# STA 250 Project Report

Yilun Zhang*

3/20/2014

## 1 INTRODUCTION

We've talked in class that multi-class classification can be reduced into a binary classification problem using the strategy called "One vs One" and "One vs All" Strategies. Professor says they are good ways to be implemented on SVM.

In "One vs One" strategy, in each time, we use two groups to be the training data and built $(g-1)g/2$ models. And then use these model to predict the test set. At last we take majority vote to determine which group the observations belong.

The "One vs All" strategy is similar to "One vs One", but built classifier on one group and all rest groups each time. Then use this $g$ groups to take majority vote.

An idea comes to me that the "One vs One" may improve the classification. Considering in a KNN classification, the "close" groups intend to claim a large k as number of neighbors. While the groups differs from each other will claim a small k. In some multi-class classification case, some of the groups are closes and others are far away. Then choosing a uniform k to classify all groups seems to be unwise. Therefore if we build $(g-1)g/2$ different models to the groups. Use cross-validation to find "best" model of each pair of groups. It's seems reasonable that this will improve the prediction than fit a uniform model on all of groups.

*lunge111@gmail.com

I use the Digit Recognizer data from Kaggle competition.[1]. The data are hand-writing digits. Every row stand for a image. Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.

Here I will try to classify 1, 3, 7, 8. Because 1 and 8 are very different,1 and 7, 3 and 8 are similar. Therefore, intuitively, different k should be used when we do KNN. And we will see whether this improves compared with use uniform k in all groups. KNN is very slow, so I will use parallel package and write own function to calculate distance function once during cross validation.

## 2 Cross-validation to determine k's

First combine 2 groups to be train set and on each training data set doing 5-folds cross-validation(with inverse distance weight). Below is a table of cross-validation error and corresponding k's. The columns are errors of 13, 17, 18, 37, 38, 78. There is an interesting feature that in every group the error of k=1 and k=2 are same, this is because when using inverse distance weight, the 1 and 2 nearest neighbor will always conclude the nearest one. According to the cross-validation errors, we would use k=(1,8,1,1,4,1) from groups 13, 17, 18, 37, 38, 78.

1,7 and 3,8 claims larger k as expected.

| | ks | errs | errs.1 | errs.2 | errs.3 | errs.4 | errs.5 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0.003753685 | 0.009678513 | 0.01342013 | 0.008700342 | 0.03324826 | 0.006913604 |
| 2 | 2 | 0.003753685 | 0.009678513 | 0.01342013 | 0.008700342 | 0.03324826 | 0.006913604 |
| 3 | 3 | 0.006570602 | 0.010598053 | 0.01437937 | 0.010637383 | 0.03021797 | 0.008886486 |
| 4 | 4 | 0.006102218 | 0.010600177 | 0.01485784 | 0.010151951 | 0.02669277 | 0.008392653 |
| 5 | 5 | 0.007977958 | 0.010138283 | 0.01821402 | 0.011598899 | 0.03223689 | 0.009875363 |
| 6 | 6 | 0.007507370 | 0.010138283 | 0.01630015 | 0.010150787 | 0.02921040 | 0.009381536 |
| 7 | 7 | 0.009385315 | 0.010600177 | 0.01773555 | 0.012084330 | 0.03374443 | 0.010865456 |

[1]http://www.kaggle.com/c/digit-recognizer/data

```
8   8 0.008448547 0.009678513 0.01677632 0.011118141 0.03172804 0.009875357
9   9 0.008916931 0.010598053 0.02012790 0.012569755 0.03173438 0.011360506
10 10 0.008916931 0.010138283 0.01869249 0.012086664 0.03172423 0.011853116
11 11 0.011733848 0.011977364 0.02204407 0.014985226 0.03173057 0.012841993
12 12 0.011733848 0.011517593 0.02204407 0.014019036 0.03172677 0.011360506
13 13 0.012202232 0.013819627 0.02348178 0.014503298 0.03223563 0.012841993
14 14 0.012202232 0.013359857 0.02348178 0.014503298 0.03223056 0.012348166
15 15 0.014077972 0.015203185 0.02443872 0.015469487 0.03324192 0.013832092
```

# 3 Prediction and Vote

Then I use k's render lowest cross-validation error in each group to do binary KNN classification. And then take majority vote. And use k=1 to do multi-classes KNN. The confusing matrix based on test data is shown below. We see the result are indeed improved.

```
#multi-class
y      1    2    3    4
  1 1089    0   16   14
  2    1 1021    1   29
  3    3   11 1047    8
  4    0   21    0  911
```

```
#One vs One
pred    1    2    3    4
   1 1092    0   13   12
   2    1 1039    1   20
   3    3   11 1047    8
   4    0   12    0  913
```

# 4 Summary

"One to One" Strategy does improve the prediction. Moreover, when computer distance matrix, it needs $\frac{n^2-n}{2}$ calculations which is the same to do in multi-classes KNN. And this strategy is very easy to parallel.

If we widen the bound of cross-validation, different classification models can be applied on groups. We can try to find classifier best distinguishes two groups.

However, when using "One to One" Strategy we build $(n-1)n/2$ models. This can introduce more variance, especially when the data size is small. And when an algorithm is better implemented with "global" features(like LDA), "One vs One" seems not applicable.

I also considered on "One vs All" strategy. Doing "One vs All" is much more computational intensive than "One vs One", especially when the number of groups is large. When doing "One vs All", the feature of the rest group is averaged from n-1 groups, so "One vs All" may be not a good method to improve classification. The good thing is that "One vs All" use all data to build model for each model.

