# CS 7641 – Machine Learning Assignment 1 -- Minming Zhao (gtID: 902685774)

## 1. Problem Statement

In this report, we will solve two binary classification problems for two datasets using R.

**Problem #1: Student alcohol addiction prediction**

To study student possibility to be alcohol addictive with relation to their school, gender, age, address, parents' job, their study time etc. Knowing better about student alcohol addiction could help families and teachers pay more attentions to those students who are more likely to be alcohol addictive and help students to study hard and let them enjoy other beautiful aspects of the life other than alcohol addiction.

**Problem #2: Adult census income prediction**

Money making is always an interesting question, isn't it? This is to predict whether one could make income more than $50k/y or not based on census data. With this study prediction, it help us understand how to make more money by locating some key features who did make over $50k/y.

## 2. Table of Content



## 3. Problem #1 Student alcohol addiction prediction

### 3.1. Dataset Attributes

| Features | Attributes |
|---|---|
| **school** | binary: school 'GP' or School 'MS' |
| **sex** | binary: 'F' - female or 'M' - male |
| **age** | numeric: from 15 to 22 |
| **address** | binary: 'U' - urban or 'R' - rural |
| **famsize** | binary: 'LE3' - less or equal to 3 or 'GT3' - greater than 3 |
| **Pstatus** | binary: 'T' - parents living together or 'A' - apart |
| **Medu** | numeric: 0 - none, 1:- primary education (4th grade), 2:5th to 9th grade, 3 secondary education or 4: higher education |
| **Fedu** | numeric: 0 - none, 1 - primary education (4th grade), 2 : 5th to 9th grade, 3 : secondary education or 4 : higher education |
| **Mjob** | nominal: 'teacher', 'health' care related, civil 'services' (e.g. administrative or police), 'at_home' or 'other' |
| **Fjob** | nominal: 'teacher', 'health' care related, civil 'services' (e.g. administrative or police), 'at_home' or 'other' |
| **traveltime** | home to school. numeric: 1 - <15 min., 2 - 15 to 30 min., 3 - 30 min. to 1 hour, or 4 - >1 hour |
| **studytime** | numeric: 1 - <2 hours, 2 - 2 to 5 hours, 3 - 5 to 10 hours, or 4 - >10 hours |
| **failures** | numeric: n if 1<=n<3, else 4 |
| **higher** | higher education wants. binary: yes or no |
| **internet** | binary: yes or no |
| **romantic** | a romantic relationship. binary: yes or no |

| | |
|---|---|
| **famrel** | family relation. numeric: from 1 - very bad to 5 - excellent |
| **freetime** | numeric: from 1 - very low to 5 - very high |
| **goout** | numeric: from 1 - very low to 5 - very high |
| **health** | numeric: from 1 - very bad to 5 - very good |
| **absences** | numeric: from 0 to 93 |
| **Grades** | numeric: from 0 to 20, output target |
| **Consumption_additive** | Alcohol addiction. binary: yes or no |

## 3.2. Learning curve to help decide train data size

We take a look at the learning curve to see what the minimum training size for each algorithm is. As we could notice from the below that SVM require least amount of training size while neural network, boosting and knn will require most training set. For sanity to obtaining best model, we later randomly assign 80% of all data as training set and remaining 20% for test set.

