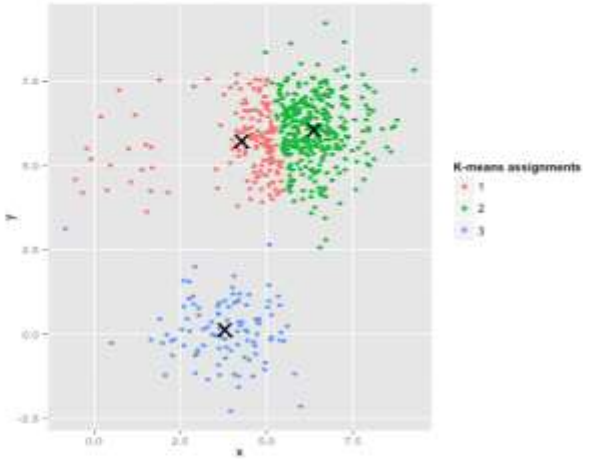


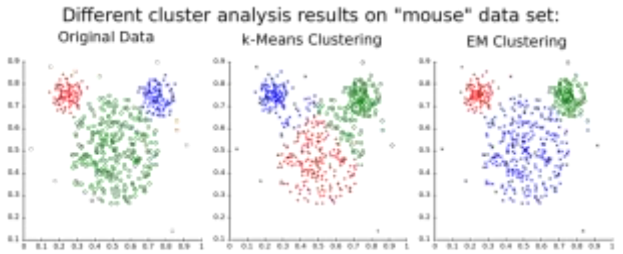
What are some failure cases of k-means?

- Bad initialization/optimization
- Spherical data (can be mitigated by embedding in a high dim space, or using polar coords)
- Data where there are no clusters
- Clusters of different variances and sparsity
  - More weight assigned to larger clusters
  - Instead use GMM, which generalities k-means for non-homoscedastic and ellipsoid cases







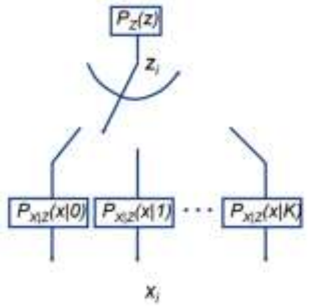


How are mixture models formulated, optimized, and used?

There are two types of random variables we are trying to model: the observed variable X and the hidden state variable Z we're "unaware of".

Observations are realized with a two step procedure:

$$P_X(x) = \sum_{c=1}^{C} P_x(x, Z=c) = \sum_{c=1}^{C} P_{X|Z}(x|c) P_Z(c) = \sum_{c=1}^{C} P_{X|Z}(x|c)\pi_c$$



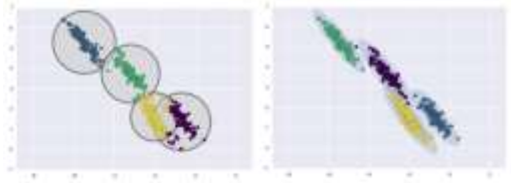
1. z is sampled from Z, with pdf P(Z=z)
2. x|z is sampled from X|Z, with pdf P(X=x|Z=z).

The components of a mixture model are as follows:

K	$=$	number of mixture components
N	$=$	number of observations
$\theta_{i=1\ldots K}$	$=$	parameter of distribution of observation associated with component i
$\phi_{i=1\ldots K}$	$=$	mixture weight, i.e., prior probability of a particular component i
$\boldsymbol{\phi}$	$=$	K-dimensional vector composed of all the individual $\phi_{1\ldots K}$; must sum to 1
$z_{i=1\ldots N}$	$=$	component of observation i
$x_{i=1\ldots N}$	$=$	observation i
$F(x\|\theta)$	$=$	probability distribution of an observation, parametrized on θ
$z_{i=1\ldots N}$	$\sim$	$\operatorname{Categorical}(\boldsymbol{\phi})$

Then, for example for a multivariate gaussian mixture model, the probability of an observation would be calculated as:

$$p(\boldsymbol{\theta}) = \sum_{i=1}^{K} \phi_i \mathcal{N}(\boldsymbol{\mu_i}, \boldsymbol{\Sigma_i})$$

