Proposal1, using SVM for prototype selection (Wine Dataset)

Huseyin Coskun/Ruben Garzon

15 Nov 2014

About this document - WORK IN PROGRESS

This document describes the exploratory analysis done with the Wine dataset.

Exploratory Analysis

We will first read the Dataset obtained from UCI ML repository.
https://archive.ics.uci.edu/ml/datasets/Wine

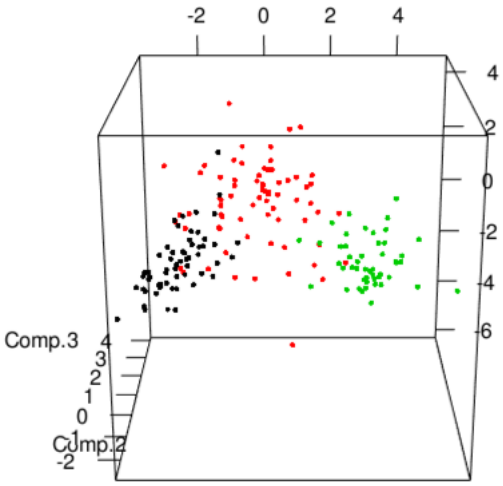
```
#Read the datasets
wine <- read.csv("../Datasets/Wine/wine.data", header=FALSE)
colnames(wine) <- c("class","Alcohol","Malic Acid","Ash","Alcalinity of Ash","Magnesium","Total Phenols
#Load required libraries
#Libraries
library(rgl)
library(caret)
```

```
## Loading required package: lattice
## Loading required package: ggplot2
```

```
library(e1071)
library(caret)
library(MASS)
```

We can plot the data with PCA to explore the distribution of classes.

```
# Log transform, just for plotting with PCA
log.wine <- log(wine[, 2:14])
# apply PCA - scale. = TRUE is highly
# advisable, but default is FALSE.
wine.pca <- princomp(log.wine,
                     center = TRUE,
                     scale. = TRUE,cor=TRUE,scores=TRUE)
#summary(wine.pca)
#plot(wine.pca, type = "l")
plot3d(wine.pca$scores[,1:3], col=wine[,1])
```



We define an auxiliar function to compute Euclidean Distance between samples and prototypes.

```
#Compute distance between samples and each prototype
computeEuclideanDissimilarities <- function (sampMatrix,prototypesMatrix)
{
        distances <- as.matrix(dist(rbind(sampMatrix,prototypesMatrix),method="euclidean"))
        elements <- nrow(sampMatrix)*nrow(prototypesMatrix)
        dissimMatrix<-distances[1:nrow(sampMatrix),(nrow(sampMatrix)+1):(nrow(sampMatrix)+nrow(prototyp
        return (dissimMatrix)
}
```

We create the core function for our analysis. Given a train and test set, and a number of prototypes do:
- Fit an svm on the trainSet. We first used svm using linear and rbf kernels. Finally we used tune.svm, which is supposed to adjust the svm parameters for us (we should investigate more on this).
- Choose the prototypes. We first choosed the prototypes among the SV, but we later realized that choosing them among the non-SV made more sense, as they are further from the classification boundary, and therefore should be more representative of the classes. We will anyway work with both options and compare them.
- Once we have the prototypes, we compute the dissimilarity Matrix, in this case using Euclidean distance.
- Fit a QDA on the dissimilarity trainset space.
- Predict with the fitted QDA on the test set, and we compute accuracy and classification error.

