NAIVE BAYES CLASSIFICATION

Naive Bayes Classification is a method used for binary or multinomial classification (but not a regression) that utilizes the Bayes' formula. Suppose there are k predictors $\mathbf{X} = (X_1, \dots, X_k)$ which are binary, categorical, or continuous. And let Y denote the response variable. By the Bayes' formula

$$\mathbb{P}(Y|\mathbf{X}) = \frac{\mathbb{P}(\mathbf{X}|Y)\mathbb{P}(Y)}{\mathbb{P}(\mathbf{X})}.$$

The Naive Bayes classification method assumes that the predictors are conditionally independent, given Y, that is,

$$\mathbb{P}(Y|\mathbf{X}) = \frac{\mathbb{P}(Y)\prod_{i=1}^{k}\mathbb{P}(X_i|Y)}{\mathbb{P}(\mathbf{X})}.$$

This conditional independence assumption is rather naive, hence the name of the technique.

In classification problem (binary or multinomial), we compute the conditional (posterior) probability $\mathbb{P}(Y|\mathbf{X})$ of each class, and classify the record into the class with the highest probability. Since we compare the posterior probabilities and the denominator $\mathbb{P}(\mathbf{X})$ is present in each expression, it can be ignored. That is, $\mathbb{P}(Y|\mathbf{X})$ is proportional to $\mathbb{P}(Y)\displaystyle\prod_{i=1}^{k}\mathbb{P}(X_i|Y)$ up to a multiplicative constant.

To estimate the prior probability $\mathbb{P}(Y = y)$ of each class y, we compute the proportion of observations in each class in the training set. To compute the empirical conditional probabilities $\mathbb{P}(X_i = x|Y = y)$ for categorical predictors, we calculate the fraction of observations in the class $Y = y$ in the training set for which $X_i = x$. If a predictor is continuous, we assume that the underlying distribution is normal (Gaussian) with estimated mean $\hat{\mu} = \bar{x}$ and estimated variance $\hat{\sigma}^2 = s^2$.

Characteristics of Naive Bayes Classifiers

1. Robust to outliers because they average out when computing posterior probabilities.
2. Handles missing values by ignoring the missing data points in calculations.
3. Robust to irrelevant predictors since $\mathbb{P}(X_i|Y)$ is almost uniformly distributed and factors out in comparisons of posterior probabilities.
4. Correlated predictors can degrade the performance of the technique. The conditional independence assumption is the key.

Example. Suppose the training data are as given in the table below.

ID	Home Owner	Marital Status	Annual Income ($K)	Defaulted Borrower
1	yes	single	125	no
2	no	married	100	no
3	no	single	70	no
4	yes	married	120	no
5	no	divorced	95	yes
6	no	married	60	no
7	yes	divorced	220	no
8	no	single	85	yes
9	no	married	75	no
10	no	single	90	yes

The prior probabilities are $\mathbb{P}(default = no) = 7/10 = 0.7$, $\mathbb{P}(defaul = yes) = 3/10 = 0.3$. The conditional probabilities are:

$$\mathbb{P}(homeowner = yes \,|\, default = no) = 3/7,$$

$$\mathbb{P}(homeowner = yes \,|\, default = yes) = 0,$$

$$\mathbb{P}(homeowner = no \,|\, default = no) = 4/7,$$

$$\mathbb{P}(homeowner = no \,|\, default = yes) = 1,$$

$$\mathbb{P}(maritalstatus = single \,|\, default = no) = 2/7,$$

$$\mathbb{P}(maritalstatus = single \,|, default = yes) = 2/3,$$

$$\mathbb{P}(maritalstatus = married \,|\, default = no) = 4/7,$$

$$\mathbb{P}(maritalstatus = married \,|\, default = yes) = 0,$$

$$\mathbb{P}(maritalstatus = divorced \,|\, default = no) = 1/7,$$

$$\mathbb{P}(maritalstatus = divorced \,|\, default = yes) = 1/3.$$

The posterior density for annual income is normal with the estimated parameters for *default=no*, $\hat{\mu}$ =sample mean= $(125 + 100 + 70 + 120 + 60 + 220 + 75)/7 = 110$, $\hat{\sigma}^2 = s^2 = 2975$, and for *default=yes*, $\hat{\mu} = (95 + 85 + 90)/3 = 90$, and $\hat{\sigma}^2 = s^2 = 25$.

