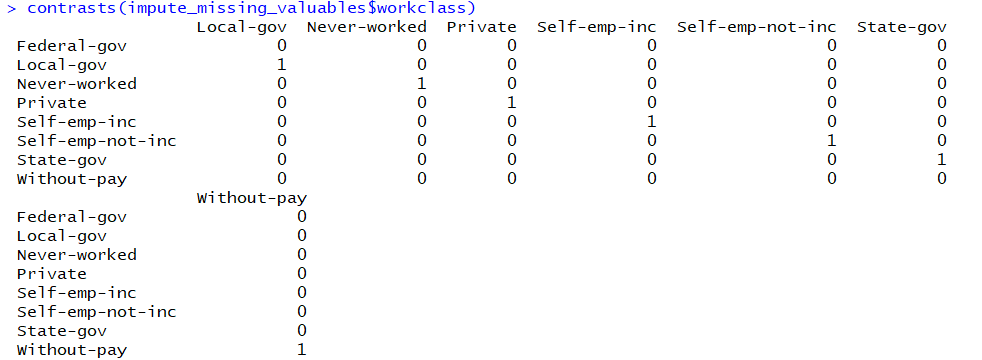
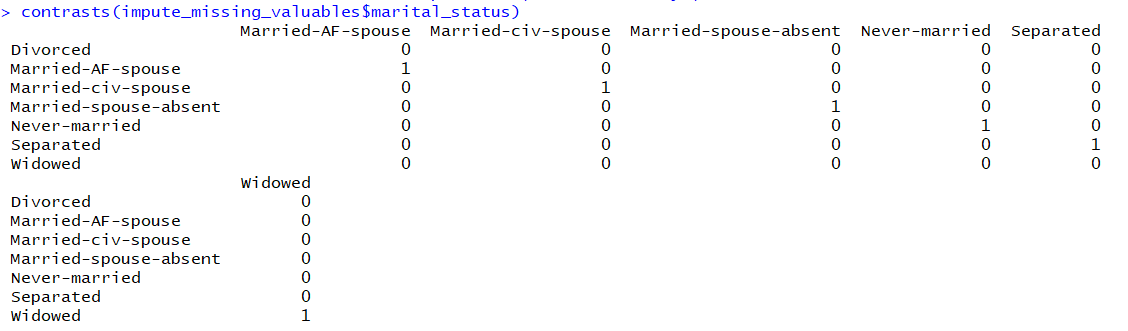
Part of the code and some ideas like dealing with over-fit problem are from the book “an introduction to statistical learning with applications in R”.

**Impute missing data**

1. I need to impute all the missing data for the training dataset and test dataset. So, I read in the training dataset (adult\_data) and have created set A to be the subset without missing value and set B to be the subset contains missing value. The same method applies to the test dataset (adult\_test).
2. Euclidean distance is used here. Since the value of fnlwgt is much larger than age value, each dimension is normalized before implementing knnImputation so that the Euclidean distance will not be dominated by one or two variables.
3. Create dummy variables. Education level is an ordinary variable. However, education level and education-num are considering the same thing, it is redundant to use both. And I decided to drop the education level column. For the other columns, although occupation can be ordinary, it is difficult to determine the order. Thus, I treated it as a categorical variable. For all the categorical variables, if they have n levels, n-1 dummy variables are created. For example:

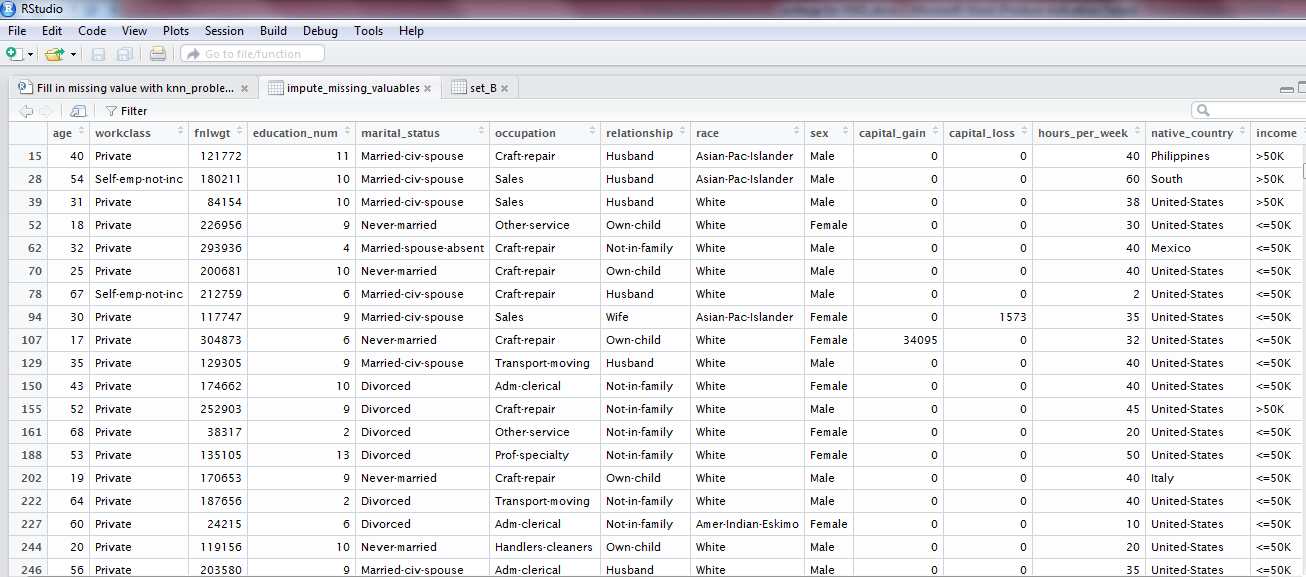




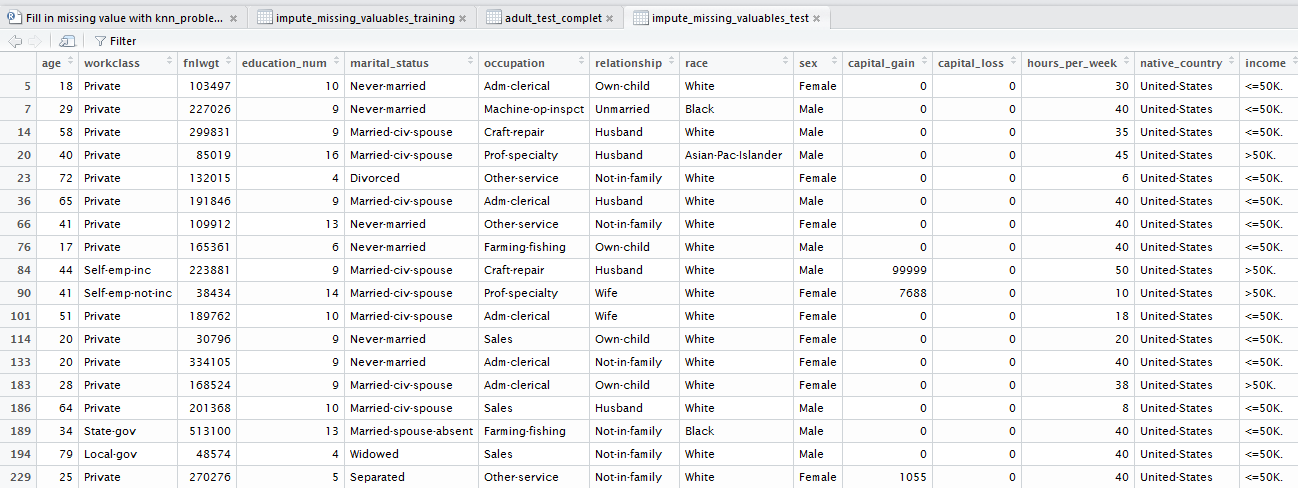
The same method applies to the variable “occupation”, "relationship", "race", "sex", "native\_country", and "income".

## The function I have used is called Fill in NA values with the values of the nearest neighbours (knnImputation {DMwR}). This function will automatically create dummy variables for the categorical variable and Euclidean distance is the default setting. I set the scale = T to normalize each dimension.

