CSE572-Lab5-key

October 3, 2022

1 CSE 572: Lab 5

In this lab, you will practice implementing the probabilistic Naive Bayes classifier.

To execute and make changes to this notebook, click File > Save a copy to save your own version in your Google Drive or Github. Read the step-by-step instructions below carefully. To execute the code, click on each cell below and press the SHIFT-ENTER keys simultaneously or by clicking the Play button.

When you finish executing all code/exercises, save your notebook then download a copy (.ipynb file). Submit the following **three** things: 1. a link to your Colab notebook, 2. the .ipynb file, and 3. a pdf of the executed notebook on Canvas.

To generate a pdf of the notebook, click File > Print > Save as PDF.

1.1 Implement Naive Bayes manually

You like to play pickup soccer at the Sun Devil Fitness Center (SDFC). However, you noticed that on some days there are enough people to play a scrimmage but on some days there are not enough people. It's more fun for you to play a scrimmage, and it's a lot of effort for you to pull yourself away from studying Data Mining, so you decide that you only want to go to the SDFC to play soccer when it's likely there will be enough players for a scrimmage. You think players' attendance might be dependent on the weather and proximity to exam weeks, so you collect some observations about these attributes on the days that you've gone to play in the past and whether or not there was a scrimmage on those days. You code that dataset below.

[1]:
```
import pandas as pd

# Create the dataframe
d = {
     'weather':␣
 ↪['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny','Rainy','Sun
     'exam-proximity':␣
 ↪['High','High','High','Medium','Low','Low','Low','Medium','Low','Medium','Medium','Medium',
     'scrimmage':␣
 ↪['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']
    }

df = pd.DataFrame(data=d)
```

[2]:
```
df
```

[2]:

	weather	exam-proximity	scrimmage
0	Sunny	High	No
1	Sunny	High	No
2	Overcast	High	Yes
3	Rainy	Medium	Yes
4	Rainy	Low	Yes
5	Rainy	Low	No
6	Overcast	Low	Yes
7	Sunny	Medium	No
8	Sunny	Low	Yes
9	Rainy	Medium	Yes
10	Sunny	Medium	Yes
11	Overcast	Medium	Yes
12	Overcast	High	Yes
13	Rainy	Medium	No

Today, the weather is Sunny and the proximity to exams is Medium. Implement a Naive Bayes classifier to decide if there is likely to be a scrimmage today and thus you should go to the SDFC.

First, calculate the prior probability of a scrimmage $P(Y = yes)$ and $P(Y = no)$

[3]:
```
p_y_yes = df[df['scrimmage'] == 'Yes'].shape[0] / df.shape[0]

p_y_yes
```

[3]:
```
0.6428571428571429
```

[4]:
```
p_y_no = df[df['scrimmage'] == 'No'].shape[0] / df.shape[0]
# alternate solution: p_y_no = 1 - p_y_yes
# YOUR CODE HERE

p_y_no
```

[4]:
```
0.35714285714285715
```

Next, we calculate the class-conditional probabilities for the weather and exam-proximity attributes: $P(weather = sunny|no)$, $P(weather = sunny|yes)$, $P(examproximity = medium|no)$, $P(examproximity = medium|yes)$.

Recall that for categorical attributes, $P(X_i = c|y) = \frac{n_c}{n}$ where n_c is number of instances where $X_i = c$ and belongs to class y and n is total number of occurrences of class y.

[5]:
```
p_sunny_no = df[(df['scrimmage'] == 'No') & (df['weather']=='Sunny')].shape[0] /
  ↪ df[df['scrimmage'] == 'No'].shape[0]

p_sunny_no
```

[5]:
```
0.6
```

[6]:
```
p_sunny_yes = df[(df['scrimmage'] == 'Yes') & (df['weather']=='Sunny')].
  ↪shape[0] / df[df['scrimmage'] == 'Yes'].shape[0]
# YOUR CODE HERE

p_sunny_yes
```

[6]:
```
0.2222222222222222
```

[7]:
```
p_medium_no = df[(df['scrimmage'] == 'No') & (df['exam-proximity']=='Medium')].
  ↪shape[0] / df[df['scrimmage'] == 'No'].shape[0]

p_medium_no
```

[7]:
```
0.4
```

[8]:
```
p_medium_yes = df[(df['scrimmage'] == 'Yes') &␣
  ↪(df['exam-proximity']=='Medium')].shape[0] / df[df['scrimmage'] == 'Yes'].
  ↪shape[0]
# YOUR CODE HERE

p_medium_yes
```

[8]:
```
0.4444444444444444
```

Question 1:

The Naive Bayes assumption is that weather (X_1) and exam proximity (X_2) are conditionally independent given the class value Y. This is true if $P(X_1|X_2, Y) = P(X_1|Y)$, i.e., the value of X_2 has no influence on the value of X_1 given Y. Thus the Naive Bayes assumption is that weather and exam proximity are independent given the variable Y (whether or not there is a scrimmage). Is this a reasonable assumption? Why or why not?

Answer: YOUR ANSWER HERE

Yes. The weather is independent of the proximity to exams given any value of Y because they are completely separate processes (the weather is not affected by exams and vice versa).

