**INTRODUCTION TO STATISTICAL LEARNING** WITH APPLICATIONS IN R

2. Statistical learning:

* Estimating f
* Trade-off b/w prediction accuracy and model interpretability
* Supervised vs. Unsupervised learning
* Regression vs. Classification
* Assessing Model Accuracy
  + Measuring quality of fit
  + Bias-variance trade-off
  + classification setting

3. Linear Regression

**LECTURE 11: TREE-BASED METHODS (REGRESSION TREES)**

1. Intro

* Main idea: segment the predictor space into a number of simple regions (e.g. rectangles), then fit a very simple model in each region.
* The set of splitting rules used to segment the predictor space can be summarized in a tree
* At a given internal node (higher in the branch), the label (Xj < t) indicates the left-hand branch, and the right-hand branch indicates X >= t
* Terminal nodes = leaves (end of the tree)
* The number in each leaf is the mean of the response for the observations that fall there
* Terminology for trees:
  + In keeping with the tree analogy, the regions R1, R2, and R3 are known as *terminal nodes*. (this is where it stops – no longer split after terminal nodes)
  + Decision trees are typically drawn upside down, in the sense that leaves are at the bottom of the tree
  + The points along the tree where the predictor space is split are reffered to as *internal nodes*

2. Tree-building algorithm

* We divide the predictor space into J non-overlapping regions, R1,…, Rj
* For every obs that falls into the region Rj, we make the same prediction, which is simply the mean of the response value for the training obs in Rj. (Take the mean of the region and use this as the predicted value of that region)
* The goal is to find boxses that minize the RSS (residual sum of square)
* There are countless ways of partition the feature space into J boxes

=> Take a top-down, greedy approach known as *recursive binary splitting* (can only give rectangles regions)

- Top down: it begins at the top of the tree and then successively splits the predictor space

- Greedy: at each step of splitting, the best split is made at that particular step, rather than looking ahead, and picking a split that will lead to a better tree in some future step.

* Splitting X = {X1,…, Xp}
* First select the predictor Xj and the cut-point s s.t. splitting the predictor space into the region Xj < s and Xj >= s leads to the greatest possible reduction in RSS
* Next, we repeat the process, looking for the best predictor and best cut-point in order to split the data further so as to minimize the RSS within each of the resulting regions
* Instead of splitting the entire predictor space, we split one of the 2 previously identified regions and get three regions
* Again, we look to split one of these 3 regions future, so as to minimize the RSS
* The process continues until a stopping criterion is reached. For instance, we may continue until no region contain more than n\_min obs.
* We predict the response for a given test observation using the mean of the training obs in the region to which the test obs belongs

3. Pruning a Tree

* The process described above may produce good predictions on the training set, but is *likely to overfit the data*
* *A smaller tree with fewer splits might lead to lower variance* and better interpretation at the cost of a little bias
* One possible alternative is to grow the tree only so long as the decrease in the RSS due to each split exceeds some threshold
* This strategy will result in smaller trees, but can be short-sighted: a seemingly worthless split early on the in tree might be followed by a very good split.

=> A better strategy is to grow a very large tree To, and then prune it back in order to obtain a subtree (CART)

* Cost -complexity pruning is used to do this
* We consider a sequence of tree indexed by a non-negative tuning parameter alpha
* For each value of alpha there is a subtree T subset of To s.t RSS + alpha|T| is as small as possible => |T| = number of terminal nodes of tree T.
* Choosing the best subtree:
  + The tuning parameter alpha controls a trade-off between the subtree’s complexity and its fit to the training data
  + We select an optimal value alpha\_hat using cross-validation
  + We then return to the full data set and obtan the subtree corresponding to alpha\_hat

Summary

* Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal nodes has fewer than n-min obs
* Apply cost complexity pruning to the large tree in order to obtain a sequence of the best subtrees as a function of alpha
* Use K-fold cv to choose alpha. For each k = 1,…,K:
  + Repeat steps 1 and 2 on the (K-1)/K-th fraction of the training data, excluding the k-th fold
  + Evaluate the mean squared prediction error on the data in the left-out k-th fold, as a function of alpha
  + Average the results, pick alpha to minimize the average error
* Return the subtree from Step 2 that corresponds to chosen value of alpha.

