Tree

1. Regression Tree

i. divide the predictor space into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.
ii. For every observation that falls into the region Rj , we make the same prediction, which is simply the ***mean*** of the response values for the training observations in Rj
iii. The goal is to find boxes R1, . . . , RJ that minimize the RSS $\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$

- ***Tree Pruning :*** *grow a very large tree T_0, and then prune it back in order to obtain a subtree.*
  (idea: using cross-validation and find a subtree that leads to the lowest test error rate.) -> but not practical
  Cost complexity pruning: *Rather than considering every possible subtree, we consider a sequence of trees indexed by a nonnegative tuning parameter α.*
  - ***Steps***
    1) Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
    2) Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
    3) Use K-fold cross-validation to choose α. That is, divide the training observations into K folds.
       a) Repeat Steps 1 and 2 on all but the k_{th} fold of the training data.
       b) Evaluate the mean squared prediction error on the data in the left-out k_{th} fold, as a function of α.
       Average the results for each value of α, and pick α to minimize the average error. $\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$
    4) Return the subtree from Step 2) that corresponds to the chosen value of α.
  - ***Properties***
    - When α = 0, then the subtree T will simply equal T_0

2. Classification Tree

predict that each observation belongs to the ***most commonly occurring class*** of training observations in the region to which it belongs. However, in the classification setting, RSS cannot be used as a criterion for making the binary splits.

1) Classification Error Rate : the fraction of the training obs. in that region that do not belong to the most common class.

$$E = 1 - \max_k(\hat{p}_{mk})$$

Here $\hat{p}_{mk}$ represents the proportion of training observations in the m^{th} region that are from the k^{th} class. However, it turns out that classification error is not sufficiently sensitive for tree-growing

2) Gini index : a measure of total variance across the K classes

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

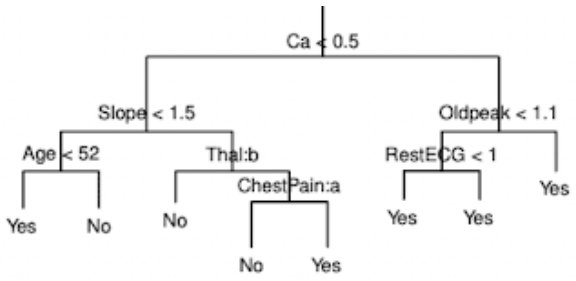
Gini index takes on a small value if all of the $\hat{p}_{mk}$'s are close to zero or one.
a small value indicates that a node contains predominantly observations from a single class.

3) Entropy

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$

Since $0 \le \hat{p}_{mk} \le 1$, it follows that $0 \le -\hat{p}_{mk} \log \hat{p}_{mk}$.
entropy will take on a value near zero if the $\hat{p}_{mk}$'s are all near zero or near one.

- ***Gini index, the entropy will take on a small value if the m_{th} node is pure.***
- *Classification error rate is preferable if prediction accuracy of the final pruned tree is the goal.*



There might be some of the splits yield two terminal nodes that have the same predicted value.
—> to *increase "**node purity**"*

Ensemble method

background: The decision trees suffer from high variance.

1. Bagging: Bootstrap aggregation, a general-purpose procedure for reducing the variance.

- ***main concept:*** *a set of n independent observations Z1,...,Zn, each with variance σ2, the variance of the mean $\bar{Z}$ of the observations is given by σ^2/n. In other words,* ***averaging a set of observations reduces variance.***

- ***method:*** *1) take many training sets from the population*
  2) build a separate prediction model using each training set
  3) average the resulting predictions

$$\hat{f}_{\text{avg}}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{b}(x)$$

But in practical, we generally do not have access to multiple training sets. Instead, we can **bootstrap**

- ***method update:*** *1) we generate B different bootstrapped training data sets*
  2) train our method on the bth bootstrapped training set, get f^*b(x)
  3) average all the predictions

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x)$$

- ***attribute:***
  1) trees are grown deep, and are not pruned
  2) each individual tree has high variance, but low bias
  3) averaging these B trees reduces the variance
  4) using a very large value of B ***will not lead to overfitting***
  5) In ***classification problem***, we can record the class by ***taking a majority vote***: the overall prediction is the most commonly occurring majority class among the B predictions.
  6) difficult to interpret the resulting model (improves prediction accuracy at the expense of interpretability)

- ***Out-of-Bag Error Estimation:*** *way to estimate the test error of a bagged model*
  - ***concept:*** without the need to perform cross-validation, since on average, each bagged tree makes use of around two-thirds of the observations. The ***remaining one-third*** of the observations not used to fit a given bagged tree are referred to as the out-of-bag (OOB) observations.
  - ***method:*** *1) predict the i^{th} observation using each of the trees in which that observation was OOB.*
    2) yield around B/3 predictions for the i^{th} observation.
    3) average these predicted responses(in regression) or take a majority vote(in classification)
    4) do the above steps for n observations and calculate the resulting OOB error

