Lab 2: Basic Classification Algorithms

First Name: Bill **Last Name:** Zhou **NetID:** wz82

Lab 2 will be graded. Upload your work to Gradescope by Feb. 18th at 11:59pm. Make sure to upload a single .pdf containing your answers and the R code you wrote to obtain them. Regrade requests should be made via Gradescope.

In this lab we will learn the basics of logistic regression and the k-nearest neighbor algorithm, and learn how to measure the accuracy of the models we build.

Part (a): Logistic regression

Go to Canvas and download the `Titanic` data set.

The Titanic data set contains information(age, gender, passenger class etc.) about 1309 passengers that travelled on the ship Titanic and whether or not they survived. We will fit a logistic regression model to predict what sorts of passengers were more likely to survive.

We will divide the `Titanic` data set into 2 parts: the training data set and the test data set. We will fit the model and learn the parameters with the training set, and then test the accuracy of our model ***using the test set*** by comparing the predicted outcomes of the data in the test set using our model and their true outcomes. The training and test sets must be created from the original data without bias. In this lab, we'll randomly sample 2/3 of the original data to be the training data and let the rest be the test data. Why must we do to calculate the accuracy for the test set?

Why → What

Train-Test split ⇒ train training data under some hypothesis to get model
→ Plug in testing X data to get prediction values ⇒ Compare with real Y in testing set

Load the titanic data set into R. There are 11 variables in the data set except the outcome variable. We suspect that `Passenger ID`, `Name`, `Ticket` and `Cabin` are not useful in predicting survival chance so we can remove them. In addition, the `Cabin` column has lots of missing data. Check if there is still any other missing data in the data set. You'll find the `Age` column also has many rows where the data is missing. Can you think of two approaches to deal with these rows?

① fill the blank cell by the average of that column
② by the average of #3 neighbors
"③" delete Age column

After figuring out how to deal with missing data, we need to check if all categorical variables have been identified by R and coerce them if not. Finally, we can create the training and test sets. Note that you need to store the outcome value of the test set separately and remove the outcome column from the test set.

Now you can fit the logistic regression model to the training data. Report the summary of your model. Once you have fitted the model, use your model to predict the outcomes of the test data. Since 'Survived' is already binary valued, you do not need to coerce the 'Survive' column to a factor. Also DO NOT remove the "Survived" column from df.test.
(Note that the program may reports error when using predict function due to some factor levels only being present in the test samples. In this case, you can do one of the following:
- remove test samples containing the offending factors
- do not coerce the variable to factor
- retry with a new random seed until you get a split that works, and keep that seed
Depends on the split of your data set, you may or may not encounter this issue. see also: https://www.r-bloggers.com/linear-logistic-regression-in-r-dealing-with-unknown-factor-levels-in-test-data/)

The logistic regression model we built will output the probability of surviving of each passenger in the test data set. To get a binary prediction of whether a passenger survives or not, we need to pick a threshold. For now we simply predict a passenger would survive if the output survival probability is greater than or equal to 0.5. What is the (overall) accuracy of your model? In addition, report the true positive and true negative rates.

Coefficients
$\beta_{Pclass} = -1.047$
$\beta_{sexmale} = -2.673$
$\beta_{Age} = -0.042$
$\beta_{SibSp} = -0.375$
$\beta_{Parch} = -0.116$
$\beta_{Fare} = 0.0039$
$\beta_{EmbarkC} = -12.33$
$\beta_{EmbarkQ} = -12.23$
$\beta_{EmbarkS} = -12.76$
Intercept = 17.54

Overall accuracy = 0.801

Real \ Pred	F	T
F	158	37
T	22	80

TN FN
FP TP

$TPR = \frac{TP}{TP+FN} = \frac{80}{117}$

$TNR = \frac{TN}{TN+FP} = \frac{158}{180}$

Now that you know how to measure accuracy, can you find a very simple classifier that would give you a decent overall accuracy (about 65% or more) by just looking at the data?

if that person is male, predict died
if that person is female, predict alive
We have 80.4 by using the algorithm above, which is appropriately high

As you can see, the data is biased. As the data gets more biased, our formula for overall accuracy becomes less indicative of how good our model is. A way to solve this is to use the following formula:

$$\texttt{balanced\_accuracy} = \frac{1}{2} \times \frac{t_p}{(t_p + f_n)} + \frac{1}{2} \times \frac{t_n}{(t_n + f_p)}$$

Using this formula, calculate the balanced accuracy of our logistic regression model on the test dataset.

b_a = 0.781

Identify which predictors are most influential to the final prediction.