Assuming the attributes are conditionally independent given Y allows us to compute $P(X|Y)$ by multiplying the class-conditional probabilities $P(X_1|Y)$ and $P(X_2|Y)$. We compute this below.

[9]:
```
p_x_yes = p_sunny_yes * p_medium_yes

p_x_yes
```

[9]:
```
0.09876543209876543
```

[10]:
```
p_x_no = p_sunny_no * p_medium_no
# YOUR CODE HERE

p_x_no
```

[10]: 0.24

Now we are ready to determine our classification. According to Bayes theorem, if $P(X|No)P(No) > P(X|Yes)P(Yes)$, then $P(No|X) > P(Yes|X)$ and we should classify Scrimmage = No and we should not go to the SDFC. If the reverse is true, then we should classify Scrimmage = Yes and we should go to the SDFC.

Below, we calculate \$P(X|No)P(No) > \$ and $P(X|Yes)P(Yes)$ and check if $P(X|No)P(No) > P(X|Yes)P(Yes)$.

[11]:
```
p_no_x = p_x_no * p_y_no

p_no_x
```

[11]: 0.08571428571428572

[12]:
```
p_yes_x = p_x_yes * p_y_yes
#YOUR CODE HERE

p_yes_x
```

[12]: 0.06349206349206349

[13]:
```
# Check if P(Y=no|X) is greater than P(Y=yes|X)
p_no_x > p_yes_x
```

[13]: True

Question 2:

Is it likely there will be a scrimmage today, and thus should you go to the SDFC to play soccer? Answer Yes or No.

Answer:

YOUR ANSWER HERE

No.

1.2 Implement Naive Bayes using Scikit-learn

In this section, we will use scikit-learn to implement Gaussian Naive Bayes to predict whether samples in the Wisconsin breast cancer dataset have the class value 'benign' or 'malignant'. Gaussian Naive Bayes estimates the class-conditional probabilities for each attribute by estimating a Gaussian probability density function for each attribute. You can read more about the Gaussian

Naive Bayes classifier (and other Naive Bayes classifiers assuming different types of probability distributions) in the sklearn documentation.

[14]:
```
import numpy as np

# Load the Wisconsin breast cancer dataset
data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/
 ↪breast-cancer-wisconsin/breast-cancer-wisconsin.data', header=None)
data.columns = ['Sample code', 'Clump Thickness', 'Uniformity of Cell Size',␣
 ↪'Uniformity of Cell Shape',
                'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare␣
 ↪Nuclei', 'Bland Chromatin',
                'Normal Nucleoli', 'Mitoses','Class']

data = data.drop(['Sample code'],axis=1)

data = data.replace('?',np.NaN)
data['Bare Nuclei'] = pd.to_numeric(data['Bare Nuclei'])

data
```

[14]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	\
0	5	1	1	
1	5	4	4	
2	3	1	1	
3	6	8	8	
4	4	1	1	
..	…	…	…	
694	3	1	1	
695	2	1	1	
696	5	10	10	
697	4	8	6	
698	4	8	8	

	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	\
0	1	2	1.0	
1	5	7	10.0	
2	1	2	2.0	
3	1	3	4.0	
4	3	2	1.0	
..	…	…	…	
694	1	3	2.0	
695	1	2	1.0	
696	3	7	3.0	
697	4	3	4.0	
698	5	4	5.0	

```
     Bland Chromatin  Normal Nucleoli  Mitoses  Class
0                  3                1        1      2
1                  3                2        1      2
2                  3                1        1      2
3                  3                7        1      2
4                  3                1        1      2
..               ...              ...      ...    ...
694                1                1        1      2
695                1                1        1      2
696                8               10        2      4
697               10                6        1      4
698               10                4        1      4

[699 rows x 10 columns]
```

After loading the dataset, we clean it by removing samples with missing data, duplicates, or outliers using the code from Labs 2-3.

[15]:
```
def inds_nans(df):
    inds = df.isna().any(axis=1)
    # print('Found {} rows that had NaN values.'.format(inds.sum()))
    return inds

def inds_dups(df):
    inds = df.duplicated()
    # print('Found {} rows that were duplicates.'.format(inds.sum()))
    return inds

def inds_outliers(df):
    # In this example, we defined outliers as values that are +/- 3 standard␣
 ↪deviations
    # from the mean value. To identify such values, we need to compute the Z␣
 ↪score for
    # every value by subtracting the feature-wise mean and dividing by the␣
 ↪feature-wise
    # standard deviation (also known as standardizing the data).
    df = df[df.columns[:-1]]
    Z = (df-df.mean())/df.std()
    # The below code will give a value of True or False for each row. The row␣
 ↪will be
    # True if all of the feature values for that row were within 3 standard␣
 ↪deviations of
    # the mean. The row will be False if at leaset one of the feature values␣
 ↪for that row
    # was NOT within 3 standard deviations of the mean.
    inlier_inds = ((Z > -3).sum(axis=1)==9) & ((Z <= 3).sum(axis=1)==9)
    # The outliers are the inverse boolean values of the above
```

```
    outlier_inds = ~inlier_inds
    # print('Found {} rows that were outliers.'.format(outlier_inds.sum()))
    return outlier_inds
```

