**Module 4 Notes**

k-NN Algorithm – classification algorithm

**Classification Errors and Metrics**

**Classification Error**

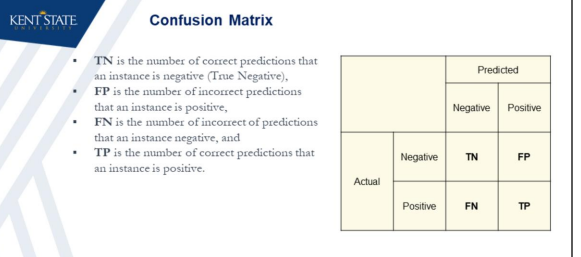
* Classifying a record as belonging to one class when it belongs to another class
* Suppose the target categorical variable has two levels: “True” and “False”. Two types of classification errors are possible
  + Observation is false but classified as true (false positive)
  + Observation is true but classified as false (false negative)
* Two group classification problem – four results
  + The observation is correctly grouped
    - True positive, true negative
  + The observation is incorrectly identified
    - False negative, false positive

**Examples**

* Declining the loan application of a creditworthy applicant (false negative)
* Approving the loan application of a customer who was not creditworthy (false positive)
* Diagnosing a patient with cancer by mistake (false positive)
* Not detecting a tumor in a patient (false negative)

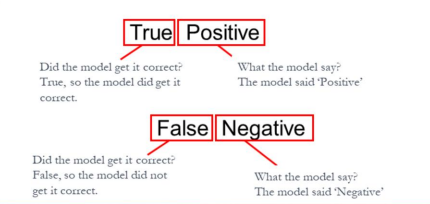
**Confusion Matrix**

* We can represent the four outcomes of our prediction with the actual by means of a confusion matrix



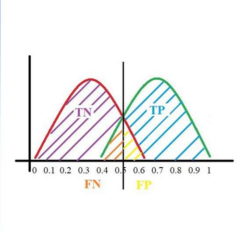
**Error Types**

* Two parts to each: whether the model got it correct or not, and what the model guessed



**Error Trade-off**

* Which error type is worse in a classification model?
* Depends on the problem/business case. Recall the previous examples: false positive results in loss for a bank while false negative leads to a patient to die
* Most classification machine learning models, including the logistic regression, provide probabilities as their output
* A threshold is then used to decide the positive and negative cases
* By varying the threshold, one can trade false positives for false negatives and vice versa
* Increasing the threshold value means that we will be more strict in classifying an observation as positive
* This results in lowering the false positives and true positives and increasing true and false negatives
* On the other hand, lowering the threshold results in decreased false and true negatives and increased false and true positives



**Recall Sensitivity**

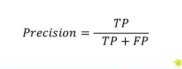
* We observed that accuracy is not a good measure of performance, especially for imbalanced data categories
* Intuitively, we should maximize the ability of a model to find all the relevant cases within a dataset. This is called “Recall” or “Sensitivity”
* Formally, recall is the number of true positives divided by the number of true positives plus the number of false negatives



* Sensitivity – the proportion of positives correctly classified
  + Denominator is the number of actual positive items in your sample
* Recall seems like a performance metric that is much better than the accuracy
* Can we rely merely on recall to express the performance a classifier?

**Precision**

* Precision is defined as the number of true positives divided by the number of true positives plus the number of false negatives
* While recall expresses the ability to find all positive instances in a dataset, precision expresses the proportion of the data points our model says was positive actually were positive
* Balancing between Precision and Recall is the same as balancing between False positives and False Negatives