Gender, Pclass = 3, Fare,

Part (b): K Nearest Neighbors

We will continue analyzing the `Titanic` data set, but with a different classification method. In particular, We will learn how to do KNN in R, and compare the performance of KNN with the logistic regression approach used in part (a).

Do the same procedures to the Titanic data set as we did in part(a), i.e.

1. Delete columns `Passenger ID`, `Name`, `Ticket` and `Cabin`.
2. Deal with missing data by removing all rows associated with a missing value (alternatively, you may replace missing values with mean/median).
3. Check for factor variables and convert them into dummy variables.

Note that we did not need the third step in part(a). Neither KNN nor linear/logistic regression deals with factor variables directly. The `lm()` and `glm()` functions in R automatically convert factors to dummies, but the built-in function `knn()` for KNN does not.

Can you think of the reason why KNN cannot deal with factor variables directly?

It's about the scale, If the scale is to large for that dimension, then the rest predictors will be useless

To convert factor variables into dummy variables, we use the function `model.matrix()`.

```
> titanic$Sex <- model.matrix( ~ Sex - 1, data=titanic )
> titanic$Embarked <- model.matrix( ~ Embarked - 1, data=titanic)
```

Why don't we just use the `as.numeric()` function to do the conversion?

It won't return binary result
and won't be factor-data type

We will divide the Titanic data set into training and test sets, again by randomly sampling 2/3 of the data to be the training set and using the rest as the test set. We will set a random seed so the the sampling is reproducible.

```
> set.seed(1)
> train_ind <- sample(1:nrow(titanic), 2/3*nrow(titanic))
```

We will now perform KNN using the `knn()` function, which is part of the `class` library. This function works rather differently from the other model fitting functions that we have encountered thus far. Rather than a two-step procedure in which we first fit the model and then we use the model to make predictions, the `knn()` function computes the predictions directly in a single command. Running `?knn` to display the help page, we see that the function requires four inputs:

1. A matrix containing the predictors associated with the training data, labeled `titanic.train` below.
2. A matrix containing the predictors associated with the data for which we wish to make predictions, labeled `titanic.test` below.
3. A vector containing the outcomes (class labels) for the training observations, labeled `train.survive` below.
4. A value for k, the number of nearest neighbors to be used by the classifier.

Before applying the `knn()` function, we normalize each predictors to have unit variance. Here we use function `scale()` to make each variables to have 0 mean and variance 1.

```
> titanic_normal <- scale(titanic[,!names(titanic) %in% 'Survived'])
> titanic.train <- titanic_normal[train_ind, ]
> titanic.test <- titanic_normal[-train_ind, ]
```

Why do we need this step?

Because we want every predictor to be equally important in our n-dimension decision boundary generating problem

We next create two variables that store the outcome values of the training/test sets.

```
train.survive = titanic$Survived[train_ind]
test.survive  = titanic$Survived[-train_ind]
```

We can now apply the `knn()` function to predict the survival of the passengers. Note that in KNN, if several observations are tied as nearest neighbors, R will randomly break the tie.

```
> knn.pred = knn(titanic.train, titanic.test, train.survive, k=1)
> table(knn.pred, test.survive)
> mean(knn.pred == test.survive)
```

Here we choose $K = 1$, which gives an error rate of about 23% (according to my experiment).