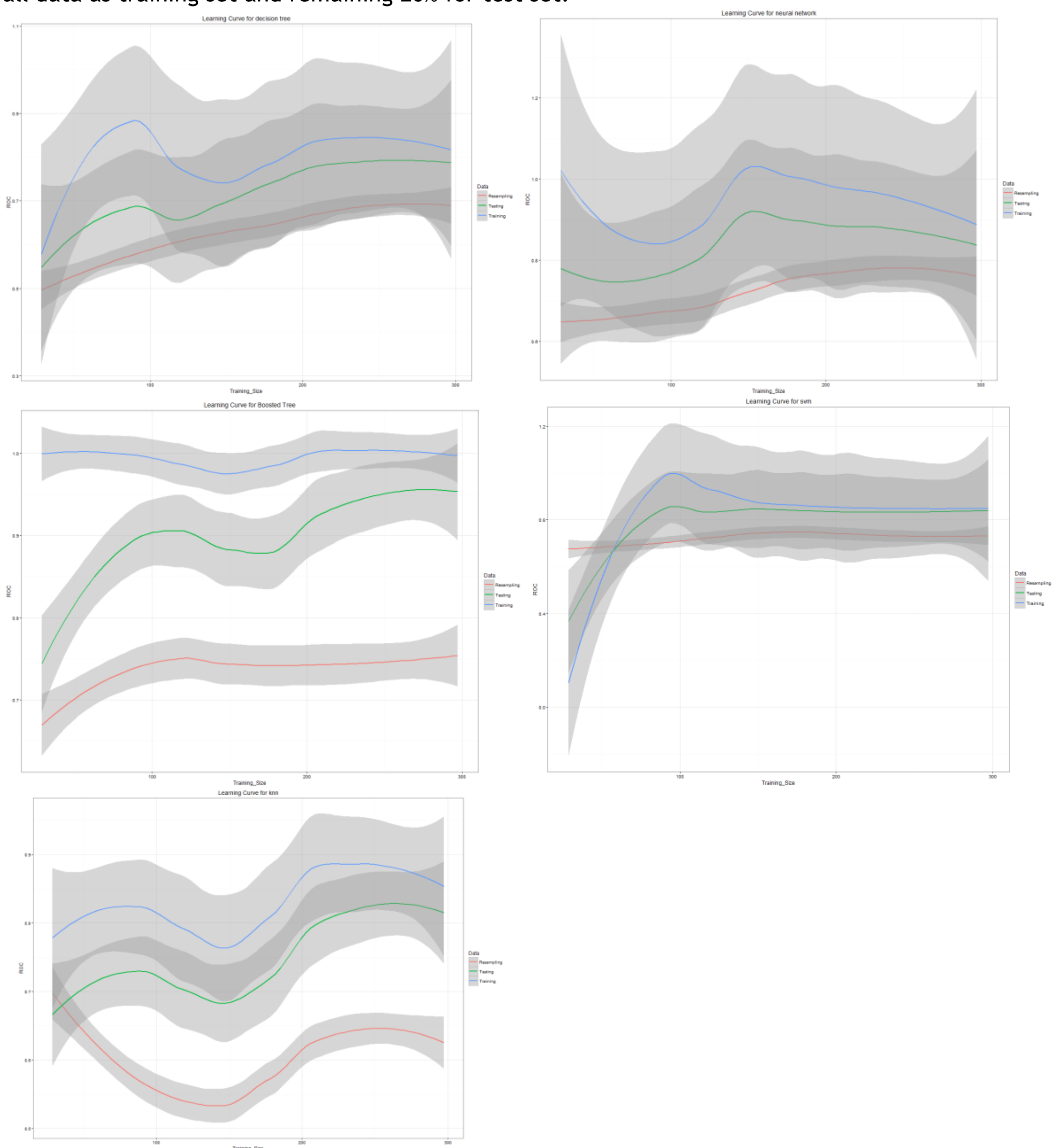



Fig.1 Learning Curves of 5 learning algorithms

Then we will randomly pick TrainData and TestData. We use histogram of TrainData and TestData to ensure they are similarly distributed. For example we could look at the age distribution in both training and testing.

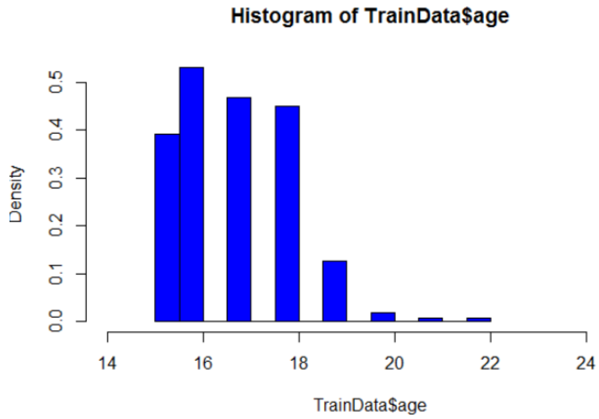


Fig.2: Example distribution of train set

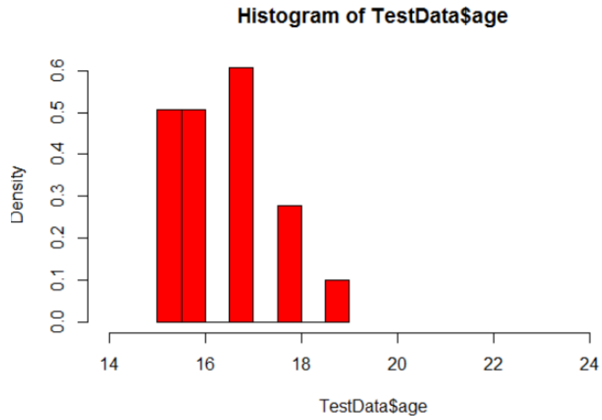


Fig.3: Example distribution of test set

## 3.3. Modeling learning algorithm

### 3.3.1. Decision Tree with some form of pruning.

Here we use package *tree* to do the decision tree model as it provide convenient cross validation as well pruning features.

The key code in r to build decision tree mode is,

```
treeMod <- tree(Consumption_additive_f ~ school + sex + age + address + famsize +
Pstatus + Medu + Fedu + Mjob + Fjob + traveltime + studytime + failures + higher
+ internet + romantic + famrel + freetime + goout + health + absences,
data=TrainData)
```

And then we could call “plot(treeMod); text(treeMod,pretty=0)” to plot the decision tree. As we can see from the below, we have 20 terminal nodes.

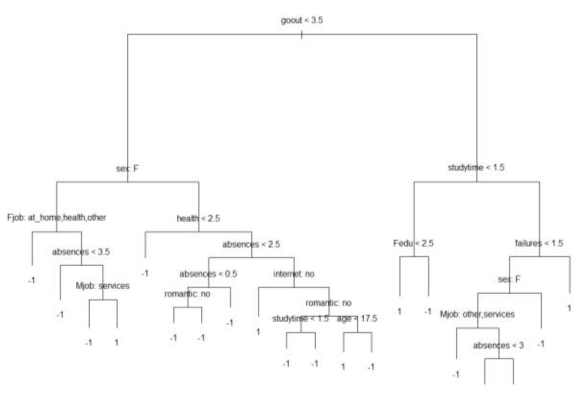


Fig.4 Decision Tree for Dataset 1

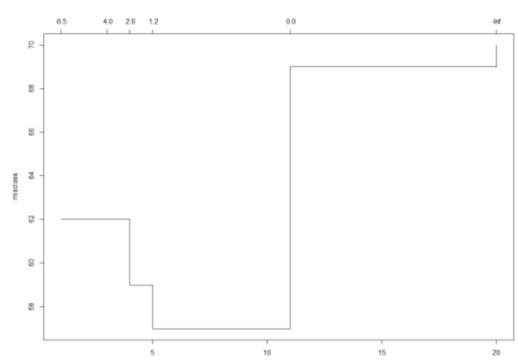


Fig.5: Pruning – misclass vs tree nodes

We could take a close look at the decision tree, it makes sense with our normal impression on how much a student drink to do with their behaviors or families. For example, students who go out often may drink more than those mostly stay at home. Or another example could be, male students tend to drink than females. Or the students who study more may not have that much time to drink alcohol as much as those who spend less time in study. This is to do with our domain knowledge, which makes sense as well. We can see we have 20 terminal nodes and we may have some overfitting situation here.