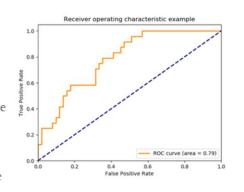


**True Positive Rate (TPR) and True Negative Rate (TNR)**

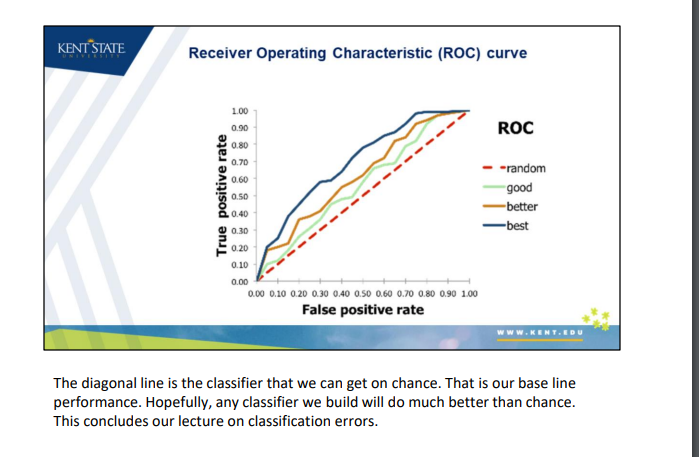
* **Recall**, also known as **Sensitivity** or **True Positive Rate (TPR),** is the proportion of positive cases which were correctly identified by the model
* In our example, it is the proportion of people that tested positive and are positive of all the people that actually are positive (i.e. what proportion of positive cases were correctly identified by the model)
* **Specificity** or **True Negative Rate (TNR)** is the proportion of negative cases which were correctly identified by the model. False Positive Rate **(FPR) = 1- TNR**
* As with sensitivity, it can be looked at as the probability that the test result is negative given that the patient is not sick

**Receiver Operating Characteristic (ROC) curve**

* ROC curves typically feature true positive rate on the Y axis, and false positive rate on the X axis
* This means that the top left corner of the plot is the “ideal” point – a false positive rate of zero, and a true positive rate of one
* This is not very realistic, but it does mean that a larger area under the curve (AUC) is usually better



* **ROC** Curve – represents our performance measures
  + Y axis = **Sensitivity** (proportion of positives classified correctly as positive)
  + X axis = **Specificity** (FPR)



**k-Nearest Neighbor (kNN) Classification**

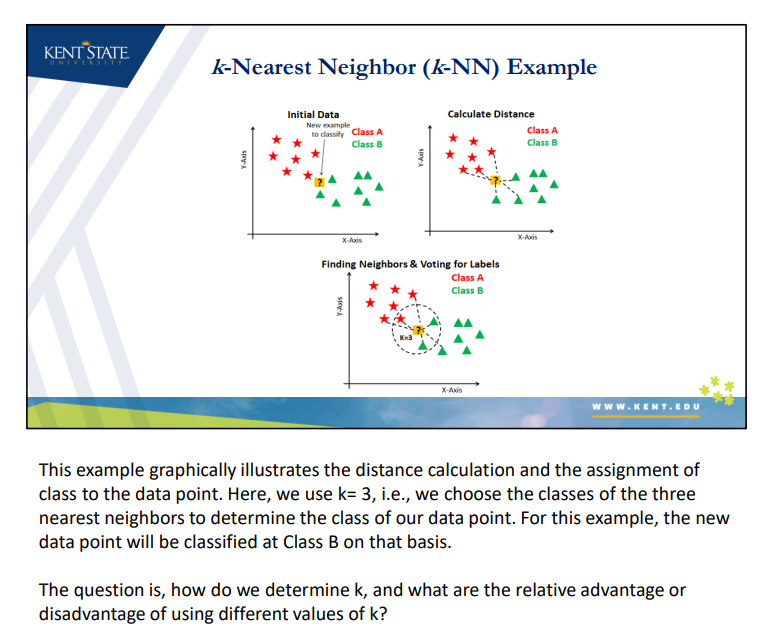
* k-Nearest Neighbor (k-NN) algorithm is a supervised classification method
* k-NN seeks to classify objects based on closest *k* training examples in the training set
* The algorithm is considered as one of the simplest machine learning methods
* Non-parametric classifier i.e. it does not require the estimation of parameters.
* Relies on finding “similar” records in the training data
* These neighbors are then used to derive a classification for the new record by voting
* Can be used for both categorical variables (classification), and continuous variables (prediction)

***k*-NN algorithm as 3 functional components:**

* A training dataset which includes a list of features (i.e. input variables) and their corresponding class labels (i.e. outputs)
* A distance metric which is used to quantify the distance (i.e. dissimilarity of records)
* The value of *k* the number of nearest neighbors from which classifications are made
* Training data – contains the list of values for the predictor (input) variables x1, x2, etc., and the class membership Y (output variable)
* Most common measure for distance calculation is the Euclidean distance
  + Square root of the sum of squared differences between the value of the variable in the training set, and the value of the variable in the new data point which needs to be classified
  + Consideration in selecting distance measures Is the realization that these measures must be calculated for each value in the training set across all input features
    - Euclidean distance has a low computational cost
* Then need a rule to assign a class to the record to be classified, based on neighbor’s classes.
  + Simplest case is k = 1, looking for the record that is closest (nearest neighbor) and classify the new record as belonging to the same class as its closest neighbor