76.4 accuracy

We explore the performance of KNN with other values of K. Run KNN for $K = 1, 3, 5, 10, 20, 50$. (You may use a for loop for convenience). Report the corresponding error rates, and find the value of K that gives the lowest error rate.

K	1	3	5	10	20	50
acc	0.764	0.744	0.749	0.788	0.784	0.794

Thus K=10 has relative low error rate, around 20%

Finally, compare the prediction performance of KNN (using your optimal K) with the logistic regression model from part(a). Which method gives a better accuracy rate in this case?

Logistic Regression

Take Home Questions

(1) We introduced *balanced accuracy* in Part (a). Is balanced accuracy the same as overall accuracy? Prove or give a counterexample.

No, As the confusion matrix below

Pred \ True	T	F
T	81	9
F	9	1

Overall_accuracy $= \frac{82}{100}$

balanced_accuracy $= \frac{1}{2} \cdot \frac{81}{90} + \frac{1}{2} \cdot \frac{1}{10} = 0.5$

, not the same

(2) Give an example of a scenario when the balanced accuracy provides a better measure than the overall accuracy.

Pred \ True	T	F
T	9	1
F	81	9

Overall: 18%

Balanced: 50%

(3) Recall in Lab 1 you tried to get the units of the ω coefficients of the linear regression model. What are the units of the coefficients of the logistic regression model.
(Hint: Recall the logistic regression model assumes:

$$\mathbb{P}(y = 1|\mathbf{x}, \beta) = \frac{1}{1 + \exp(-\beta^T \mathbf{x})}$$

Try to find an expression $f(p)$ of $p = \mathbb{P}(y = 1|\mathbf{x}, \beta)$ which satisfies $f(p) = \beta^T \mathbf{x}$. We call $f(p)$ "logit" or "log-odds". Recall the interpretation of β coefficients in the linear regression model is the change of outcome variable when a predictor changes by 1 unit. Interpret β coefficients of the logistic regression model in a similar way.)

$\beta_i x_i$ should be unitless, Thus β_i's unit is 1/ x_i's unit

For every one year of age, a passenger's logit of surviving on Titanic will decrease (or increase) by a factor of what?

-0.042 per year older

(4) Go to Canvas and download the Cancer data set: `pros.xls` and `pros_meta.txt`. We want to build a logistic regression model to predict whether the tumor has penetrated the prostatic capsule.

Load the data into R. Note the 'VOL' column has lots of missing data and you need to remove these instances first. We will fit a logistic regression model with the data and test the accuracy of our model.
Divide the data set into training and test data. Fit a logistic regression model with the training data. Report the summary of your model.

$\beta_{AGE} = -0.025$	$\beta_{PROS} = 0.5101$	$\beta_{PSA} = 0.022$	Intercept $= -6.34$
$\beta_{RACE} = -0.6973$	$\beta_{DCAPS} = 0.6443$	$\beta_{GLEASON} = 0.9229$	

(5) Now let's check the accuracy of your model in Q2. Remember the logistic regression model outputs a probability of penetration p for each patient. We could pick a threshold

p_0 so that we predict the outcome to be 1 if $p \geq p_0$. Hence different p_0 gives different classifiers.

Define FPR $= \frac{f_p}{f_p+t_n}$ to be the false positive rate: the proportion of negative instances that are falsely predicted to be positive instances. Moreover, define TPR $= \frac{t_p}{t_p+f_n}$ to be the true positive rate: the proportion of positive instances that are also predicted correctly. Each p_0 gives a new pair of (FPR, TPR).

Pick a list of p_0 and plot the corresponding (FPR, TPR) pairs on a graph. We obtain the FPR vs. TPR plot (i.e., the ROC curve) for our logistic regression model as shown in Figure 1. (If you generate this plot yourself, it will look slightly different due to the randomness in splitting in to training and test datasets. But the general trend of the curve will be the same.)