1. The first few rows of the adult.data that had missing information with missing information fully imputed are as follows:



Similarly, the first few rows of the adult.test that had missing information with missing information fully imputed are as follows:



**Predictive modeling \_ Decision Tree**

1. To implement decision tree in R, factor predictors must have at most 32 levels. So I created dummy variables for native\_country using the same method as described in problem 1. In the test data, none the observations’ native\_country is Holand\_Netherlands while there are some training observations come from Holand\_Netherlands. So I add one column “Is Holand\_Netherlands” to the test data and set its value to be 0 for all test observations.
2. How a decision variable was chosen is as follows: the split which maximizes the reduction in impurity is chosen, the data set split and the process repeated. Splitting continues until the terminal nodes are too small or too few to be split.
3. The classification error rate was used to guide the cross-validation and pruning process, this is indicated by setting FUN = prune.misclass.
4. cv.tree function then reports the number of terminal nodes of each tree considered (size) as well as the corresponding error rate. Tree size vs the corresponding error rate was plotted as Figure 1. From the figure, I see tree size 5 has the lowest corresponding error and is a relatively small tree size. I then prune the tree to 5-node tree using prune.misclass function. Prunning the tree is used to solve the overfitting problem.

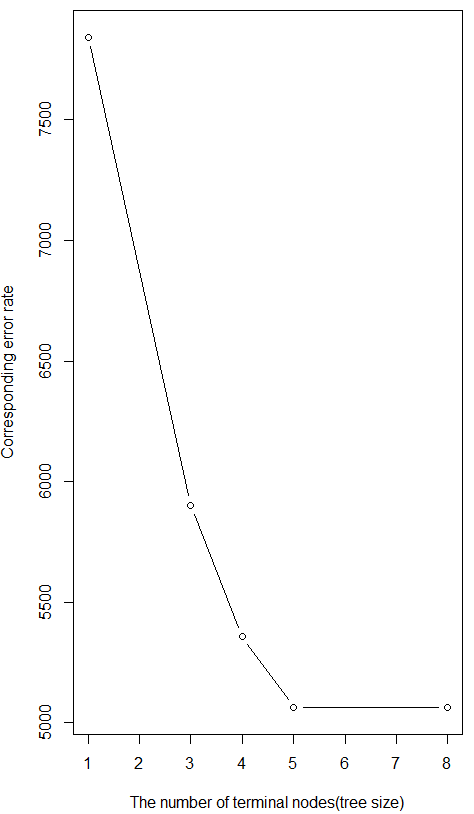
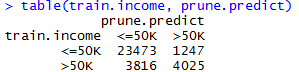


Figure 1: Tree size vs corresponding error rate

1. After the pruning, accuracy for the training data is (23473+ 4025)/32561=0.8445

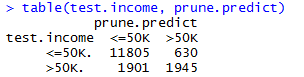
The following code tells us details about the prediction (prune.predict) vs the true value (train.income) for the training dataset.



1. For the test data, the accuracy is (11805+1945)/16281=0.8445

The following code tells us details about the prediction (prune.predict) vs the true value

(test.income) for the test dataset.



**Predictive modeling \_ Random Forest**

1. I constructed my bootstrapping datasets in terms of instances using the following methods, which is from the book “an introduction to statistical learning with applications in R”.