We could do some pruning by easily calling cv.tree function in package *tree*. Figure 5 lower X axis is the count of terminal nodes and upper X axis is the count of folds, i.e. # of pieces the data is split in the cross validation. This shows how this misclassification error average against the terminal node counts and number of folds. Hence the optimal number of terminal nodes should be one which gives lowest misclass while

smallest number of terminal nodes since usually the fewer nodes, the simpler decision tree model is. In this case, we can try best terminal nodes count = 5.
Thus we could get the plot of the pruned decision tree as below.

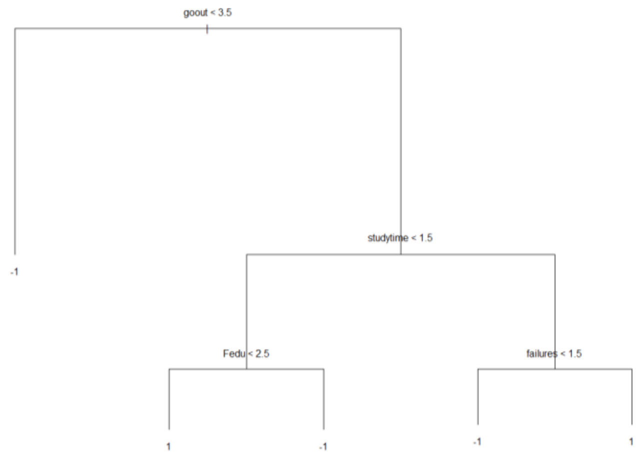



Fig.6: Pruned Decision Tree

Because we do pruning, we are avoiding overfitting in the decision tree. Hence it is expected that we will lose some classification accuracy here using pruning for Training Dataset. But we should expect to improve accuracy in Test Dataset.

Table 1: Error rate for decision tree

| | Without Pruning | With Pruning |
|---|---|---|
| Training set | 0.1139241 | 0.1360759 |
| Testing set | 0.1898734 | 0.1772152 |

From the testing dataset, we can see the misclassification error reduces from 19% to 17.7%, which implies we did avoid overfitting and improve the generalization procedure for pruning.

### 3.3.2. Neural Networks

Here We use package *neuralnet* and need to install it at the first time by running install.packages (“neuralnet”).
Step1: At the first step, we need to normalize the data before training a neural network. Here I used min-max scale method to scale the data to interval of [0,1].
Step2: We are going to build neural network model. Here we use logistic function as the activation function as we have classification at the end. Here we have 22 input at the first layer and target to have 1 output at the end. Usually we target to have 2/3 nodes of previous layers so we decide to have 10 and 5 layers as hidden layers. I would like to have activation function been applied to the output thus the linear.output set to FALSE.

```
nn <- neuralnet(Consumption_additive ~ school.f + sex.f + age + address.f +
famsize.f + Pstatus.f + Medu + Fedu + Mjob.f + Fjob.f + traveltime + studytime +
failures + higher.f + internet.f + romantic.f + famrel + freetime + goout +
health + absences + Grades, data=TrainData,  act.fct = "logistic", hidden =
c(10,5), linear.output=FALSE)
```

And we could try different layers and nodes as well as the activation function to see the best neural network model.

Table 2: Best neural network model

| Act.function | Layers | TrainErrorRate | TestErrorRate |
|---|---|---|---|
| **logistic** | c(10) | 0.01898734177 | 0.2278481013 |
| **logistic** | c(10,5) | 0.01898734177 | 0.1392405063 |
| **logistic** | c(15,10,5) | 0.03164556962 | 0.1772151899 |
| **tanh** | c(10,5) | 0.01898734177 | 0.2278481013 |

And we could plot the neural network as below.

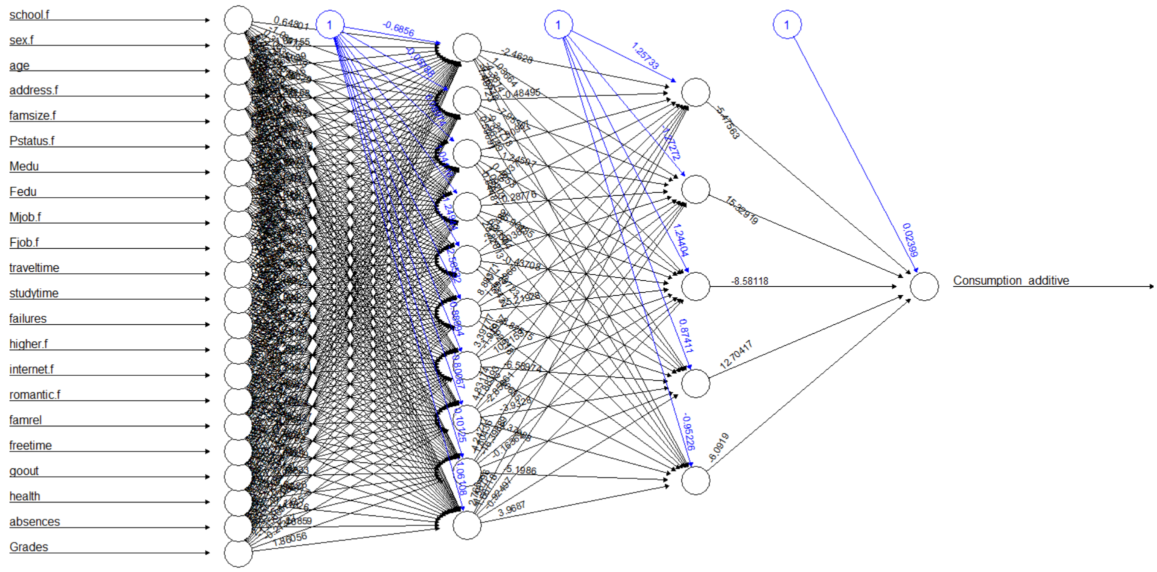



Fig.7: Neural Network plot for dataset #1

Step3: We check the accuracy of this neural network model.