[16]:
```
# Select only the rows at index locations that were not nans, duplicates, or␣
 ↪outliers
data_clean = data.loc[~((inds_nans(data) | inds_dups(data)) |␣
 ↪inds_outliers(data)),:]

data_clean
```

[16]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	\
0	5	1	1	
1	5	4	4	
2	3	1	1	
3	6	8	8	
4	4	1	1	
..	…	…	…	
693	3	1	1	
694	3	1	1	
696	5	10	10	
697	4	8	6	
698	4	8	8	

	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	\
0	1	2	1.0	
1	5	7	10.0	
2	1	2	2.0	
3	1	3	4.0	
4	3	2	1.0	
..	…	…	…	
693	1	2	1.0	
694	1	3	2.0	
696	3	7	3.0	
697	4	3	4.0	
698	5	4	5.0	

	Bland Chromatin	Normal Nucleoli	Mitoses	Class
0	3	1	1	2
1	3	2	1	2
2	3	1	1	2
3	3	7	1	2
4	3	1	1	2
..	…	…	…	…
693	2	1	2	2
694	1	1	1	2
696	8	10	2	4

```
697                10                 6        1       4
698                10                 4        1       4

[399 rows x 10 columns]
```

Next we normalize the data using the code from Lab 3 so the features will have approximately normal distributions.

[17]:
```
from sklearn import preprocessing

# Normalize the feature columns
data_clean[data_clean.columns[:-1]] = preprocessing.
 ↪normalize(data_clean[data_clean.columns[:-1]], norm='l2')
```

```
/Users/hkerner/anaconda3/envs/landcover-mapping/lib/python3.6/site-
packages/ipykernel_launcher.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  after removing the cwd from sys.path.
/Users/hkerner/anaconda3/envs/landcover-mapping/lib/python3.6/site-
packages/pandas/core/indexing.py:1734: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  isetter(loc, value[:, i].tolist())
```

[18]:
```
data_clean
```

[18]:
```
     Clump Thickness  Uniformity of Cell Size  Uniformity of Cell Shape  \
0           0.753778                 0.150756                  0.150756
1           0.319438                 0.255551                  0.255551
2           0.538816                 0.179605                  0.179605
3           0.380235                 0.506979                  0.506979
4           0.609994                 0.152499                  0.152499
..               …                        …                         …
693         0.588348                 0.196116                  0.196116
694         0.566947                 0.188982                  0.188982
696         0.233126                 0.466252                  0.466252
697         0.233285                 0.466569                  0.349927
698         0.221201                 0.442401                  0.442401

     Marginal Adhesion  Single Epithelial Cell Size  Bare Nuclei  \
0             0.150756                     0.301511     0.150756
```

```
1              0.319438                      0.447214     0.638877
2              0.179605                      0.359211     0.359211
3              0.063372                      0.190117     0.253490
4              0.457496                      0.304997     0.152499
..                  …                             …            …
693            0.196116                      0.392232     0.196116
694            0.188982                      0.566947     0.377964
696            0.139876                      0.326377     0.139876
697            0.233285                      0.174964     0.233285
698            0.276501                      0.221201     0.276501

     Bland Chromatin  Normal Nucleoli   Mitoses  Class
0           0.452267          0.150756  0.150756      2
1           0.191663          0.127775  0.063888      2
2           0.538816          0.179605  0.179605      2
3           0.190117          0.443607  0.063372      2
4           0.457496          0.152499  0.152499      2
..               …                 …       …    …
693         0.392232          0.196116  0.392232      2
694         0.188982          0.188982  0.188982      2
696         0.373002          0.466252  0.093250      4
697         0.583212          0.349927  0.058321      4
698         0.553001          0.221201  0.055300      4

[399 rows x 10 columns]
```

Split the data into a training and test set with 70% train and 30% test.

[19]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data_clean[data_clean.
 ↪columns[:-1]],
                                                    data_clean[data_clean.
 ↪columns[-1]],
                                                    test_size=0.3,
                                                    random_state=0)
```

Use the GaussianNB object in sklearn to fit a Gaussian Naive Bayes classifier and predict the class labels for the test set based on probabilities estimated from the training set.

[20]:
```
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()
```

[21]:
```
# Fit the model parameters using the training data
gnb = gnb.fit(X_train, y_train)
```

[22]:
```
# Predict the test set classes using the trained model
y_pred = gnb.predict(X_test)
```

Compute the accuracy of this model on the test set.

[23]:
```
from sklearn.metrics import accuracy_score
# YOUR CODE HERE

print('Accuracy on test data is %.2f' % (accuracy_score(y_test, y_pred)))
```

```
Accuracy on test data is 0.87
```

Compute the accuracy of this model on the training set.

[24]:
```
# YOUR CODE HERE
y_pred_train = gnb.predict(X_train)

print('Accuracy on training data is %.2f' % (accuracy_score(y_train,␣
 ↪y_pred_train)))
```

```
Accuracy on training data is 0.89
```