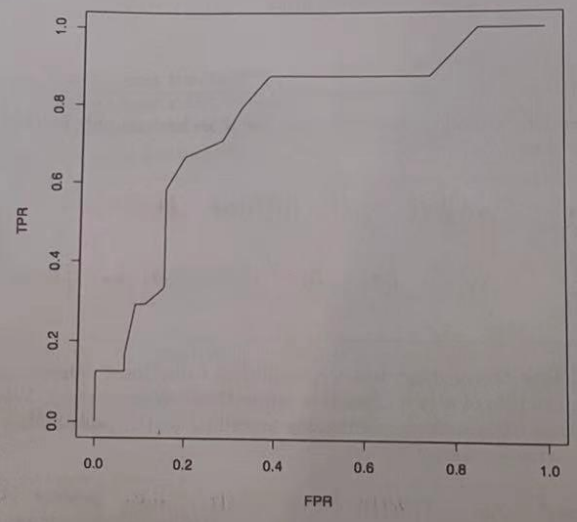


Figure 1: ROC curve: False Positive Rate vs. True Positive Rate

Next, assume we have a balanced data set, i.e. with 50% positive instances and 50% negative ones. What would the FPR vs. TPR plot look like for the following two classifiers:

(a) You were guessing the prediction randomly. (a) FPR and TPR will always be around Thus the plot will be several dots shatter around (0.1, 0.1

(b) You were predicting only one of the classes for everything.

5a. TPR & FPR will always be around 50%
in this case
Thus the plot will be several point stuck around
(0.5, 0.5)
(b) will be a straight line
TPR = 0.5

What would the plots in 5a and 5b look like if we have an unbalanced data set, say $p\%$ positive class?

5a: points will gather around (P, P)
5b will be another straight line TPR = P

(6) In predicting cancer diagnosis, we would like to be more conservative and prefer to error on the side of a false diagnosis rather than missing cancer. What would you do in this case when making predictions according to the probabilities output by your logistic regression model?

Decrease the threshold so that more patient will be classified as cancer, to reduce FN

More on Logistic Regression

1. Will the fitted coefficients and the classification change if we duplicate the whole training set while using the logistic regression? What happens if we only duplicate a subset of the training set? Give intuitive explanations and veriy your result using the `Titanic` data set.

divide error

practice, when we perform the classific... $g(i) = 0$. Explain briefly why this step ...

No, if we duplicate everything by 2, then β^* will be half of original β

Also, No, if we duplicate X_i by 2, then β_i^* will be $\frac{\beta_i}{2}$, to diminishing the duplication effect

2. Duplicating a subset of the training set may be used to fix an imbalanced data set. A common approach is called the oversampling, where we resample or duplicate the minority class to balance the data set. State one potential limitation of the oversampling method.

It can possibly cause overfitting

3. Recall from the lecture that the logistic regression can be extended to the multi-class classification tasks. Consider the following version of the `One-Versus-All` mathod. Suppose we have K classes, for each class i where $0 \le i \le K-1$, we perform a logistic regression and determine the weights $\beta_0^{(i)}$ and $\beta^{(i)}$. We define

$$\mathbb{P}(y = i \mid \mathbf{x}, \beta) = \frac{e^{\beta_0^{(i)} + \beta^{(i)} \cdot \mathbf{x}}}{\sum_{j=0}^{K-1} e^{\beta_0^{(j)} + \beta^{(j)} \cdot \mathbf{x}}} \tag{1}$$

In the case of binary classification, i.e. $K = 2$, how does the predicted probability using `One-Versus-All` differ from the original logistic regression?

They will return different coefficients, however, the classification result should be the same

Recall that logistic regression for 2 classes has

$$\mathbb{P}(y = 1 \mid \mathbf{x}, \beta) = \frac{1}{1 + e^{-(\beta_0 + \beta \cdot \mathbf{x})}}$$

What are the relationships between β_0, β, $\beta_0^{(i)}$ and $\beta^{(i)}$ in the case of binary classification?