Suppose we would like to predict the default status for a person who is not a home owner, who is single, and whose annual income is $\$120K$. We write

$$\mathbb{P}(\mathbf{X} \,|\, default = no) = \mathbb{P}(homeowner = no \,|\, default = no) \times \mathbb{P}(maritalstatus = single \,|\, default = no) \times$$
$$\mathbb{P}(annualincome = \$120K \,|\, defaul = no) = (4/7)(2/7)\frac{1}{\sqrt{(2\pi)(2975)}}\, e^{-\frac{(120-110)^2}{(2)(2975)}} = 0.001215,$$

and

$$\mathbb{P}(\mathbf{X}\,|\,default = yes) = \mathbb{P}(homeowner = no\,|\,default = yes)\times\mathbb{P}(maritalstatus = single\,|\,default = yes)\times$$

$$\mathbb{P}(annualincome = \$120K\,|\,defaul = yes) = (1)(2/3)\frac{1}{\sqrt{(2\pi)(25)}}\,e^{-\frac{(120-90)^2}{(2)(25)}} = (8.1)(10)^{-10}.$$

Hence,

$$\mathbb{P}(default = no\,|\,\mathbf{X}) = \mathbb{P}(default = no)\mathbb{P}(\mathbf{X}\,|\,default = no)/\mathbb{P}(\mathbf{X})$$
$$= (0.7)(0.001215)/\mathbb{P}(\mathbf{X}) = 0.000851/\mathbb{P}(\mathbf{X}),$$

and

$$\mathbb{P}(default = yes\,|\,\mathbf{X}) = \mathbb{P}(default = yes)\mathbb{P}(\mathbf{X}\,|\,default = yes)/\mathbb{P}(\mathbf{X})$$
$$= (0.3)(8.1)(10)^{-10}/\mathbb{P}(\mathbf{X}) = (2.43)(10)^{-10}/\mathbb{P}(\mathbf{X}).$$

We can see that $\mathbb{P}(default = no\,|\,\mathbf{X}) > \mathbb{P}(default = yes\,|\,\mathbf{X})$ and so we predict *default=no* for this person. □

Example. We use the naive Bayes binary classifier for the data "pneumonia_data.csv".

