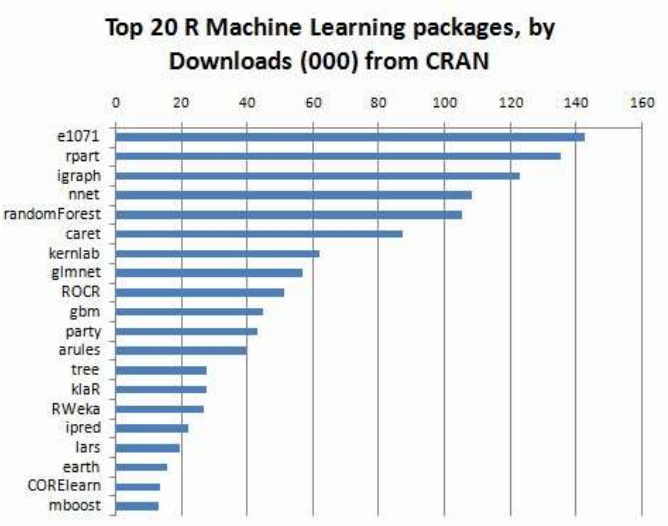
Machine Learning Package Selection

Below are the Open-source machine learning packages available:
The recent analysis(1) shows top 20 R Machine learning packages and how many of these are being downloaded from Jan to May 2015 by analyzing daily downloads in CRAN.



Since the goal of the assignment task is to use machine learning package for classification task and to predict the test result (which may be positive or negative) based on the other attributes. I am selecting package *caret* as it provides functions for performing classification task and for creating predictive models. The caret package contains tools for feature selection, model tuning using resampling, data splitting, pre-processing and other functionality. Other packages have different functions for example, e1071 package has functions for latent class analysis, fuzzy clustering, short time Fourier transform, support vector machines, etc.

- **Steps involved in data preparation** :

1. Read the illness.txt file in R Studio. Row and columns in the data are transposed and columns are named. All columns except test_result is changed to numeric from factor data.
2. Split the data into training and testing data sets. The function createDataPartition in the caret package splits the data within groups of the data, and conducts sampling within the classes. 75% of data goes to training dataset and the remaining goes to the testing data set.

- Two different classification algorithms chosen are:

a) k-Nearest Neighbors algorithm
b) Random Forest algorithm

k-Nearest Neighbor

knn algorithm(2) assumes all instances correspond to points in the n-dimensional space. The distance metric used to determine the nearest neighbors of an instance is the standard Euclidean distance. The knn algorithm assigns the maximum class value represented in the k nearest training examples for larger values of k. For example, considering a set of positive and negative training examples, along with a query instance x, to be classified. The 1-nearest neighbor algorithm classifies x, negative, whereas 10-nearest neighbor classifies it as positive.

Algorithm description(3):

- Find the K nearest neighbors in the training data set based on the Euclidean distance.

- Predict the class value(+ve or -ve) by finding the most common class among the K nearest neighbors.
- Calculate the accuracy as
  Accuracy = (No. of correctly classified examples / No. of testing examples) X 100
- Determine k based on the highest cross–validated accuracy

kNN requires variables to be normalized or scaled. caret provides facility to preprocess data using the function preprocess(). Pre-processing operations chosen are centering and scaling. The function trainControl is be used to specify the 10-fold cross-validation. The parameter *method* in The function train() in the caret package is used to specify the type of model(ex. knn, rf), preprocessing functions and tuneLength.

Fig1. shows the result for k-Nearest Neighbors where accuracy is used for estimating performance. The statistic Kappa takes into account the expected error rate due to unbalanced classes. Fig2. Plot yields Number of Neighbours Vs accuracy for 10-fold cross-validation.