$\frac{e^{\beta_0^0 + \beta^0 x}}{e^{\beta_0^1 + \beta^1 x} + e^{\beta_0^0 + \beta^0 x}} = \frac{1}{1 + e^{-(\beta_0 + \beta \cdot x)}}$ Thus $\frac{1}{1 + e^{(\beta_0^0 - \beta_0^1) + (\beta^0 - \beta^1)x}} = \frac{1}{1 + e^{-\beta_0 + \beta \cdot x}}$

divide $e^{\beta_0^1 + \beta^1 x}$

$\beta_0 = \beta_0^1 - \beta_0^0$, $\beta = \beta^1 - \beta^0$

In practice, when we perform the classification using (1), we will fix a class i as the base class and set $\beta^{(i)} = 0$. Explain briefly why this step is performed. Is the choice of i important for classification?

To reduce the computation workload of $\beta \cdot X$
The choice of i does not matters
Because $P(y=i \mid X, \beta)$ will provide relative P for each class
and $\sum_{i=1}^{N} P(y=i \mid X, \beta) = 1$, selecting different i will only leads to different coept but same prob distribution

4. For simplicity, we consider the logistic regression with threshold 0.5. What is the formula for the classification boundary?

$y=1$ if $e^{-\beta_0 - \beta X} < 1 \Rightarrow \beta_0 + \beta X = 0$

Suppose the logistic regression gives β_0 and β as the fitted coefficients. How does the classification result change if we predict instead using the weights $c\beta_0$ and $c\beta$ for some constant $c > 0$? Will the classification change if $c < 0$.

If threshold = 0.5 then for any $c > 0$, result won't change
for any $c < 0$ result will be flipped

(Optional) Use the observation above, explain heuristically why the logistic regression can perform badly when fitted by the maximum likelihood, when the data *can* be linearly separated?

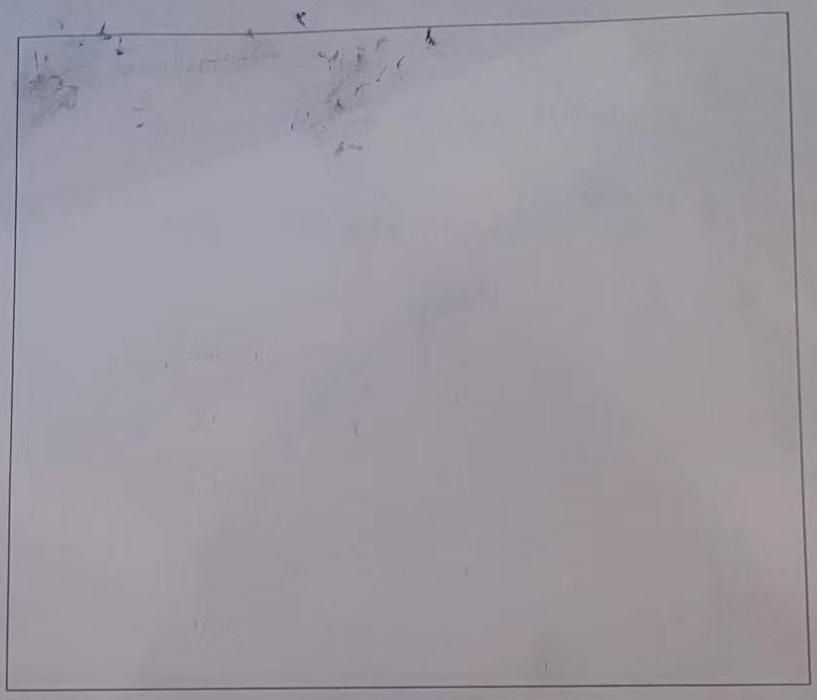
(Optional) Test this phenomenon on some simulated data in R. Give a printout of the warning messages when you run the logistic regression.

KNN

1. Compare the performance of KNN with and without normalization of the `Titanic` data set.

Create a plot of the value of K vs. the associating error rate, where $K \in [1, 20]$, with normalization. Next, make the same plot but without normalizing the data set. Compare the two plots. (You may attach additional page for printing out the 2 plots.)

What can you say about the accuracy rates with and without normalization?



