Lab 06: Generative classifiers: Naive Bayes

As discussed in class, a Naive Bayes classifier works as follows:

$$\begin{aligned} p(y \mid \mathbf{x}; \theta) &= \frac{p(\mathbf{x} \mid y; \theta) p(y; \theta)}{p(\mathbf{x}; \theta)} \\ &\propto p(\mathbf{x} \mid y; \theta) p(y; \theta) \\ &\approx p(y; \theta) \prod_j p(x_j \mid y; \theta) \end{aligned}$$

We will use Naive Bayes to perform diabetes diagnosis and text classification.

Example 1: Diabetes classification

In this example we predict wheter a patient with specific diagnostic measurements has diabetes or not. As the features are continuous, we will model the conditional probabilities $p(x_j \mid y; \theta)$ as univariate Gaussians with mean $\mu_{j,y}$ and standard deviation $\sigma_{j,y}$.

The data are originally from the U.S. National Institute of Diabetes and Digestive and Kidney Diseases (NIDDK) and are available from Kaggle (https://www.kaggle.com/uciml/pima-indians-diabetes-database)

In [25]:
```
import csv
import math
import random
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
```

Data manipulation

First we have some functions to read the dataset, split it into train and test, and partition it according to target class (y).

In [26]:
```
# Load data from CSV file
def loadCsv(filename):
    data_raw = pd.read_csv(filename)
    headers = data_raw.columns
    dataset = data_raw.values
    return dataset, headers

# Split dataset into test and train with given ratio
def splitDataset(test_size,*arrays,**kwargs):
    return train_test_split(*arrays,test_size=test_size,**kwargs)

# Separate training data according to target class
# Return key value pairs array in which keys are possible target
variable values
# and values are the data records.

def data_split_byClass(dataset):
    Xy = {}
    for i in range(len(dataset)):
        datapair = dataset[i]
        # datapair[-1] (the last column) is the target class for
this record.
        # Check if we already have this value as a key in the ret
urn array
        if (datapair[-1] not in Xy):
            # Add class as key
            Xy[datapair[-1]] = []
        # Append this record to array of records for this class k
ey
        Xy[datapair[-1]].append(datapair)
    return Xy
```

Model training

Next we have some functions used for training the model. Parameters include mean and standard deviation, used to partition numerical variables into categorical variables, as well as

In [27]:
```
# Parameters of a Gaussian are its mean and standard deviation

def mean(numbers):
    return sum(numbers)/float(len(numbers))

def stdev(numbers):
    avg = mean(numbers)
    variance = sum([pow(x-avg,2) for x in numbers])/float(len(num
bers)-1)
    return math.sqrt(variance)

# Calculate Gaussian parameters mu and sigma for each attribute o
ver a dataset

def get_gaussian_parameters(X,y):
    parameters = {}
    unique_y = np.unique(y)
    for uy in unique_y:
        mean = np.mean(X[y==uy],axis=0)
        std = np.std(X[y==uy],axis=0)
        py = y[y==uy].size/y.size
        parameters[uy] = {'prior':py,'mean':mean,'std':std}
    return parameters, unique_y

def calculateProbability(x, mu, sigma):
    sigma = np.diag(sigma**2)
    x = x.reshape(-1,1)
    mu = mu.reshape(-1,1)
    exponent = np.exp(-1/2*(x-mu).T@np.linalg.inv(sigma)@(x-mu))
    return ((1/(np.sqrt(((2*np.pi)**x.size)*np.linalg.det(sigm
a))))*exponent)[0,0]
```

Model testing

Next some functions for testing the model on a test set and computing its accuracy. Note that we assume

$$p(y \mid \mathbf{x}; \theta) \propto p(\mathbf{x} \mid y; \theta),$$

which means we assume that the priors $p(y)$ are equal for each possible value of y.

In [28]:
```
# Calculate class conditional probabilities for given input data
vector

def predict_one(x,parameters,unique_y,prior = True):
    probabilities = []
    for key in parameters.keys():
        probabilities.append(calculateProbability(x,parameters[ke
y]['mean'],parameters[key]['std'])*(parameters[key]['prior']**(fl
oat(prior))))
    probabilities = np.array(probabilities)
    return unique_y[np.argmax(probabilities)]

def getPredictions(X, parameters, unique_y,prior=True):
    predictions = []
    for i in range(X.shape[0]):
        predictions.append(predict_one(X[i],parameters,unique_y,p
rior))
    return np.array(predictions)

# Get accuracy for test set

def getAccuracy(y, y_pred):
    correct = len(y[y==y_pred])
    return correct/y.size
```

Experiment

Here we load the diabetes dataset, split it into training and test data, train a Gaussian NB model, and test the model on the test set.