Table 3: Neural Network error rate

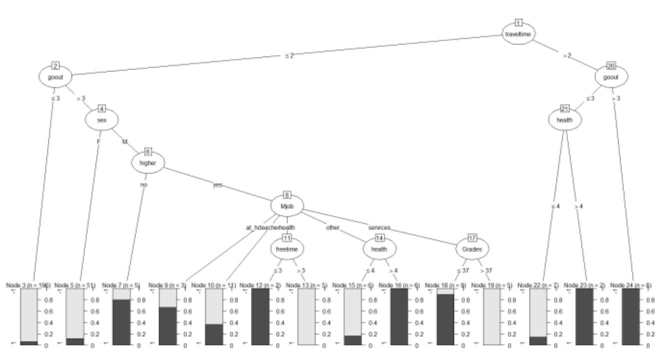
| | error rate | error rate avg. from 10 times |
|---|---|---|
| Training set | 0.01898734177 | 0.02974683544 |
| Testing set | 0.1392405063 | 0.2113924051 |

As we can see, the misclassification in training dataset is only 1.9% and the misclassification in test dataset is 13.9%. Comparing to Decision tree, which is 17~18% misclassification, here the neural network model we have built here is better.

Step4: Cross Validation by running 10 times of training and testing dataset split and then take the average misclassification error. With 10 times repeated run, we can get an idea how the neural model performs over other learning algorithm. We get average test dataset misclassification error about 21%, which a bit worse than decision tree we have. Also the training set data error is only 0.03, which leads us to think neural network here may be overfitting already. But we have to admit that we have tried many combination of different layers and nodes in neural networks. However neural networks seems performs a bit slow in this case.

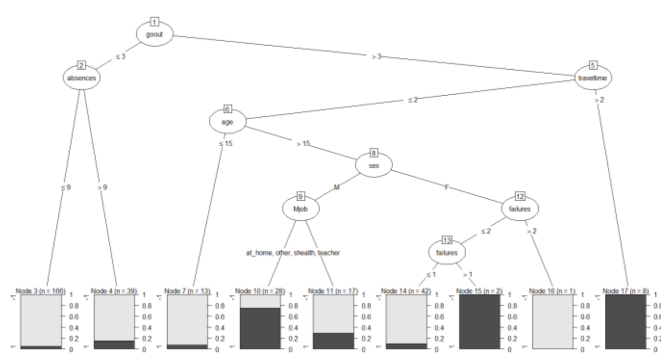
### 3.3.3. Boosting.

Here we use package C50 to do decision boosting as it provide a more convenient way to setting the boosting parameter.



Testing Error rate= 0.2278481

Fig. 8: Decision tree with some Pruning



Testing Error rate= 0.1898734

Fig.9: Boosting with Pruning

Firstly we do a decision tree again using C50 with some pruning. Then use Boost for the decision tree by assigning trials = 10. Also try to use feature selection (winnow=TRUE) to do more aggressive in pruning.

```
treeMod_boosting10 <- C5.0(TrainData[-23],
TrainData$Consumption_additive.f,trials = 10,control =
C5.0Control(noGlobalPruning = FALSE,earlyStopping = TRUE,CF=0.25,winnow=FALSE))
```

Hence the boost improves the accuracy by reducing error rate from 22.78% to 19%.

### 3.3.4. Support Vector Machines

We use package caret to do SVM. We use radial basic function kernel at the first and then use linear kernel. As always we will need to scale the data to regularize the values. From lecture, we know the larger C, the more fit in training set, which may leads to over fit the training set. Thus we will watch the error of test set to locate the best C.

**Gaussian kernel**

here we will try different C to find best model. We will do 10 folds and repeat 5 times for cross validation.

```
> ctrl <- trainControl(method="repeatedcv",   # 10 fold cross validation
                       repeats=5,          # do 5 repititions of cv
                       summaryFunction=twoClassSummary,
                       classProbs=TRUE)    # Use AUC to pick the best model #
> svm.tune1 <- train(x=TrainX,y= TrainY,method = "svmRadial", # Radial kernel
+                     tuneLength = 9, # 9 values of the cost function
+                     preProc = c("center","scale"),  # Center and scale data
+                     metric="ROC",trControl=ctrl)
```

Hence for different C, we can pick the C which gives largest ROC. The final values used for the model were sigma = 0.02648622 and C = 1. Resampling results across tuning parameters:

| C | ROC | Sens | Spec |
|---|---|---|---|
| 0.25 | 0.8074256 | 0.9549231 | 0.3006667 |
| 0.50 | 0.8075333 | 0.9518769 | 0.3106667 |
| 1.00 | 0.8079744 | 0.9556923 | 0.2786667 |
| 2.00 | 0.7979231 | 0.9604615 | 0.2580000 |
| 4.00 | 0.7848410 | 0.9581231 | 0.2173333 |
| 8.00 | 0.7725179 | 0.9635077 | 0.1760000 |
| 16.00 | 0.7591333 | 0.9666462 | 0.1626667 |
| 32.00 | 0.7586718 | 0.9666462 | 0.1466667 |
| 64.00 | 0.7592513 | 0.9659077 | 0.1706667 |

Similarly we step further to fine tune to find best C and sigma for our model by giving C a range c(0.75, 0.9, 1, 1.1, 1.25) around 1 and sigma range c(.01, 0.0264, 0.3). Similarly we find the best C = 1.25 and sigma = 0.01. Then the train set and testing set error is as below.

Table 4: SVM error rate

| | error rate of Radial kernel (Gaussian) | error rate of Linear kernel (no kernel) |
|---|---|---|
| Training set | 0.1234177 | 0.1202532 |
| Testing set | 0.164557 | 0.1898734 |

Hence the misclassification error of training set is around 0.123 while misclassification error of test dataset is 0.164.