knn Results:

```
k-Nearest Neighbors

283 samples
  8 predictor
  2 classes: 'negative', 'positive'

Pre-processing: centered, scaled
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 255, 255, 255, 254, 254, 255, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa      Accuracy SD  Kappa SD
   5  0.7457231  0.3492988  0.05882204   0.1590233
   7  0.7422368  0.3372588  0.06946483   0.1926461
   9  0.7491744  0.3573019  0.06957301   0.1861966
  11  0.7491303  0.3464311  0.07373942   0.2025643
  13  0.7431779  0.3342070  0.07432437   0.1942183
  15  0.7374308  0.3062924  0.07402362   0.2060998
  17  0.7374308  0.3025815  0.08065532   0.2336981
  19  0.7458113  0.3129279  0.08949556   0.2565765
  21  0.7422429  0.3024610  0.08230220   0.2439653
  23  0.7220398  0.2452825  0.07358231   0.2199481
  25  0.7196147  0.2385809  0.07030233   0.2142020
  27  0.7184182  0.2309423  0.07521650   0.2157680
  29  0.7244587  0.2460842  0.06769869   0.2029295
  31  0.7210135  0.2389731  0.07839958   0.2265040
  33  0.7174862  0.2191928  0.07009983   0.2042605
  35  0.7150642  0.2136690  0.06490245   0.1894462
  37  0.7149410  0.2112583  0.06009876   0.1678481
  39  0.7149380  0.2078805  0.06502886   0.1827078
  41  0.7127714  0.1980467  0.07134707   0.1971552
  43  0.7103463  0.1847175  0.07566576   0.2208956

Accuracy was used to select the optimal model using  the largest value.
The final value used for the model was k = 9.
```

Fig1.

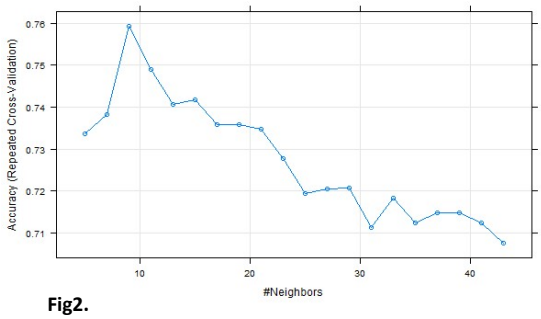


Fig2.

To estimate performance for classification, a confusionMatrix function is used which is a cross-tabulation of the predicted and observed values. e1071 package is installed to use confusionMatrix. Performance for 2-class classification models is also estimated using Sensitivity and Specificity in the package caret, which are directly related to the false positive and false negative rate of a method. Fig4. shows the ROC curve that plots the sensitivity (eg. true positive rate) by one minus specificity (eg. the false positive rate). The area under the ROC curve is a Common metric of performance.

```
Confusion Matrix and Statistics

          Reference
Prediction negative positive
  negative       50       14
  positive       14       15

               Accuracy : 0.6989
                 95% CI : (0.595, 0.7897)
    No Information Rate : 0.6882
    P-Value [Acc > NIR] : 0.461

                  Kappa : 0.2985
 Mcnemar's Test P-Value : 1.000

            Sensitivity : 0.7812
            Specificity : 0.5172
         Pos Pred Value : 0.7812
         Neg Pred Value : 0.5172
             Prevalence : 0.6882
         Detection Rate : 0.5376
   Detection Prevalence : 0.6882
      Balanced Accuracy : 0.6492

       'Positive' Class : negative
```

Fig3.

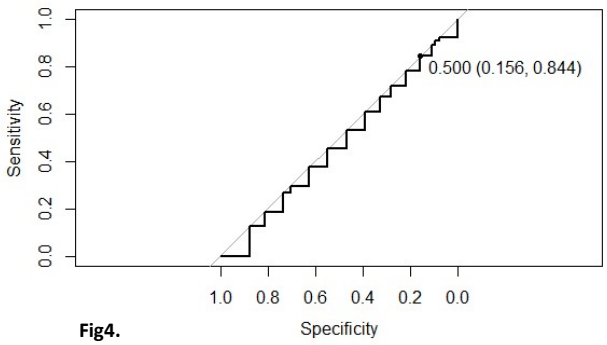


Fig4.

Random Forest

Random Forests(4) grows many classification trees. To classify a new object from an input vector, put the input vector down each of the trees in the forest. Each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

- trainControl and train functions are used to specify the method = “rf”.
- All other parameters like preprocessing and data preparation remains same as used in knn algorithm.
- same random number seed is set prior to the rf model that is idenditcal to the seed used for the knn model to ensure the same resampling sets are used.

randomForest Results:

Fig6. Plots the accuracy for randomly selected parameters.

```
> rfFit
Random Forest

283 samples
  8 predictor
  2 classes: 'negative', 'positive'

Pre-processing: centered, scaled
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 255, 254, 254, 255, 255, 254, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa      Accuracy SD  Kappa SD
  2     0.7594524  0.3874831  0.07412339   0.1968622
  3     0.7584763  0.3913787  0.07197538   0.1876615
  4     0.7680031  0.4175023  0.07465061   0.1905324
  5     0.7537584  0.3856620  0.08009738   0.1999777
  6     0.7586024  0.4005636  0.08004108   0.1999943
  7     0.7561394  0.3916476  0.07928025   0.1977391
  8     0.7467357  0.3710202  0.07899197   0.1960704

Accuracy was used to select the optimal model using  the largest value.
The final value used for the model was mtry = 4.
```

Fig5.