In SAS:

```
proc import out=pneumonia datafile="./pneumonia_data.csv" dbms=csv replace;

/*SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS*/
proc surveyselect data=pneumonia rate=0.8 seed=999454
out=pneumonia outall method=srs;
run;

data train (drop=selected);
set pneumonia;
if selected=1;
run;

data test (drop=selected);
set pneumonia;
if selected=0;
run;
```

```
/*COMPUTING PRIOR PROBABILITIES*/
proc freq data=train noprint;
 table pneumonia/out=priors;
run;

data priors;
set priors;
 percent=percent/100;
 if pneumonia='no' then call symput('prior_no', percent);
 if pneumonia='yes' then call symput('prior_yes', percent);
run;

/*COMPUTING POSTERIOR PROBABILITIES FOR CATEGORICAL PREDICTORS*/
proc freq data=train noprint;
 table pneumonia*gender/out=gender_perc nocum list;
run;

data gender_perc;
set gender_perc;
 percent=percent/100;
if pneumonia='no' and gender='F' then call symput('female_no', percent);
if pneumonia='no' and gender='M' then call symput('male_no', percent);
if pneumonia='yes' and gender='F' then call symput('female_yes', percent);
if pneumonia='yes' and gender='M' then call symput('male_yes', percent);
 run;

proc freq data=train noprint;
 table pneumonia*tobacco_use/out=tobacco_use_perc nocum list;
run;

data tobacco_use_perc;
set tobacco_use_perc;
 percent=percent/100;
if pneumonia='no' and tobacco_use='no' then call symput('tobacco_no_no', percent);
if pneumonia='no' and tobacco_use='yes' then call symput('tobacco_yes_no', percent);
if pneumonia='yes' and tobacco_use='no' then call symput('tobacco_no_yes', percent);
if pneumonia='yes' and tobacco_use='yes' then call symput('tobacco_yes_yes', percent);
 run;

/*COMPUTING MEAN AND STANDARD DEVIATION FOR NUMERICAL PREDICTORS*/
proc means data=train mean std noprint;
 class pneumonia;
```

```
  var age PM2_5;
output out=stats;
run;

data stats;
 set stats;
if pneumonia='no' and _stat_='MEAN' then
  do;
   call symput('age_mean_no',age);
   call symput('PM2_5_mean_no',PM2_5);
  end;
if pneumonia='no' and _stat_='STD' then
  do;
   call symput('age_std_no',age);
   call symput('PM2_5_std_no',PM2_5);
  end;
if pneumonia='yes' and _stat_='MEAN' then
  do;
   call symput('age_mean_yes',age);
   call symput('PM2_5_mean_yes',PM2_5);
  end;
if pneumonia='yes' and _stat_='STD' then
  do;
   call symput('age_std_yes',age);
   call symput('PM2_5_std_yes',PM2_5);
  end;
run;

/*COMPUTING POSTERIOR PROBABILITIES FOR TESTING DATA*/
data test;
set test;
if (gender='F' and tobacco_use='no') then
do;
pred_prob_no=&prior_no*&female_no*&tobacco_no_no*1/(2*3.14)
*1/(&age_std_no*&PM2_5_std_no)
*exp(-(age-&age_mean_no)**2/(2*&age_std_no**2)
-(PM2_5-&PM2_5_mean_no)**2/(2*&PM2_5_std_no**2));
pred_prob_yes=&prior_yes*&female_yes*&tobacco_no_yes*1/(2*3.14)
*1/(&age_std_yes*&PM2_5_std_yes)*exp(-(age-&age_mean_yes)**2/(2*&age_std_yes**2)
-(PM2_5-&PM2_5_mean_yes)**2/(2*&PM2_5_std_yes**2));
end;
if (gender='M' and tobacco_use='no') then
```

```
do;
pred_prob_no=&prior_no*&male_no*&tobacco_no_no*1/(2*3.14)
*1/(&age_std_no*&PM2_5_std_no)
*exp(-(age-&age_mean_no)**2/(2*&age_std_no**2)
-(PM2_5-&PM2_5_mean_no)**2/(2*&PM2_5_std_no**2));
pred_prob_yes=&prior_yes*&male_yes*&tobacco_no_yes*1/(2*3.14)
*1/(&age_std_yes*&PM2_5_std_yes)
*exp(-(age-&age_mean_yes)**2/(2*&age_std_yes**2)
-(PM2_5-&PM2_5_mean_yes)**2/(2*&PM2_5_std_yes**2));
end;
if (gender='F' and tobacco_use='yes') then
do;
pred_prob_no=&prior_no*&female_no*&tobacco_yes_no*1/(2*3.14)
*1/(&age_std_no*&PM2_5_std_no)
*exp(-(age-&age_mean_no)**2/(2*&age_std_no**2)
-(PM2_5-&PM2_5_mean_no)**2/(2*&PM2_5_std_no**2));
pred_prob_yes=&prior_yes*&female_yes*&tobacco_yes_yes*1/(2*3.14)
*1/(&age_std_yes*&PM2_5_std_yes)
*exp(-(age-&age_mean_yes)**2/(2*&age_std_yes**2)
-(PM2_5-&PM2_5_mean_yes)**2/(2*&PM2_5_std_yes**2));
end;
if (gender='M' and tobacco_use='yes') then
do;
pred_prob_no=&prior_no*&male_no*&tobacco_yes_no*1/(2*3.14)
*1/(&age_std_no*&PM2_5_std_no)
*exp(-(age-&age_mean_no)**2/(2*&age_std_no**2)
-(PM2_5-&PM2_5_mean_no)**2/(2*&PM2_5_std_no**2));
pred_prob_yes=&prior_yes*&male_yes*&tobacco_yes_yes*1/(2*3.14)
*1/(&age_std_yes*&PM2_5_std_yes)
*exp(-(age-&age_mean_yes)**2/(2*&age_std_yes**2)
-(PM2_5-&PM2_5_mean_yes)**2/(2*&PM2_5_std_yes**2));
end;
run;

/*COMPUTING PREDICTION ACCURACY*/
data test;
 set test;
  if pred_prob_no < pred_prob_yes then pred_class='yes';
  else pred_class='no';
  if pneumonia=pred_class then pred=1; else pred=0;
 run;
```

```
 proc sql;
  select mean(pred) as accuracy
   from test;
quit;
```

