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?

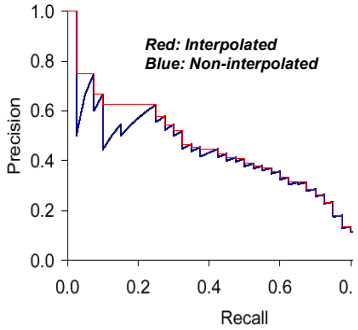
- In general, for averages to be valid, you need the denominator values to be in the same scaled units
- This exactly what the harmonic mean does: it is the reciprocal of the arithmetic mean of the reciprocals of the given set of observations
    - Precision and recall both have true positives in the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

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

$$\left(\frac{x_1^{-1} + \ldots + x_n^{-1}}{n} \right)^{-1} \qquad \left(\frac{1^{-1} + 4^{-1} + 4^{-1}}{3} \right)^{-1} = \frac{3}{\frac{1}{1} + \frac{1}{4} + \frac{1}{4}} = \frac{3}{1.5} = 2$$

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)



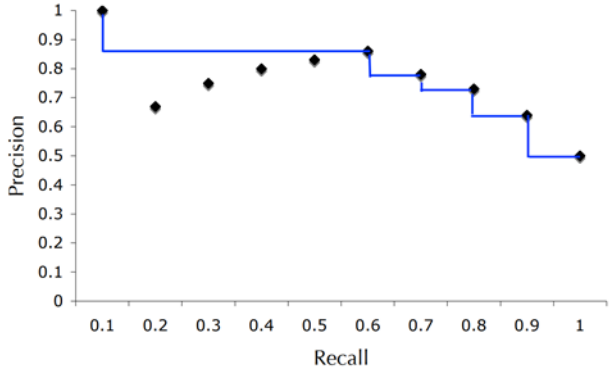
$$\text{AP} = \sum_n (R_n - R_{n-1}) P_n$$

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

Calculate the accuracy, precision, recall, and F1 score for the following binary confusion matrix.
Complete the following P/R@K table assuming there are 10 true positives, and draw the PR curve.

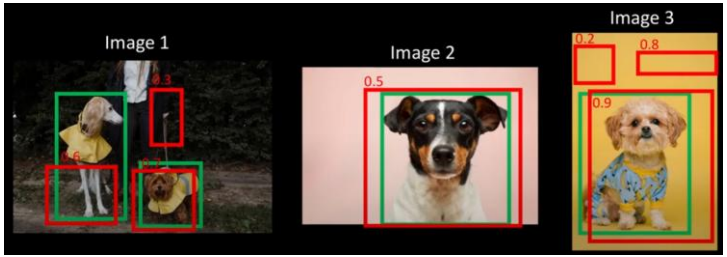
		Actual		
		Bipolar	**Not Bipolar**	Total
Predicted	**Bipolar**	10	11	21
	Not Bipolar	9	52	61
	Total	19	63	82

- Accuracy
  - 62/82
- Precision
  - 10/21
- Recall
  - 10/19
- F1 Score
  - (2*(10/21)*(10/19)) / ((10/21)+(10/19))



How is mAP calculated in the context of retrieval vs object detection?

- **Retrieval**
  - The “standard” case; check the top K retrieved images in the test set for a retrieval query (eg, an object class) and compute the precision/recall for each K by considering if they are relevant to your class. Thus, you get the PR curve, and calculate the AP.
- **Object detection**
  - For a given object class, precision/recall value calculated for each detection (after sorting by descending confidence) over every image the test dataset
    - True/false positive detections are based on a chosen IoU threshold
  - Plotting gives a class PR curve from which you can calculate the AP (by taking the AUC)
  - This is repeated for every class
- In both cases, mAP can be computed by averaging the APs for each class. For object detection, this might also be averaged over different IoU thresholds.



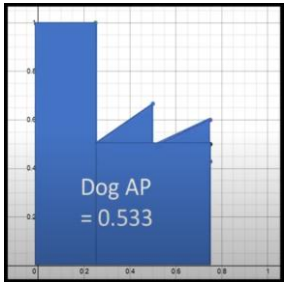
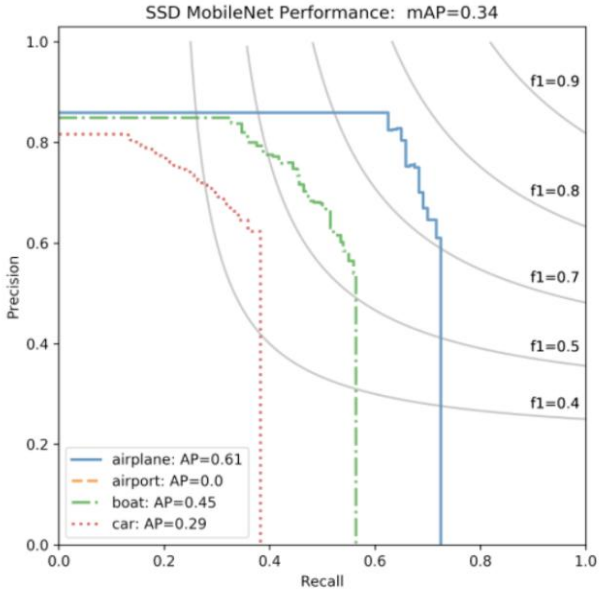


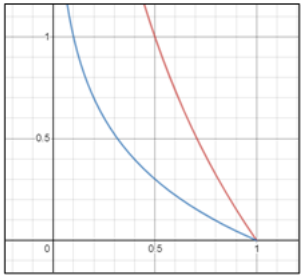
Image	Confidence	TP or FP	Precision	Recall
Image 3	0.9	TP	1 / 1	1 / 4
Image 3	0.8	FP	1 / 2	1 / 4
Image 1	0.7	TP	2 / 3	2 / 4
Image 1	0.6	FP	2 / 4	2 / 4
Image 2	0.5	TP	3 / 5	3 / 4
Image 1	0.3	FP	3 / 6	3 / 4
Image 3	0.2	FP	3 / 7	3 / 4



Precision-recall curve for SSD model for 4 object classes, where IoU threshold is 0.5. Van Etten, A. (2019, January)

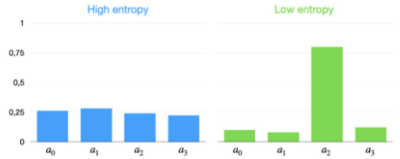
Define Shannon information, entropy, cross entropy, KL/JS Divergence. What are their relationships?