In [30]:
```
# Load dataset

filename = 'diabetes.csv'
dataset, headers = loadCsv(filename)
#print(headers)
#print(np.array(dataset)[0:5,:])

# Split into training and test

X_train,X_test,y_train,y_test = splitDataset(0.4,dataset[:,:-1],d
ataset[:,-1])
print("Total =",len(dataset),"Train =", len(X_train),"Test =",len
(X_test))

# Train model

parameters, unique_y = get_gaussian_parameters(X_train,y_train)
prediction = getPredictions(X_test,parameters,unique_y)
print("Accuracy with Prior =",getAccuracy(y_test,prediction))

# Test model

prediction = getPredictions(X_test,parameters,unique_y,prior = Fa
lse)
print("Accuracy without Prior =",getAccuracy(y_test,prediction))
```

```
Total = 768 Train = 460 Test = 308
Accuracy with Prior = 0.7564935064935064
Accuracy without Prior = 0.7597402597402597
```

Exercise In lab / take home work (20 points)

Find out the proportion of the records in your dataset are positive vs. negative. Can we conclude that $p(y=1) = p(y=0)$? If not, add the priors $p(y=1)$ and $p(y=0)$ to your NB model. Does it improve the result?

In [ ]:
```
# Code implement here
```

Explain that you can conclude that $p(y=1) = p(y=0)$? If not, add the priors $p(y=1)$ and $p(y=0)$ to your NB model. Does it improve the result? (double click to explain)

Example 2: Text classification

This example has been adapted from a post by Jaya Aiyappan, available at Analytics Vidhya (https://medium.com/analytics-vidhya/naive-bayes-classifier-for-text-classification-556fabaf252b#:~:text=The%20Naive%20Bayes%20classifier%20is,time%20and%20less%20training%20da

We will generate a small dataset of sentences that are classified as either "statements" or "questions."

We will assume that occurance and placement of words within a sentence is independent of each other (i.e., the features are conditionally independent given y). So the sentence "this is my book" is the same as "is this my book." We will treat words as case insensitive.

In [31]:
```
# Generate text data for two classes, "statement" and "question"

text_train = [['This is my novel book', 'statement'],
              ['this book has more than one author', 'statement
'],
              ['is this my book', 'question'],
              ['They are novels', 'statement'],
              ['have you read this book', 'question'],
              ['who is the novels author', 'question'],
              ['what are the characters', 'question'],
              ['This is how I bought the book', 'statement'],
              ['I like fictional characters', 'statement'],
              ['what is your favorite book', 'question']]

text_test = [['this is the book', 'statement'],
             ['who are the novels characters', 'question'],
             ['is this the author', 'question'],
            ['I like apples']]

# Load training and test data into pandas data frames

training_data = pd.DataFrame(text_train, columns= ['sentence', 'c
lass'])
print(training_data)
print('\n-----------------------------------------\n')
testing_data = pd.DataFrame(text_test, columns= ['sentence', 'cla
ss'])
print(testing_data)
```

```
                              sentence      class
0                This is my novel book  statement
1  this book has more than one author  statement
2                       is this my book   question
3                       They are novels  statement
4               have you read this book   question
5              who is the novels author   question
6               what are the characters   question
7        This is how I bought the book  statement
8          I like fictional characters  statement
9           what is your favorite book   question

-----------------------------------------

                         sentence      class
0                this is the book  statement
1  who are the novels characters   question
2              is this the author   question
3                   I like apples       None
```