accuracy
0.733333

In R: All the values for predictors have to be numeric, so we need to replace all string values with numeric values before running the technique.

```
pneumonia.data<- read.csv(file="./pneumonia_data.csv", header=TRUE, sep=",")

pneumonia.data$pneumonia<- ifelse(pneumonia.data$pneumonia=="yes",1,0)
pneumonia.data$gender<- ifelse(pneumonia.data$gender=='M',1,0)
pneumonia.data$tobacco_use<- ifelse(pneumonia.data$tobacco_use=='yes',1,0)

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(1012312)
sample <- sample(c(TRUE, FALSE), nrow(pneumonia.data), replace=TRUE, prob=c(0.8,0.2))
train<- pneumonia.data[sample,]
test<- pneumonia.data[!sample,]

test.x<- data.matrix(test[-5])
test.y<- data.matrix(test[5])

#FITTING NAIVE BAYES BINARY CLASSIFIER
library(e1071)
nb.class<- naiveBayes(as.factor(pneumonia) ~ gender + age + tobacco_use + PM2_5, data=train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.y<- as.numeric(predict(nb.class, test.x))-1

match<- c()
for (i in 1:length(pred.y))
   match[i]<- ifelse(test.y[i]==pred.y[i], 1, 0)
print(paste('accuracy=', round(mean(match)*100, digits=2),'%'))
```

"accuracy= 74.28 %"

In Python:

```
import pandas
from sklearn.model_selection import train_test_split
from sklearn import metrics

pneumonia_data=pandas.read_csv('./pneumonia_data.csv')
code_gender={'M':1,'F':0}
code_tobacco_use={'yes':1,'no':0}
code_pneumonia={'yes':1,'no':0}

pneumonia_data['gender']=pneumonia_data['gender'].map(code_gender)
pneumonia_data['tobacco_use']=pneumonia_data['tobacco_use'].map(code_tobacco_use)
pneumonia_data['pneumonia']=pneumonia_data['pneumonia'].map(code_pneumonia)

X=pneumonia_data.iloc[:,0:4].values
y=pneumonia_data.iloc[:,4].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=9994445)

#FITTING NAIVE BAYES BINARY CLASSIFIER
from sklearn.naive_bayes import GaussianNB
gauss_nb=GaussianNB()
gauss_nb.fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred = gauss_nb.predict(X_test)
print('Accuracy:', round(metrics.accuracy_score(y_test, y_pred)*100, 2),'%')
```

```
Accuracy: 71.68 %
```

□

Example. For the data "movie_data.csv" we fit a naive Bayes multinomial classifier.

