MACHINE LEARNING WITH R

Class 2 Intensive

Machine Learning Algorithms

Machine learning algorithms are largely used to:

- Predict
- Classify
- Cluster

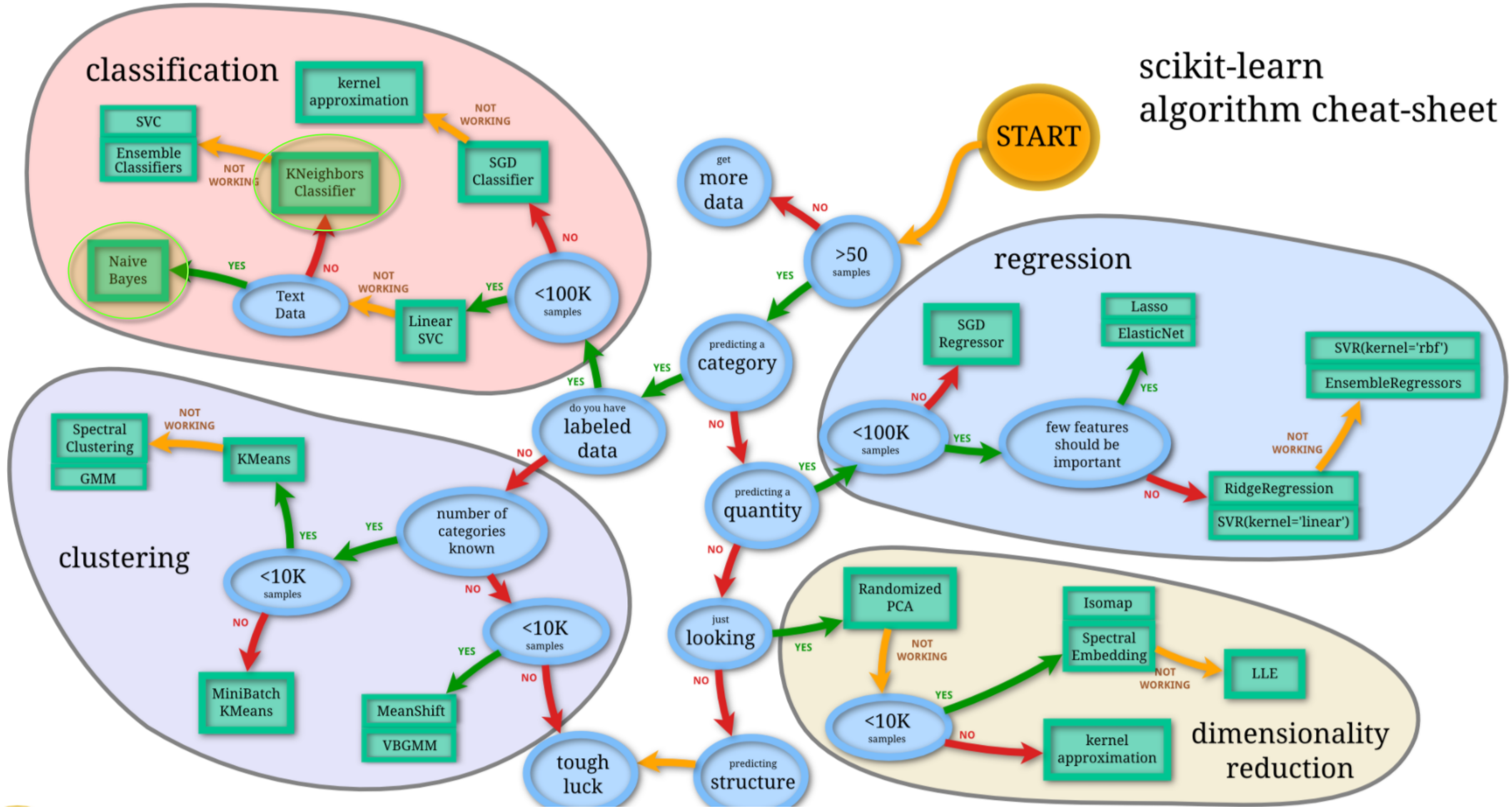
Supervised Learning

- Predicting target feature from other features using available data by training the model
- Predictive models
- Classification: Which category an input belongs to used when dealing with nominal data with levels
  - kNN
  - naïve Bayes
  - Decision Trees
  - Classification Rule Learners
- Numeric Predictions:
  - Linear Regression
  - Regression Trees
  - Model Trees
- Dual:
  - Neural Networks
  - Support Vector Machines

Unsupervised Learning

- Insights gained by analyzing and summarizing available data. There is no target feature, no feature is more important
- Descriptive models
- Pattern Discovery/Market Basket Analysis
  - Association Rules
- Clustering: Identifying groups
  - k-means Clustering