**Linear Kernel**

Linear kernel is similar process with Gaussian kernel except to explicitly define it by method = svmLinear in the key train function. We try different C as well by giving tuneGrid c(0.9, 1, 1.1, 1.25,1.5,2)

```
> grid2<- expand.grid(C = c(0.9, 1, 1.1, 1.25,1.5,2))
> svm.tune2 <- train(x=TrainX,y= TrainData$Consumption_additive,method =
"svmLinear",preProc = c("center","scale"),metric="ROC",tuneGrid =
grid2,trControl=ctrl)
```

Similarly ROC was used to select the optimal model using the largest value. The final value used for the model was C = 1.1. The training and testing error rate is in Table X. Hence the misclassification error of training set is around 0.12 while misclassification error of test dataset is 0.189.

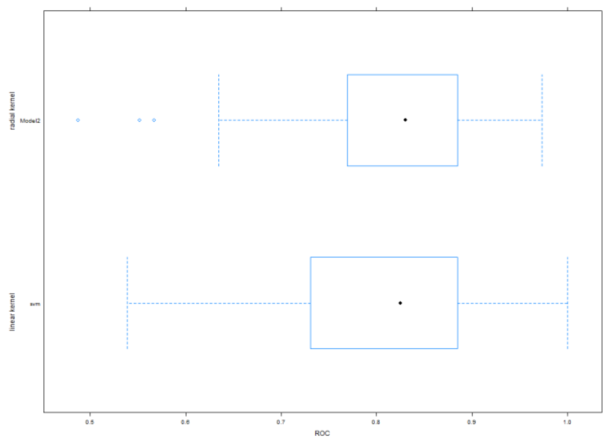


Fig. 10: SVM kernel function ROC comparison

Therefore we can see the Gaussian kernel performs a little better accuracy than liner kernel for testing dataset. It seems that linear kernel here overfits a bit in trainset as liner kernel provides a bit better accuracy in training set than Gaussian kernel. As to the processing time, both kernel performs similarly quick as we don't have big size of data in this problem. And it did performs faster than neural network.

### 3.3.5. k-Nearest Neighbors.

Here we use package *class* to do knn learning. The key function to do knn algorithm in r is as below by using package class.

```
pred_test_knn = knn(TrainData[,-23], TestData[,-23], cl=TrainData[,23], k, l = 0,
prob = FALSE, use.all = TRUE)
pred_train_knn = knn.cv(TrainData, cl=TrainData[,23], k, l = 0, prob = FALSE,
use.all = TRUE)
```

Table 5: KNN error rate

| k value | TrainDataErr | TestDataErr | k value | TrainDataErr | TestDataErr |
|---|---|---|---|---|---|
| **1** | 0.136076 | 0.227848 | **7** | 0.164557 | 0.177215 |
| **2** | 0.167722 | 0.291139 | **8** | 0.167722 | 0.164557 |
| **3** | 0.129747 | 0.253165 | **9** | 0.177215 | 0.177215 |
| **4** | 0.148734 | 0.253165 | **10** | 0.180380 | 0.164557 |
| **5** | 0.161392 | 0.189873 | **11** | 0.174051 | 0.164557 |
| **6** | 0.170886 | **0.164557** | **12** | 0.183544 | 0.164557 |

Hence we can pick can k = 6 which can give us best Test data error.

## 4. Second dataset #2 - adult dataset

### 4.1. Data attributes

| **age** | **continuous.** |
|---|---|
| **workclass** | Private, Self-emp-not-inc, Self-emp-inc, Federal, Local, State, Wo pay, Never-worked. |
| **fnlwgt** | continuous. |
| **education** | Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool. |
| **education-num** | continuous. |
| **marital-status** | Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse. |
| **occupation** | Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport etc. |
| **relationship** | Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried. |

| | |
|---|---|
| **race** | White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black. |
| **sex** | Female, Male. |
| **capital-gain** | continuous. |
| **capital-loss** | continuous. |
| **hours-per-week** | continuous. |
| **native-country** | United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China etc. |

## 4.2. Learning curve to help decide train data size

As we did previously, we take a look at the learning curve to see what the minimum training size for each algorithm is. As we could notice from the below that neural networks require least amount of training size while boosting and knn will require most training set size. For sanity to obtaining best model, we later randomly assign 80% of all data as training set and remaining 20% for test set.