In SAS:

```
proc import out=movie datafile="./movie_data.csv" dbms=csv replace;

/*SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS*/
proc surveyselect data=movie rate=0.8 seed=121800
out=movie outall method=srs;
run;

data train (drop=selected);
set movie;
if selected=1;
run;

data test (drop=selected);
set movie;
if selected=0;
run;

/*COMPUTING PRIOR PROBABILITIES*/
proc freq data=train noprint;
 table rating/out=priors;
run;

data priors;
set priors;
 percent=percent/100;
 if rating='very bad' then call symput('prior_very_bad', percent);
 if rating='bad' then call symput('prior_bad', percent);
 if rating='okay' then call symput('prior_okay', percent);
 if rating='good' then call symput('prior_good', percent);
 if rating='very good' then call symput('prior_very_good', percent);
run;

/*COMPUTING POSTERIOR PROBABILITIES FOR CATEGORICAL PREDICTORS*/
proc freq data=train noprint;
 table rating*gender/out=gender_perc
   nocum list;
run;

data gender_perc;
set gender_perc;
```

```
 percent=percent/100;
if rating='very bad' and gender='F' then call symput('female_very_bad', percent);
if rating='very bad' and gender='M' then call symput('male_very_bad', percent);
if rating='bad' and gender='F' then call symput('female_bad', percent);
if rating='bad' and gender='M' then call symput('male_bad', percent);
if rating='okay' and gender='F' then call symput('female_okay', percent);
if rating='okay' and gender='M' then call symput('male_okay', percent);
if rating='good' and gender='F' then call symput('female_good', percent);
if rating='good' and gender='M' then call symput('male_good', percent);
if rating='very good' and gender='F' then call symput('female_very_good', percent);
if rating='very good' and gender='M' then call symput('male_very_good', percent);
run;

proc freq data=train noprint;
 table rating*member/out=member_perc nocum list;
run;

data member_perc;
set member_perc;
 percent=percent/100;
if rating='very bad' and member='no' then call symput('member_no_very_bad', percent);
if rating='very bad' and member='yes' then call symput('member_yes_very_bad', percent);
if rating='bad' and member='no' then call symput('member_no_bad', percent);
if rating='bad' and member='yes' then call symput('member_yes_bad', percent);
if rating='okay' and member='no' then call symput('member_no_okay', percent);
if rating='okay' and member='yes' then call symput('member_yes_okay', percent);
if rating='good' and member='no' then call symput('member_no_good', percent);
if rating='good' and member='yes' then call symput('member_yes_good', percent);
if rating='very good' and member='no' then call symput('member_no_very_good', percent);
if rating='very good' and member='yes' then call symput('member_yes_very_good', percent);
 run;

/*COMPUTING MEAN AND STANDARD DEVIATION FOR NUMERICAL PREDICTORS*/
proc means data=train mean std noprint;
 class rating;
  var age nmovies;
output out=stats;
run;

data stats;
 set stats;
if rating='very bad' and _stat_='MEAN' then
```

```
  do;
   call symput('age_mean_very_bad',age);
   call symput('nmovies_mean_very_bad',nmovies);
  end;
if rating='very bad' and _stat_='STD' then
  do;
   call symput('age_std_very_bad',age);
   call symput('nmovies_std_very_bad',nmovies);
  end;
if rating='bad' and _stat_='MEAN' then
  do;
   call symput('age_mean_bad',age);
   call symput('nmovies_mean_bad',nmovies);
  end;
if rating='bad' and _stat_='STD' then
  do;
   call symput('age_std_bad',age);
   call symput('nmovies_std_bad',nmovies);
  end;
if rating='okay' and _stat_='MEAN' then
  do;
   call symput('age_mean_okay',age);
   call symput('nmovies_mean_okay',nmovies);
  end;
if rating='okay' and _stat_='STD' then
  do;
   call symput('age_std_okay',age);
   call symput('nmovies_std_okay',nmovies);
  end;
if rating='good' and _stat_='MEAN' then
  do;
   call symput('age_mean_good',age);
   call symput('nmovies_mean_good',nmovies);
  end;
if rating='good' and _stat_='STD' then
  do;
   call symput('age_std_good',age);
   call symput('nmovies_std_good',nmovies);
  end;
if rating='very good' and _stat_='MEAN' then
  do;
   call symput('age_mean_very_good',age);
```

```
   call symput('nmovies_mean_very_good',nmovies);
  end;
if rating='very good' and _stat_='STD' then
  do;
   call symput('age_std_very_good',age);
   call symput('nmovies_std_very_good',nmovies);
  end;
run;

/*COMPUTING POSTERIOR PROBABILITIES FOR TESTING DATA*/
data test;
set test;
if (gender='F' and member='no') then
do;
pred_prob_very_bad=&prior_very_bad*&female_very_bad*&member_no_very_bad*1/(2*3.14)
*1/(&age_std_very_bad*
&nmovies_std_very_bad)*exp(-(age-&age_mean_very_bad)**2/(2*&age_std_very_bad**2)
-(nmovies-&nmovies_mean_very_bad)**2/(2*&nmovies_std_very_bad**2));
pred_prob_bad=&prior_bad*&female_bad*&member_no_bad*1/(2*3.14)
*1/(&age_std_bad*&nmovies_std_bad)*exp(-(age-&age_mean_bad)**2/(2*&age_std_bad**2)