2. What will be the prediction result if we set K to be the total number of training data points?

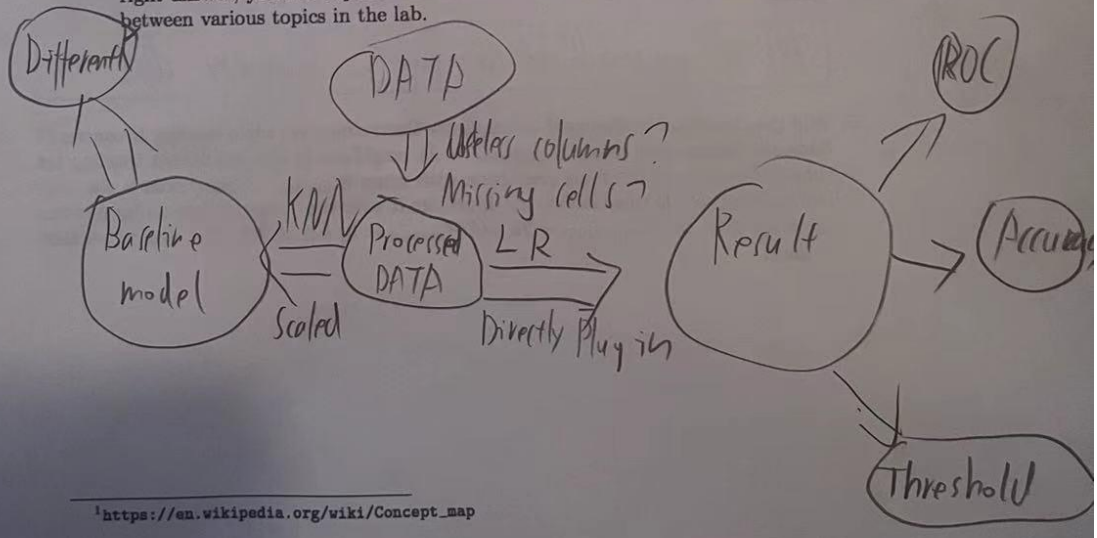
All the point will be classified as majority class

3. Will the classification change if we duplicate the training set while keeping K constant? Someone claims that using $K_{\text{duplicated}} = 2k$ neighbors in the duplicated training set should always yield the same prediction with using $K_{\text{original}} = k$ neighbors in the original training set. Is this claim true? Illustrate it with the `Titanic` data set by plotting the error rates for $K_{\text{duplicated}} = 2k$ and $K_{\text{original}} = k$, $k \in [1, 30]$. (Set `use.all=FALSE` in `knn`).

The result will change, in this case K^* is equivalent to $\frac{K}{2}$ in the original model.

The claim is also not true
As plot shown

4. Make a **concept map**[1] for the topics covered by this lab. While there is no single right answer, you are expected to demonstrate that you understand the relationships between various topics in the lab.

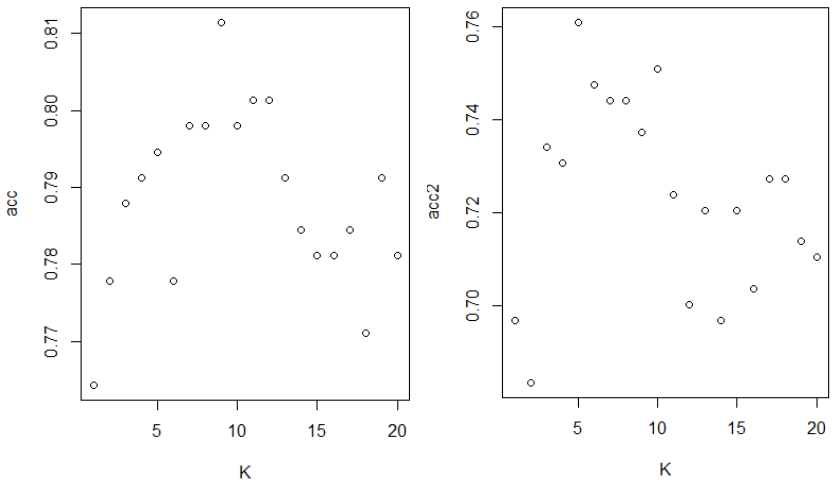


[1] https://en.wikipedia.org/wiki/Concept_map

Appendix:

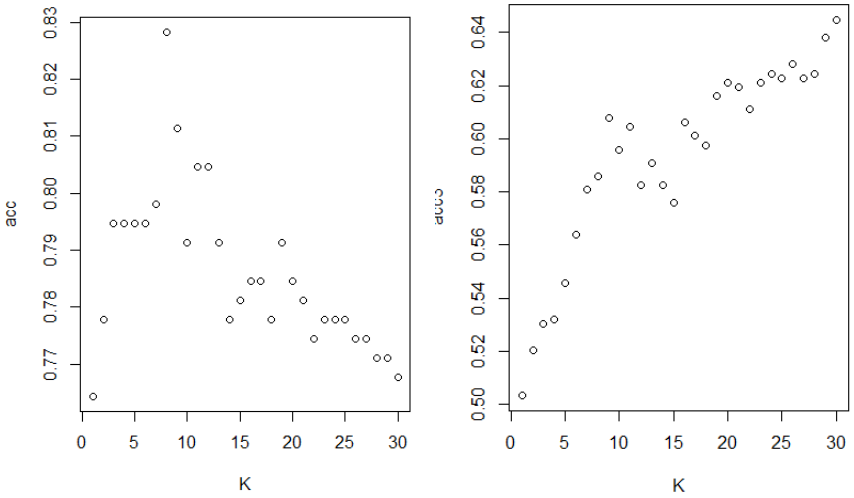
1. page 11 comparison between scaled, unscaled KNN