**Making Predictions**

* *k*-NN is considered as a lazy learning algorithm (as opposed to an eager algorithm) where all computations are delated until a prediction needs to be made
  + We compute its distances from the entire set of training records only at the time of prediction
  + Prohibits us from using this algorithm for real-time prediction of a large number of records
* In fact, the algorithm doesn’t learn but simply memorize the data
* When a classification is needed for a given record, the algorithm:
  + 1. Computes the distance of the record to other training records
  + 2. Then identifies *k* nearest neighbors in the training set
  + 3. Finally, makes classification based on the class of the *k* identified neighbors
* K > 1 neighbors:
  + Find the nearest *k* neighbors to the record to be classified
  + Use a majority decision rule to classify the record, where the record is classified as a member of the majority class of the *k* neighbors
  + Ex. majority of the neighbors belong to class A, then the observation is Class A



* Advantage of choosing *k* > 1 is that the higher values of *k* provide smoothing that reduces the risk of overfitting due to noise in the training data
  + If *k* is too low, we may be fitting to the noise in the data
  + If *k* is too high, we may miss out on ability to capture the local structure of the data
* How is *k* chosen?
  + We choose the *k* with the best classification performance on the validation set

***k*-NN: Prediction Using Majority Voting**

* In this approach, all votes are equal. For each class, we count how many of the *k* neighbors have that class. We return the class with the majority of the votes
* In case of a tie vote, different strategies can be followed:
  + A) Randomly choose one of the classes (least accurate)
  + B) Choose the class of the record that is closest to the observation (as if *k* was 1)
  + Decrease *k* by 1 and decide the class
* “Majority’ definition is linked to the notion of a cutoff value applied to the class membership probabilities.
  + Simple majority rule is equivalent to setting the cutoff value to 0.5

***k-*NN: Prediction Using Inverse Distance-Weighted Voting**

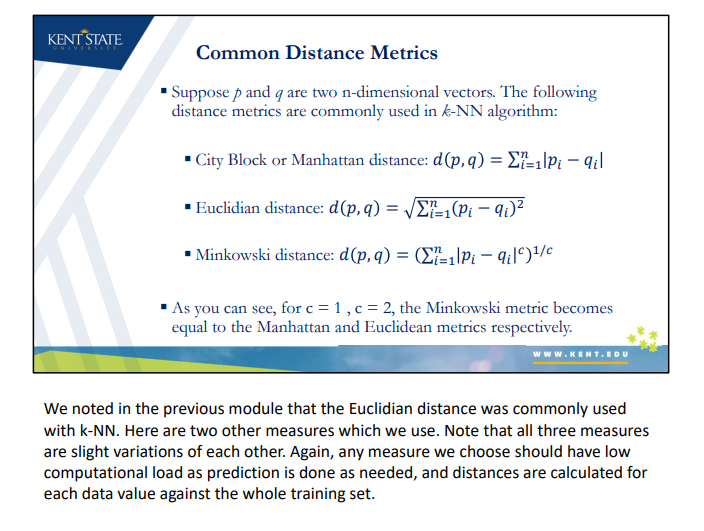
* In this approach, closer neighbors get higher votes
* While there are better-motivated methods, the simplest version is to take a neighbor’s vote to be the inverse of its distance to the given observation
* Then, we sum the votes and return the class with the highest vote

**Distance Metrics and Data Normalization**

* Distance is a proxy for similarity
* Important to normalize distance data used for distance so that measurements can be compared
  + Ex. cost of living indices for countries

***k-*Nearest Neighbor (*k*-NN) Classification**

* A distance metric is used to quantify the distance (i.e. dissimilarity) between different observations
* Calculating the distance between the observation to be predicted and the training samples is the first step of the *k*-NN algorithm
* We now introduce some of the common distance metrics that used by *k*-NN algorithms.