-(nmovies-&nmovies_mean_bad)**2/(2*&nmovies_std_bad**2));
pred_prob_okay=&prior_okay*&female_okay*&member_no_okay*1/(2*3.14)
*1/(&age_std_okay*&nmovies_std_okay)*exp(-(age-&age_mean_okay)**2/(2*&age_std_okay**2)
-(nmovies-&nmovies_mean_okay)**2/(2*&nmovies_std_okay**2));
pred_prob_good=&prior_good*&female_good*&member_no_good*1/(2*3.14)
*1/(&age_std_good*&nmovies_std_good)*exp(-(age-&age_mean_good)**2/(2*&age_std_good**2)
-(nmovies-&nmovies_mean_good)**2/(2*&nmovies_std_good**2));
pred_prob_very_good=&prior_very_good*&female_very_good*&member_no_very_good*1/(2*3.14)
*1/(&age_std_very_good
*&nmovies_std_very_good)*exp(-(age-&age_mean_very_good)**2/(2*&age_std_very_good**2)
-(nmovies-&nmovies_mean_very_good)**2/(2*&nmovies_std_very_good**2));
end;

if (gender='M' and member='no') then
do;
pred_prob_very_bad=&prior_very_bad*&male_very_bad*&member_no_very_bad*1/(2*3.14)
*1/(&age_std_very_bad
*&nmovies_std_very_bad)*exp(-(age-&age_mean_very_bad)**2/(2*&age_std_very_bad**2)
-(nmovies-&nmovies_mean_very_bad)
**2/(2*&nmovies_std_very_bad**2));
pred_prob_bad=&prior_bad*&male_bad*&member_no_bad*1/(2*3.14)
*1/(&age_std_bad*&nmovies_std_bad)*exp(-(age-&age_mean_bad)**2/(2*&age_std_bad**2)
```

```
-(nmovies-&nmovies_mean_bad)**2/(2*&nmovies_std_bad**2));
pred_prob_okay=&prior_okay*&male_okay*&member_no_okay*1/(2*3.14)
*1/(&age_std_okay*&nmovies_std_okay)*exp(-(age-&age_mean_okay)**2/(2*&age_std_okay**2)
-(nmovies-&nmovies_mean_okay)**2/(2*&nmovies_std_okay**2));
pred_prob_good=&prior_good*&male_good*&member_no_good*1/(2*3.14)
*1/(&age_std_good*&nmovies_std_good)*exp(-(age-&age_mean_good)**2/(2*&age_std_good**2)
-(nmovies-&nmovies_mean_good)**2/(2*&nmovies_std_good**2));
pred_prob_very_good=&prior_very_good*&male_very_good*&member_no_very_good*1/(2*3.14)
*1/(&age_std_very_good
*&nmovies_std_very_good)*exp(-(age-&age_mean_very_good)**2/(2*&age_std_very_good**2)
-(nmovies-&nmovies_mean_very_good)**2/(2*&nmovies_std_very_good**2));
end;

if (gender='F' and member='yes') then
do;
pred_prob_very_bad=&prior_very_bad*&female_very_bad*&member_yes_very_bad*1/(2*3.14)
*1/(&age_std_very_bad
*&nmovies_std_very_bad)*exp(-(age-&age_mean_very_bad)**2/(2*&age_std_very_bad**2)
-(nmovies-&nmovies_mean_very_bad)
**2/(2*&nmovies_std_very_bad**2));
pred_prob_bad=&prior_bad*&female_bad*&member_yes_bad*1/(2*3.14)
*1/(&age_std_bad*&nmovies_std_bad)*exp(-(age-&age_mean_bad)**2/(2*&age_std_bad**2)
-(nmovies-&nmovies_mean_bad)**2/(2*&nmovies_std_bad**2));
pred_prob_okay=&prior_okay*&female_okay*&member_yes_okay*1/(2*3.14)
*1/(&age_std_okay*&nmovies_std_okay)*exp(-(age-&age_mean_okay)**2/(2*&age_std_okay**2)
-(nmovies-&nmovies_mean_okay)**2/(2*&nmovies_std_okay**2));
pred_prob_good=&prior_good*&female_good*&member_yes_good*1/(2*3.14)
*1/(&age_std_good*&nmovies_std_good)*exp(-(age-&age_mean_good)**2/(2*&age_std_good**2)
-(nmovies-&nmovies_mean_good)**2/(2*&nmovies_std_good**2));
pred_prob_very_good=&prior_very_good*&female_very_good*&member_yes_very_good*1/(2*3.14)
*1/(&age_std_very_good
*&nmovies_std_very_good)*exp(-(age-&age_mean_very_good)**2/(2*&age_std_very_good**2)
-(nmovies-&nmovies_mean_very_good)**2/(2*&nmovies_std_very_good**2));
end;

if (gender='M' and member='yes') then
do;
pred_prob_very_bad=&prior_very_bad*&male_very_bad*&member_yes_very_bad*1/(2*3.14)
*1/(&age_std_very_bad
*&nmovies_std_very_bad)*exp(-(age-&age_mean_very_bad)**2/(2*&age_std_very_bad**2)
-(nmovies-&nmovies_mean_very_bad)**2/(2*&nmovies_std_very_bad**2));
pred_prob_bad=&prior_bad*&male_bad*&member_yes_bad*1/(2*3.14)
```

```
*1/(&age_std_bad*&nmovies_std_bad)*exp(-(age-&age_mean_bad)**2/(2*&age_std_bad**2)
-(nmovies-&nmovies_mean_bad)**2/(2*&nmovies_std_bad**2));
pred_prob_okay=&prior_okay*&male_okay*&member_yes_okay*1/(2*3.14)
*1/(&age_std_okay*&nmovies_std_okay)*exp(-(age-&age_mean_okay)**2/(2*&age_std_okay**2)
-(nmovies-&nmovies_mean_okay)**2/(2*&nmovies_std_okay**2));
pred_prob_good=&prior_good*&male_good*&member_yes_good*1/(2*3.14)
*1/(&age_std_good*&nmovies_std_good)*exp(-(age-&age_mean_good)**2/(2*&age_std_good**2)
-(nmovies-&nmovies_mean_good)**2/(2*&nmovies_std_good**2));
pred_prob_very_good=&prior_very_good*&male_very_good*&member_yes_very_good*1/(2*3.14)
*1/(&age_std_very_good
*&nmovies_std_very_good)*exp(-(age-&age_mean_very_good)**2/(2*&age_std_very_good**2)
-(nmovies-&nmovies_mean_very_good)**2/(2*&nmovies_std_very_good**2));
end;
run;

/*COMPUTING PREDICTION ACCURACY*/
data test;
 set test;
 max_prob=max(pred_prob_very_bad, pred_prob_bad,
 pred_prob_okay, pred_prob_good, pred_prob_very_good);
 if max_prob=pred_prob_very_good then pred_class='very good';
 if max_prob=pred_prob_very_bad then pred_class='very bad';
 if max_prob=pred_prob_bad then pred_class='bad';
 if max_prob=pred_prob_okay then pred_class='okay';
 if max_prob=pred_prob_good then pred_class='good';
 if pred_class=rating then pred=1; else pred=0;
run;

proc sql;
  select mean(pred) as accuracy
   from test;
quit;
```

