

Business Problem 1:

Prepare a classification model using Naive Bayes for salary data

Data Description:

age -- age of a person

workclass -- A work class is a grouping of work

education -- Education of an individuals

maritalstatus -- Marital status of an individuals

occupation -- occupation of an individuals

relationship --

race -- Race of an Individual

sex -- Gender of an Individual

capitalgain -- profit received from the sale of an investment

capitalloss -- A decrease in the value of a capital asset

hoursperweek -- number of hours work per week

native -- Native of an individual

Salary -- salary of an individual

Solution: In the given business problem, we need to classify the data given based on that the salary which is lesser than or greater than fifty thousand is classified.

Importing essential libraries:

Codes:

```
import pandas as pd
```

```
import numpy as np
```

```
from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.naive_bayes import MultinomialNB
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import confusion_matrix
```

Data Loading:

Codes:

```
Salary_test=pd.read_csv("E:\Data\Assignments\imade\Naivebayes\SalaryData_Test.csv")
```

```
Salary_train=pd.read_csv("E:\Data\Assignments\imade\Naivebayes\SalaryData_Train.csv")
```

```
colnames_test = list(Salary_test.columns)
```

```
colnames_train = list(Salary_train.columns)
```

Getting Dummies:

As the data consists categorical data, we need to convert them into numerical, so that we create dummies for the categorical data.

Codes:

```
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
```

```
le = LabelEncoder()
```

```
Salary_test['workclass'] = le.fit_transform(Salary_test['workclass'])
```

```
Salary_test['education'] = le.fit_transform(Salary_test['education'])
```

```
Salary_test['maritalstatus'] = le.fit_transform(Salary_test['maritalstatus'])
```

```
Salary_test['occupation'] = le.fit_transform(Salary_test['occupation'])
```

```
Salary_test['relationship'] = le.fit_transform(Salary_test['relationship'])
```

```
Salary_test['race'] = le.fit_transform(Salary_test['race'])
```

```
Salary_test['sex'] = le.fit_transform(Salary_test['sex'])
```

```
Salary_test['Salary'] = le.fit_transform(Salary_test['Salary'])
```

```
Salary_test['native'] = le.fit_transform(Salary_test['native'])
```

```
Salary_train['workclass'] = le.fit_transform(Salary_train['workclass'])
Salary_train['education'] = le.fit_transform(Salary_train['education'])
Salary_train['maritalstatus'] = le.fit_transform(Salary_train['maritalstatus'])
Salary_train['occupation'] = le.fit_transform(Salary_train['occupation'])
Salary_train['relationship'] = le.fit_transform(Salary_train['relationship'])
Salary_train['race'] = le.fit_transform(Salary_train['race'])
Salary_train['sex'] = le.fit_transform(Salary_train['sex'])
Salary_train['Salary'] = le.fit_transform(Salary_train['Salary'])
Salary_train['native'] = le.fit_transform(Salary_train['native'])
```

Defining input and outputs:

In this problem we need to classify the salaries which are lesser and greater than 50 thousand, so the salary variables becomes the output or target variable and the rest becomes the input variables.

Codes:

```
test_predictors = colnames_test[:13]
```

```
test_target = colnames_test[13]
```

```
train_predictors = colnames_train[:13]
```

```
train_target = colnames_train[13]
```

Splitting:

To get the confusion matrix, we need to split the data into Train and Test.

Splitting the test data set:

Codes:

```
DXtrain,DXtest,Dytrain,Dytest=train_test_split(Salary_test[test_predictors],Salary_test[test_target],test_size=0.3, random_state=0)
```

```
Dgnb = GaussianNB()
```

```
Dmnb = MultinomialNB()
```

Test data Prediction:

Codes:

```
Dpred_gnb = Dgnb.fit(DXtrain,Dytrain).predict(DXtest)
```

```
Dpred_mnb = Dmnb.fit(DXtrain,Dytrain).predict(DXtest)
```

Getting confusion matrix:

To check the accuracies we need to form the confusion matrix.