2. ***Random Forests*** *: improvement over bagged trees by decorrelating the trees*

- ***main concept:***
  1) when building decision trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors ($m \approx \sqrt{p}$)
  2) suppose that there is one very strong predictor, most or all of the trees will use this strong predictor in the top split, hence the predictions from the bagged trees will be highly correlated

3. ***Boosting****: trees are grown sequentially*

- ***main concept:***
  1) each tree is grown using information from previously grown trees
  2) Boosting does not involve bootstrap sampling; instead each tree is fit on a modified version of the original data set.
- ***method:***
  1) Set $f(x) = 0$ and $r_i = y_i$ for all i in the training set.
  2) For b = 1,2,...,B, repeat:
     a. Fit a tree $\hat{f}^b$ with d splits (d+1terminal nodes) to the training data (X, r)
     b. Update $\hat{f}$ by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda\hat{f}^{b}(x)$$

     c. Update the residuals

$$r_i \leftarrow r_i - \lambda\hat{f}^{b}(x_i)$$

  3) Output the boosted model $\hat{f}(x) = \sum_{b=1}^{B}\lambda\hat{f}^{b}(x)$

boosting can overfit if B is too large.
We use cross-validation to select B.

shrinkage parameter λ: controls the rate at which boosting learns, Very small λ can require using a very large value of B to achieve good performance.

- ***attribute:***
  1) learns slowly
  2) each tree can be rather small with just a few terminal nodes
  3) fit a decision tree to the residuals from the model

SVM

1. Maximal Margin Classifier

- ***main concept:***
  1) maximal margin hyperplane: the separating hyperplane that is farthest from the training observations
  2) If β0,β1,...,βp are the coefficients of the maximal margin hyperplane, find the sign of f (x∗) = β0 + β1 x∗1 + β2 x∗2 + · · · + βp x∗p .
- ***method:***

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,M}{\text{maximize}} \; M$$

$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M \;\; \forall\, i = 1, \ldots, n.$$

- ***attributes:***
  1) the maximal margin hyperplane depends directly on the support vectors, but not on the other observations (the maximal margin hyperplane depends directly on only a small subset of the observations)
  2) in many cases no separating hyperplane exists, and so there is no maximal margin classifier, so the optimization problem has no solution with M > 0

2. Support Vector Classifier (soft margin classifier)

- ***main concept:***
  1) generalization of the maximal margin classifier to the ***non-separable case***
  2) classification in the ***two-class setting,*** if the ***boundary*** between the two classes ***is linear.***
  3) consider a classifier based on a hyperplane that ***does not perfectly separate the two classes*** so that can achieve ***greater robustness to individual obs.*** and ***better classification of most of the training observations***
  4) Rather than seeking the largest possible margin; instead, allowing some observations to be on the incorrect side of the margin or even the incorrect side of the hyperplane.
- ***method:***

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n,\, M}{\text{maximize}} \; M$$

$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M(1 - \epsilon_i)$$

$$\epsilon_i \geq 0, \;\; \sum_{i=1}^{n} \epsilon_i \leq C,$$

C: nonnegative, bounds the sum of the εi's. chosen via cross–validation

ε1,...,εn are slack variables: allow individual observations to be on the wrong side of the margin or the hyperplane

- ***attributes:***
  1) When C is small, low bias but high variance. When C is larger, more biased but may have lower variance.
- ***<compare to LDA>:***

  SVC: decision rule is based only on a potentially small subset of the training observations

  LDA: depends on the mean of all of the observations within each class, as well as the within-class covariance matrix computed using all of the observations.

3. Support Vector Machines

- ***main concept:***
  1) ***enlarging the feature space*** using functions of the predictors, such as quadratic and cubic terms, to address ***non-linearity problem***

$$\underset{\beta_0,\beta_{11},\beta_{12},\ldots,\beta_{p1},\beta_{p2},\epsilon_1,\ldots,\epsilon_n,\, M}{\text{maximize}} \; M$$

$$\text{subject to } y_i \left(\beta_0 + \sum_{j=1}^{p} \beta_{j1} x_{ij} + \sum_{j=1}^{p} \beta_{j2} x_{ij}^2 \right) \geq M(1 - \epsilon_i)$$

$$\sum_{i=1}^{n} \epsilon_i \leq C, \;\; \epsilon_i \geq 0, \;\; \sum_{j=1}^{p} \sum_{k=1}^{2} \beta_{jk}^2 = 1.$$

problems: end up with a huge number of features, then computations would become unmanageable

2) ***enlarging the feature space*** using ***kernels***

kernel: function that quantifies the similarity of two observations.

