To illustrate three classification methods (Boosting, SVM and Random Forest), I simulated a dataset as an example. Please refer to the first a few lines of “classification.r”. The simulated training set has \( n = 100 \) observation and \( p = 500 \) covariates. Only the first five variables are relevant to classification, and the remaining 495 variables are just noises. There is another big test dataset to (honestly) test the performance of the methods.

**Boosting**

We need to install the R package “gbm” to run boosting. In “gbm”, the weak learner is classification (or regression) tree. There are two most important tuning parameters in boosting: number of trees and the depth of each tree. Usually, one specifies a series of values for each parameter, then use the 10-fold cross-validation to evaluate the performance of the model with each pair of tuning parameters.

The depth of tree means how many layers a tree has. For example, Figure 1 shows a tree with depth= 1 and Figure 2 shows a tree with depth= 3.

The depth determines the level of interactions between variables in the final model. When the depth of tree is one, there is only one variable is considered, so there will be its main effect only in the final model. When the depth is more than one, say three, several variables are considered. If these variables are different (illustrated in Figure 1), then the interaction between these variables are considered in the final model. For example, in Figure 2, the tree considers the V3-V4 two-way interaction, V3-V2 two-way interaction, V3-V4-V5 three-way interaction and V3-V2-V5 three-way interaction.

In “gbm”, the depth of all trees are set to be the same. Then if one choose depth= 1, the final boosting model considers main effects only; if one choose depth= 2, the final boosting model considers two-way interaction; if one choose depth= 3, the final boosting model considers three-way interaction, and so on so forth.
Next, we go through the R codes for running boosting.

```
fit1 = gbm(y.tr∼., n.trees=10000, distribution='adaboost', interaction.depth = 1, cv.folds=10, data=data.tr)
```

This line of code tells R to fit a boosting model. “y.tr∼.” means including all covariates in the model. The “n.trees” specifies the number of trees in boosting. The “distribution” specifies the type of outcome. If the outcome is binary (for classification problem), we can choose ‘adaboost’ or ‘bernoulli’. If the outcome is continuous, we can choose ‘gaussian’, and for count data, we can choose ‘poisson’. The “interaction.depth=1” means to fit a main effect only model. The “cv.folds=10” means to conduct 10-fold cross validation.

After fitting the model, we can use “fit1$cv.error” to output the cross validation error (not necessary to be misclassification error). We can choose the best tuning parameters with the smallest validation error.

Please refer to “classification.r” for making prediction on future data and examples of
fitting model with two-way interactions.

**Support Vector Machine**

We need to install the R package “e1071” to run SVM. There are two important things to tune in SVM. One is the selection of kernel, the other is the cost parameter (trade-off between training-data-fitting and future-prediction).

The most popular kernels are linear kernel \( K(x, y) = \|x - y\|^2 \) and radial (gaussian) kernel \( K(x, y) = \exp\{\|x - y\|^2 / \sigma^2\} \). The following code finds the best cost parameter when linear kernel is used.

```r
  tune1 = tune(svm, train.x=as.matrix(x.tr), train.y=as.factor(y.tr), kernel='linear', range=list(cost=2^seq(-5,5,length=100)), control=tune.control(sampling='cross', cross=10))
```

Note that the candidate cost parameter values are uniformly distributed on log-scale. This is usually preferred. If we take the values uniformly distributed on the original scale,
then the difference between 10000 and 10000.1 is the same as the difference between 0.1 and 0.2, but obviously the second pair is much more different than the first pair. “sampling=‘cross’, cross=10” means to conduct 10-fold cross-validation.

The object “tune1” gives the best cost parameter value. Then we can fit the “optimal” SVM using the following code:

```r
fit1.svm = svm(x.tr, as.factor(y.tr), kernel='linear', cost=0.03125)
```

When the radial kernel is used, we have two tuning parameters: \( \sigma \) in the radial kernel and cost parameter:

```r
tune2 = tune(svm, train.x=as.matrix(x.tr), train.y=as.factor(y.tr), kernel='radial', range=list(cost=2^seq(-5:5), gamma=2^(-5:5)), control=tune.control(sampling='cross', cross=10))
```

Note that, when we have two tuning parameters, the computation cost is much higher than that of single tuning parameter.

Please refer to “classification.r” for making predictions on new data.

You can see that, the performance of SVM is not very good. A possible reason for this is that SVM conducts the analysis based on all variables, which include many irrelevant ones.

**Random Forest**

First, I briefly describe the idea of Random Forest. There are two most important ideas in Random Forest. Consider a setting that we have \( B \) (say \( B=1,000 \)) i.i.d. dataset. Then we fit a same model on dataset and take the average of these models. Intuitively, the average model is better than the model fitted on the single dataset because the variance of estimator is reduced by taking average. Now, if we only have one dataset, we can use bootstrap to mimic the case we have many independent datasets. To be specific, the Random Forest method draws \( B \) bootstrap samples, fit a tree on each bootstrap sample, and then take the average. This is the first most important idea in Random Forest, and
is also the reason that it is called forest (many trees). Now we compare the case with $B$ bootstrap samples to the case with $B$ i.i.d. datasets. A big difference is that $B$ i.i.d. dataset are independent, but the $B$ bootstrap samples, and the $B$ trees fitted on these bootstrap samples, are not. The higher the correlation between trees is, the less effective the average will be. The second most important idea in Random Forest is to reduce the correlation between trees fitting on different bootstrap samples. What it does is, when considering a split on a tree in a bootstrap sample, only a randomly selected subset of variables is considered. As a consequence of this, in different bootstrap samples, different variables are used to build up the tree, which can reduce the correlation between trees. This is also why it is called random forest.

We need the R package “randomForest” to run Random Forest. The most important tuning parameter is the number of randomly selected variables for consideration in each split of the tree:

$$tune.rf = \text{tuneRF}(x=x.tr, y=\text{as.factor}(y.tr), \text{ntree}=1000, \text{mtryStart}=10, \text{stepFactor}=1, \text{nodesize}=10)$$

The “tree” specifies the number of trees in the forest. The “mtryStart” specifies the number of randomly selected variables for consideration in each split. The “stepFactor=1” means that the number of randomly selected variables for consideration is the same for all splits in one tree. The “nodesize=10” means each terminal node (the node at bottom, or leaf) of the tree should contain more than 10 observations.

The object tune.rf gives the cross-validation error and we can choose the best mtryStart to be the one with the smallest CV-error.

Once the optimal parameter is selected, we can fit the final Random Forest model:

$$\text{fit.rf} = \text{randomForest}(x=x.tr, y=\text{as.factor}(y.tr), \text{ntree}=1000, \text{mtry}=10, \text{nodesize}=10, \text{importance}=T)$$

Please refer to “classification.r” for making predictions on new data.