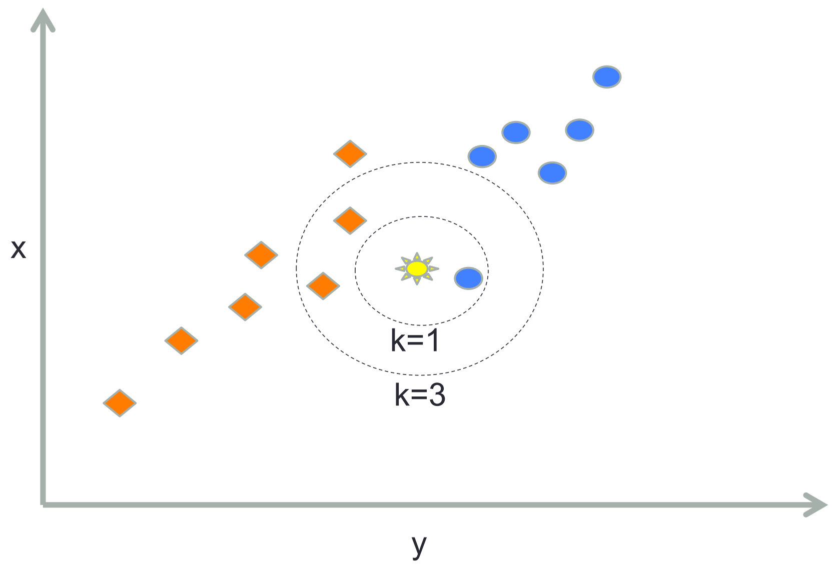
Algorithms



- http://scikit-learn.org/stable/tutorial/machine_learning_map/

k-Nearest Neighbors (kNN)

- K-nearest neighbor (kNN) is a nonparametric lazy learning method. From a nonparametric view, it does not make any assumptions about data distribution. In terms of lazy learning, it does not require an explicit learning phase for generalization.



kNN in R

To perform kNN in R, we need:

- A matrix containing the predictors associated with the training data
- A matrix containing the predictors associated with the data for which we wish to make predictions.
- A vector containing the class labels for the training observations.
- A value for **K** , the number of nearest neighbors to be used by the classifier.

kNN: Decide on distance metrics

- Euclidean Distance
  - Path a bird would take, real-value vectors
- Cosine Similarity
  - Real-value vectors are indicated as
    - -1: exact opposites
    - 0: independent
    - 1: exactly the same
- Jaccard Distance or Similarity
  - Distance between set of objects and says how similar the sets are
  - Friends, shopping list
- Mahalanobis Distance
  - Similar to Euclidean but takes into account correlation and is scale invariant.
  - real-value vectors
- Hamming Distance
  - Distance between strings, words or DNA
  - Distance between shoe and hose is 3, olive and ocean is 4
- Manhattan Distance
  - Path a pedestrian would take, real-value vectors

kNN: Split into training and test dataset

- n.point<-nrow(iris)
- sampling.rate<-0.7

- training <- sample(1: n.point, sampling.rate * n.point, replace = FALSE)

- testing <- setdiff(1: n.point, training)

- iris_train<-subset(iris[training,])
- iris_test<-subset(iris[testing,])

kNN: Class labels

- spc_test<-iris_test[,5]

- spc_train<-iris_train[,5]

kNN: Clean data

- iris_test<-iris_test[,-5]

- iris_train<-iris_train[,-5]

kNN: Scaling Data

- Because the KNN classifier predicts the class of a given test observation by identifying the observations that are nearest to it, the scale of the variables matters. Any variables that are on a large scale will have a much larger effect on the distance between the observations, and hence on the KNN classifier, than variables that are on a small scale.
- A good way to handle this problem is to standardize the data so that all standardize variables are given a mean of zero and a standard deviation of one. Then all variables will be on a comparable scale. The scale() function does just this.

kNN: Standardize data

- min-max normalization

$$X_{new} = \frac{X - \min(X)}{\max(X) - \min(X)}$$

- z-score standardization

$$\mathrm{X}_{new} = \frac{\mathrm{X} - \mu}{\sigma} = \frac{\mathrm{X} - \mathrm{Mean}(\mathrm{X})}{\mathrm{StdDev}(\mathrm{X})}$$

kNN: Standardize data

- iris_test = as.data.frame(scale(iris_test))
- iris_train = as.data.frame(scale(iris_train))

kNN in R

Using knn() function from library(class)

- set.seed(101)
- predict_species<-knn(iris_train, iris_test, spc_train, k=1)

kNN: Pick an evaluation metric

- Misclassification rate
- misclass.error<-mean(spc_test != predict_species)
- print(misclass.error)

- library(gmodels)
- CrossTable(x=spc_test, y = predict_species, prop.chisq = FALSE)

kNN: Value of k and data evaluation

- predict_species<-NULL
- error.rate<-NULL
- for (i in 1:10){
-   set.seed (101)
-   predict_species<-knn(iris_train, iris_test, spc_train, k=i)
-   error.rate[i]<-mean(spc_test !=predict_species)
- }
- print(error.rate)
- k<-1:10

- error.df<-data.frame(error.rate, k)
- error.df
- plot(k, error.rate, type = "b", col="red")

kNN: Optimize k

Pick the value that gives the best optimization from the previous step or get a good starting point using square room of number of variables:

```
predict_species<-knn(iris_train, iris_test, spc_train,
k=round(sqrt(ncol(iris))))
```

kNN: Advantages

- Simple and effective
- The cost of the learning process is zero
- It is nonparametric, which means that you do not have to make the assumption of data distribution
- Fast training phase
- You can classify any data whenever you can find similarity measures of given instances

kNN: Disadvantages

- Does not produce a model, which limits the ability to find novel insights in relationships among features and it is hard to interpret the classified result.
- It is an expensive computation for a large dataset and requires a large amount of memory.
- Slow classification phase
- The performance relies on the number of dimensions. Therefore, for a high dimension problem, you should reduce the dimension first to increase the process performance
- Nominal features and missing data require additional processing