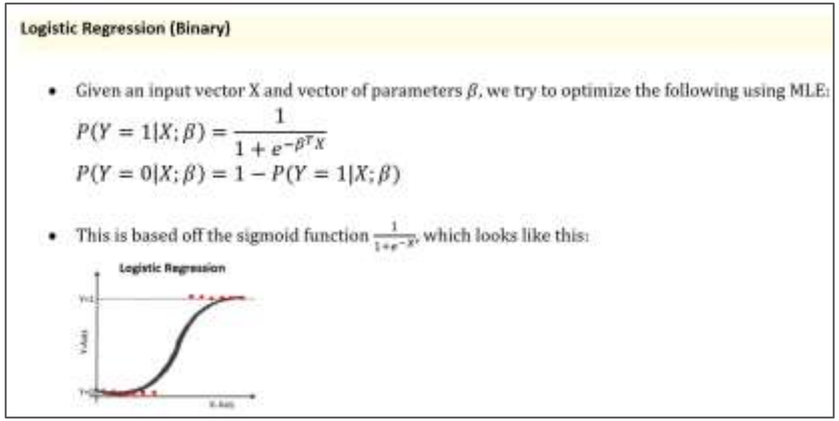


The EM algorithm is an iterative method to find the MLE or MAP, when models depends on unobserved latent variables. So, it's a natural fit for mixture models.

Applications of mixture models:

- Model a distribution for a dataset, and be able to understand the approximate probability of future values.
- Clustering the data into K groups (ie, more sophisticated way to do k-means with soft assignment; argmax for hard)
  - Given a data point, can compute the probability from each component distribution for each z by P(X,Z)=P(X|Z)P(Z)
  - It can be shown that k-means can fit spheres while GMMs can fit ellipsoids with non-identity covariances

Define, and compare/contrast logistic regression and softmax regression

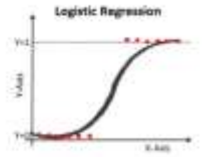


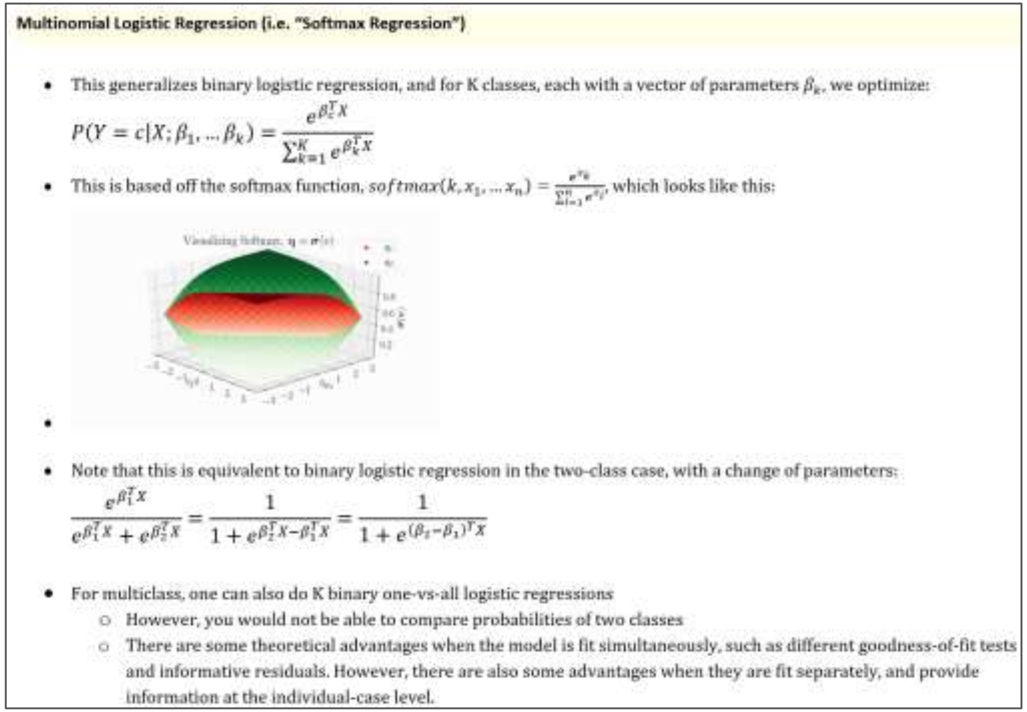
Logistic Regression (Binary)