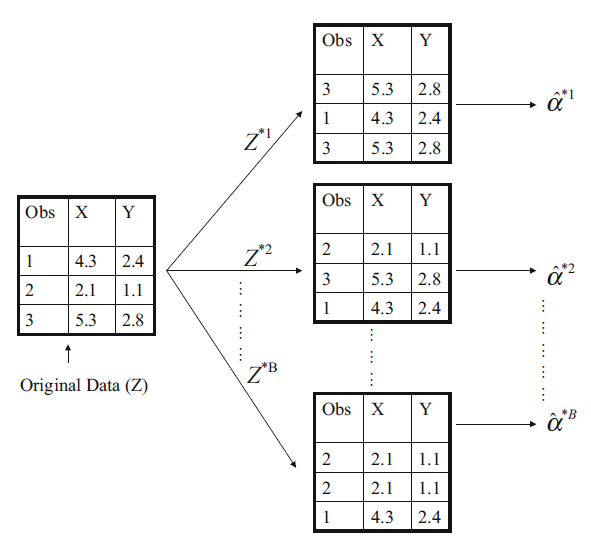
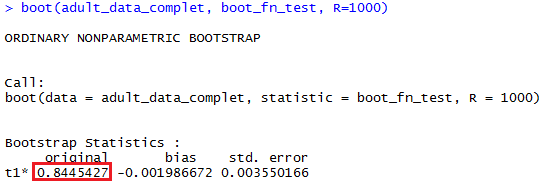


Figure 2. Schematic of Bootstrap

Each bootstrap data set contains 32561 observations, sampled with replacement from the imputed training data set (adult\_data\_complet). Boot function created 1000 bootstrap data set. Each bootstrap data set was used to build a decision tree. The Decision Tree method used in Problem 2 is used here. The tree was also pruned to be a 5 node tree. Using these trees, the income for the imputed test data can be calculated. Average accuracy for the test dataset was calculated by mean (test.income == prune.predict), where test.income is the true classifier and prune.predict is the predicted classifier. Average accuracy over all test cases using the above bootstrapped dataset is 0.8445427.



1. Random Forests were used to de-correlate the trees. Build 1000 decision trees on bootstrapped training sample, but when building these trees, each time a split in a tree is considered, a random sample of 7 predictors is chosen as split candidates from the full set of 53 predictors. I choose 7 because 7 is about the same as the square root of 53. I have 53 predictors because to implement decision tree in R, factor predictors must have at most 32 levels. So I created dummy variables for native\_country using the same method as described in problem 1. Thus I have 53 predictors.

Apply Random Forest method to predict the incomes for all of the test cases in the census dataset.

The average accuracy over all test cases is computed by

mean (trandomtree.predict == test.income),where test.incomeis the true classifier and randomtree.predict is the predicted classifier from Random Forest.

Average accuracy over all test cases using Random Forest is 0.8655488

1. The number of trees has changed for bootstrapping and Random Forest methods to check its influence on the average accuracy. Average accuracy over all test cases using the above methods is plotted as follows:



Figure 3. Bootstrap and random Forest results for imputed adult\_test dataset

Bootstrapping training dataset in terms of features can improve the average accuracy more than bootstrapping training dataset in terms of the instances. Random Forest has high accuracy than bootstrapping. Average accuracy doesn’t change much as the number of trees increases from 100 to 2000. Random Forest has a higher accuracy than decision tree in problem 2, with the average accuracy using Random Forest is about 0.866 and that using decision tree is 0.845.