Test data Confusion Matrix:

Codes:

Gaussian Naive Bayes:

```
confusion_matrix(Dytest,Dpred_gnb)
```

```
print ("Accuracy",(3223+370)/(3223+370+162+763))
```

Accuracy = 79.53

Multinomial Naive Bayes:

```
confusion_matrix(Dytest,Dpred_mnb)
```

```
print ("Accuracy",(3258+232)/(3258+232+2584+127+901))
```

Accuracy = 49.14

Splitting the test data set:

Codes:

```
DXtrain,DXtest,Dytrain,Dytest=train_test_split(Salary_train[train_predictors],Salary_train[train_target],test_size=0.3, random_state=0)
```

```
Dgnb = GaussianNB()
```

```
Dmnb = MultinomialNB()
```

Train data Prediction:

Codes:

```
Dpred_gnb = Dgnb.fit(DXtrain,Dytrain).predict(DXtest)
```

```
Dpred_mnb = Dmnb.fit(DXtrain,Dytrain).predict(DXtest)
```

Test data Confusion Matrix:

Codes:

Gaussian Naive Bayes:

```
confusion_matrix(Dytest,Dpred_gnb)
```

```
print ("Accuracy",(6463+720)/(6463+720+1531+335))
```

Accuracy = 79.37

Multinomial Naive Bayes:

```
confusion_matrix(Dytest,Dpred_mnb)
```

```
print ("Accuracy",(6526+471)/(6526+471+272+1780))
```

Accuracy = 77.32

So we can conclude that the Accuracy of the Guassian Naive Bayes of both Test data and Train data are more accurate than Multinomial Naive Bayes.

```

8 import pandas as pd
9 import numpy as np
10 from sklearn.naive_bayes import GaussianNB
11 from sklearn.naive_bayes import MultinomialNB
12 from sklearn.model_selection import train_test_split
13 from sklearn.metrics import confusion_matrix
14
15 Salary_test = pd.read_csv("E:\\Data\\Assignments\\i made\\Naivebayes\\SalaryData_Test.csv")
16 Salary_train = pd.read_csv("E:\\Data\\Assignments\\i made\\Naivebayes\\SalaryData_Train.csv")
17
18 colnames_test = list(Salary_test.columns)
19 colnames_train = list(Salary_train.columns)
20
21 #.....Get Dummies.....
22 from sklearn.preprocessing import LabelEncoder, OneHotEncoder
23 le = LabelEncoder()
24
25 Salary_test['workclass'] = le.fit_transform(Salary_test['workclass'])
26 Salary_test['education'] = le.fit_transform(Salary_test['education'])
27 Salary_test['maritalstatus'] = le.fit_transform(Salary_test['maritalstatus'])
28 Salary_test['occupation'] = le.fit_transform(Salary_test['occupation'])
29 Salary_test['relationship'] = le.fit_transform(Salary_test['relationship'])
30 Salary_test['race'] = le.fit_transform(Salary_test['race'])
31 Salary_test['sex'] = le.fit_transform(Salary_test['sex'])
32 Salary_test['Salary'] = le.fit_transform(Salary_test['Salary'])
33 Salary_test['native'] = le.fit_transform(Salary_test['native'])
34
35
36 Salary_train['workclass'] = le.fit_transform(Salary_train['workclass'])
37 Salary_train['education'] = le.fit_transform(Salary_train['education'])
38 Salary_train['maritalstatus'] = le.fit_transform(Salary_train['maritalstatus'])
39 Salary_train['occupation'] = le.fit_transform(Salary_train['occupation'])
40 Salary_train['relationship'] = le.fit_transform(Salary_train['relationship'])
41 Salary_train['race'] = le.fit_transform(Salary_train['race'])
42 Salary_train['sex'] = le.fit_transform(Salary_train['sex'])
43 Salary_train['Salary'] = le.fit_transform(Salary_train['Salary'])
44 Salary_train['native'] = le.fit_transform(Salary_train['native'])
45
46 # Definig input and output
47 test_predictors = colnames_test[:13]
48 test_target = colnames_test[13]
49
50 train_predictors = colnames_train[:13]
51 train_target = colnames_train[13]
52
53 # Splitting
54 DXtrain, DXtest, Dytrain, Dytest = train_test_split(Salary_test[test_predictors], Salary_test[test_target], test_size=0.3, random_state=0)
55
56 Dgnb = GaussianNB()
57 Dmnb = MultinomialNB()
58
59 # Prediction
60 Dpred_gnb = Dgnb.fit(DXtrain, Dytrain).predict(DXtest)
61 Dpred_mnb = Dmnb.fit(DXtrain, Dytrain).predict(DXtest)
62
63 # Confusion Matrix
64 # GaussianNB
65 confusion_matrix(Dytest, Dpred_gnb)
66 print ("Accuracy", (3223+370)/(3223+370+162+763)) # 79.53
67
68 # MultinomialNB
69 confusion_matrix(Dytest, Dpred_mnb)
70 print ("Accuracy", (3258+232)/(3258+232+2584+127+901)) # 49.14
71
72
73 DXtrain, DXtest, Dytrain, Dytest = train_test_split(Salary_train[train_predictors], Salary_train[train_target], test_size=0.3, random_state=0)
74
75 Dgnb = GaussianNB()
76 Dmnb = MultinomialNB()
77
78 # Prediction
79
80 Dpred_gnb = Dgnb.fit(DXtrain, Dytrain).predict(DXtest)
81 Dpred_mnb = Dmnb.fit(DXtrain, Dytrain).predict(DXtest)
82
83 # Confusion Matrix
84 # GaussianNB
85 confusion_matrix(Dytest, Dpred_gnb)
86 print ("Accuracy", (6463+720)/(6463+720+1531+335)) # 79.37
87
88 # MultinomialNB
89 confusion_matrix(Dytest, Dpred_mnb)
90 print ("Accuracy", (6526+471)/(6526+471+272+1780)) # 77.32
91