- ***method***

  <General Function>: $f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i K(x, x_i)$

  αi is nonzero only for the support vectors in the solution
  S is the collection of indices of these support points

  A. Linear Kernel

  $$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j} \qquad f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i \langle x, x_i \rangle$$

  B. Polynomial Kernel

  $$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d$$

  - d is a positive integer, when d > 1, leads to a much more flexible decision boundary. When the support vector classifier is combined with a non-linear kernel, the resulting classifier is known as a ***support vector machine***.
  - no matter how large d is, the max # of parameters (alpha) we need to learn from the data is n

  C. Radial Kernel

  $$K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2)$$

  - γ is a positive constant, when γ is large, means more local behavior
  - If a test observation $x^* = (x_1^*, \ldots, x_p^*)$ is far from a training observation, then the sigma of distance would be large, so the K would be tiny. This means that xi will play virtually no role in f(x*).
  - observations that are far from x* will play essentially no role in the predicted class label for x* —> ***local behavior***

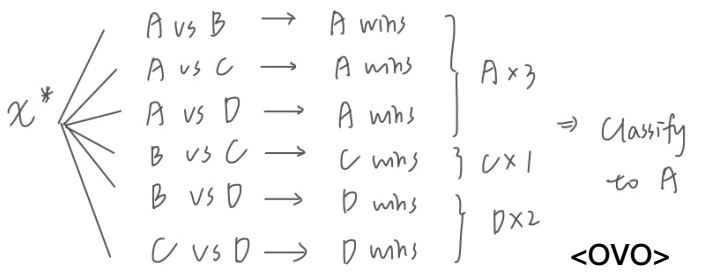
- ***SVM with more than 2 classes***

  A. One-Versus-One Classification

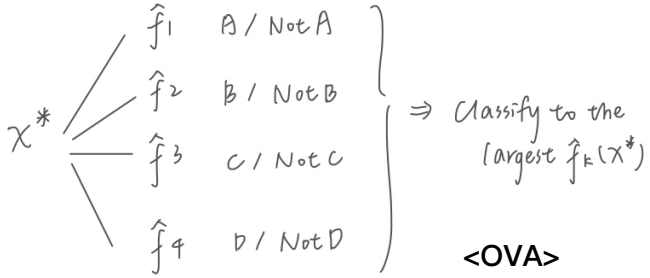
  - ***method***
    1) constructs C^k_2 SVMs, each of which compares a pair of classes
    2) tally the number of times that the test observation is assigned to each of the K classes.
    3) assigning the test observation to the class which it was most frequently assigned in these pairwise classifications

  B. One-Versus-All Classification

  - ***method***
    1) fit K(# of classes) SVMs, each time comparing one of the K classes to the remaining K − 1 classes
    2) Let β0k, β1k, . . . , βpk denote the parameters that result from fitting an SVM comparing the k^{th} classes
    3) assign the obs to the class for which $\beta_{0k} + \beta_{1k} x_1^* \cdots + \beta_{pk} x_p^*$ is largest (x^* denote a test observation)



<OVO>



<OVA>

4. Relationship to Logistic Regression

It turns out that one can rewrite the formula in SVC for fitting the support vector classifier

f(X) = $\beta_0 + \beta_1 X_1 + \ldots.. + \beta_p X_p$ as

$$\underset{\beta_0, \beta_1, \ldots, \beta_p}{\text{minimize}} \left\{ \sum_{i=1}^{n} \max[0, 1 - y_i f(x_i)] + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}$$

<Loss + Penalty>

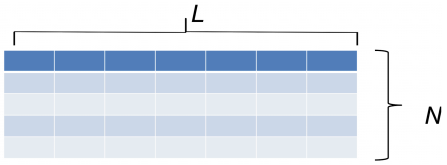
- When **λ is large** then β_1~β_p are small, more violations are tolerated, and a **low-variance but high-bias** classifier will result.
- When **λ is small** then few violations to the margin will occur, this amounts to a **high-variance but low-bias** classifier

$$L(\mathbf{X}, \mathbf{y}, \beta) = \sum_{i=1}^{n} \max[0, 1 - y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip})]$$

5. Evaluation Metrics

- ***Hamming Loss:*** The fraction of the wrong labels to the total number of labels

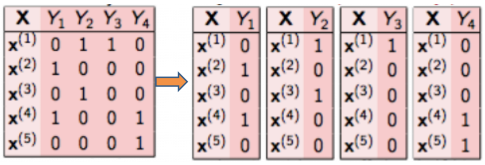
$$\frac{1}{NL} \sum_{i=1}^{N} \sum_{j=1}^{L} I(\hat{y}_{ij} \neq y_{ij})$$



Multi-Label Classification

1. ***Problem Transformation:*** *Transform multi-label problem into single-label problem(s)*

- ***main concept:*** treats each label as a separate binary or multi-class classification problem.