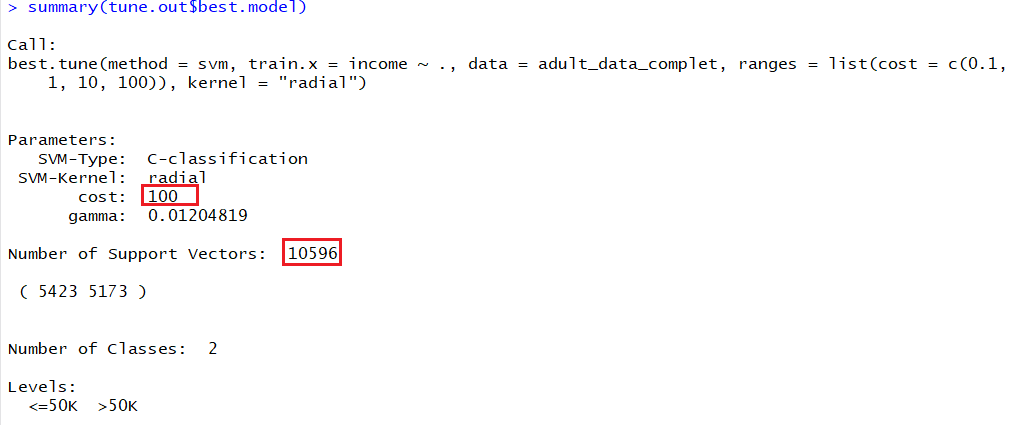
**Predictive modeling \_ SVM**

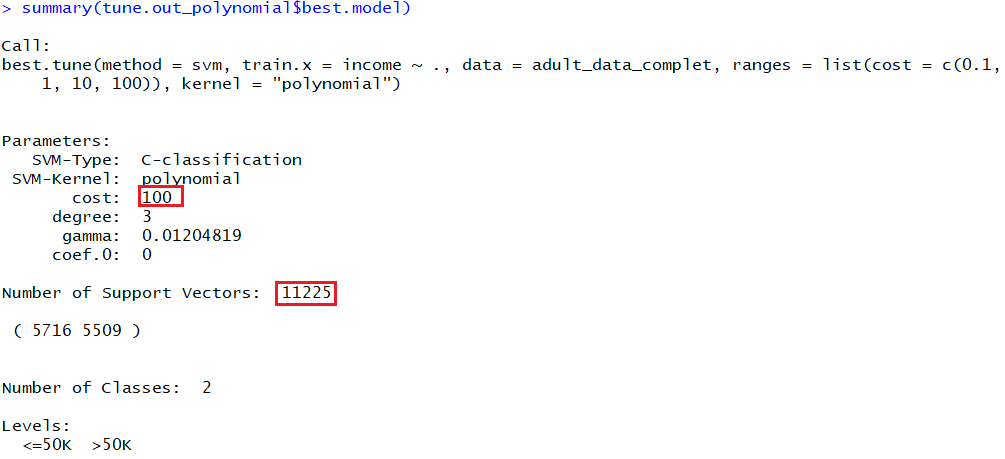
1. Imputed training and testing dataset from problem 1 are used here. native\_country feature is used instead of its dummy variables to reduce the dimensions. Since imputed test dataset’s level is different than that of the imputed training dataset, I first set them to be the same using

levels(adult\_test\_complet$native\_country) = levels(adult\_data\_complet$native\_country).

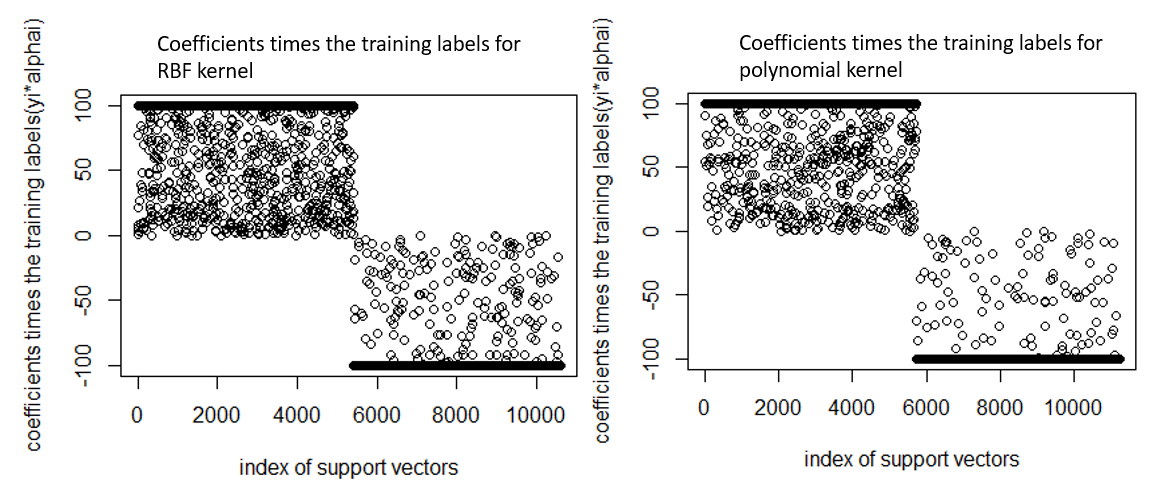
adult\_test\_complet is the imputed test dataset while adult\_data\_complet is the imputed training dataset.