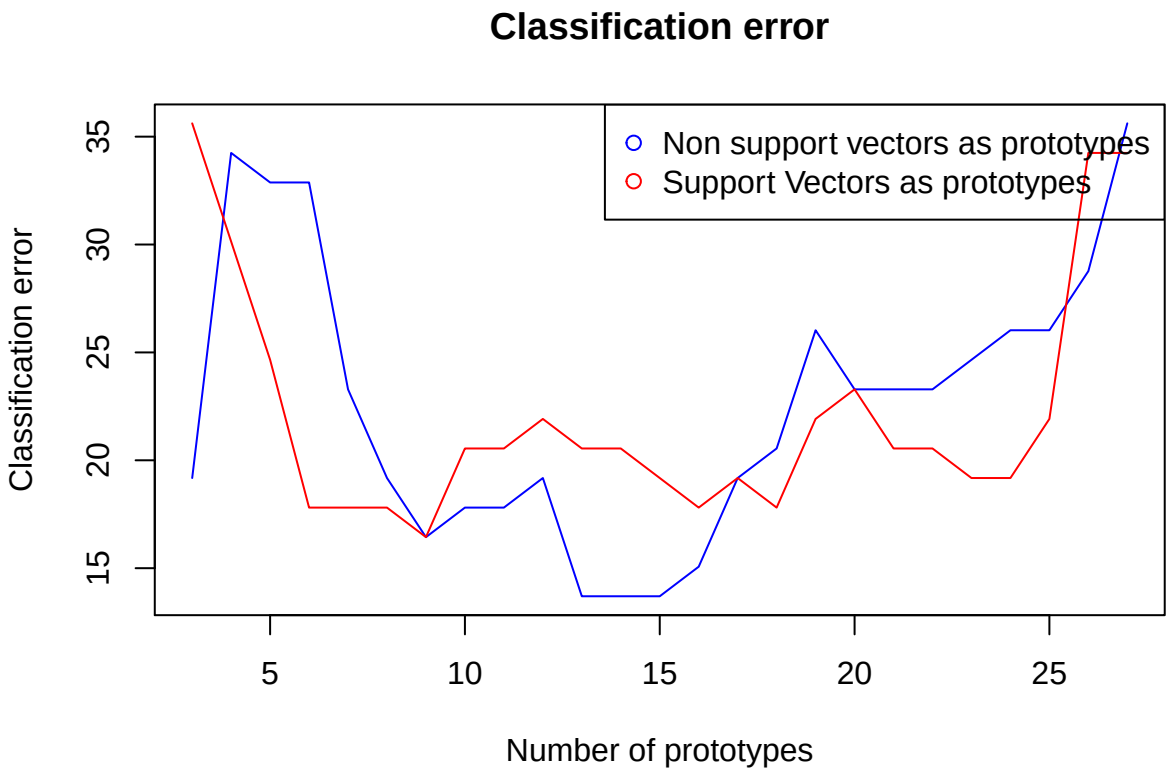
```
#Run analysis, basically:
#Take the train and test set (they are generated outside this function)
#Fit an SVM
#Choose the prototypes
#Compute dissimilarity matrix
#Fit QDA with dissimilarity matrix
#Compute accuracy on the test set
runAnalysis <- function(numPrototypes,useSV)
{
#         print(c('Iterating with ',numPrototypes,' prototypes'))

         svmfit <- tune.svm(class~.,data=trainingSet)
         svmfit$best.model
        if (useSV == TRUE)
        {
                #Here we use the support vectors as prototypes
```

```
#                print ("Using support vectors")
                prototypes <- svmfit$best.model$ind
        }
        else
        {       #Here we use the non-support vectors as prototypes
#                print ("Using non-support vectors as prototypes")
                allSamples<- 1:nrow(wine)
                prototypes <- allSamples[-svmfit$best.model$ind]

        }

        prototypes<-prototypes[1:numPrototypes]
        prototyp<-trainingSet[prototypes,]
        trainSetDissimilarities <- computeEuclideanDissimilarities (trainingSet[,-1],prototyp[,-1])

        dissSpace<-as.data.frame(cbind(trainingSet$class,trainSetDissimilarities))
        colnames(dissSpace)[1]<-"class"
        qda.fit <-qda(class~.,data=dissSpace)

        testSetDissimilarities <- computeEuclideanDissimilarities (testSet[,-1],prototyp[,-1])
        testSetDissSpace <- as.data.frame(cbind(testSet$class,testSetDissimilarities))
        colnames(testSetDissSpace)<-colnames(dissSpace)
        qda.testpred <- predict(qda.fit, testSetDissSpace)
#        print(table(qda.testpred$class,testSet$class))
        cf<-confusionMatrix(qda.testpred$class,testSet$class)
        acc <- cf$overall['Accuracy']
#        print(acc)
        return(acc)
}
```

Once we have the core analysis function, we will apply it to the WINE dataset.
First we generate the train and test set. The paper reports alpha=0.6 (train/test) per class.

Then we run the analysis for different number of prototypes and both for SV and for non-SV as prototypes.
We finally plot the classification errors obtained in both cases.

```
#Now we will run the analysis for different number of prototypes
#Using the support vectors as prototypes and the non-support vectors as prototypes
#Finally plot the classification errors obtained for both cases
protoRange <- 3:27
accuraciesNonSV <- lapply(protoRange,runAnalysis, useSV=FALSE)
plot(protoRange,100*(1-as.numeric(accuraciesNonSV)),type="l",col="blue",main='Classification error',ylab
#max(as.numeric(accuraciesNonSV))

accuraciesSV <- lapply(protoRange,runAnalysis,useSV=TRUE)
lines(protoRange,100*(1-as.numeric(accuraciesSV)),type="l",col="red",main='Classification error',ylab='
legend("topright", legend =c("Non support vectors as prototypes","Support Vectors as prototypes"), col=
```




```
#max(as.numeric(accuraciesSV))
```

Questions

Question 1:

Maybe the complete approach is incorrect. Should we apply the SVM into the dissimilarity space? We don't think so.

Question 2:

We plotted the dataset with PCA. By checking the shape of the clusters (classes), we decided to use a non-linear kernel. Does it make sense? (Finally we used tune.svm)

Question 3:

We initially selected a subset of the Support Vectors as prototypes (random subset increasing 3, 4,5 ...,) to reproduce the work in the paper. We then thought that probably it makes more sense to choose the samples that are not support vectors, as they are probably more representative of groups of samples.

Question 4:

In the paper, they use alpha=0.6 for training/testing distribution, per class. This results in a dissimilarity space with around 30 samples (for one class) and Nprototype features. We cannot fit QDA with more than 26 prototypes due to the limited number of samples in some of the classes. How did they reach a graphic

from 3 to 54 prototypes on the Wine dataset? Our QDAs fail with more than 27 prototypes. Is qda doing some kind of CV? How did they compute 54 prototypes with 35 samples? 0.6*59=35,4

Question 5:

In the paper they talk about Classification error. We assumed class. error as (1-Accuracy). Is this correct?

Question 6:

Do we need to scale variables in some moment? SVM seems to scale them for us, but we should check if needed. Should we standarize when computing dissimilarity matrix?

Question 7:

We get different results due to the random sampling. The correct approach would be do 10-fold validation (per class) and compute average of accuracies obtained?

Question 8:

We compared against a QDA on the input sample space. We got 100% Accuracy. Is it an easy dataset? Something wrong?

Pending/Future work:

- Standarization/Normalization in the different steps.
- Analyse svm tuning parameters in detail.
- Analyse qda tuning parameters in detail.
- Try different prototype selection subsets (now we just use sequencial selection among the SVs or the non-SVs)
- Another improvement could be done by plotting the dissimilarity space with PCA, trying to see if the decision boundaries in that space can be linear or not.