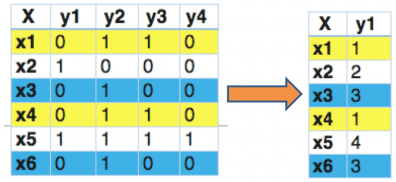


The problem is broken into 4 different single class classification problems. Any classifier (ex. Naïve Baye's, LR, and SVM) can be used for predicting.

- ***main concept:*** the first classifier is trained just on the input data and then each next classifier is trained on the input space and all the previous classifiers in the chain.

colored: input space
white: target variable

- ***main concept:*** transforms the problem into a multi-class problem with one multi-class classifier is trained on all unique label combinations found in the training data.



x_1 and x_4 have the same labels, similarly, x_3 and x_6 have the same set of labels. Label powerset transforms this problem into a single multi–class problem

2. ***Adapted Algorithms:*** *adapting conventional algorithms to directly perform multi-label classification*

The most famous adapted algorithm is Multilabel kNN (MLkNN)

3. ***Ensemble approaches:*** *Combining multiple classifiers*

AdaBoost.MH and AdaBoost.MR

Unsupervised Learning

The goal is to discover interesting things about the measurements:

- Is there an informative way to visualize the data?
- Can we discover subgroups among the variables or among the observations?

1. ***Principal Components Analysis:*** *When faced with a large set of correlated variables, principal components allow us to summarize this set with a smaller number of representative variables*

- ***main concept:***
  1) Principal components analysis(PCA) refers to the process by which principal components are computed
  2) PCA produces a ***low-dimensional representation*** of a dataset. It finds a sequence of linear combinations of the variables that have ***maximal variance***, and are ***mutually uncorrelated***.
  3) Each of the dimensions found by PCA is a linear combination of the p features
  4) PCA was performed after ***standardizing*** each variable to have mean zero and standard deviation one

- ***components:***
  1) **First Principal Component**
     - a set of features $X_1, X_2, \ldots, X_p$ is the normalized linear combination of the features

       $Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + \cdots + \phi_{p1}X_p$, by normalized, we mean that $\text{sigma}(\phi_{j1}^2) = 1$

	X_1	X_2	.....	X_p	Z_1
data1	x_{11}	x_{12}	.....	x_{1p}	z_{11}
⋮	⋮	⋮	⋮	⋮	⋮
datan	x_{n1}	x_{n2}	.....	x_{np}	z_{1n}

     - the elements $\varphi_{11}, \ldots, \varphi_{p1}$ are the loadings of the first principal component
     - Given a $n \times p$ data set, the linear combination of the sample feature values of the form

       $z_{i1} = \phi_{11}x_{i1} + \phi_{21}x_{i2} + \cdots + \phi_{p1}x_{ip}$

       the first principal component loading vector solves the optimization problem:

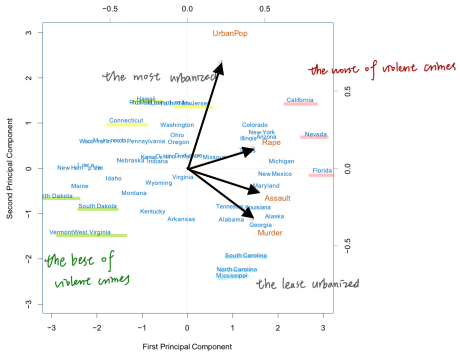
       $$\underset{\phi_{11},\ldots,\phi_{p1}}{\text{maximize}} \left\{ \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{p}\phi_{j1}x_{ij}\right)^2 \right\} \text{ subject to } \sum_{j=1}^{p}\phi_{j1}^2 = 1 \longrightarrow \frac{1}{n}\sum_{i=1}^{n} z_{i1}^2$$

  2) **Second Principal Component**
     - linear combination of $X_1 \sim X_p$ that has max variance of all linear combinations that are uncorrelated with Z_1

       $z_{i2} = \phi_{11}x_{i1} + \phi_{22}x_{i2} + \cdots + \phi_{p2}x_{ip}$

     - constraining Z_2 to be uncorrelated with Z_1 is equivalent to constraining the direction φ_2 to be orthogonal (perpendicular) to the direction φ_1, so once we have found φ_1, there is only one possibility for $\varphi 2$