- Choosing alpha:

* as we increase tree size MSE decrease
* but look at the CV error and pick tree size with smallest CV error.

- Implementation in R: rpart (CART), tree, party, partykit

**LECTURE 12: CLASSIFICATION TREES / ENSEMBLE METHODS**

**I. Classification trees**

- Similar to a regression tree, except that it is used to predict a categorical variable

- For a classification tree, we predict that each obs belong to the most commonly occurring class of training observations in the region to which it belongs.

- Details of classification trees:

* we use a recursive binary splitting to grow a classification tree (top down approach
* Regression tree: RSS (residual sum of square) as splitting criteria: find split that minimize RSS of the tree
* But for classification tree, RSS may not be used as a criterion.
* Alternative: classification error rate (proportion of the misclassified)
* This is simply the fraction of the training obs in that region that do not belong to the most common class: 1 – max(pmk) where pmk  = proportion of training obs in the m-th region that are from the k-th class.
* However, classification error is not sufficiently sensitive for tree growing. In practice, two other measures are preferable.

Gini index (commonly used)

* = sum of k = 1 to K of pmk(1 - pmk)
* takes on small value if all the pmk are close to 0 or 1
* Gini index is referred to as a measure of node purity – a small value indicates that node contains predominantly observations from a single class

=> wants as small as possible

Cross – Entropy

* An alternative to the Gini index is cross-entropy: sum of pmk\*log(pmk)
* Gini and cross-entropy very similar numerically

Pruning: cost complex pruning. Classification error or weighted node purity as the lost function.

- Tree usually requires bigger sample size than linear to get the same accuracy.

- Tree-based methods work better than linear if the truth is non-linear.

Pros and cons

* Trees can be displayed graphically, and are easily interpreted even by a non-expert
* However, a single tree generally does not have the same level of predictive accuracy as some of the other regression and classification approaches
* A small change in the data can cause a large change in the final estimated tree

=> By aggregating many decision trees, the predictive performance of trees can be substantially improved.

**II. Bagging and RF**

1. Ensemble methods

* Get data => Fit some models => Combine models
* Ensemble methods use collections of models to get better predictive performance than a single model
* Wisdom of crowds (Bagging, Random forest)
* Wisdom of weighted crowds of experts (Boosting)

2. Bagging

* Bootstrap aggregation, or bagging, is a general purpose procedure for reducing the variance of a statistical learning method.
* Recall that given a set of n independent observation Z1,…, Zn each with variance sigma^2, the variance of the mean Z\_bar of the observations is given by sigma^2/n
* Averaging a set of obs reduces variance
* However, this is not practical b/c we generally do not have access to multiple training sets
* Instead, we can bootstrap, by taking repeated samples from the training data set.
* We generate B different bootstrapped training data sets
* For regression: we train out method on the b-th bootstrapped training set in order to get the predictions at a point x. We average all the predictions to obtain f(x).
* For classification: for each test obs, we record the class predicted by each of the B trees and take a majority vote.

3. Out-of-bag error estimation

* We can predict the response for the i-th obs using each of the trees in which that obs was OOB
* this will yield around B/3 predictions for the it hobs, we then take average/majority vote to get a single prediction
* An OOB prediction can be obtained in this way for each of the n obs, from which the overall OOB MSE or classification error can be computed.