```

Business Problem 2:

Build a naive Bayes model on the data set for classifying the ham and spam

Solution: In the given business problem, we need to classify before the classification we need to clean the data, since the given data is raw or in unstructured format and consists of unwanted data.

Importing essential libraries:

Codes:

```
import pandas as pd
```

```
import numpy as np
```

```
from sklearn.feature_extraction.text import CountVectorizer,TfidfTransformer
```

Loading the data:

Codes:

```
email_data=pd.read_csv("E:\\Data\\Assignments\\imade\\Naivebayes\\ham_spam.csv",engine = "python")
```

cleaning data:

Codes:

```
import re
```

```
stop_words = []
```

```
with open("E:\\Data\\Assignments\\i made\\TextMining\\stop.txt") as f:
```

```
    stop_words = f.read()
```

Splitting the entire string by giving separator as "\n" to get list of all stop words:

Codes:

```
stop_words = stop_words.split("\n")
```

Transform a count matrix to a normalized tf or tf-idf representation:

Creating a matrix of token counts for the entire text document:

Codes:

```
def split_into_words(i):  
    return [word for word in i.split(" ")]
```

splitting data into train and test data sets:

Codes:

```
from sklearn.model_selection import train_test_split  
  
email_train,email_test = train_test_split(email_data,test_size=0.3)
```

Preparing email texts into word count matrix format:

Codes:

```
emails_bow = CountVectorizer(analyzer=split_into_words).fit(email_data.text)
```

For all messages:

Codes:

```
all_emails_matrix = emails_bow.transform(email_data.text)  
  
all_emails_matrix.shape
```


(5559, 6661)

For training messages:

Codes:

```
train_emails_matrix = emails_bow.transform(email_train.text)
```

```
train_emails_matrix.shape
```

(3891, 6661)

For testing messages:

Codes:

```
test_emails_matrix = emails_bow.transform(email_test.text)
```

```
test_emails_matrix.shape
```

(1668, 6661)

Preparing a naive bayes model on training data set without TFIDF matrices:

Codes:

```
from sklearn.naive_bayes import MultinomialNB as MB
```

```
from sklearn.naive_bayes import GaussianNB as GB
```