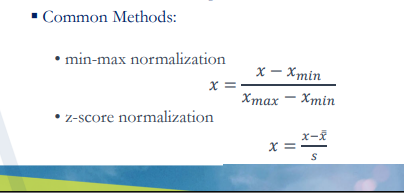


**Sensitivity to Scale I**

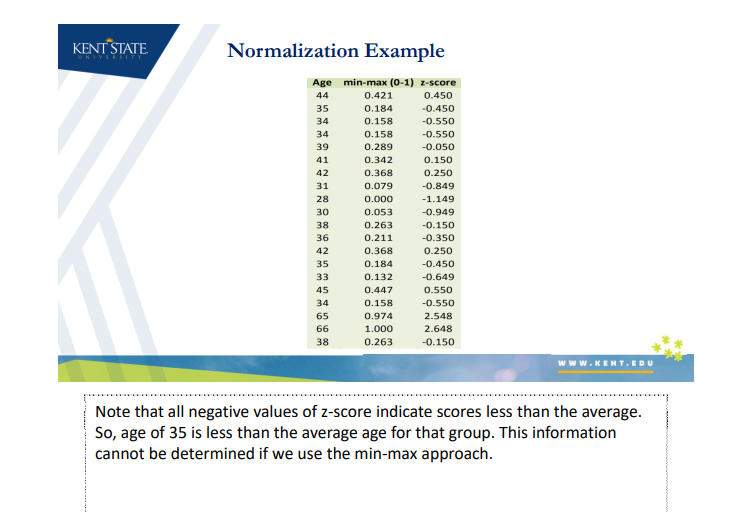
* Calculation of distance is very sensitive to the scale of individual attributes
* Suppose we want to calculate the distance between the records of patients using only two attributes (variables), weight and height:
  + Patient 1: weight = 66kg height=1.90m
  + Patient 2: weight = 63kg height=1.45m
  + Patient 3: weight = 65kg height=1.50m
* Which of the patients 1 and 2 is the closest match (i.e. nearest neighbor) to patient 3? Assume City Block is used as the distance metric
* Distance (patient 3, patient 1) = |65-66| + |1.50-1.90| = 1.40
* Distance (patient 3, patient 2) = |65-63| + |1.50-1.45| = 2.05
  + So patient 1 is the closer match to patient 3
* As we can see, the effect of eight differences was completely masked by weight, due to different scales of the variables
* What if height was expressed in cm instead of m?
  + Distance (patient 3, patient 1) = |65-66| + |150-190| = 41
  + Distance (patient 3, patient 2) = |65-63| + |150-145| = 7
    - Completely a different results?
* Result is affected by units of measurement
  + One approach to solve this is the concept of standardization/normalization
    - Convert all values of a variable into a unitless range of numbers

**Normalization**

* To avoid dependency of distance calculations to scale of data, in practice we usually normalize the data before calculating distances
* Normalization helps to prevent attributes with large ranges out-weighting attributes with small ranges

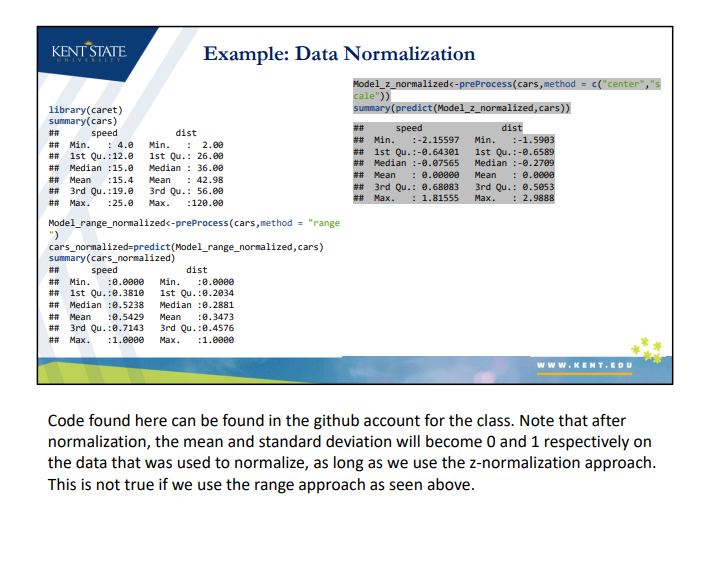


* Z-score normalization is more robust due to standard deviation in denominator
  + Once data has been standardized, the data will have a mean of 0 and a variance of 1
    - Allows for direct comparison