Scaled Unscaled



2.page 12 comparison between 2k 2n and k,n(duplicate or not) for KNN

Duplicated one Original one



R-code:

```
titanic=read.csv("~/lab2/titanic.csv")

titanic2=subset(titanic,select= -c(PassengerId,Name,Ticket,Cabin))


titanic2$Age[is.na(titanic2$Age)]=mean(titanic2$Age,na.rm=TRUE)


is.factor(titanic2$Sex)

is.factor(titanic2$Embarked)

set.seed(97376)

train=sample(1:nrow(titanic2),2/3*nrow(titanic2))

titanic2.train=titanic2[train_ind,]

titanic2.test=titanic2[-train_ind,]

glm(formula = Survived~Pclass+Sex+Age+SibSp+Parch+Fare+Embarked, data=titanic2.train,family =
binomial )


library(ROCR)


titanic2_predicted_prob=predict(titanic2_model1,titanic2.test,type='response')

titanic2_class_prediction=as.integer(titanic2_predicted_prob>0.5)

Test_true_class=titanic2.test$Survived

TP=TN=FP=FN=0

for( i in 1:297){
  if(Test_true_class[i]==0){
    if(titanic2_class_prediction[i]==0){TN=TN+1}
    else{FP=FP+1}
    }
  if(Test_true_class[i]==1){
```

```
   if(titanic2_class_prediction[i]==1){TP=TP+1}
   else{FN=FN+1}
  }
}
simple_classifier_pred=int(c(1:297))
for( i in 1:297){
  if(titanic2.test$Sex[i]=='female')
  {
    simple_classifier_pred[i]=1
  }
  else
  {
    simple_classifier_pred[i]=0
  }
}

simple_classifier_accuracy=mean(simple_classifier_pred==Test_true_class)

b_a=0.5*(TP/(TP+FN))+0.5*(TN/(TN+FP))




#KNN

#titanic2$Sex=model.matrix(~Sex-1,data=titanic2)
#titanic2$Embarked=model.matrix( ~ Embarked - 1, data=titanic2)
install.packages('fastDummies')
library('fastDummies')
```

```
titanic2=dummy_cols(titanic2,select_columns = 'Sex')
titanic2=dummy_cols(titanic2,select_columns = 'Embarked')

titanic_knn=subset(titanic2,select= -c(Sex,Embarked))

set.seed(8964)
train_knn_ind=sample(1:nrow(titanic_knn),2/3*nrow(titanic_knn))
titanic_knn=scale(titanic_knn[,!names(titanic_knn)%in%'Survived'])
titanic_knn.train=titanic_knn[train_knn_ind,]
titanic_knn.test=titanic_knn[-train_knn_ind,]
knn_train.survive = titanic$Survived[train_knn_ind]
knn_test.survive = titanic$Survived[-train_knn_ind]


knn.pred = knn(titanic_knn.train, titanic_knn.test, knn_train.survive, k=1)
table(knn.pred, knn_test.survive)
mean(knn.pred == knn_test.survive)
for (i in c(1,3,5,10,20,50)){
  knn.pred = knn(titanic_knn.train, titanic_knn.test, knn_train.survive, k=i)
  table(knn.pred, knn_test.survive)
  acc=mean(knn.pred == knn_test.survive)
  print(acc)
}

cancer=read.csv("~/lab2/pros.csv")
cancer2=subset(cancer,select= -c(VOL))

set.seed(1989)
train_ind=sample(1:nrow(cancer2),2/3*nrow(cancer2))
```

```
cancer2.train=cancer2[train_ind,]
cancer2.test=cancer2[-train_ind,]

acc=c(1:20)
acc2=c(1:20)
acc3=c(1:30)
for (i in c(1:20)){
  knn.pred = knn(titanic_knn.train, titanic_knn.test, knn_train.survive, k=i)
  table(knn.pred, knn_test.survive)
  acc[i]=mean(knn.pred == knn_test.survive)
  print(acc)
}


#train_knn_ind=sample(1:nrow(titanic_knn),2/3*nrow(titanic_knn))
titanic_knn2=subset(titanic2,select= -c(Sex,Embarked,Survived))

titanic_knn2.train=titanic_knn2[train_knn_ind,]
titanic_knn2.test=titanic_knn2[-train_knn_ind,]


for (i in c(1:20)){
  knn.pred = knn(titanic_knn2.train, titanic_knn2.test, knn_train.survive, k=i)
  table(knn.pred, knn_test.survive)
  acc2[i]=mean(knn.pred == knn_test.survive)
  #
}

plot(c(1:20),acc,xlab='K')
```

```
plot(c(1:20),acc2,xlab='K')

knn.pred = knn(titanic_knn2.train, titanic_knn2.test, knn_train.survive, k=891)
table(knn.pred, knn_test.survive)
print(mean(knn.pred == knn_test.survive))



for (i in c(1:20)){
  knn.pred = knn(titanic_knn.train, titanic_knn.test, knn_train.survive, k=i,)
  table(knn.pred, knn_test.survive)
  acc[i]=mean(knn.pred == knn_test.survive)
  print(acc)
}

install.packages("dplyr")
library("dplyr")

titanic_knn3.train=rbind(titanic_knn.train,titanic_knn.train)
titanic_knn3.test=rbind(titanic_knn.test,titanic_knn.test)

knn_train3.survive=rbind(knn_train.survive,knn_train.survive)
knn_test3.survive=rbind(knn_test.survive,knn_test.survive)
for (i in c(1:30)){
  knn.pred = knn(titanic_knn3.train, titanic_knn3.test, knn_train3.survive, k=2*i,use.all=FALSE)
  table(knn.pred, knn_test3.survive)
  acc3[i]=mean(knn.pred == knn_test.survive)
  print(acc3)
}
```

```
plot(c(1:30),acc3,xlab="K")

for (i in c(1:30)){
  knn.pred = knn(titanic_knn.train, titanic_knn.test, knn_train.survive, k=i,)
  table(knn.pred, knn_test.survive)
  acc[i]=mean(knn.pred == knn_test.survive)
  print(acc)
}

plot(c(1:30),acc,xlab="K")
```