1. Soft margin SVMs are used here. A cost argument allows us to specify the cost of a violation to the margin. When the cost argument is small, then the margins will be wide and many support vectors will be on the margin or will violate the margin. When the cost argument is large, then the margins will be narrow and there will be few support vectors on the margin or violating the margin. The e1071 library in R includes a built-in function, tune(), to perform cross validation and provide the best model with cost parameter tunned from a range of c(0.1,1,10,100).
2. tune.out$best.model stores the best model using the radial basis function (RBF) kernel, which has the lowest cross-validation error rate. tune.out\_polynomial$best.model stores the best model using the polynomial kernel. summary(tune.out$best.model) and summary(tune.out\_polynomial$best.model) show for both kernel, cost = 100 have the lowest cross-validation error rate. summary function also tells us the number of support vectors. For RBF kernel, the number of support vectors is 10596 while for polynomial kernel, that number is 11225.





1. The best models were then used by predict function to predict the incomes for all of the imputed test cases in the census dataset. Moreover, tune.out$best.model$coefs and tune.out\_polynomial$best.model$coefs can provide the corresponding coefficients times the training labels, which is  in the following function. 



tune.out$best.model$labels tells us that for the training data, <= 50 K is labeled as 1, and >50 K is labeled as 2.

The resulting alpha’s for RBF is as Figure 4. x\_axis is the index of the resulting support vectors in the data matrix. The training labels for index from 1 to 5423 are 1 and for index from 5424 to 10596 are 2. Thus, using the above results (coefficients times the training labels) and the following code, the resulting alpha is plotted:

for (i in 5424:10596) { tune.out$best.model$coefs[i] = tune.out$best.model$coefs[i]/2 }

plot(tune.out$best.model$coefs)

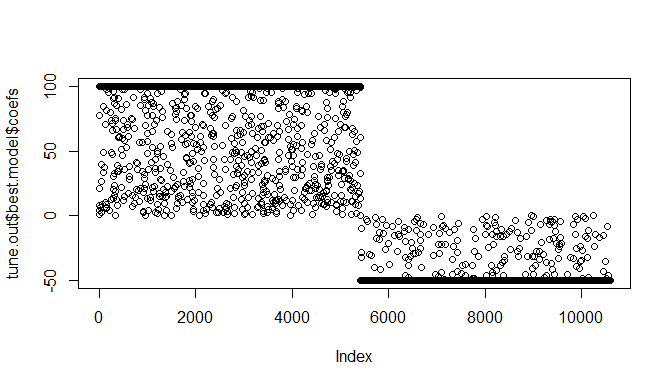


Figure 4. The resulting alpha’s for RBF kernel

The same method applies to plot the resulting alpha’s for polynomial kernel, the result is shown as Figure 5

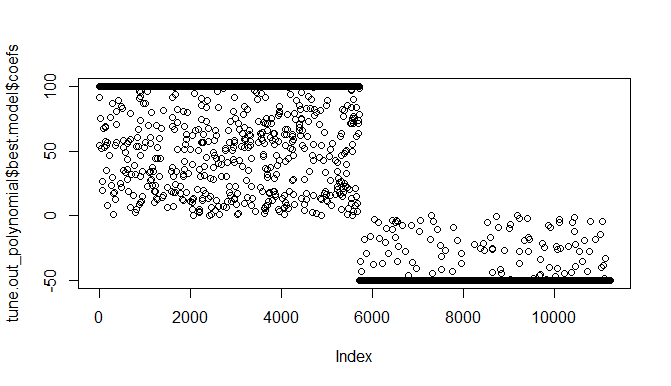


Figure 5. The resulting alpha’s for polynomial kernel

1. Table and mean function can tell us the accuracy for svm using polynomial kernel and the radial basis function (RBF) kernel. test.income is the true classifier while radial\_predict and polynomial\_predict are predicted classifiers using RBF and polynomial kernel respectively. The average accuracy for RBF kernel is 0.854 while that for polynomial kernel is 0.848. RBF kernel has higher accuracy than polynomial kernel.