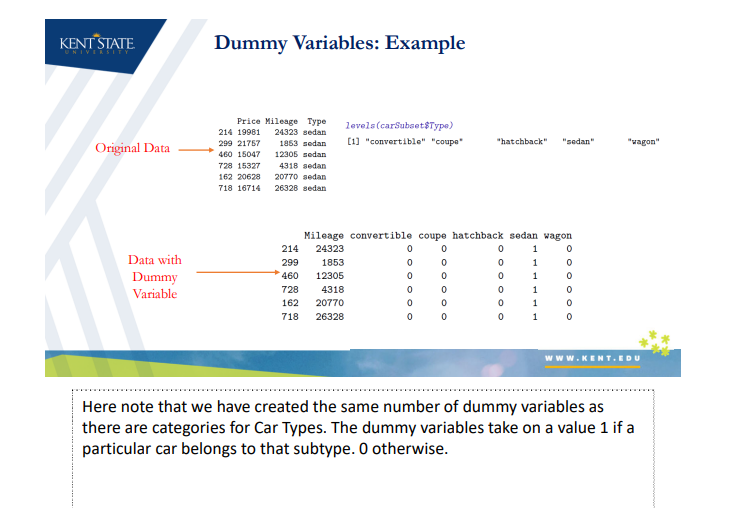
**Data Transformation: Normalization**

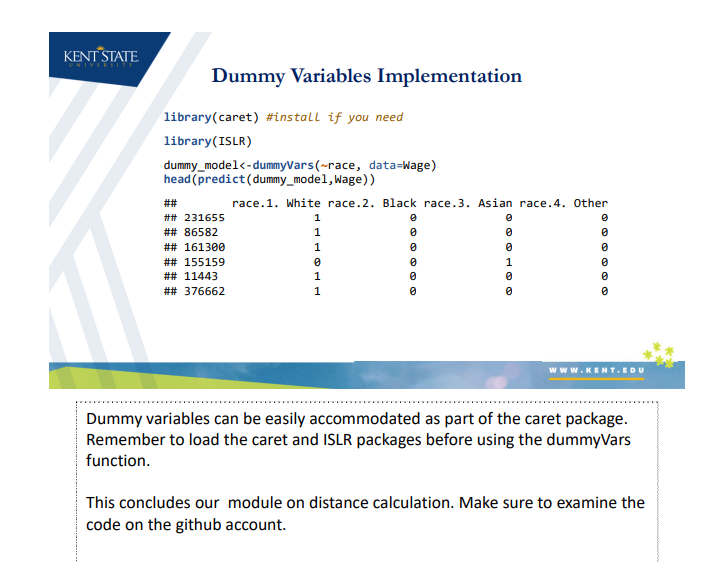
* The preProcess() function that is in the ‘caret’ package is a powerful method which has implemented a number of data processing and transformation methods
* The function implements min-max normalization using “range” as the method or z-score scaling when using “center” and “scale” as input method parameters
* The function, however, creates a model which needs to be applied to the data using the predict() function



**Distance Metrics for Categorical Variables**

* Numerical operations such as those involved in calculation of distance metrics are not possible for categorical variables
* Categorical variables, however, can be converted to dummy variables to perform numerical operations
* Dummy variables are artificially defined variables designed to convert a categorical variables with multiple levels into individual binary variables
* The process also is referred to as “One hot encoding”





***k*-NN Model Tuning and Optimization**

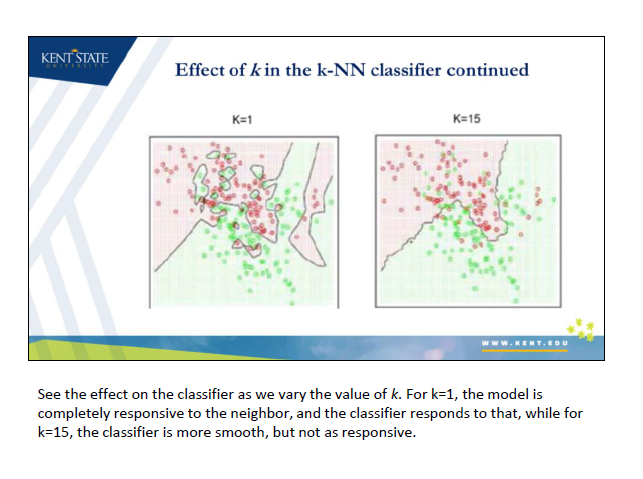
* Model Tuning – determining the values of the model parameters for optimal results
  + *K*-NN – the parameter is *k*, the number of nearest neighbors to consider