accuracy
0.278146

In R:

```r
movie.data<- read.csv(file="./movie_data.csv", header=TRUE, sep=",")

movie.data$gender<- ifelse(movie.data$gender=='M',1,0)
movie.data$member<- ifelse(movie.data$member=='yes',1,0)

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(444625)
sample <- sample(c(TRUE, FALSE), nrow(movie.data), replace=TRUE, prob=c(0.8,0.2))
train<- movie.data[sample,]
test<- movie.data[!sample,]

test.x<- data.matrix(test[-5])
test.y<- data.matrix(test[5])

#FITTING NAIVE BAYES BINARY CLASSIFIER
library(e1071)
nb.multiclass<- naiveBayes(as.factor(rating) ~ age + gender + member + nmovies, data=train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.y<- as.numeric(predict(nb.multiclass, test.x))

print(paste('accuracy=', round((1-mean(test.y!=pred.y))*100, digits=2), '%'))
```

```
"accuracy= 32.53 %"
```

In Python:

```
import pandas
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from statistics import mean

movie_data=pandas.read_csv('./movie_data.csv')
code_gender={'M':1,'F':0}
code_member={'yes':1,'no':0}
code_rating={'very bad':1,'bad':2,'okay':3,'good':4,'very good':5}

movie_data['gender']=movie_data['gender'].map(code_gender)
movie_data['member']=movie_data['member'].map(code_member)
movie_data['rating']=movie_data['rating'].map(code_rating)

X=movie_data.iloc[:,0:4].values
y=movie_data.iloc[:,4].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=457752)

#FITTING NAIVE BAYES BINARY CLASSIFIER
gnb=GaussianNB()
gnb.fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred = gnb.predict(X_test)
y_test=pandas.DataFrame(y_test,columns=['rating'])
y_pred=pandas.DataFrame(y_pred,columns=['predicted'])
df=pandas.concat([y_test,y_pred],axis=1)

match=[]
for i in range(len(df)):
    if df['rating'][i]==df['predicted'][i]:
        match.append(1)
    else:
        match.append(0)

print('accuracy=', round(mean(match)*100,2),'%')
```

```
accuracy= 36.84 %
```

□