4. Random Forests

* Rf provide an improvement over bagged trees by decorrelating the trees – reduce the variance when we average the trees.
* We build a number of decision trees on bootstrapped training samples
* But when building these decision trees, each time a split in tree is considered, **a random selection of m predictors** is chose as split candidates from the full set of p predictors.
* A fresh selection of m predictions is taken at each split, and typically we choose m ~ square root (p) .
  + - if m = p => bagging
* Example: Gene Expression data
  + We applied RF to a high-dimensional biological data set consisting of expression measurements of 4718 genes measured on tissue samples from n = 349 patients
  + Each of the patient samples has a qualitative label with 15 different levels: either normal or one of 14 different types of cancer
  + We use RF to predict cancer type based on the 500 genes that have the largest variance in the training set
  + We randomly divided the obs into a training and a test set, and applied RF to the training set for 3 differnet values of the number of splitting variables m.

**III. Boosting**

1. Boosting

* Boosting is a general approach that can be applied to many statistical learning methods for regression or classification
* In bagging, each tree is built on a bootstrap data set, independent of the other trees
* Boosting works in a similar way, except that the trees are grown sequentially: each tree is grown using info from previously grown trees.
* Idea behind Boosting algorithm:
  + Given the current model, we fit a decision tree to the residuals from the model. We then add this new decision tree into the fitted function in order to update the residuals.
  + Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d.
  + By fitting small trees to the residuals, we slowly improve f in areas where it does not perform well
  + The shrinkage parameter lambda slows the process down even further, allowing more and diffent shaped trees to attack the residual.
  + Learn slowly!

2. Tuning parameters for boosting

* The number of trees B
  + Boosting can overfit if B is too large
  + Cross-validation to select B
* The shrinkage parameter lambda
  + A small positive number (typical values are 0.01 or 0.001)
  + Control the rate at which boosting learns
  + Very small lambda can require using a large B to achieve good performance
* The number of splits d in each tree
  + Controls the complexity of the boosted ensemble
  + Often d = 1 works well (additive model)
  + More generally d is the interaction depth, and controls the interaction order of the boosted model

3. Boosting for Classification: AdaBoost

* Combine weak learners to make a strong learner
* Basic idea: repeatedly fit classification trees to weighted versions of the training data and update the weights in order to better classify previously misclassified observations
* Weighting

**LECTURE 13: SUPPORT VECTOR MACHINE**

SVM: Efficient learning algorithms for non-linear functions

* quadratic optimization: easy to solve

Linear: Maximal margin classifier => support vector classifier => SVM

- **What is hyperplane?**

* a hyperplane in p dimensions is a flat affine subspace of dimension p – 1
* General question for hyperplane: Beta\_0 + Beta\_1\*X\_1 + … + Beta\_p\*X\_p = 0
* in p = 2 dimensions a hyperplane is a line
* If Beta\_0 = 0, the hyperplane goes through the origin
* The vector Beta = (Beta1,…, Beta\_p) = normal vectors: points in a direction orthogonal to the surface of a hyperplane

**Maximal margin classifier**

- Among all separating hyperplanes, find the one that makes the **biggest gap** or margin between the 2 classes (perpendicular distance between decision boundary and the closest data point)

=> find the separating hyperplane s.t. no data points are allowed to be within the streak (the streak as wide as possible).

- Lagrangian dual objective function: solving this function will solve the original problem.

**Support vectors**

* the alpha parameters are 0 for all the dat points that are not on the margin
* The decision function is fully specified by a small subset of training data – the support vectors
* Support vectors (SV) are the data points that lie closest to the decision surface
* SVs are elements in the training set that would change the position of the hyperplane if removed
* SVs are critical elements of the training set.
* only the data points with non-zero alpha values can contribute to the decision boundary.

Non-separable data

- The data are not separable by a linear boundary. this is often the case, unless n < p.

Noisy data

- sometimes the data are separable, but are noisy

- This can lead to a poor solution for the maximal margin classifier

- the support vector classifier maximizes a *soft margin*

- Relax the hard margin constraints to give a soft margin and allows some of the training set data points to be misclassified.