**Parameter versus Hyperparameter**

* A model parameter is a configuration variable that is internal to the model and whose value can be estimated from data
* Model parameters:
  + Are required by the model when making predictions
  + Are estimated or learned from the data
  + Are often not set manually by the practitioner
  + Are often not saved as part of the learned model
* Example: Coefficients of variables in linear regression models
* Difference between parameters for a model that are estimated directly from data (coefficients of linear regression), and those determined outside the data
  + Linear regression – parameters are coefficients of Y-intercept and slope for variables
    - Estimated directly from data
  + *k*-NN – number of neighbors is parameter
    - not estimated from the data – rather, chosen from the modeler
* Model-tuning is based on parameters that are not estimated from the data like *k*
  + Also called **hyperparameters**
* A model hyperparameter, on the other hand, is a configuration that is external to the model and whose value cannot be estimated from data
* Choosing right hyperparameters can significantly improve the performance of a model
* Example of hyperparameter: the number neighbours, *k*, in a *k*-NN model
* It is a common practice to select a model’s hyperparameters by trying different values and evaluating the performance (i.e. grid search) on validation set (e.g. cross validation)
  + Allows us to determine what value of the parameter allows the best prediction

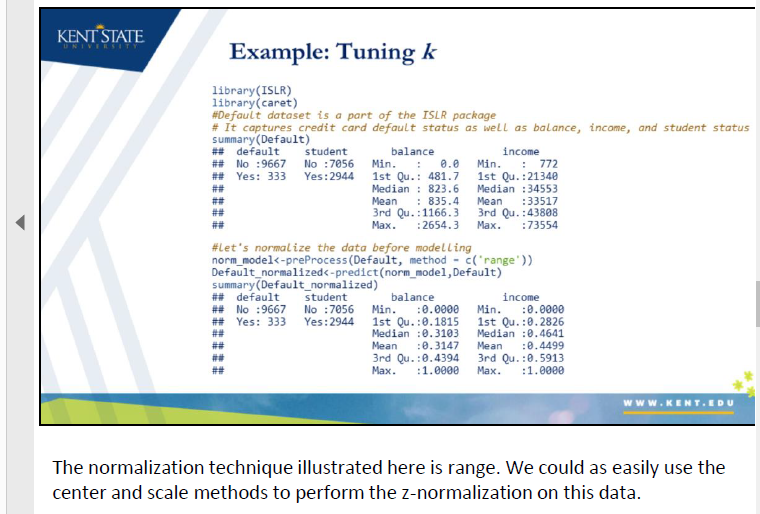
**Effect of *k* in the *k*-NN classifier**

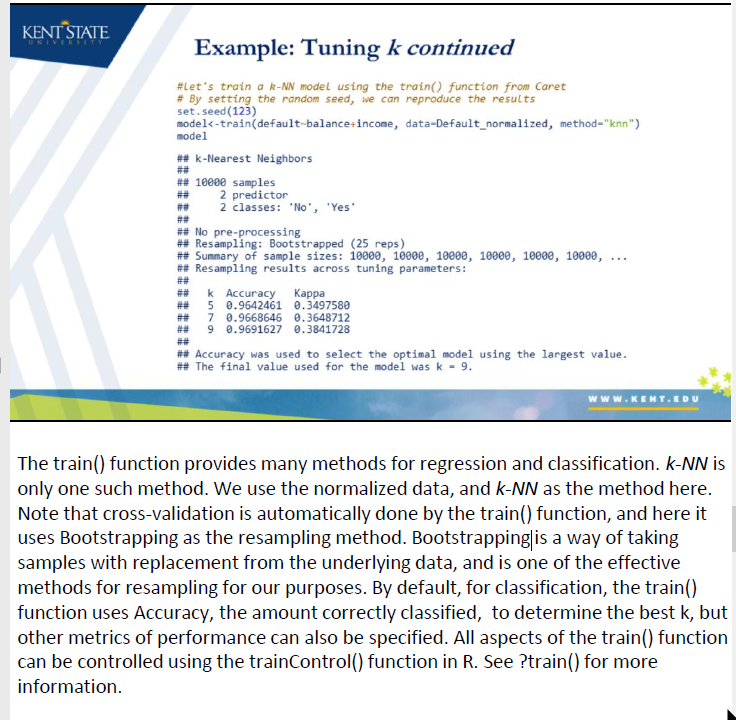
* If *k* is chosen to be too small, the model may overfit and the noise and outliers may significantly affect the predictions
* On the other hand, if *k* is chosen to be too large, some of the neighbors may be irrelevant to the prediction. This normally results in underfitting
* The optimal value of *k* depends on the classification task and the data and should be determined by the modeler in advance
* We want to balance between overfitting to the predictor information and ignoring this information completely
  + The more complex and irregular the structure of the data, the lower the optimum value of *k*
  + Typically, the values of *k* fall in the range 1 to 20