- Given an input vector X and vector of parameters β, we try to optimize the following using MLE:

$$P(Y=1|X;\beta) = \frac{1}{1+e^{-\beta^T X}}$$
$$P(Y=0|X;\beta) = 1 - P(Y=1|X;\beta)$$

- This is based off the sigmoid function $\frac{1}{1+e^{-x}}$, which looks like this:



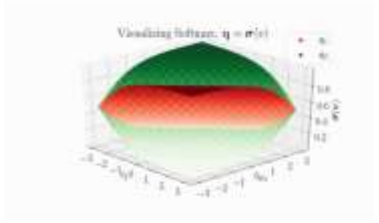


Multinomial Logistic Regression (i.e. "Softmax Regression")

- This generalizes binary logistic regression, and for K classes, each with a vector of parameters β_k, we optimize:

$$P(Y=c|X;\beta_1,\dots\beta_k) = \frac{e^{\beta_c^T X}}{\sum_{k=1}^{K} e^{\beta_k^T X}}$$

- This is based off the softmax function, $softmax(k, x_1, \dots x_n) = \frac{e^{x_k}}{\sum_{i=1}^{n} e^{x_i}}$, which looks like this:



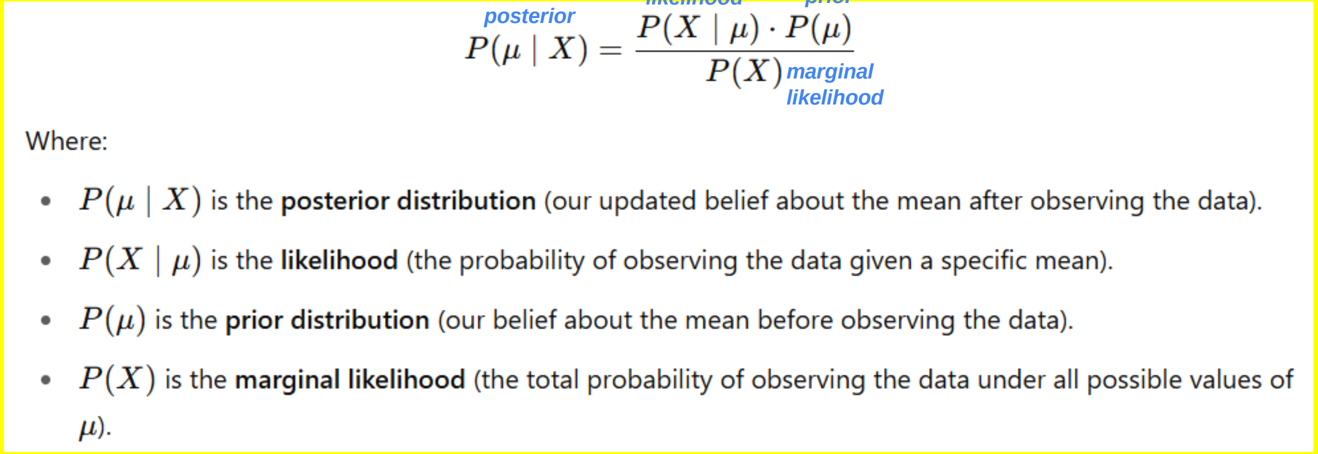
-

- Note that this is equivalent to binary logistic regression in the two-class case, with a change of parameters:

$$\frac{e^{\beta_1^T X}}{e^{\beta_1^T X} + e^{\beta_2^T X}} = \frac{1}{1+e^{\beta_2^T X - \beta_1^T X}} = \frac{1}{1+e^{(\beta_2-\beta_1)^T X}}$$

- For multiclass, one can also do K binary one-vs-all logistic regressions
  - However, you would not be able to compare probabilities of two classes
  - There are some theoretical advantages when the model is fit simultaneously, such as different goodness-of-fit tests and informative residuals. However, there are also some advantages when they are fit separately, and provide information at the individual-case level.