# 5 CODE

```
setwd("/media/Windows8_OS/14winter/250/project/")
train=read.csv("train.csv")
test=read.csv("test.csv")
library(parallel)
i=sample(1:42000,10000)
training=train[i,]
test=train[-i,]
j=sample(1:32000,10000)
test=test[j,]
#subset into 4 group
one=subset(training,label==1)
eight=subset(training,label==8)
```

```
three=subset(training,label==3)
seven=subset(training,label==7)
one$label=1
three$label=2
seven$label=3
eight$label=4
head(one)
training=rbind(one,three,seven,eight)
testing=subset(test,test$label%in%c(1,3,7,8))
testing[testing$label==3,]$label=2
testing[testing$label==7,]$label=3
testing[testing$label==8,]$label=4

##save n(n-1)/2 pairs to a list, they are:
#[1] 1 2
#[1] 1 3
#[1] 1 4
#[2] 2 3
#[2] 2 4
#[3] 3 4
pair=list();k=1
for(i in 1:3)
  for(j in (i+1):4){
    pair[[k]]=c(i,j);k=k+1;print(c(i,j))}
cl <- makeCluster(4,"FORK")
v=clusterSplit(cl,pair)


##find k nearest neighbor
findknn<-function(matr,k){
```

```
  #given  a distance matrix matr, find k-nn
  a=apply(matr,1,function(x) data.frame(num=order(x)[1:k],dis=x[order(x)[1:k]]))
  a
}
KNN<-function(matr=NULL,testdat,traindat,k,
              trainlabel,dismethod=c("euclidean","manhattan","mahalanobis")){
  require(StatMatch)
  if(is.null(matr)){
    if(dismethod=="mahalanobis")
      matr=mahalanobis.dist(testdat,traindat)
    if(dismethod=="euclidean")
      matr=as.matrix(dist(rbind(testdat,traindat)))[1:nrow(testdat),
                                                    -(1:nrow(testdat))]
    if(dismethod=="manhattan")
      matr=as.matrix(dist(rbind(testdat,traindat),method="manhattan"))[1:nrow(testdat
                                                                          (nrow(testdat)
  }
  knn=findknn(matr,k)
  sapply(knn, function(x,trainlabel) {count=trainlabel[x$num]
                                      as.integer(names(sort(tapply(1/x$dis,count,sum)
         ,trainlabel=trainlabel)
}


#cross validation
kfoldCV<-function(traindat,nfold,k=1:15,trainlabel,dismethod=c("mahalanobis","euclide
{require(StatMatch)
 random=sample(1:nrow(traindat))
 group=split(random,cut(1:nrow(traindat),nfold))
 if(dismethod=="mahalanobis")
```

```
   dmatr=mahalanobis.dist(traindat)
 if(dismethod=="euclidean")
   dmatr=as.matrix(dist(traindat))
 if (dismethod=="manhattan")
   dmatr=as.matrix(dist(traindat,method="manhattan"))
 a=k
 for(i in k){
   err=sapply(group, function(x,dmatr,label){
     pred=KNN(matr=dmatr[x,-x],k=i,trainlabel=label[-x])
     sum(pred!=label[x])/length(pred)
   },dmatr=dmatr,label=trainlabel)
   a[i]=mean(err)
 }
 list(k=which.min(a),minerr=min(a),df=data.frame(ks=k,errs=a))
}
#put 2 group into a subset
# use 5 folds  cross validation to find the best k for each pair
#you may not want to run this ,very slow
CV=clusterApply(cl,v,function(x,trian,nfolds)
  sapply(x,function(t,train,nfolds) {
    dat=train[train$label==t[[1]]|train$label==t[[2]],]
kfoldCV(traindat=dat[,-1],nfold=nfolds,trainlabel=dat$label,dismethod="euclidean")
},
train=train,nfolds=nfolds),train=training,nfolds=5
)



#predict the testing, save the result in a list
#taking very long time
```

```
Binaryfit<-function(train, test, trlabel,ngroup=4,
                 seq=c(1,8,1,1,4,3))
{predlist=list()
 pred=list()
 p=1
 for(i in 1:(ngroup-1)){
   for(j in (i+1):ngroup){
     pred[[j]]=KNN(testdat=test,
  traindat=train[train$label==i|train$label==j,][,-1],k=seq[p],trainlabel=train$label
  dismethod="euclidean");p=p+1}
   predlist[[i]]=pred}
 predlist
}
m=Binaryfit(train=training,test=testing[,-1])

a
#vote
mt=matrix(rep(0,6*4172),ncol=6)
p=1
name1=rep(0,6)
for(i in 1:3)
  for(j in (i+1):4){
    mt[,p]=sapply(m[[i]][[j]],function(x) ifelse(x==i,1,-1))
    name1[p]=paste(as.character(i),as.character(j))
    p=p+1
  }


newmt=as.data.frame(mt)
names(newmt)=name1
```

```
head(newmt)
pred=apply(newmt,1,function(x) {m=matrix(rep(0,36),6)
                                for(i in 1:5)
                    m[,i]=c(rep(0,i),x[(-0.5*i^2+6.5*i-5):(-0.5*i^2+5.5*i)])
                                m=t(m+t(-m))
                                which.max(apply(m,1,sum))
})
y=KNN(testdat=testing[,-1],traindat=training[,-1],
      trainlabel=training[,1],k=1,dismethod="euclidean")
table(y,testing[,1])
table(pred,testing[,1])
```