**COSC 757 Data Mining Lab 5: Neural Networks, SVM, Boosting, and Bagging**

**Spring 2016**

**Classification Using Neural Networks**

In this part of the lab we will be classifying the iris dataset from Lab 3 using a neural network classifier. The package NNET created by Ripley provides methods for using feed-forward neural networks with a single hidden layer, and for multinomial log-linear models.

1) Download and install the nnet package

2) Load the iris dataset:

> data(iris)

3) Using the method that we used in Lab 3 as well as the SVM part of this lab, set up your training and testing datasets for the classification exercise.

> set.seed(1234)

> ind <- sample(2, nrow(iris), replace=TRUE, prob=c(0.7,0.3))

> trainData <- iris[ind==1,]

> testData <- iris[ind==2,]

4) Now let’s build two networks. The first (nn1) is a very simple network and the second (nn2) we will use 2 hidden layer nodes, initial weights in [-0.1, 0.1], weight decay 5e-4 (decay is penalty for larger weights; this is for avoiding overfit), and 200 iterations.

> nn1 <- nnet(Species~., trainData, size = 0, skip=TRUE, linout=TRUE)

> nn2 <- nnet(Species~., trainData, size=2, rang=0.1, decay=5e-4, maxit=200)

5) We can test the neural networks and generate a confusion matrix using the following command:

> table(testData$Species,predict(nn1, testData, type = "class"))

> table(testData$Species,predict(nn2, testData, type = "class"))

How do your networks compare in terms of their prediction accuracy? How do they with the classifiers that were used in Lab 3 in their ability to properly classify the iris dataset?

6) Now repeat the experiment by generating new test and training datasets to see if you can obtain the same results. Remember to change the random seed before creating the training and testing datasets.

**Classification Using SVM**

The e1071 package was the first implementation of SVM in R. The svm() function provides an interface to libsvm [13], complemented by visualization and tuning functions. libsvm is a fast and easy-to-use implementation of the most popular SVM formulation of classification (C and ), and includes the most common kernels (linear, polynomial, RBF, and sigmoid). Multi-class classification is provided using the one-against-one voting scheme. It also includes the computation of decision and probability values for predictions, shrinking heuristics during the fitting process, class weighting in the classification mode, handling of sparse data, and cross-validation.

The R implementation is based on the S3 class mechanisms. It basically provides a training function with standard and formula interfaces, and a predict() method. In addition, a plot() method for visualizing data, support vectors, and decision boundaries is provided. Hyperparameter tuning is done using the tune() framework, which performs a grid search over specified parameter ranges.

1) Load the e1071 package

This lab will be demonstrated on the cats dataset where we will use SVM to classify the sex of domestic cats based on two numeric variables Bwt (body weight in kg) and Hwt (Heart weight in g).

2) Load the cats dataset from the MASS package

> library(MASS)

> data(cats)

3) Now create a support vector machine model using the svm() function where the parameter "Sex~." indicates the attribute (column) of the dataset to be used as instance classes.

> model <- svm(Sex~., data = cats)

4) To inspect the parameters and the number of support vectors of the model type

> print(model)

> summary(model)

5) To see the built model with a scatter plot of the input, the plot() function can be used. This function optionally draws a filled contour plot of the class regions. The main parameters of this function are listed below.

> plot(model,cats)

6) Now create a training and testing dataset to tune the svm model. The first (trainData) is the training dataset and consists of 70% of the tuples. The second (testData) is the testing dataset and consists of 30% of the tuples. Do create the testing and training datasets, issue the following commands

> set.seed(1234)

> ind <- sample(2, nrow(cats), replace=TRUE, prob=c(0.7,0.3))

> trainData <- cats[ind==1,]

> testData <- cats[ind==2,]

7) Now recreate the model with the training dataset and test it with the test dataset

> model <- svm(Sex~., data = trainData)

> prediction <- predict(model, testData[,-1])

8) Create the confusion matrix using the table() function.