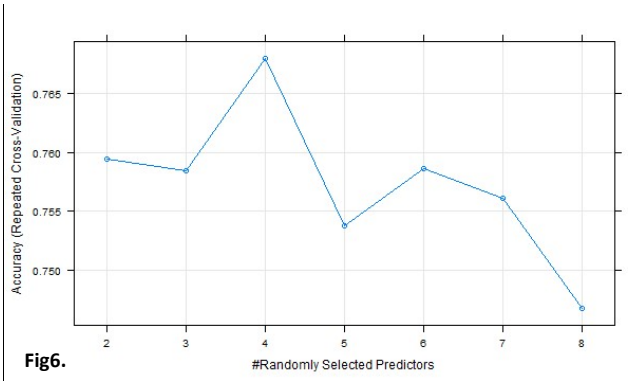


Fig6.

Similar to knn, performance of random forest is estimated using the parameters accuracy and confusionMatrix. ROC curve in Fig8. is plotted using the built-in function, twoClassSummary by computing the sensitivity and specificity.

```
Confusion Matrix and Statistics

          Reference
Prediction negative positive
  negative       53       13
  positive       11       16

               Accuracy : 0.7419
                 95% CI : (0.6408, 0.8271)
    No Information Rate : 0.6882
    P-Value [Acc > NIR] : 0.1568

                  Kappa : 0.3871
 Mcnemar's Test P-Value : 0.8383

            Sensitivity : 0.8281
            Specificity : 0.5517
         Pos Pred Value : 0.8030
         Neg Pred Value : 0.5926
             Prevalence : 0.6882
         Detection Rate : 0.5699
   Detection Prevalence : 0.7097
      Balanced Accuracy : 0.6899

       'Positive' Class : negative
```

Fig7.

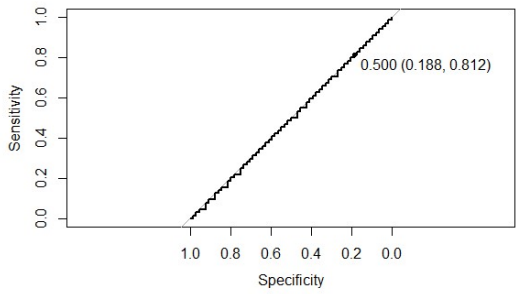


Fig8.

Caret functions used in R to determine repeated crossvalidation, preprocessing the data and the model type are:

For knn:

- ctrl <- trainControl(method="repeatedcv",repeats = 3)
- knnFit <- train(test_result~., data = illnessTrain, method = "knn", trControl = ctrl, preProcess = c("center","scale"),tuneLength = 20)

For Random Forest:

- ctrl <- trainControl(method="repeatedcv",repeats = 3)
- rfFit <- train(test_result ~ ., data = illnessTrain, method = "rf", trControl = ctrl, preProcess = c("center","scale"), tuneLength = 20)

Results:

- Based on the highest crossvalidated accuracy, k(no. of nearest neighbors) is determined as 9 for knn and k=4 for Random Forest.
- As seen in the confusion Matrix and statistics output accuracy is 0.6989, whereas Random forest has slightly higher accuracy of 0.7419.
- However in the confusion Matrix output, both knn and randomForest models determine "Positive" result level: Negative.
- Random Forest gives better predictions than knn algorithm due to the accuracy rate.

Reference List:

(1) Bhavya Geethika. Top 20 R Machine Learning and Data Science packages. *Data Mining, Analytics, Big Data, and Data Science.* Available from: http://www.kdnuggets.com/2015/06/top-20-r-machine-learning-packages.html [Accessed 8th Oct 2015].
(2) Tom M.Mitchell. *Machine Learning.* McGraw-Hill Science/Engineering/Math; (March 1, 1997).
(3) Max Kuhn, Ph.D. *Predictive Modeling with R and the caret Package.* Available from: https://www.r-project.org/nosvn/conferences/useR-2013/Tutorials/kuhn/user_caret_2up.pdf [Accessed on 9th Oct 2015].
(4) Leo Breiman and Adele Cutler. *Random Forests*. Available from: https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm [Accessed on 11th Oct 2015