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



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

- **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.

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

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

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

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

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

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

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

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

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

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.

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

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

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

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

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

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

- 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.
- There are two versions of MSE: one for predictions over a labeled dataset, and one for an estimator

$$\text{MSE}(\hat{\theta}) = \text{E}_\theta\left[(\hat{\theta} - \theta)^2\right]$$

$$MSE(X, Y) = \frac{1}{n}||X - Y||_2^2 = \frac{1}{n}(\sqrt{\sum_i (X_i - Y_i)^2})^2 = \frac{1}{n}\sum_i (X_i - Y_i)^2$$

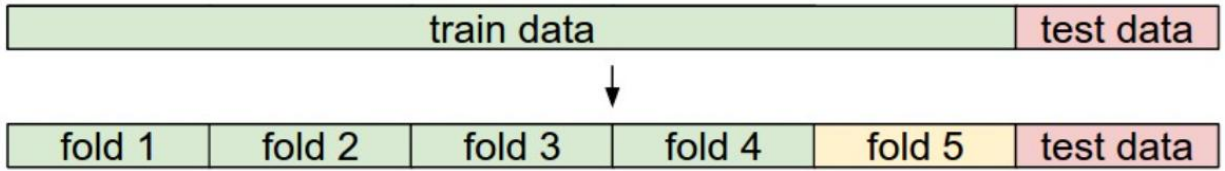
Experimental Designs and Paradigms

What is k-fold cross-validation, and what are its pros and cons?

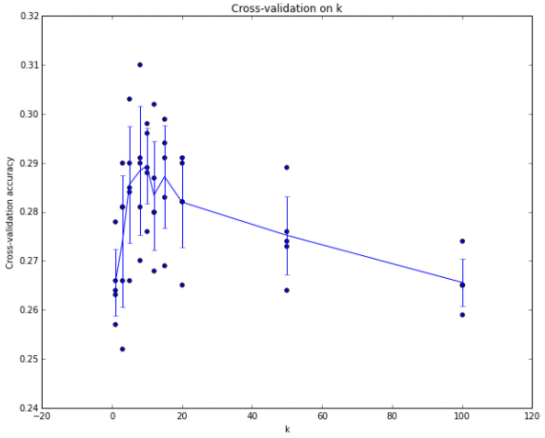
For example, in 5-fold cross-validation, we would split the training data into 5 equal folds, use 4 of them for training, and 1 for validation. We would then iterate over which fold is the validation fold, evaluate the performance, and finally average the performance across the different folds.

Pros: hyperparameter tuning becomes more reliable and less noisy

Cons: computationally expensive, linear to the number of folds



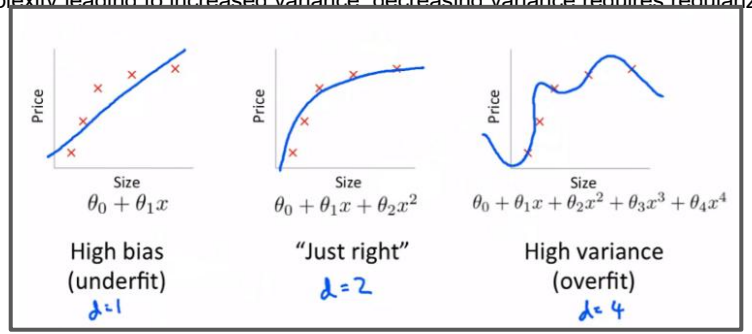
Common data splits. A training and test set is given. The training set is split into folds (for example 5 folds here). The folds 1-4 become the training set. One fold (e.g. fold 5 here in yellow) is denoted as the Validation fold and is used to tune the hyperparameters. Cross-validation goes a step further and iterates over the choice of which fold is the validation fold, separately from 1-5. This would be referred to as 5-fold cross-validation. In the very end once the model is trained and all the best hyperparameters were determined, the model is evaluated a single time on the test data (red).



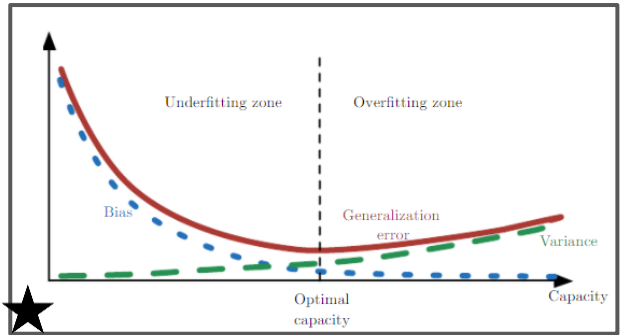
What is bias and variance? Explain their relationship, mathematical definition, and application in ML.

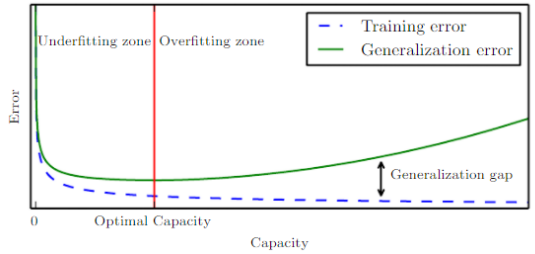
- **Bias** $\text{bias}(\hat{\theta}_m) = \mathbb{E}(\hat{\theta}_m) - \theta$.
  - The bias for a parameter estimator is the difference between its expected value and the true underlying parameter.
  - For ML: **on average how correct**/high quality your trained parameters will be to a true "oracle". It should be **asymptotically unbiased**; as the number of training sample approaches infinity, the model should be unbiased. Otherwise, there's **systematic error**.
- **Variance** $\text{Var}(\hat{\theta})$
  - Given a parameter estimator, variance characterizes how **consistent** it would be given different training datasets sampled from the underlying data-generating process.
  - For ML: if you resampled for different training datasets, would those trained parameters be similar to each other (low variance), or would it overfit to each training dataset samples' statistics, leading to significantly different parameters (high variance)?
- **Mean Squared Error**
  - The MSE for an estimator is its expected squared difference between the true underlying parameter. It can be shown that it decomposes to the bias and variance.
- **Bias variance trade-off & inverse relationship**
  - In general, high variance = overfitting and high bias = underfitting.
  - Decreasing bias tends to require model complexity leading to increased variance; decreasing variance requires regularization leading to increasing bias.