> tab <- table(pred = prediction, true = testData[,1])

9) With this information, it is possible to compute the sensitivity, the specificity and the precision of the model to the test set. Model accuracy rates can be computed using the classAgreement() function:

> classAgreement(tab)

10) You can visualize the results using the plot() function

> plot(model,trainData)

> plot(model,testData)

11) Finally, the tune() function can be used to tune hyperparameters of statistical methods. This approach uses a grid search over the supplied parameter ranges. The svm() takes the following parameters:

* **data:** an optional data frame containing the variables in the model. If this option is used, the parameters x and y described below, aren't necessary;
* **x:** a data matrix, a vector, or a sparse matrix that represents the instances of the dataset and their respective properties. Rows represent the instances and columns represent the properties;
* **y:** a response vector with one label for each row (instance) of x;
* **type:** sets how svm() will work. The possible values for classification are: C, nu and one (for novelty detection);
* **kernel:** defines the kernel used in training and prediction. The options are: linear, polynomial, radial basis and sigmoid;
* **degree:** parameter needed if the kernel is polynomial (default: 3);
* **gamma:** parameter needed for all types of kernels except linear (default: 1/(data dimension));
* **coef0:** parameter needed for polynomial and sigmoid kernels (default: 0);
* **cost:** cost of constraint violation (default: 1). This is the ‘C’-constant of the regularization term in the Lagrange formulation;
* **cross:** specifies the cross-validation. A k > 0 is necessary. In this case, the training data is performed to assess the quality of the model: the accuracy rate for classification;
* **probability:** logical indicating whether the model should allow for probability predictions.

To find the best parameters for gamma and cost we issue the following command:

> tuned <- tune.svm(Sex~., data = trainData, gamma = 10^(-6:-1), cost = 10^(1:2))

and to see the result:

> summary(tuned)

12) Now repeat your experiment using the tuned gamma and cost parameters and see if tuning the model can improve the results.

**Bagging**

The adabag package in R contains methods for both bagging and boosting. In the first example, we will use bagging to classify the Vehicle dataset available in the mlbench package.

1) First, you must install and the adabag, mlbench. and rpart packages.

> install.packages(‘adabag’)

> install.packages(‘mlbench’)

> install.packages(‘rpart’)

2) Then load the adabag package. This should automatically load the additional required packages.

> library(adabag)

3) Now load the Vehicle dataset from the mlbench package

> data(Vehicle)

4) Set a variable “l” to the length of the Vehicle dataset

> l <- length(Vehicle[,1])

5) Create a sample index to randomly sample 2/3 of the Vehicle dataset

> sub <- sample(1:l,2\*l/3)

6) Now use the bagging function to create a classification model using the sampled dataset. In the following example, we use 15 regression trees (mfinal=15) with a maximum depth of 5 (maxdepth=5) and a minimum number of records for an attribute to split is 15 (minsplit=15).

> Vehicle.bagging <- bagging(Class ~.,data=Vehicle[sub, ], mfinal=15, control = rpart.control(maxdepth=5, minsplit=15))

7) Use the model that you just created to test the prediction accuracy.

> Vehicle.bagging.pred <- predict.bagging(Vehicle.bagging,newdata=Vehicle[-sub, ], newmfinal=10)

8) View the resulting confusion table

Vehicle.bagging.pred$confusion

9) View the resulting prediction error

Vehicle.bagging.pred$error

10) Experiment with different values of mfinal, maxdepth, and minsplit to see if you can increase the accuracy of the model.

**Boosting**

1) Now let’s use boosting to classify the same Vehicle dataset and see if we get similar results. Here we will use the boosting function with 15 regression trees with the same parameters as above.

> Vehicle.adaboost <- boosting(Class ~.,data=Vehicle[sub, ],mfinal=15, coeflearn="Zhu",

control=rpart.control(maxdepth=5, minsplit=15))

2) Now test the prediction accuracy of the boosted model.

> Vehicle.adaboost.pred <- predict.boosting(Vehicle.adaboost,newdata=Vehicle[-sub, ])

3) View the resulting confusion table

> Vehicle.adaboost.pred$confusion

4) View the resulting prediction error

> Vehicle.adaboost.pred$error

<http://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Classification/SVM>

<http://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Packages/nnet>

https://cran.r-project.org/web/packages/adabag/adabag.pdf