- ***Sample:***



- ***Another Interpretation of Principal Components:***
  1) loading vectors span a plane along which the observations have the highest variance
  2) principal components provide low-dimensional linear surfaces that are closest to the observations
  3) First principal component loading vector is line in p-dimensional space that is closest to the n observations

- ***Scaling:***
  1) If the variables are in different units, scaling each to have standard deviation equal to one is recommended
  2) If they are in the same units, you might or might not scale the variables

- ***Proportion Variance Explained:***
  - Proportion of Variance Explained (PVE)
    - The total variance present in a data set is defined as $\sum_{j=1}^{p}\text{Var}(X_j) = \sum_{j=1}^{p}\frac{1}{n}\sum_{i=1}^{n}x_{ij}^2$
    - The variance explained by the m_{th} principal component is $\text{Var}(Z_m) = \frac{1}{n}\sum_{i=1}^{n}z_{im}^2$
    - $$PVE_m = \frac{\text{Var}(Z_m)}{\sum_{j=1}^{p}\text{Var}(X_j)} = \frac{\sum_{i=1}^{n}z_{im}^2}{\sum_{j=1}^{p}\sum_{i=1}^{n}x_{ij}^2}$$ *PVEs sum to one*

2. Clustering:

- ***main concept:*** *find* ***homogeneous subgroups*** *among the observations*

A. K-Means Clustering: *partitioning a data set into K distinct, non-overlapping clusters*

- ***method***
  1) first specify the desired number of clusters K
  2) K-means algorithm will assign each observation to exactly one of the K clusters
  3) let "within-cluster variation" is as small as possible

  <Within-Cluster Variation> (here we use squared Euclidean distance)

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

  |Ck| denotes the number of observations in the kth cluster

  4) Algorithms

  - Randomly assign a number, from 1 to K, to each of the observations.These serve as initial cluster assignments for the observations.
  - Iterate until the cluster assignments stop changing:
    - For each of the K clusters, compute the cluster ***centroid***. The k_{th} cluster centroid is the vector of the p feature means for the observations in the k_{th} cluster.
    - Assign each observation to the cluster whose centroid is closest (where closest is defined using Euclidean distance).

> - *Algorithm above is guaranteed to decrease the value of the sigma(W(Ck)) at each step*
> - *Algorithm above is not guaranteed to give the global minimum (since it does not consider all possible clusters)*

B. K-Medoids Clustering

- ***main concept:***

Similar to K-means, but in each step we do not compute the centroid of each cluster. We compute the ***medoid***, which is a data point that has the smallest average dissimilarity with other data points in the cluster.
(medoid needs to be a "data point", not a "value")

<K-Means v.s. K-Medoids>

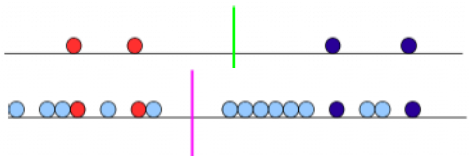
- *K-Medoids run longer($O(n^2)$) since it needs to calculate the distances between all pairs of nodes, and K-means only need to calculate the average*
- *K-Medoids is more robust to noise. ex. (1,1), (1,2), (2,1), (1000,1000). K-Medoids would pick a data point from the first three points and K-Means would find a*

C. Hierarchical Clustering

- ***main concept:***
  1) an alternative approach which does not require that we commit to a particular choice of K
  2) results in an attractive tree-based representation of the observations, called a dendrogram.
- ***sample (bottom-up clustering):***
  1) built starting from the leaves and combining clusters up to the trunk
  2) The earlier (lower in the tree) fusions occur, the more similar the groups of observations are to each other
  3) The height of this fusion, as measured on the vertical axis, indicates how different the two observations are
  4) cannot draw conclusions about the similarity of two observations based on their proximity along the horizontal axis.
  5) one single dendrogram can be used to obtain any number of clusters
- ***define the dissimilarity between two clusters:***
  - ***linkage*** : *defines the dissimilarity between two groups of observations*
    1) **Complete**: ***Maximal*** *intercluster dissimilarity. Compute* ***all pairwise*** *dissimilarities between the observations in cluster A and the observations in cluster B, and record the* ***largest*** *of these dissimilarities.*
    2) **Average**: ***Mean*** *intercluster dissimilarity. Compute* ***all pairwise*** *dissimilarities between the observations in cluster A and the observations in cluster B, and record the* ***average*** *of these dissimilarities.*
    3) **Single**: ***Minimal*** *intercluster dissimilarity. Compute* ***all pairwise*** *dissimilarities between the observations in cluster A and the observations in cluster B, and record the* ***smallest*** *of these dissimilarities.*
    4) **Centroid:** *Dissimilarity between the* ***centroid*** *for cluster A (a mean vector of length p) and the* ***centroid*** *for cluster B. Centroid linkage can result in undesirable inversions.*

Semi-Supervised Learning

- ***main concept:***
  1) Unlabeled data can give a better sense of the class separation boundary
  2) Supervised learning + Additional unlabeled data
  3) Unsupervised learning + Additional labeled data



- ***Inductive & Transductive:***
  1) Transductive: Produce label only for the available unlabeled data (output is not a classifier)
  2) Inductive: Not only produce label for unlabeled data, but also produce a classifier.