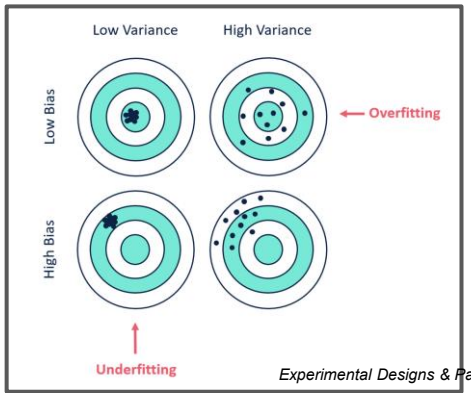
TLDR: Bias = accuracy on train set, too high leads to overfitting
Variance = robustness
Want a sweet spot where it's not 100% accurate on train spot, but does well on validation clips



$$\begin{aligned} \text{MSE} &= \mathbb{E}[(\hat{\theta}_m - \theta)^2] \\ &= \text{Bias}(\hat{\theta}_m)^2 + \text{Var}(\hat{\theta}_m) \end{aligned}$$





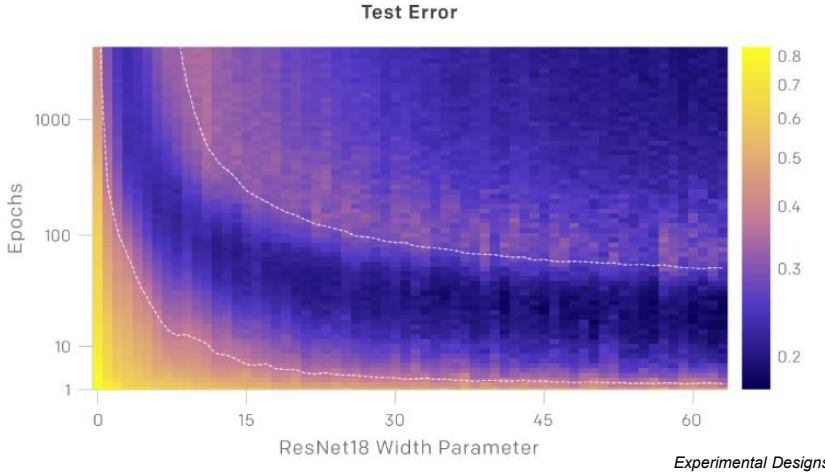


What is the double descent phenomenon?

- 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

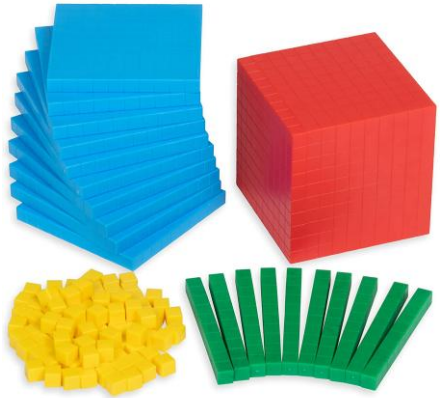
Several factors contribute to this phenomenon:

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.



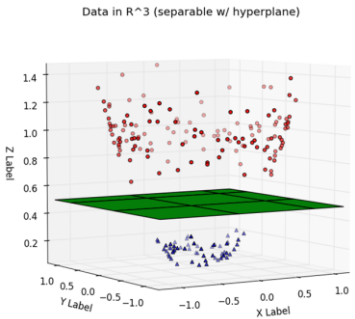
What is the “curse/blessing of dimensionality”?

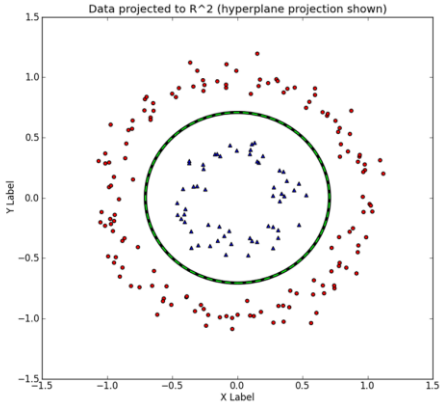
- The **curse of dimensionality** refers the problem that, when the dimensionality increases, the **volume of the space increases so fast** that the available data become **sparse**.
  - This sparsity is problematic for any method that requires **statistical significance**.
  - In order to obtain a statistically 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.

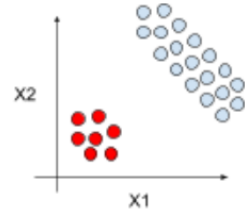
- ***To capture 50% of 1D 10x1 grid, need 5 data points***
- ***To capture 50% of 2D 10x10 grid, need 50 data points***
- ***To capture 50% of 3D 10x10x10 grid, need 500 data points***





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

Compare/contrast generative vs discriminative methods.

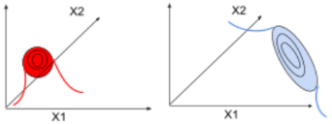


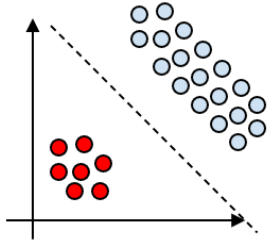
The fundamental difference is that:

- **Discriminative models** learn the (hard or soft) **boundary** between classes; conditional distribution
- **Generative models** model the **distribution** of individual classes; joint distribution

For example, the data to the right has two features X1, X2 and two classes, red and blue.

- A **generative** model would explicitly model **P(X,Y)**; in the most direct way, you would have the pdfs for P((x1,x2),red) and P((x1,x2),blue), perhaps using a gaussian mixture model
  - Classification on X then be performed by arg max over Y of P(X, Y)
- A **discriminative** model would implicitly or explicitly model **P(Y|X)**; given features X, you would have probability of it being red or blue.
  - We don't have p(X); otherwise, we would be in the generative scenario since P(X,Y)=P(Y|X)P(X)
  - Classification done by arg max over Y of P(Y|X)