Multinomial Naive Bayes

Codes:

```
classifier_mb = MB()
```

```
classifier_mb.fit(train_emails_matrix,email_train.type)
```

```
train_pred_m = classifier_mb.predict(train_emails_matrix)
```

```
accuracy_train_m = np.mean(train_pred_m==email_train.type)
```

Accuracy = 98%

```
test_pred_m = classifier_mb.predict(test_emails_matrix)
```

```
accuracy_test_m = np.mean(test_pred_m==email_test.type)
```

Accuracy = 96%

Gaussian Naive Bayes

```
classifier_gb = GB()
```

```
classifier_gb.fit(train_emails_matrix.toarray(),email_train.type.values) # we need to convert  
tfidf into array format which is compatible for gaussian naive bayes
```

```
train_pred_g = classifier_gb.predict(train_emails_matrix.toarray())
```

```
accuracy_train_g = np.mean(train_pred_g==email_train.type)
```

Accuracy = 90%

```
test_pred_g = classifier_gb.predict(test_emails_matrix.toarray())
```

```
accuracy_test_g = np.mean(test_pred_g==email_test.type)
```

Accuracy = 83%

Learning Term weighting and normalizing on entire emails:

```
tfidf_transformer = TfidfTransformer().fit(all_emails_matrix)
```

Preparing TFIDF for train emails:

```
train_tfidf = tfidf_transformer.transform(train_emails_matrix)
```

```
train_tfidf.shape
```

(3891, 6661)

Preparing TFIDF for test emails:

```
test_tfidf = tfidf_transformer.transform(test_emails_matrix)
```

```
test_tfidf.shape
```

(1668, 6661)

Preparing a naive bayes model on training data set:

```
from sklearn.naive_bayes import MultinomialNB as MB
```

```
from sklearn.naive_bayes import GaussianNB as GB
```

Multinomial Naive Bayes:

```
classifier_mb = MB()
```

```
classifier_mb.fit(train_tfidf,email_train.type)
```

```
train_pred_m = classifier_mb.predict(train_tfidf)
```

```
accuracy_train_m = np.mean(train_pred_m==email_train.type)
```

Accuracy = 96%

```
test_pred_m = classifier_mb.predict(test_tfidf)
```

```
accuracy_test_m = np.mean(test_pred_m==email_test.type)
```

Accuracy = 95%

Gaussian Naive Bayes:

```
classifier_gb = GB()
```

```
classifier_gb.fit(train_tfidf.toarray(),email_train.type.values)
```

[we need to convert tfidf into array format which is compatible for gaussian naive bayes]

```
train_pred_g = classifier_gb.predict(train_tfidf.toarray())
```

```
accuracy_train_g = np.mean(train_pred_g==email_train.type)
```

Accuracy = 91%

```
test_pred_g = classifier_gb.predict(test_tfidf.toarray())
```

```
accuracy_test_g = np.mean(test_pred_g==email_test.type)
```

Accuracy = 85%

Hence we can conclude that the Accuracy of the Multinomial Naïve Bayes model for both training and testing data sets gives a better results or accuracies than Guassian Naive Bayes model.

```
3 import pandas as pd
4 import numpy as np
5 from sklearn.feature_extraction.text import CountVectorizer,TfidfTransformer
6
7 # Loading the data set
8
9 email_data = pd.read_csv("E:\\Data\\Assignments\\i made\\Naivebayes\\ham_spam.csv",engine = "python")
10
11
12 # cleaning data
13 import re
14 stop_words = []
15 with open("E:\\Data\\Assignments\\i made\\TextMining\\stop.txt") as f:
16     stop_words = f.read()
17
18
19 # splitting the entire string by giving separator as "\n" to get list of
20 # all stop words
21 stop_words = stop_words.split("\n")
22
```

```

45 email_data.text = email_data.text.apply(cleaning_text)
46
47 # removing empty rows
48 email_data.shape
49 email_data = email_data.loc[email_data.text != " ",:]
50
51
52 # CountVectorizer
53 # Convert a collection of text documents to a matrix of token counts
54
55 # TfidfTransformer
56 # Transform a count matrix to a normalized tf or tf-idf representation
57
58 # creating a matrix of token counts for the entire text document
59
60 def split_into_words(i):
61     return [word for word in i.split(" ")]
62
63
64 # splitting data into train and test data sets
65 from sklearn.model_selection import train_test_split
66
67 email_train,email_test = train_test_split(email_data,test_size=0.3)
68
69