Explain how the bayes rule can be used as a stastical model to update prior beliefs



$$\overset{\text{posterior}}{P(\mu \mid X)} = \frac{\overset{\text{likelihood}}{P(X \mid \mu)} \cdot \overset{\text{prior}}{P(\mu)}}{\underset{\text{marginal likelihood}}{P(X)}}$$

Where:

- $P(\mu \mid X)$ is the **posterior distribution** (our updated belief about the mean after observing the data).
- $P(X \mid \mu)$ is the **likelihood** (the probability of observing the data given a specific mean).
- $P(\mu)$ is the **prior distribution** (our belief about the mean before observing the data).
- $P(X)$ is the **marginal likelihood** (the total probability of observing the data under all possible values of μ).

- X is the new observation, and μ is our current understanding of the world
- Expresses that the current understanding can be updated with new observations
- Can be used iteratively as Bayesian updating; posterior distribution from one iteration becomes the prior for the next iteration.
  - Bayes' Rule is like a mathematical GPS for uncertainty—it continuously reorients your understanding of the world as new information arrives.
- Generally, the likelihood is easier to compute than the posterior (since it's directly given by the model)
- Examples
  - Machine learning: Updating the model's parameters (e.g., weights) as new data arrives
  - Stock market prediction: Continuously updating predictions based on new stock price data
  - Medical diagnosis: Iteratively updating the likelihood of a disease given new test results over time

What is Bayes' classifier? What is the Naive Bayes classifier? What are their pros and cons?

Bayes' classifier takes a probabilistic approach, centered around Bayes' rule:

$$p(Y|\mathbf{X}) = \frac{p(\mathbf{X}|Y)\,p(Y)}{p(\mathbf{X})} \Leftrightarrow posterior = \frac{likelihood * prior}{evidence}$$

Ultimately, given some x we want to choose the most likely y-label. The decision rule is:

$\hat{y} = argmax_{i\in[1,\dots,k]}\, p(\mathbf{x}|Y=i) * p(Y=i)$

Note that the denominator is ignored for classification purposes, since it's just a constant. To solve this, one needs to estimate the distribution for X|Y and Y.

- Y follows a multinoulli distribution and can be estimated with MLE, or assumed to be uniform across all classes.
- X|Y is a n-dimensional distribution, and needs to be fit. For example, one option is to fit k-many (one for each class conditional) n-dimensional gaussians, and estimate the parameters with MLE.

Naïve Bayes

If we assume that the $X_1, \dots, X_n$ in $\mathbf{X}$ are independent from one another, then fitting X|Y becomes easier. In this case,

$$\begin{aligned} p(\mathbf{X}|Y)p(Y) = P(\mathbf{X},Y) = p(X_1,\dots X_n,Y) &= p(X_1|X_2,\dots,X_n,Y)p(X_2,\dots,X_n,Y) \\ &= p(X_1|X_2,\dots,X_n,Y)\cdots p(X_{n-1}|X_n,Y)p(X_n|Y)p(Y) \\ &= p(Y)\prod_i^n p(X_i|Y)\text{, due to independence} \end{aligned}$$

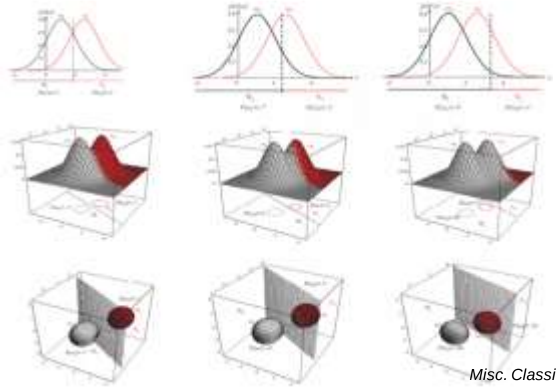
This makes estimating the distribution of X|Y easier and less data-dependent.

For example, we can fit $k * n$ many 1-dimensional gaussians and just multiply the probabilities, which avoids the curse of dimensionality.