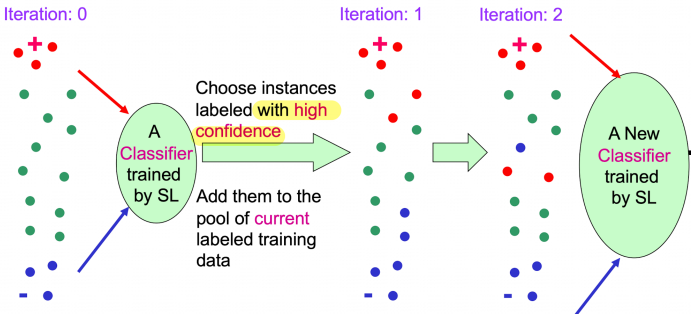
==========***Here we only talk about Inductive Semi-Supervised Learning***==========

- ***Data Based Method***
  - ***main concept:*** Discover an inherent geometry in the data, and exploit it in finding a good classifier.
- ***Classifier Based Method***
  - ***main concept:*** Start from initial classifier(s), and iteratively enhance it (them)

A. Self-Training

- ***method***
  1) Train supervised model on labeled data L
  2) Test on unlabeled data U
  3) Add the most confidently classified members of U to L
  4) Repeat

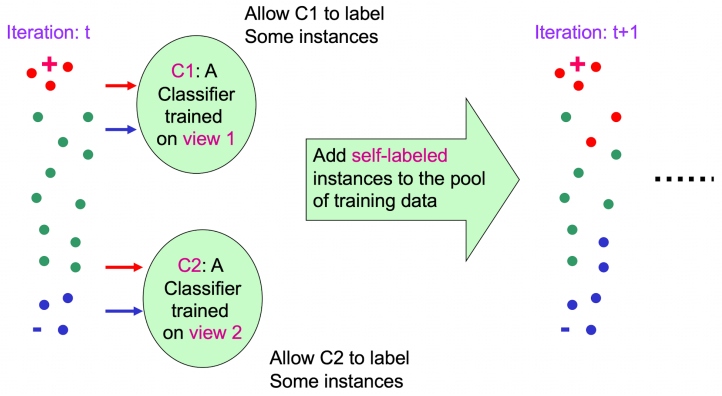
  (we can reduce weight of unlabeled data to increase power of more accurate labeled data)



- ***Advantages:***
  1) The simplest semi-supervised learning method, Often used in real tasks like natural language processing.
  2) A wrapper method, applies to existing (complex) classifiers.
- ***Disadvantages:***
  1) Early mistakes could reinforce themselves.

B. Co-Training

- ***assumption:*** if instances contain two sufficient sets of features
- ***method***



- Two views are independent given the label
  ➡ Reduces the probability of the models agreeing on incorrect labels
- Two views are consistent:

C. Cluster-And-Label

- ***assumption:*** Clusters coincide with decision boundaries (would have a poor result if this assumption is wrong)
- ***method***
  1) Cluster of labeled and unlabeled data
  2) For each cluster, train a classifier based on the labeled points within that cluster
  3) Label all data in each cluster using the classifier designed for that cluster
  4) Train a model based on the whole data (that is now labeled)

D. Active Learning

- ***method***
  1) proceeds in rounds, each round has a current model
  2) The current model is used to assess informativeness of unlabeled examples
  3) The most informative(the most uncertain about) example(s) is/are selected, the labels are obtained
  4) The (now) labeled example(s) is/are included in the training data. The model is re-trained using the new training data
  5) The process repeat until we have budget left for getting labels

Neural Networks and Deep Learning

1. Perceptron: *an algorithm for* **binary classification** *that uses a* **linear prediction function**

$$f(\mathbf{x}) = \begin{cases} 1, & \boldsymbol{\beta}^T\mathbf{x} + \beta_0 \geq 0 \\ -1, & \boldsymbol{\beta}^T\mathbf{x} + \beta_0 < 0 \end{cases}$$

- β=(β1, β2 ,..., βp), these parameters are called weights
- β parameters are unknown. This is what we have to learn