Generative	Discriminative
• Learns P(X,Y)=P(X|Y)P(Y) (implicity or explicitly) • Models the distribution of individual classes • Less direct towards classification • More probabilistic modeling involved, more assumptions taken about the underlying distribution • Can require **more assumptions** • Applies to both supervised and unsupervised settings • Considers all points before modeling distribution • Access to the joint distribution; ability to draw samples from feature distribution given labels. • Offers a **rich representation** of the independence relationships in the dataset • Generally good for things like outlier detection • **Examples: GMMs, Bayesian Networks (e.g. Naive Bayes), GANs, VAEs, diffusion models**	• Learns P(Y|X) (implicity or explicitly) • Learns the decision boundary between classes • More direct way toward classification; the idea is to avoid solving an easier problem by first solving a harder problem as an intermediate step. • **Fewer assumptions** are taken about the data • Usually inherently supervised in nature • Generally, only considers points near decision boundary • Tends to produce **higher classification accuracy** • Offers a rich representation of the **decision boundaries** in the dataset • **Examples: K-NN, Logistic regression, SVMs, Neural Networks, Random Forest**

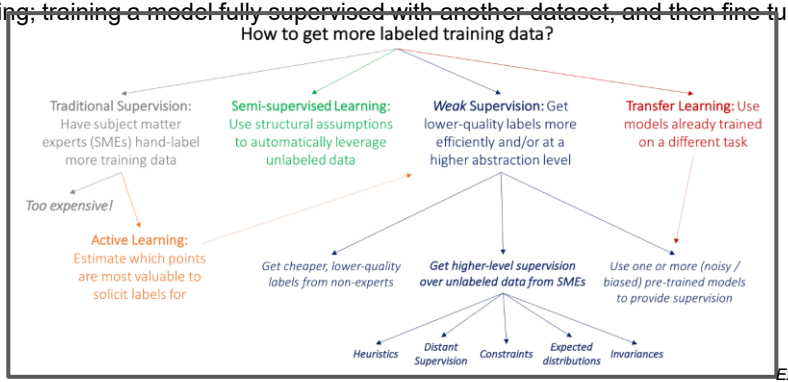
What are the different levels of supervision that a ML model can have? Give some examples of some prevalent methods. When should you consider which to use?

In machine learning, given a dataset of input features {X_i} and labels {Y_j}. The nature of these sets lead to the following learning paradigms:

- **Fully Supervised**
  - All labels are provided for the training set. Each X_i has a corresponding Y_i, which is assumed to be ground truth.
- **Semi Supervised**
  - Some of the labels are provided for a strict subset of {X_i}. That is, only some X_i have a corresponding Y_i, which is assumed to be ground truth.
  - One technique is psuedolabeling -- predicting labels for the unlabeled features, and then training on them. This is called **bootstrapping**.
- **Weakly Supervised**
  - Noisy, limited, or imprecise sources are used to provide supervision signal for labeling training data. Each X_i has a corresponding Y_i, but Y_i could be noisy or inaccurate.
  - Some techniques are outlier detection, temporal smoothing, and modeling the noise.
- **Self Supervised**
  - Labels are generated automatically, using the inherent structure of the unlabeled training samples.
  - Examples include: Autoencoders, inpainting, superresolution. {Y_i} is the empty set, {}.

The usage generally depends on the type of data you have available and the cost of annotation.

- Semi, weakly, and self supervision can provide more training data when fully supervised from experts would otherwise be too costly.
- Another solution to this issue of obtaining labels is through transfer learning; training a model fully supervised with another dataset, and then fine-tuning it fully supervised with your smaller target dataset.



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.

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

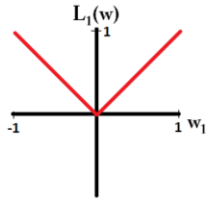
$$R_{\text{emp}}(h) = \frac{1}{n} \sum_{i=1}^{n} L(h(x_i), y_i)$$

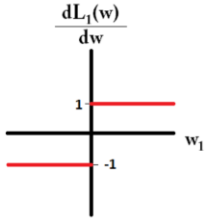
Compare and contrast L1/L2 regularization. What other names are they known by?

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

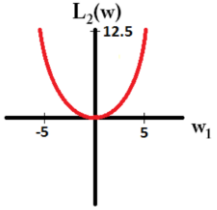
- L1 Regularization (i.e. Lasso regression):
  - Derivative/gradient is either 1, -1, or 0
  - Leads to sparsity (more weights being 0)
- L2 Regularization (i.e. Ridge regression):
  - Derivative/gradient is reduced linearly as the weight goes towards 0, so smaller gradient steps are taken (the motivation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

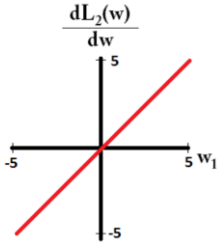
$$L_1(w) = \Sigma_i |w_i|$$

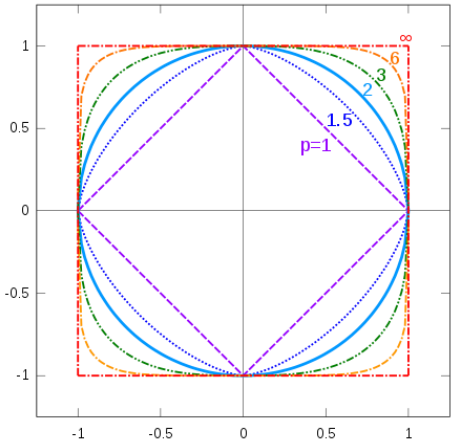




$$L_2(w) = \frac{1}{2}\Sigma_i w_i^2$$







Unit circles for different Lp norms

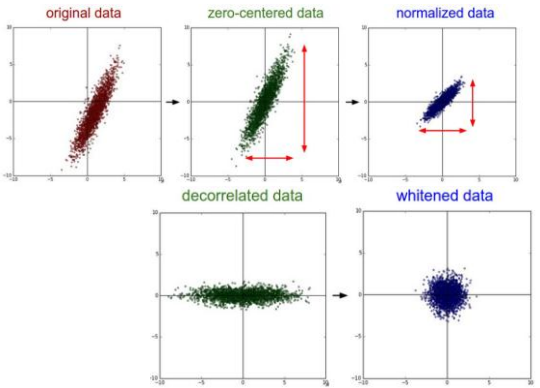
Statistical Data Processing

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?