Pros of Bayes' Classifier:
- Technically optimal, if distribution estimates are accurate
- Generative and gives "exact' per-class probabilities (e.g. unlike SVMs)

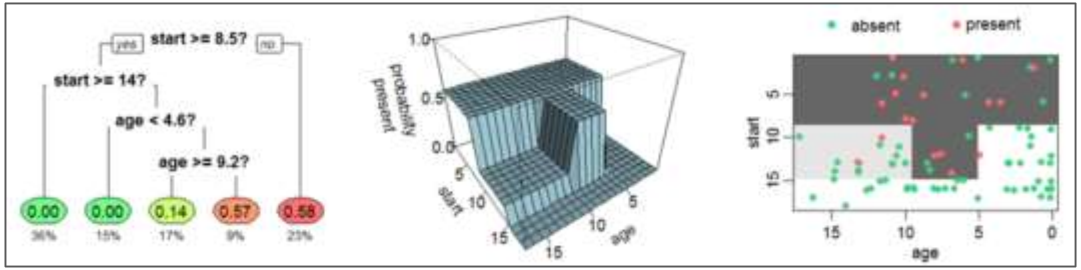
Cons:
- Hard to estimate probability distribution for X|Y (need a lot of data)
- Naive bayes assumption requires independence, which probably won't hold in most cases

Suppose you use gaussians to model X|Y. The figure shows the binary classification case for x in R^1,R^2,R^3 in the first, second, and third rows respectively.
The decision boundary given by the BDR is a hyperplane, and is weighed by the distribution of Y. As the black class becomes more likely (left to right), it occupies more space and pushes the decision boundary further away from it.



What are decision trees? Explain its pros/cons.

Decision trees are binary trees where each non-leaf node directs the input vector to either its left or right node, based on two per-node parameters: the feature dimension, and threshold c. The leaf nodes are classification or regression prediction outputs.



Decision trees can be learned using a training dataset $\{(x_i, y_i)\}_{i=1}^N, x_i \in \mathbb{R}^K$.

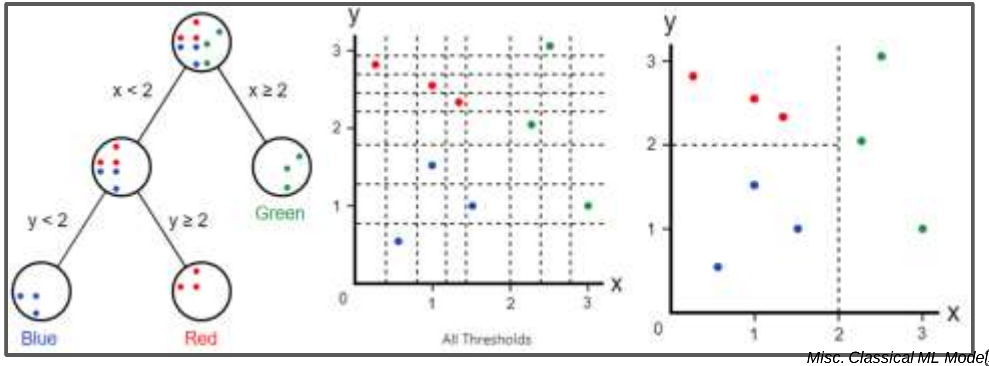
- Recursively, decision nodes are added to the tree, where each node is parametrized by the feature dimension in $\{1, \ldots, k\}$, and a threshold c.
  - There are several ways to decide this, but a common approach is just brute force: try every possible threshold for every dimension, and pick the one with the highest **Gini Gain**. At a high level, this allows us to pick the "best" split.
- A node is no longer split if all points in it have the same class, or if all splits are equally good.

Pros:

- Decision Trees are easily **interpretable**, and mirrors human decision making
- **Built-in feature selection**; unused features can be discarded, and features on top are the most informative
- Handles both categorical and numerical data easily; robust to scale & heterogeneous features
- Capable of learning **nonlinear**, complicated decision boundaries
- **Simple** to use "out-of-the-box"

Cons:

- Can be very prone to **overfitting**
- Learning an optimal decision is known to be NP-complete, so in practice all algorithms will be **suboptimal** (e.g. using greedy methods)



Question Goes Here

- Answer goes here