Feature expansion

- Enlarge the space of features by including transformations

- Go from p features to M features > p

- Lead to the non-linear decision boundaries in the original space

**Non-linearities and kernals**

- Polynomials get wild fast

- There is more elegant and controlled way to introduce nonlinearities in support vector classifiers – through the use of kernels

**Kernels and SVM**

- If we can compute inner-products between obs, we can fit a SV classifier

Polynomial kernel

Radial kernal

**Which to use: SVM or logistic regression?**

- When classes are (nearly) separable, SVM does beter than LR

- When not, LR (with ridge penalty) and SVM are similar

- If you wish to estimate probabilities, LR is the choice

- For nonlinear boundaries, kernel SVMs are popular

- SVM should influence the optimality of the separating hyperplane

**LECTURE 14: BLACK-BOX MODELS**

**Interpreting black-box models**

- *Global interpretation* (interpret how we make predictions for obs => observation-specific)

* variable importance: identify the variables with the largest overall impact
* partial dependence plots: the typical influence of a feature on the response variable across all obs (visualize of some high influenced variables) – try to report overall influence => may not give us a view of the underlying model
* individual conditional expectation curves: the dependence of predicted response on a feature for each obs separately –

- *Local interpretation* (interpreting things related to variables – local impact)

* Given a new observation, what were the most influential variables that determined the predicted outcome?
* lime (Local Interpretable Model-agnostic Explanations)
* Vignette: Understanding `lime`
* breakdown plot: another tool for local interpretation

Partial dependence plots: averaging over ICE curves

- plot one function (takes only 1 or 2 arguments => easy to viz)

- Subvector Xs of l < p predictors . C is the complement set of S => C (union) S = {1,2,…,p}

- The partial dependence of f(X) on Xs is f(Xs) = E[f(Xs, Xc)]

ICE curve

* special case: data has symmetric structure ( Y vs. predictor X), 2 underlied subgroups (subgroup 1: Y increase as X increase and subgroup 2: Y decrease as X increase) => Partial dependence plot: will be flat (average influence of predictor X flats) and does not influence but actually X does affect the response variable if we plot ICE).
* If just look at PDP might not see the whole story => need to look at ICE too.
* centered version: look at the trend more clearly

Local Interpretation

* Assumption: The complex models are linear on a local scale
* *Fit a simple model around a single obs* that mimic how the global model behaves at that locality
* The simple model can be used to explain the prediction
* LIME algorithm creates explainers (pick important features that determine the outcomes)

Breakdown plot

- breakdown f(X) function into mean prediction + sum of contribution of all the variables

**LECTURE 15: UNSUPERVISED LEARNING (CLUSTERING)**

**I - Intro & K-means Clustering**

**1. SL vs. UL**

- Supervised learning (SL)

* observed both features X1,…, Xp and a response Y
* the goal is then to predict Y using X1,…,Xp

- Unsupervised learning (UL)

* only observe features X1,…,Xp
* Goals:
  + find interesting pattern about data
  + informative way to visualize data?
  + subgroups among variables/obs?
* Methods:
  + Clustering: discovering unknown subgroups
  + Principal Component Analysis: most commonly used tool for dimension reduction

2. Why UL?

* more subjective than SL
* no simple goal for the analysis i.e. prediction
* customization: E.g:
  + subgroups of breast cancer patients grouped by their gene expression measurements
  + groups of shoppers characterized by their browsing and purchase histories
* advantages: easier to obtain unlabeled data

3. PCA vs. Clustering

* PCA looks for a low-dimensional rep of the obs that explains a good fraction of the variance
* Clustering looks for homogenous subgroups among the obs

4. Clustering

* refers to a very broad set of techniques for finding subgroups/clusters in a data set
* seek a partition of the data into distinct groups so that the obs within each groups are quite similar
* What do we need for clustering?
  + proximity measure = dissimilarity measure: small if xi, xj are similar.
    - take 2 obs and take distance between these obs (Euclidean, Manhattan, Minkowski)
  + criterion function to evaluate a clustering (Cluster Evaluation)
    - intra-cluster cohesion:
      * how near the data points in a cluster to the cluster centroid?
      * Sum of squared error = commonly used measure
    - inter-cluster separation:
      * different cluster centroids should be far away from one another

TWO CLUSTERING METHODS: K-means clustering (partition obs into a pre-specified number of clusters + Hierarchical clustering (tree-like visual rep => dendrogram, don’t know in adv how many clusters we want)

a) K-means Clustering: requires us to pre-specify the number of clusters

- A good clustering is one for which the within-cluster variation is the smallest.