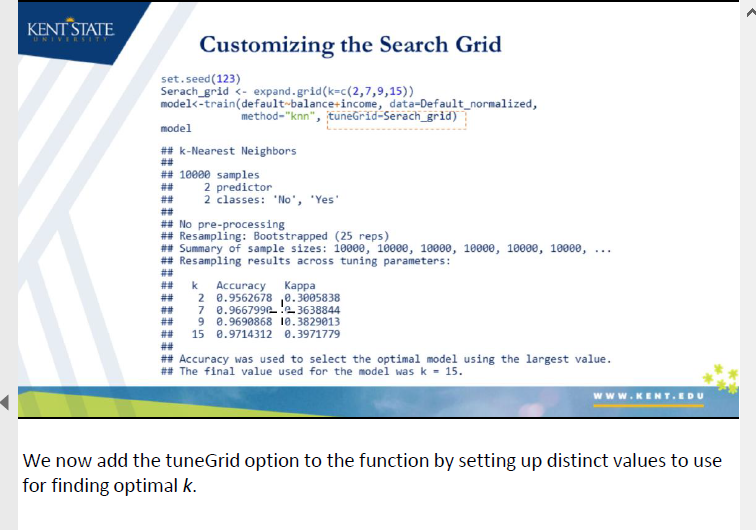
**Caret for Hyperparameter Optimization**

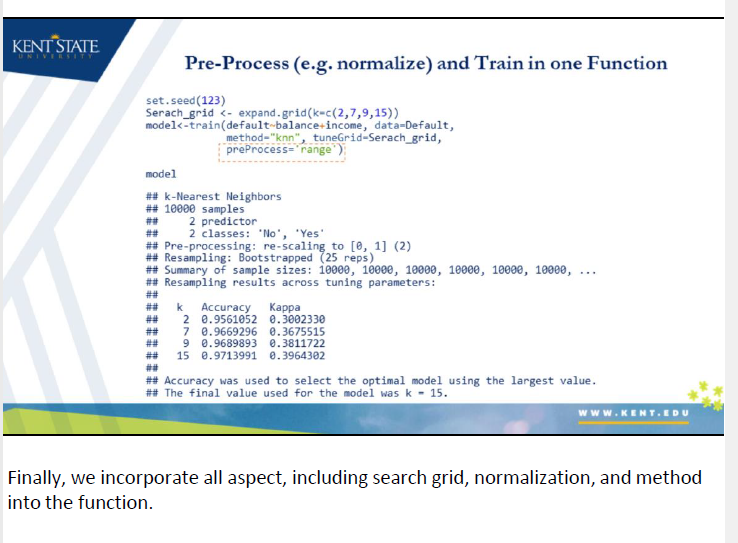
* Caret package provides a wrapper for a large number of machine learning models (more than 200 models as of now)
* The train() function will automatically perform a grid search over a pre-defined set of hyperparameter values. It then selects the best hyperparameter values using cross validations and train a model
* The hyperparameter for the *k*-NN models is *k*
* The output shows the performance of the model for different *k* values. Accuracy is used as the default metric to choose the best model





* Boostrapping – a way of taking samples with replacement from the underlying data