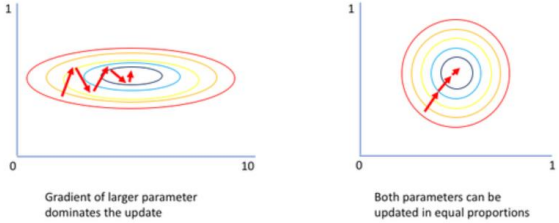
$$X' = a + \frac{(X - X_{\min})(b - a)}{X_{\max} - X_{\min}}$$

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

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

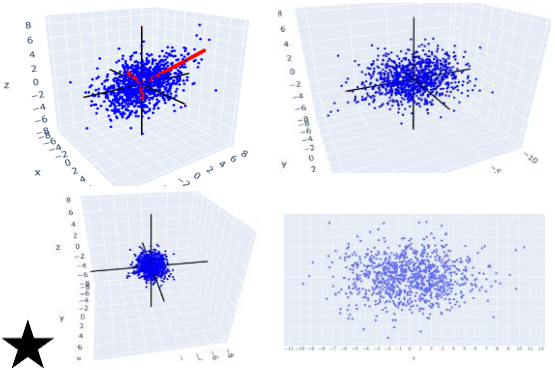


- **Centering**: Subtract mean to each feature
- **Normalization**: Restrict all values in a dimension to be within a range [a,b].
  - useful when we use features that represent different characteristics and are on very different scales but of equal importance (eg number of rooms in a house and house 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

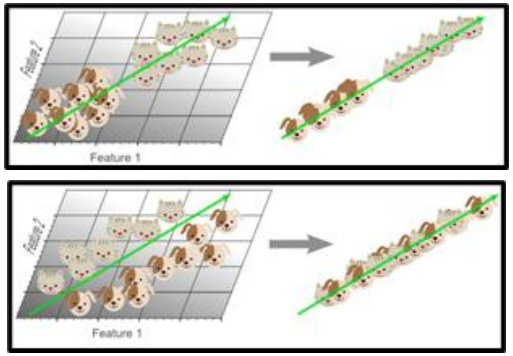


Explain what PCA does, its steps and their first principles, what it's used for, and some pitfalls/relevant issues.

- 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

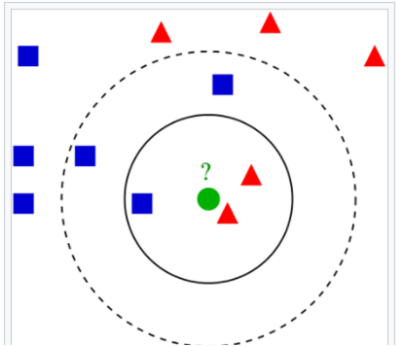
Give a list of classification algorithms.

- K-Nearest Neighbors (possibly after k-means clustering)
- Bayes classifier (e.g. naive Bayes)
- Neural Networks
- Decision trees (e.g. gradient boosted or random forest)
- SVM

How can K nearest neighbors be used for classification and regression? What are some other applications?

First, find the k nearest neighbors of your input in your training set. Then:

- For classification, output the class by taking a vote of the classes of those neighbors
  - A validation set can be evaluated by a confusion matrix and associated metrics like precision, recall, etc.
- For regression, average the values of those neighbors
  - A validation set can be evaluated by MSE error
- Applications include anomaly detection and retrieval (e.g. recommender systems)



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

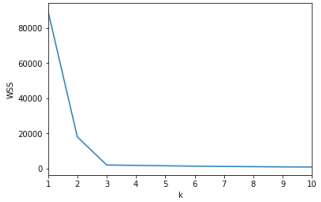
Explain the K-Means Clustering problem, an algorithm for it, and its applications.

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

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

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:

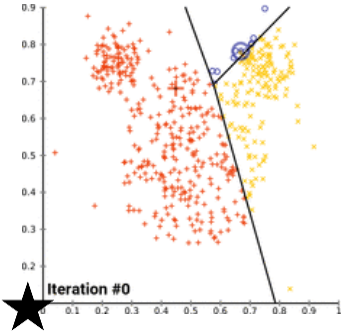
- **Assignment Step**: Assign each observation to the cluster with the nearest centroid (based on euclidean distance)
- **Update Step**: Recalculate the centroid by taking the mean of the points in each cluster

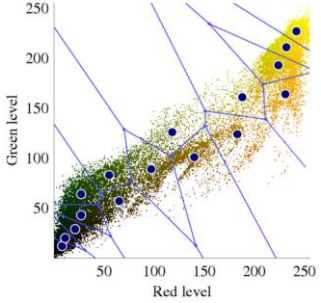


The value of k can be chosen based on "**the elbow method**", which checks for when the **within-cluster-variance** suddenly drops for values of k.

Applications to k-means:

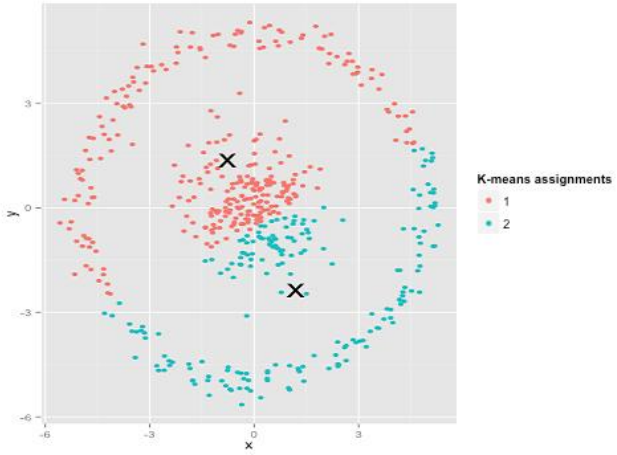
- The resulting clusters can be used to discover/visualize groups within your dataset
  - Eg, discover types of customers, genres of books, fraud detection groups, groups of colors (for image segmentation)
- Voronoi diagram can be used to quantize/discretize the dataset's vectors into k bins
- Centroids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