- ***method:***
  1) Initialize all weights β to 0 or randomly
  2) Iterate through the training data. For each training instance, classify the instance.
     a. If the prediction was correct, don't do anything.
     b. If the prediction was wrong, modify the weights by using the ***update rule***.(ex. If the classifier predicted positive but it should have been negative, shift the weights so that the value moves toward negative.)
  3) Repeat step 2 some number of times

- ***update rule:***

$$\boldsymbol{\beta}(i+1) = \boldsymbol{\beta}(i) + 0.5[y(i)-f(\boldsymbol{\beta}^T(i)\mathbf{x}(i)+ \beta_0(i))]\,\mathbf{x}(i) \longrightarrow \beta(i+1) = \beta(i) + 0.5[\text{“true label”} - \text{“predict label”}]*x(i)$$

- If true label = -1 and predict label = +1 : 0.5*-2 = -1, so the weights will move away from x(i) to make $\beta^Tx(i) + \beta_0$ more negative and x(i) is more likely to be classified correctly.
- If true label = +1 and predict label = -1 : 0.5*2 = 1, so the weights will move towards x(i) to make $\beta^Tx(i) + \beta_0$ more positive and x(i) is more likely to be classified correctly.

<Perceptron v.s. SVC>

- *both are* ***linear classifiers***
- ***SVC's*** *optimization* ***needs all data*** *to yield a classifier, so it is a batch algorithm.*
- ***Perceptron*** *adapts its weights online with* ***the data points that are presented to it****. Thus it is an online algorithm.*

- ***attributes:***
  1) If the training instances are linearly separable, the perceptron algorithm will eventually find weights W such that the classifier gets everything correct.
  2) If the training instances ***are not linearly separable***, the classifier will always ***get*** some predictions ***wrong***. So we need to implement some type of stopping criteria(ex. maximum number of iterations or epochs)

2. Multi-Class Perceptron: *A maximum of $K = 2^S$ classes can be encoded as vectors*

Example:
Assume that biases are kept zero and $w_1(1) = [1\ 1]^T$, $w_2(1) = [1\ -1]^T$.
Assume that $x(1) = [1\ 0]^T$ is in a class encodes as $C_1 = [1\ -1]^T$ and $x(2) = [0\ -1]^T$ is in a class encoded as $C_0 = [-1\ -1]^T$.
Update the weights according to the perceptron rule.

$W_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ (First Hyperplane, Second Hyperplane)

$e(1) = \begin{bmatrix} 1 \\ -1 \end{bmatrix} - f(w_1^T x + 0)$ ∵ bias = 0 (assumption)

$= \begin{bmatrix} 1 \\ -1 \end{bmatrix} - f\left(\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix}\right)$

$= \begin{bmatrix} 1 \\ -1 \end{bmatrix} - f\left(\begin{bmatrix} 1 \\ 1 \end{bmatrix}\right)$

$= \begin{bmatrix} 0 \\ -2 \end{bmatrix}$ → the first hyperplane doesn't need to change; → the second hyperplane needs to change

$W_2 = W_1 + 0.5\,x(1)\,e^T(1)$

$= \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} + 0.5\begin{bmatrix} 1 \\ 0 \end{bmatrix}\begin{bmatrix} 0 & -2 \end{bmatrix}$

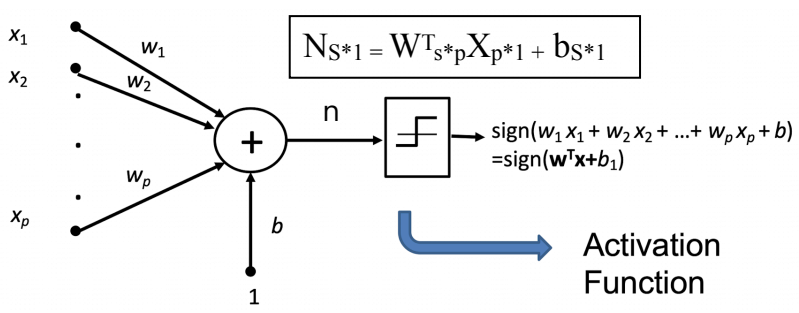
$= \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} + 0.5\begin{bmatrix} 0 & -2 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix}$

- ***learning rate:***

$$\mathbf{W}(i+1) = \mathbf{W}(i) + \alpha\mathbf{x}(i)\mathbf{e}^T(i) \qquad \mathbf{b}^T(i+1) = \mathbf{b}^T(i) + \alpha\mathbf{e}^T(i)$$