=> Within-cluster variation (WCV)

- Algorithm:

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

=> The algorithm is guaranteed to decrease the value of the objective function at each step, but not guaranteed to give the global minimum.

- Pros and Cons of K-means:

* The users need to specify K
* sensitive to outliers
* not suitable for discovering clusters that are not hyper-ellipsoids

b) Hierarchical clustering: does not require that we commit to a particular choice of K

- Agglomerative (bottom-up) algorithm: begin with each element as a separate cluster and merge them into successively clusters

- Divisive (top-down) algorithm: begin with the whole set and proceed to divide it into successively smaller clusters.

*Agglomerative (Bottom up)*

* Start with each point in its own cluster
* Identify the closest 2 clusters and merge them
* Repeat
* Ends when all points are in a single cluster

Monotonicity property: The dissimilarity b/w merged clusters is monotone increasing with the level of the merger.

- Common ways to measure cluster distance:

* (Closest 2 datapoints) Single linkage: Minimal inter-cluster dissimilarity (potentially long and skinny clusters)
* (Longest 2 datapoints) Complete linkage: Maximal inter-cluster dissimilarity (compact clusters)
* (Mean) Average linkage: Mean inter-cluster dissimilarity
* Centroid linkage: Dissimilarity between the centroids

- Choice of dissimilarity measure:

* so far have used Euclidean distance
* an alternative is correlation-based distance which considers 2 obs to be similar if their features are highly correlated
* This is an unusual use of correlation, which is normally computed b/w variables, here it is computed b/w the obs profiles for each pair of obs.

**LECTURE 16: NEURAL NETWORKS**

- Neural networks = a very flexible supervised learning algorithm with an analogy of brain neuron structure

- Deep learning: a powerful set of techniques in training deep newral networks (high-level features)

- Neural networks and deep learning have proven highly successful across a wide range of problems including image recognition, speech recognition, and natural language processing.

- Inspired by biological neural networks:

* a series of interconnected neurons
* neurons interact with their neighbors through synapses to dendrites on other neurons.
* If the sum of the input signals into one neuron surpasses a certain threshold, the neuron sends an action potential at the axon hillock and transmits this electrical signal along the axon.

- A single neuron:

* if have a lot of layers in the network and change the 1st layer => impact on the network is very small.

- From biological to artificial:

* first layer = input layer (original) => hidden layers (vanilla neural network: only single hidden layer) => output layer
* neuron => node in a directed acyclic graph (DAG)
* synapse => multiplier (weight) on each edge
* electrochemical pulse => activation function
* biological neural network => collection of neurons into a DAG

- Fitting neural networks

* Find theta that minimizes the loss function



* The generic approach to minimize loss function is by gradient descent, called “back-propagation” in this setting: G(x) = argmaxf(k)
* An idea of mini batch gradient descent can be used to deal with large training data – estimate the gradient using a randomly chosen mini-batch of the training data.

- Model tuning

* weight decay: analogous to ridge regression
* dropout is an additional regularization method (to control overfitting)

- More hidden layers: Given enough hidden neurons, any continuous function can be realized by a neural network with one hidden layer.

- Deeper networks: perform well if dimensions are efficiently large (i.e. large datasets)

- Variants of neural networks:

* Convolutional neural network (image processing)
* recurrent neural network (text and speech processing)

- Implementations in R: keras, tfruns