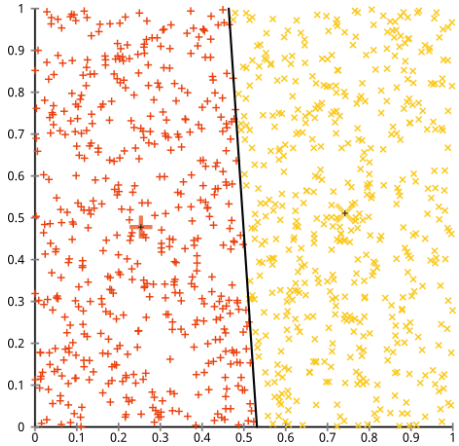


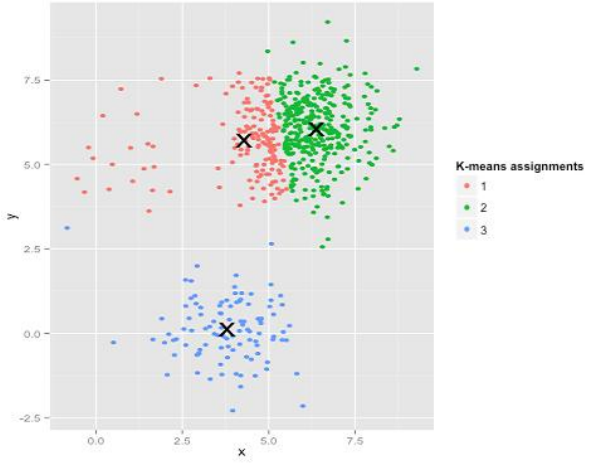


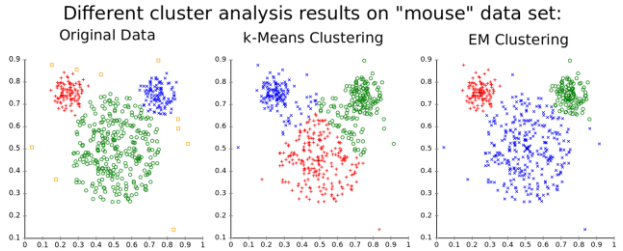
What are some failure cases of k-means?

- 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









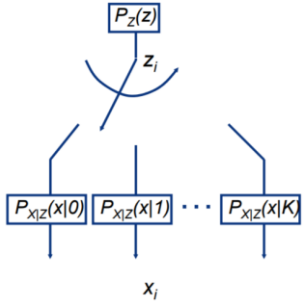
How are mixture models formulated, optimized, and used?

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:

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).

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



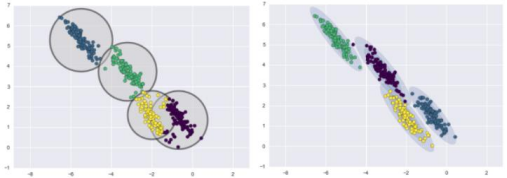
The components of a mixture model are as follows:

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
$z_{i=1\ldots N}$	$\sim$	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

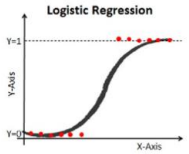
Define, and compare/contrast logistic regression and softmax regression

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:

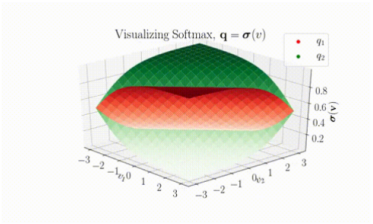


Multinomial Logistic Regression (i.e. "Softmax Regression")

- 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:



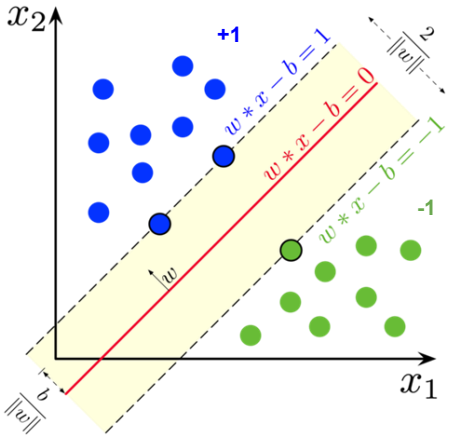
- Note that this is equivalent to binary logistic regression in the two-class case, with a change of parameters:

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

- 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 what Support Vector Machines (SVMs) are, how they work, and what their pros/cons are.

SVMs are **non-probabilistic binary linear classifiers**, which solves an optimization problem to find a hyperplane to linearly separate labeled training data with maximum margin.



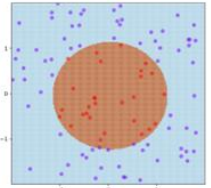
To learn an SVM classifier:

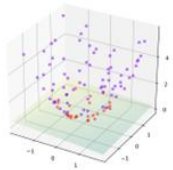
- Standardize the data
- There are 3 hyperplanes of form wx-b, as shown in the figure; their distance is 2/||w||. We want to maximize that distance while maintaining that data points lie on their respective side of the margin as much as possible. To do this, the following is minimized: $\lambda\|\mathbf{w}\|^2 + \left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i - b)\right)\right]$

  This follows since $1 - y_i(w^T x_i - b) \leq 0$ should hold if a point is on the correct side.
- Lambda determines the tradeoff between ensuring the data points lie on their correct side (may lead to overfitting) and large margin (may lead to underfitting/ignoring features). Note that if all points are perfectly linearly separable then the second term should be zero.
- The resulting w*, b* (e.g. from SGD or quadratic programming) can be used for classification: $x \mapsto sign(w^{*T}x - b^*)$

$$\text{by replacing } w^T x_i = \langle w, x_i \rangle \text{ with } K(w, x_i) = \langle \Phi(w), \Phi(x_i) \rangle$$

Additionally, the kernel method can be used





Some strengths of SVMs:

- Makes little to no assumptions about data distribution
- Training is easy; no local optima since the function is **convex**
- Effectiveness scales relatively well to high dimensions
- Relatively **robust to outliers**, due to maximal margins & because non-support data points are effectively "ignored"
- **Fast** at inference

Some weaknesses:

- **Requires a good kernel function and/or feature engineering**
- **No probability output for estimate**

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

$$P(\mu \mid X) = \frac{P(X \mid \mu) \cdot P(\mu)}{P(X)}$$

posterior: $P(\mu \mid X)$ — likelihood: $P(X \mid \mu)$ — prior: $P(\mu)$ — 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.
- 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:

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

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

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

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,

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

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

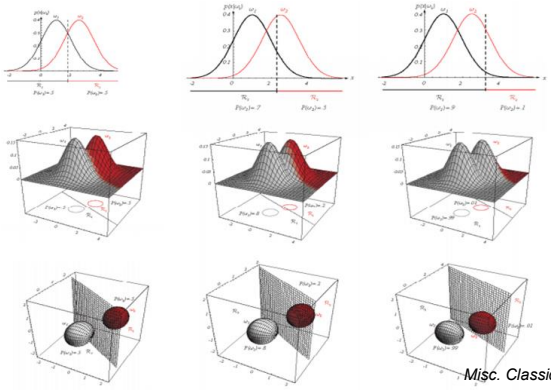
For example, we can fit $k * n$ many 1-dimensional gaussians and just multiply the probabilities, which avoids the curse of dimensionality.

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.