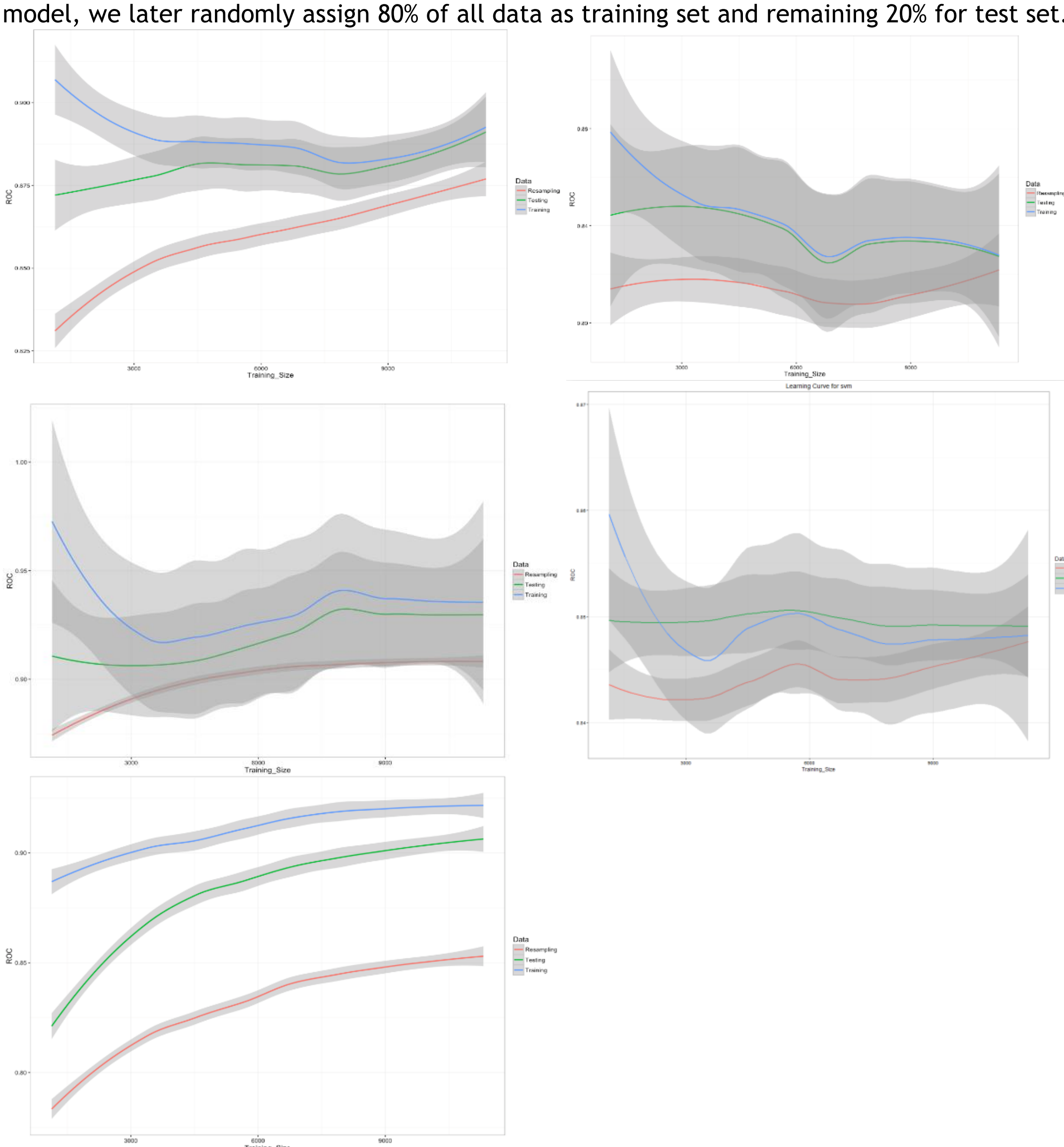



Fig. 11: Learning Curves for 5 algorithms (upper left right to bottom: decision tree, neural networks, boosting, svm, knn)

Similaryly we will ensure randomly selected train and test set are similar distributed set. For example we could look at the age distribution of training and testing set.

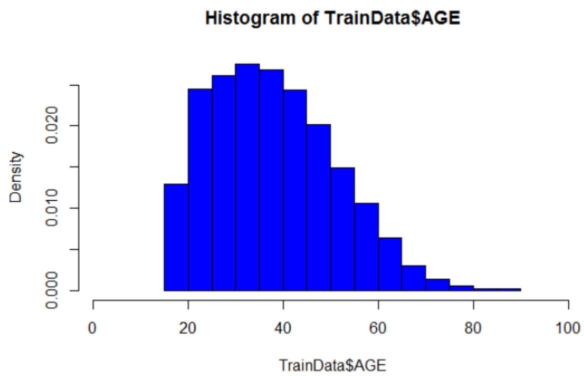



Fig.12: Example distribution of train set

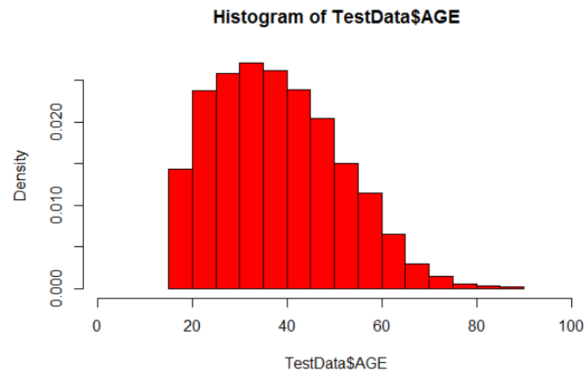



Fig.13: Example distribution of test set

## 4.3. Modeling learning algorithm

### 4.3.1. Decision tree

Similarly we use package tree to do decision tree algorithm.

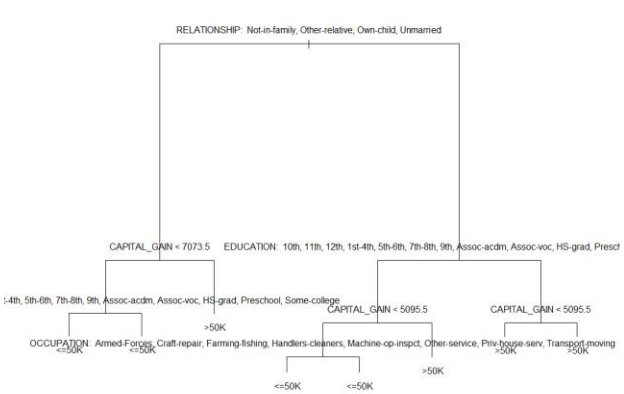



Fig. 14: Decision Tree without pruning for Dataset #2

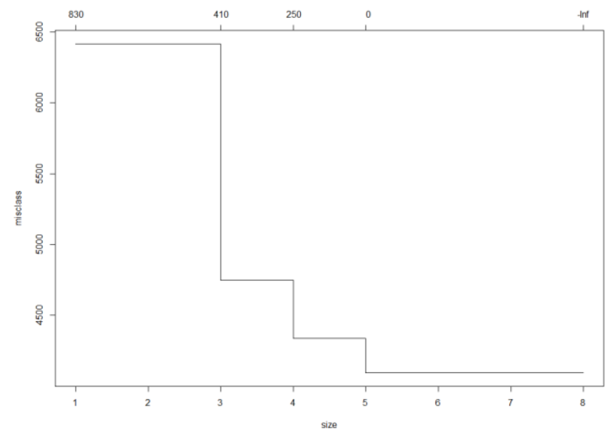



Fig.15: Pruning – misclass vs tree nodes

Table 6: Error rate for decision tree

| | Without Pruning | With Pruning |
|---|---|---|
| Training set | 0.157031 | 0.157031 |
| Testing set | 0.1665838 | 0.1665838 |

The cross validation and pruning Fig. tell us we should stay with minimum terminal nodes 5. Thus we will have the same training and testing error. The tree could be concise a little bit as the ending branch at the first one had same outputs. Thus it will looks like below.