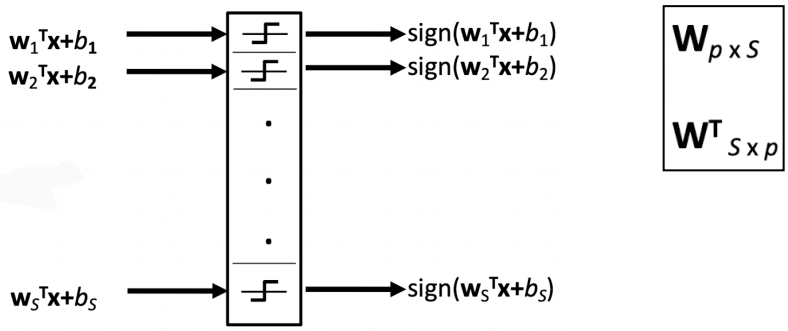
α is called the learning rate or step size, it controls how large a "step" you take.

A. Single Neuron



$$N_{S*1} = W^T{}_{s*p}X_{p*1} + b_{S*1}$$

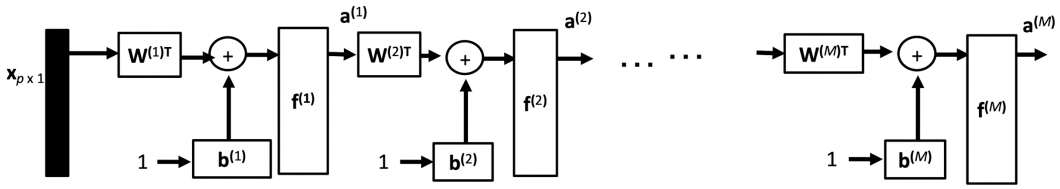
B. *Multiple Neuron*



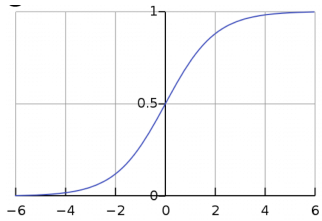
- ***methods:***
  1) acts like a simple neural system: it adapts its weights based on the error that is a result of mismatch between its output and the desired output
  2) This general process is the basis of supervised learning with ***a large class of neural networks***.
  3) If the training set is not separable, the Perceptron rule never converges.
     a. General Solution: ***represent the data in a feature space where they are linearly separable.*** (use ***layers*** of perceptrons and adjust their weights to learn the feature representations)

3. ***Multi-Layer Perceptron (MLP):*** *consists of multiple layers of Perceptrons*

- ***main concept:***
  1) Each layer may have a different number of neurons
  2) ***Different types of activation functions can be used***, not just the sign (threshold) function
  3) can essentially be represented as nested functions: $g(x) = g^{(M)}(g^{(M-1)}(\ldots(g^{(3)}(g^{(2)}(g^{(1)}(x))))...))$
     - Each g(i) is a layer of neurons with its weight matrix W(i) and bias vector b(i) and activation function f(i)
     - $a^{(i)}$ (the output of the ith layer) = $f^{(i)}(W^{(i)T}a^{(i-1)}+b^{(i)}) = g^{(i)}(a^{(i-1)})$
  4. Weights in each layer can be learned so that the ***MLP is used for either classification or regression tasks***

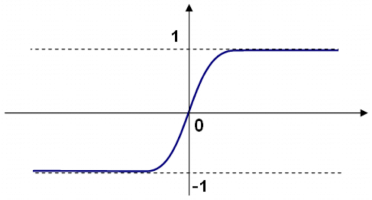


- ***type of Activation Functions:***
  1) **Sigmoid Function**: commonly used for both classification and regression tasks



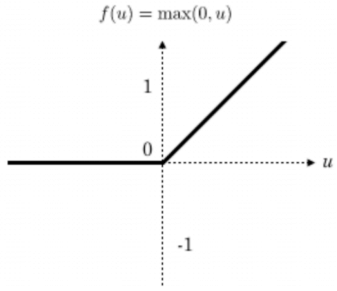
$$f(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$

  2) **Tanh Sigmoid Function**: commonly used for both classification and regression tasks



$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x}-1}{e^{2x}+1}$$

  3) **ReLu (Rectified Linear Unit)**: commonly used for modern practices for training deep neural networks



- ***powerful for Regression:***
  - any smooth function can be approximated with any precision using an MLP with only two layers:
    a. A ***hidden layer*** of smooth nonlinear functions (e.g. sigmoids)
    b. An ***output layer*** of ***linear function***
    3
- ***powerful for Classification:***
  - MLPs can be used for classification usually with at least:
    a. A ***hidden layer*** of smooth nonlinear functions (e.g. sigmoids)
    b. An ***output layer*** of ***sigmoid, softmax*** followed by threshold functions
  - Classes are usually encoded using Binary encoding/ One hot encoding
- ***training MLPs:***

A. Backpropagation