Give an example of Bayes classifier for sentiment analysis on the following dataset, using laplace smoothing, and a bag-of-words model.
(unique =20)

	Cat	Documents
Training	-	just plain boring
	-	entirely predictable and lacks energy
	-	no surprises and very few laughs
	+	very powerful
	+	the most fun film of the summer
Test	?	predictable with no fun

$$\hat{\theta}_i = \frac{x_i + \alpha}{N + \alpha d} \qquad (i = 1, \ldots, d)$$

** note: denominator is necessary since you add 1 to all d "buckets" or categories from the multinomial distribution*

$$P(-) = \frac{3}{5} \qquad P(+) = \frac{2}{5}$$

$$P(\text{"predictable"}|-) = \frac{1+1}{14+20} \qquad P(\text{"predictable"}|+) = \frac{0+1}{9+20}$$

$$P(\text{"no"}|-) = \frac{1+1}{14+20} \qquad P(\text{"no"}|+) = \frac{0+1}{9+20}$$

$$P(\text{"fun"}|-) = \frac{0+1}{14+20} \qquad P(\text{"fun"}|+) = \frac{1+1}{9+20}$$

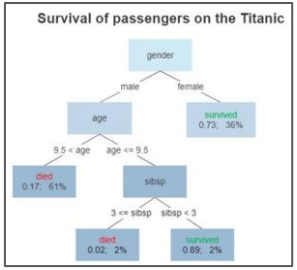
$$P(-)P(S|-) = \frac{3}{5} \times \frac{2 \times 2 \times 1}{34^3} = 6.1 \times 10^{-5}$$

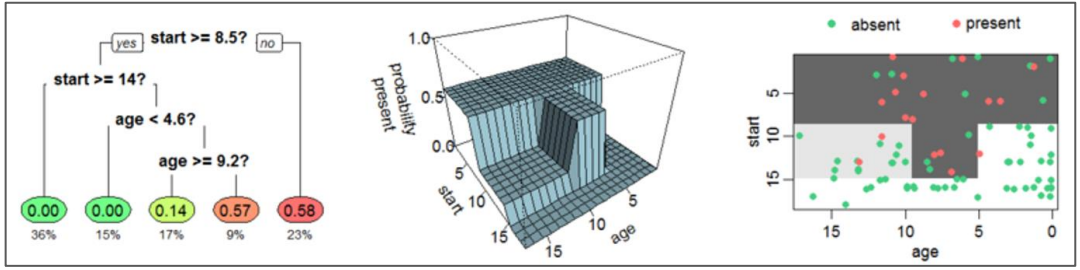
$$P(+)P(S|+) = \frac{2}{5} \times \frac{1 \times 1 \times 2}{29^3} = 3.2 \times 10^{-5}$$

The model thus predicts the class *negative* for the test sentence.

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 $\{(x_i, y_i)\}_{i=1}^{N}, x_i \in \mathbb{R}^K$.

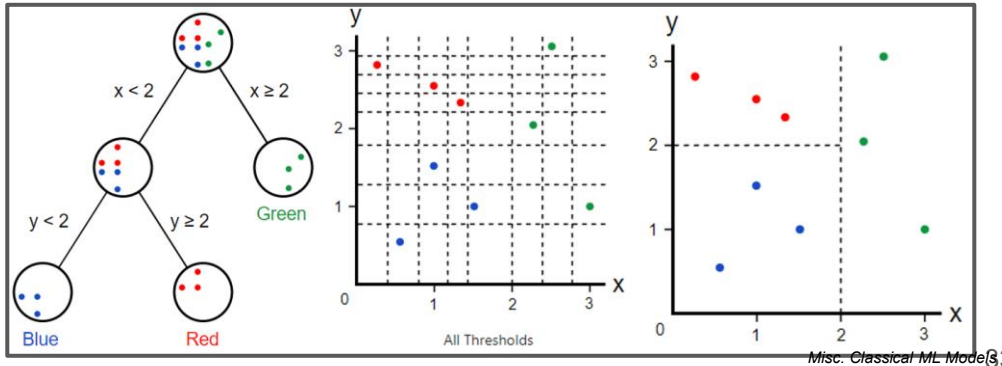
- Recursively, decision nodes are added to the tree, where each node is parametrized by the feature dimension in $\{1, \ldots, k\}$, and a threshold c.
  - There are several ways to decide this, but a common approach is just brute force: try every possible threshold for every dimension, and pick the 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)



What is a Random Forest?

Random Forests are essentially ensembles of T decision trees. This generally reduces variance/overfitting.

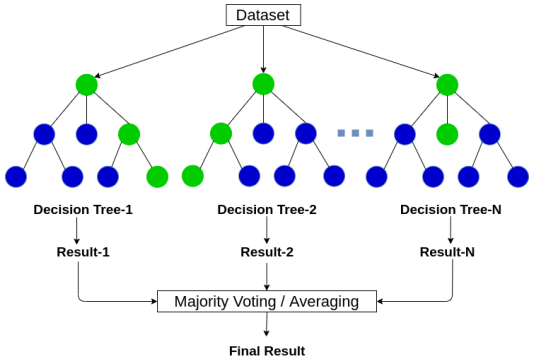
Given a training dataset of size N, for each decision tree:

- Sample, **with replacement**, N training examples
- Train a decision tree with those N samples
    - In the training, each time we make a new decision node, only try a subset of the features (e.g.), to make each tree more random and reduce correlation between trees.