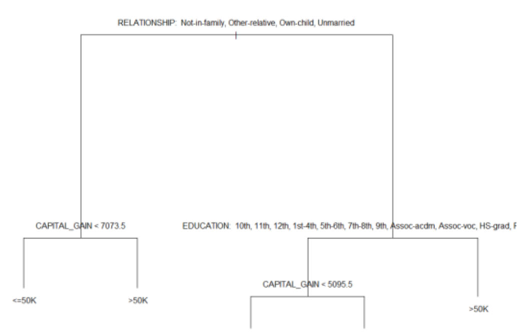



Fig. 16: Pruned Decision tree for dataset #2

### 4.3.2. Neural Network

As always, we will firstly normalize the data and I use max-min method to savcle it to [0,1] like previous dataset. We also use kernlab to se neuralnet function. The neural network model is very slow here due to large amount of data size. Thus I just do one try with one single hidden layer with 4 nodes and activation function tanh.

```
nn <- neuralnet(SALARY_CLASS_n ~ AGE + WORKCLASS_n + FNLWGT + EDUCATION_n +
EDUCATION_NUM + MARITAL_STATUS_n + OCCUPATION_n + RELATIONSHIP_n + RACE_n + SEX_n
+ CAPITAL_GAIN + CAPITAL_LOSS + HOURS_PER_WEEK + NATIVE_COUNTRY_n,
data=TrainData,  act.fct = "tanh", hidden = 4, linear.output=T)
```

The cross validation was done by repeating the same process 10 times and the average error we got is around 68%. The error rate here is very big, I believe it is due to I have not tried to select a best hidden layers and the node numbers of each layer.

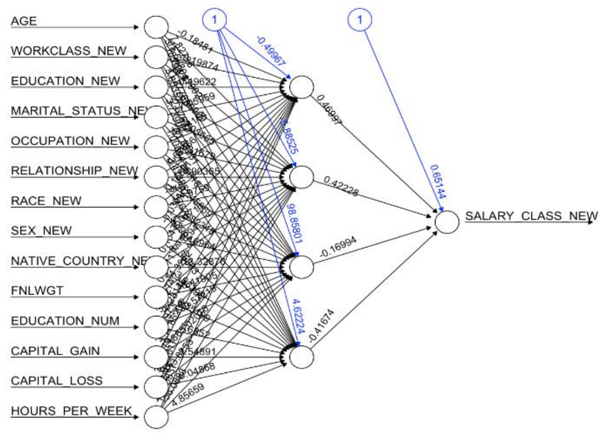



Fig. 17: Neural network plot for dataset#2

### 4.3.3. Boosting

Here similar to the first dataset, we use package C50 to do decision boosting as it provide a more convenient way to setting the boosting parameter.
Firstly we build the decision tree using C50 without pruning. The code to build decision tree model using C50 is as below

```
treeMod <- C5.0(TrainData[-15], TrainData$SALARY_CLASS_f,control =
C5.0Control(noGlobalPruning = TRUE,earlyStopping = FALSE,CF=0.25,winnow=FALSE))
```

| actual default | predicted default <=50K | >50K | Row Total |
|---|---|---|---|
| <=50K | 4232<br>0.701 | 336<br>0.056 | 4568 |
| >50K | 490<br>0.081 | 975<br>0.162 | 1465 |
| Column Total | 4722 | 1311 | 6033 |

Test data error rate = 0.1369136416
Fig.18: Decision Tree w/o pruning

| actual default | predicted default <=50K | >50K | Row Total |
|---|---|---|---|
| <=50K | 4237<br>0.702 | 331<br>0.055 | 4568 |
| >50K | 492<br>0.082 | 973<br>0.161 | 1465 |
| Column Total | 4729 | 1304 | 6033 |

Test data error rate = 0.1364163766
Fig.19: Decision Tree with pruning

| actual default | predicted default <=50K | >50K | Row Total |
|---|---|---|---|
| <=50K | 4231<br>0.701 | 337<br>0.056 | 4568 |
| >50K | 483<br>0.080 | 982<br>0.163 | 1465 |
| Column Total | 4714 | 1319 | 6033 |

Test data error rate = 0.135919111
Fig.20: Decision tree with boosting

As to do more aggressively in pruning, we could play with winnow option. However the winnow option sometimes could help improve accuracy sometimes it does not. Because the winnow option is erroneously removing predictors that can improve the accuracy of the model. Within the cross-validation loop, the winnowing process thinks that it is improving the accuracy, but that is not holding up once other samples are used to evaluate performance. In this case (different than the first dataset), winnow option is not helping improving the accuracy.

### 4.3.4. SVM

Here we use Kernlab package and use function ksvm to build the SVM model. This function provides cross validation tuning as well as defining the C and sigma for the Gaussian kernel. Similar to what we have done before, we will try different parameters to find best model. Here we try different C for SVM functions.
For computational sanity, we pick cross validation = 3, one can pick higher number such as 10 as we do before.