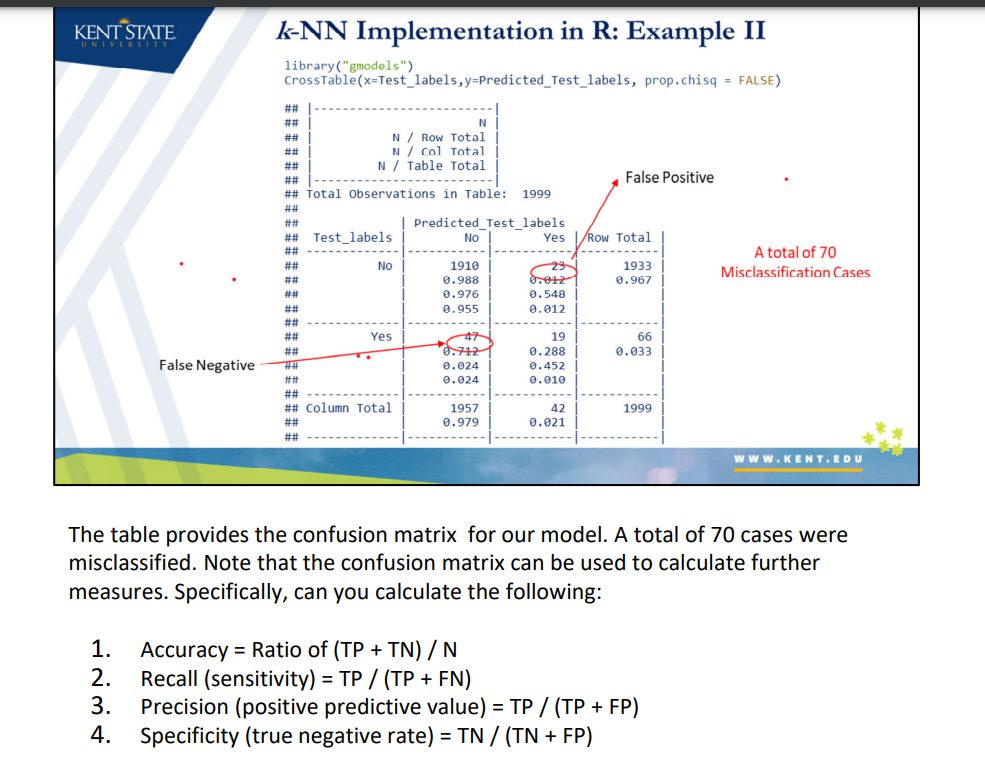


***k*-NN Implementation in R**

* *k*-NN classification algorithm is implemented in the “Class” package
* The knn() function identifies the *k*-nearest neighbors using Euclidean distance where *k* is a user-specified number
* The function knn() takes the training and test set at the same time and returns the predicted class values for the test dataset
* Numeric attributes should be normalized before passing to the function

**Confusion Matrix**

* As we discussed, confusion matrix is a two by two table that contains four outcomes produced by a binary classifier
* A Confusion Matrix can be used to create the many different measures of a model’s fitness including: accuracy, precision, and recall
* CrossTable() function that is part of the package ‘gmodels’ can be used to compile an informative confusion matrix
* Provides us with details needed to produce several outcomes, including:
  + How many observations were correctly classified (accuracy)
  + True positive rate (recall or sensitivity)
  + Positive predictive value (precision)
  + Specificity (false negative)
* Algorithms with high recall can predict with high accuracy the positive or relevant values
  + High precision indicates what proportion of positives are truly identified as positives



**Data Mining for Business Analytics – K-Nearest Neighbors**

* Identify *k* records in the training dataset that are similar to a new record we wish to classify
  + Assign new record to the predominant class among the *k* records
* K-NN Algorithm does not make assumptions about the form of the relationship between Y (class membership) and the predictors (X1, X2, etc.)
  + Nonparametric – does not involve estimation of parameters in an assumed function form
  + Instead draws similarities between the predictor values of the records in the dataset
* Rule to assign a class to the new record based on the classes of its neighbors
  + K = 1 - look for the record that is closest and use that class
* 1-nearest neighbor can be extended to *k* > 1 neighbors as follows:
  + Find the nearest *k* neighbors to the record to be classified
  + Use a majority decision rule to classify the record, where the record is classified as a member of the majority class of the *k* neighbors
* Higher values of *k* provide smoothing, reduces the risk of overfitting due to noise in the training data
* Generally use odd numbers to avoid ties
* Once *k* is chosen, we re-run the algorithm with both training and testing sets, in order to generate classifications of new records
* All categorical variables must be converted to dummy variables before k-NN can take place