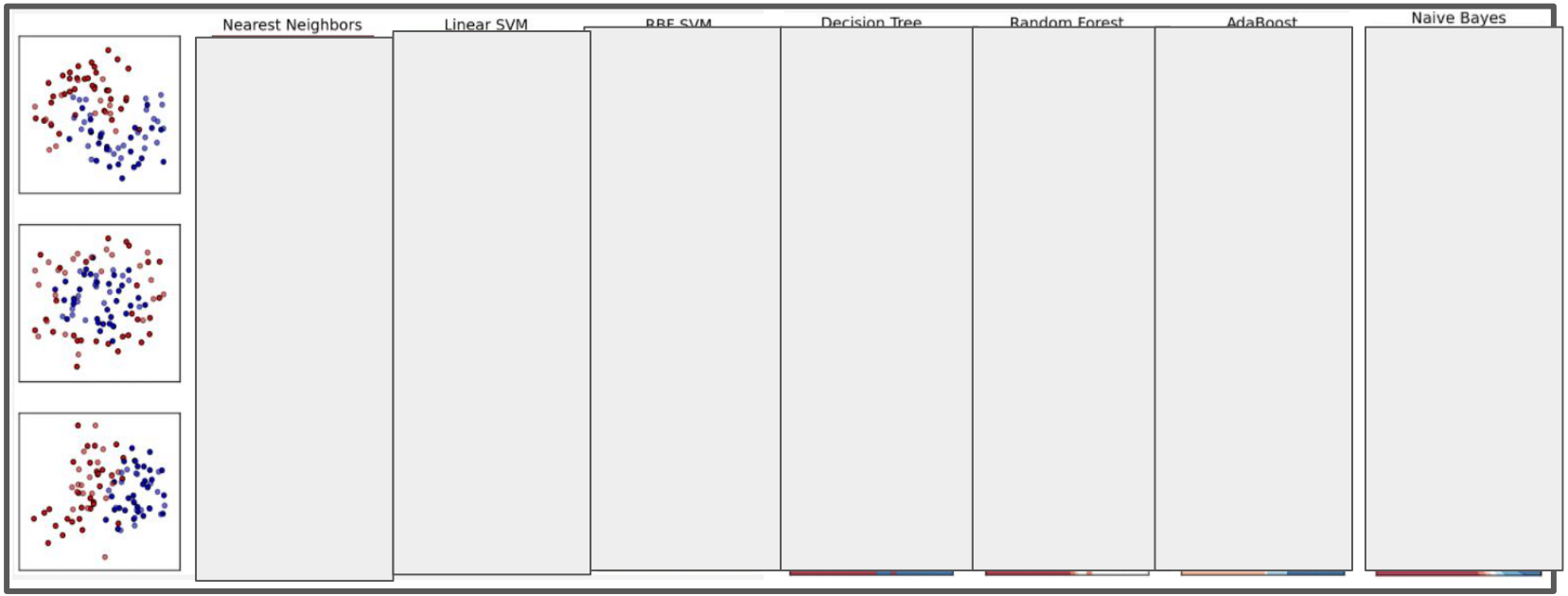
Sampling with replacement and using only a subset of features helps inject more randomness. Then, the final result is either averaged (for regression) or voted upon (for classification).

Some terminology:

- Sampling with replacement is called **bootstrapping**
- The technique of creating an ensemble using bootstrapping is called **bootstrap aggregating** (**bagging**)
- The technique of only trying a subset of the features is called feature bagging.



For the 3 given datasets, visually the decision boundaries for: NN, linear SVM, RBF SVM, Decision Trees, Random Forest, Adaboost, and Naive Bayes.
In relative terms, which ones do you think would perform the best?



At a high level, what is a Bayes Filter? What are Kalman filters?

- **Bayes Filters** estimate an unknown probability density function (PDF) recursively over time, using incoming measurements and a mathematical process model.
  - For noisy measurements are observed over time, Bayes filters give estimates that tend to be more accurate than with a single measurement alone, using temporal smoothing
  - They allow one to integrate a mathematical motion model, prior knowledge, and noisy measurements into a predictive framework which accounts for noise
  - Bayes filters take the Markov assumption: the current robot/environment state only depends on the previous one, and not any more timesteps back.
- **Kalman filters** are an important type of Bayes Filter which takes several assumptions:
  - The prior pdf is Gaussian
  - The motion & observation models are linear in the state, and affected by Gaussian noise
  - The process noise and measurement noise are independent of each other, of the state, and across time
- Given the previous state, current control input, and current measurements, there are two steps which are repeated:
  - **Prediction Step**: Using the previous state, motion model, and noise model, the filter produces a prediction of what the current state is.
  - **Update Step:** Using the next (noisy) measurement, the current estimated state is refined/updated.

- **Pros of Bayes filters**:
  - Well understood, very general formulation with many applications for time-series analysis in engineering, robotics (motion planning/control), signal processing, finance, and economics
  - In CV, can be used for temporal smoothing, eg depth estimation or tracking
  - Can predict future values (using multiple predict steps)
  - Provides a built-in way to integrate prior knowledge about state and motion model
  - Under assumptions, can be proven that it's the best possible linear estimator in the minimum mean-square-error sense
  - Can be run in real time
- **Cons of Bayes filters:**
  - Requires motion model (e.g. based on physics), which may not be apparent. In these cases, an RNN/LSTM might work better. In fact, they can substitute for a Kalman filter in theory, and are more general/can be arbitrarily complex with many parameters.
    - In alternative is to keep the motion model as identity.
  - Assumptions might not hold

Predict:

$$\hat{\mathbf{x}}_{t|t-1} = \mathbf{F}_t\hat{\mathbf{x}}_{t-1|t-1} + \mathbf{B}_t\mathbf{u}_t$$

Current state updated using previous state, motion models, and control input

$$\mathbf{P}_{t|t-1} = \mathbf{F}_t\mathbf{P}_{t-1|t-1}\mathbf{F}_t^T + \mathbf{Q}_t$$

Current state covariance increases, using estimation error and process error distributions

Update:

$$\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + \mathbf{K}_t\left(\mathbf{y}_t - \mathbf{H}_t\hat{\mathbf{x}}_{t|t-1}\right)$$

$$\mathbf{K}_t = \mathbf{P}_{t|t-1}\mathbf{H}_t^T\left(\mathbf{H}_t\mathbf{P}_{t|t-1}\mathbf{H}_t^T + \mathbf{R}_t\right)^{-1}$$

$$\mathbf{P}_{t|t} = (\mathbf{I} - \mathbf{K}_t\mathbf{H}_t)\mathbf{P}_{t|t-1}$$

Refine current state using observations; this decreases covariance.
The Kalman gain intuitively describes how much we trust our observations.
It is multiplied with the "innovation term" y-Hx, which is the difference between the observation and prediction.

$\hat{\mathbf{x}}$: Estimated state.
$\mathbf{F}$: State transition matrix (i.e., transition between states).
$\mathbf{u}$: Control variables.
$\mathbf{B}$: Control matrix (i.e., mapping control to state variables).
$\mathbf{P}$: State variance matrix (i.e., error of estimation).
$\mathbf{Q}$: Process variance matrix (i.e., error due to process).
$\mathbf{y}$: Measurement variables.
$\mathbf{H}$: Measurement matrix (i.e., mapping measurements onto state).
$\mathbf{K}$: Kalman gain.
$\mathbf{R}$: Measurement variance matrix (i.e., error from measurements).