- **Gaussian Kernel**

For Gaussian Kernel, we use kernel = "rbfdot", then the key code to build SVM is,

```
svm_mdl <- ksvm(SALARY_CLASS_n~.,data = TrainData,scaled = TRUE,kernel ="rbfdot",
kpar=list(sigma=0.05), C=C1,cross=3)
```

The different C will gives the training set error and test error as below,

Table 7: SVM Gaussian error rate

| C | TrainDataErr | TestDataErr | C | TrainDataErr | TestDataErr |
|---|---|---|---|---|---|
| **0.25** | 0.158067 | 0.159456 | **8.00** | 0.139044 | **0.151500** |
| **0.50** | 0.154254 | 0.156473 | **16.00** | 0.134195 | 0.151997 |
| **1.00** | 0.150773 | 0.154649 | **32.00** | 0.129554 | 0.154318 |
| **2.00** | 0.147540 | 0.153158 | **64.00** | 0.123917 | 0.156141 |
| **4.00** | 0.143189 | 0.151832 | | | |

As we could see from different C, best C = 8 which gives lowest Test Data Error = 0.1515. Then we run with C = 8 again and we could see the cross validation error is 0.15545.

- **Linear Kernel**

For Linear Kernel, i.e. no kernel but just use x directly, we use kernel = "vanilladot", then the key code to build SVM is,

```
svm_mdl <- ksvm(SALARY_CLASS_n~.,data = TrainData,scaled = TRUE,
kernel="vanilladot", kpar=list(sigma=0.05), C=C1, cross=3)
```

Table 8: SVM Linear kernel error rate

| C | TrainDataErr | TestDataErr | C | TrainDataErr | TestDataErr |
|---|---|---|---|---|---|
| **0.25** | 0.18997886 | 0.19244157 | **4.00** | 0.18977164 | 0.19293884 |
| **0.50** | 0.19006175 | 0.19277308 | **8.00** | 0.18977164 | 0.19293884 |
| **1.00** | 0.18985453 | **0.19277308** | **16.00** | 0.18981309 | 0.19293884 |
| **2.00** | 0.18989598 | 0.19293884 | **32.00** | 0.18981309 | 0.19293884 |

Hence, best C = 1 which gives lowest Test Data Error = 0.19277. Then we run with C = 8 again and we could see the cross validation error is 0.190145.
As we can see the Gaussian kernel has better accuracy at test dataset as well as training set. Also a noticeable difference between kernel Gaussian and linear kernel is the processing time. The linear kernel took almost five times more clock time to process the grid of C parameter. The final C = 64 even take more than 30 mins and still it could not generate a result.

### 4.3.5. kNN

Here similarly we use package class to do knn learning. The key function to do knn algorithm in r is as below by using package class.

```
pred_test_knn = knn(TrainData[,-15], TestData[,-15], cl=TrainData[,15], k, l = 0,
prob = FALSE, use.all = TRUE)
pred_train_knn = knn.cv(TrainData[,-15], cl=TrainData[,15], k, l = 0, prob =
FALSE, use.all = TRUE)
```

Table 9: kNN error rate

| k value | TrainDataErr | TestDataErr | k value | TrainDataErr | TestDataErr |
|---|---|---|---|---|---|
| **1** | 0.27825438 | 0.28692193 | **25** | 0.2087944 | **0.20968009** |
| **4** | 0.25355381 | 0.25178187 | **30** | 0.2090845 | 0.20984585 |

| | | | | | |
|---|---|---|---|---|---|
| **8** | 0.22379709 | 0.22708437 | **35** | 0.20954039 | 0.21100613 |
| **10** | 0.21642008 | 0.22028841 | **40** | 0.20999627 | 0.21183491 |
| **15** | 0.21016205 | 0.21034311 | **45** | 0.21140536 | 0.21249793 |
| **20** | 0.20858718 | 0.21084038 | **50** | 0.21198558 | 0.21382397 |

So the best k =25. The Test data error is 0.2097.

## 5. Conclusion

After running 5 learning algorithms, we could conduct a comparison between them to see the accuracy as well processing time needed.

| Machine Learning in R | Decision Tree w/ Pruning | Neural Network | Boosting | SVM-Gaussian | SVM-Linear | kNN |
|---|---|---|---|---|---|---|
| **Package used** | tree | kernlab | C5.0 | caret or ksvm | caret or ksvm | class |
| **Dataset #1 rough est. Minimum Training size** | 300 | 400 | 500 | 150 | 150 | 600 |
| **Dataset #1 Training error rate** | 13.6% | 3.0% | 8.5% | 12.3% | 12.0% | 17.1% |
| **Dataset #1 Test error rate** | 17.7% | 21.1% | 19.0% | 16.5% | 19.0% | 16.5% |
| **Dataset #2 rough est. Minimum Training size** | 9000 | 3000 | 12000 | 10000 | 10000 | 16000 |
| **Dataset #2 Training error rate** | 15.7% | N/A | N/A | 13.9% | 19.0% | 20.9% |
| **Dataset #2 Test error rate** | 16.7% | 68.0% | 13.6% | 15.2% | 19.3% | 21.0% |
| **Processing speed** | fast | median-slow | median | fast | fast | fast-median |

As we can see from above, almost all the learning algorithms achieve similar accuracy in testing set, that is the error is around 15% ~ 20% for both datasets. The exception is the neural network for dataset # 2. As we have covered previously, I think this is due to the fact that I have not searched different hidden layers and nodes as computational time consuming. All the algorithms has done cross validation in the modelling.

For the decision tree of dataset #2, the pruning does not improved the accuracy in testing set because we are already at optimal terminal nodes counts.

Generally speaking, I think SVM Gaussian algorithm performs best from test set error rate and the processing speed point of view. And it seems to require least amount of training set.

## 6. References:

https://cran.r-project.org/web/packages/C50/C50.pdf
http://www.cs.bc.edu/~alvarez/ML/statPruning.html
http://www.patricklamle.com/Tutorials/Decision%20tree%20R/Decision%20trees%20in%20R%20using%20C50.html
https://www.r-bloggers.com/the-5th-tribe-support-vector-machines-and-caret/
http://topepo.github.io/caret/miscellaneous-model-functions.html#yet-another-k-nearest-neighbor-function
https://cran.r-project.org/web/packages/class/class.pdf