```

```

70 # Preparing email texts into word count matrix format
71 emails_bow = CountVectorizer(analyzer=split_into_words).fit(email_data.text)
72
73
74 # For all messages
75 all_emails_matrix = emails_bow.transform(email_data.text)
76 all_emails_matrix.shape # (5559,6661)
77 # For training messages
78 train_emails_matrix = emails_bow.transform(email_train.text)
79 train_emails_matrix.shape # (3891,6661)
80
81
82
83 # For testing messages
84 test_emails_matrix = emails_bow.transform(email_test.text)
85 test_emails_matrix.shape # (1668,6661)
86
87 ##### Without TFIDF matrices #####
88 # Preparing a naive bayes model on training data set
89
90 from sklearn.naive_bayes import MultinomialNB as MB
91 from sklearn.naive_bayes import GaussianNB as GB
92
93 # Multinomial Naive Bayes
94 classifier_mb = MB(0)
95 classifier_mb.fit(train_emails_matrix,email_train.type)
96 train_pred_m = classifier_mb.predict(train_emails_matrix)
97 accuracy_train_m = np.mean(train_pred_m==email_train.type) # 98%
98
99 test_pred_m = classifier_mb.predict(test_emails_matrix)
100 accuracy_test_m = np.mean(test_pred_m==email_test.type) # 96%

```

```

113 # Gaussian Naive Bayes
114 classifier_gb = GB()
115 classifier_gb.fit(train_emails_matrix.toarray(),email_train.type.values) # we need to convert tfidf into array format
116 train_pred_g = classifier_gb.predict(train_emails_matrix.toarray())
117 accuracy_train_g = np.mean(train_pred_g==email_train.type) # 90%
118
119 test_pred_g = classifier_gb.predict(test_emails_matrix.toarray())
120 accuracy_test_g = np.mean(test_pred_g==email_test.type) # 83%
121
122
123 #####3
124
125 # Learning Term weighting and normalizing on entire emails
126 tfidf_transformer = TfidfTransformer().fit(all_emails_matrix)
127
128 # Preparing TFIDF for train emails
129 train_tfidf = tfidf_transformer.transform(train_emails_matrix)
130
131 train_tfidf.shape # (3891, 6661)
132
133 # Preparing TFIDF for test emails
134 test_tfidf = tfidf_transformer.transform(test_emails_matrix)
135
136 test_tfidf.shape # (1668, 6661)
137
138 # Preparing a naive bayes model on training data set
139
140 from sklearn.naive_bayes import MultinomialNB as MB
141 from sklearn.naive_bayes import GaussianNB as GB
142

```

```

143 # Multinomial Naive Bayes
144 classifier_mb = MB()
145 classifier_mb.fit(train_tfidf,email_train.type)
146 train_pred_m = classifier_mb.predict(train_tfidf)
147 accuracy_train_m = np.mean(train_pred_m==email_train.type) # 96%
148
149 test_pred_m = classifier_mb.predict(test_tfidf)
150 accuracy_test_m = np.mean(test_pred_m==email_test.type) # 95%
151
152 # Gaussian Naive Bayes
153 classifier_gb = GB()
154 classifier_gb.fit(train_tfidf.toarray(),email_train.type.values) # we need to convert tfidf into array format
155 train_pred_g = classifier_gb.predict(train_tfidf.toarray())
156 accuracy_train_g = np.mean(train_pred_g==email_train.type) # 91%
157 test_pred_g = classifier_gb.predict(test_tfidf.toarray())
158 accuracy_test_g = np.mean(test_pred_g==email_test.type) # 85%
159
160 # inplace of tfidf we can also use train_emails_matrix and test_emails_matrix instead of term inverse document
161

```