In [32]:
```
# Partition training data by class

stmt_docs = [train['sentence'] for index,train in training_data.i
terrows() if train['class'] == 'statement']
question_docs = [train['sentence'] for index,train in training_da
ta.iterrows() if train['class'] == 'question']
all_docs = [train['sentence'] for index,train in training_data.it
errows()]

# Get word frequencies for each sentence and class

def get_words(text):
    # Initialize word list
    words = [];
    # Loop through each sentence in input array
    for text_row in text:
        # Check the number of words. Assume each word is separate
d by a blank space
        # so that the number of words is the number of blank spac
es + 1
        number_of_spaces = text_row.count(' ')
        # loop through the sentence and get words between blank s
paces.
        for i in range(number_of_spaces):
            # Check for for last word
            words.append([text_row[:text_row.index(' ')].lower
()])
            text_row = text_row[text_row.index(' ')+1:]
            i = i + 1
        words.append([text_row])
    return np.unique(words)

# Get frequency of each word in each document

def get_doc_word_frequency(words, text):
    word_freq_table = np.zeros((len(text),len(words)), dtype=int)
    i = 0
    for text_row in text:
        # Insert extra space between each pair of words to preven
t
        # partial match of words
        text_row_temp = ''
        for idx, val in enumerate(text_row):
            if val == ' ':
                 text_row_temp = text_row_temp + '  '
            else:
                  text_row_temp = text_row_temp + val.lower()
        text_row = ' ' + text_row_temp + ' '
        j = 0
        for word in words:
            word = ' ' + word + ' '
            freq = text_row.count(word)
            word_freq_table[i,j] = freq
            j = j + 1
        i = i + 1

    return word_freq_table
```

In [33]:
```
# Get word frequencies for statement documents

word_list_s = get_words(stmt_docs)
word_freq_table_s = get_doc_word_frequency(word_list_s, stmt_doc
s)
tdm_s = pd.DataFrame(word_freq_table_s, columns=word_list_s)
print(tdm_s)
```

```
   are  author  book  bought  characters  fictional  has  how  i
is  like  \
0    0       0     1       0           0          0    0    0  0
1     0
1    0       1     1       0           0          0    1    0  0
0     0
2    1       0     0       0           0          0    0    0  0
0     0
3    0       0     1       1           0          0    0    1  1
1     0
4    0       0     0       0           1          1    0    0  1
0     1

   more  my  novel  novels  one  than  the  they  this
0     0   1      1       0    0     0    0     0     1
1     1   0      0       0    1     1    0     0     1
2     0   0      0       1    0     0    0     1     0
3     0   0      0       0    0     0    1     0     1
4     0   0      0       0    0     0    0     0     0
```

In [10]:
```
# Get word frequencies over all statement documents

freq_list_s = word_freq_table_s.sum(axis=0)
freq_s = dict(zip(word_list_s,freq_list_s))
print(freq_s)
```

```
{'are': 1, 'author': 1, 'book': 3, 'bought': 1, 'characters': 1,
'fictional': 1, 'has': 1, 'how': 1, 'i': 2, 'is': 2, 'like': 1, '
more': 1, 'my': 1, 'novel': 1, 'novels': 1, 'one': 1, 'than': 1,
'the': 1, 'they': 1, 'this': 3}
```

In [34]:
```
# Get word frequencies for question documents

word_list_q = get_words(question_docs)
word_freq_table_q = get_doc_word_frequency(word_list_q, question_
docs)
tdm_q = pd.DataFrame(word_freq_table_q, columns=word_list_q)
print(tdm_q)
```

```
   are  author  book  characters  favorite  have  is  my  novels
read  the  \
0    0       0     1           0         0     0   1   1       0
0    0
1    0       0     1           0         0     1   0   0       0
1    0
2    0       1     0           0         0     0   1   0       1
0    1
3    1       0     0           1         0     0   0   0       0
0    1
4    0       0     1           0         1     0   1   0       0
0    0

   this  what  who  you  your
0     1     0    0    0     0
1     1     0    0    1     0
2     0     0    1    0     0
3     0     1    0    0     0
4     0     1    0    0     1
```

In [35]:
```
# Get word frequencies over all question documents

freq_list_q = word_freq_table_q.sum(axis=0)
freq_q = dict(zip(word_list_q,freq_list_q))
print(freq_q)
print(freq_list_s)
print(freq_list_q)
```

```
{'are': 1, 'author': 1, 'book': 3, 'characters': 1, 'favorite':
1, 'have': 1, 'is': 3, 'my': 1, 'novels': 1, 'read': 1, 'the': 2,
'this': 2, 'what': 2, 'who': 1, 'you': 1, 'your': 1}
[1 1 3 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 3]
[1 1 3 1 1 1 3 1 1 1 2 2 2 1 1 1]
```

In [36]:
```
# Get word probabilities for statement class
a = 1
prob_s = []
for count in freq_list_s:
    #print(word, count)
    prob_s.append((count+a)/(sum(freq_list_s)+len(freq_list_s)*
a))
prob_s.append(a/(sum(freq_list_s)+len(freq_list_s)*a))

# Get word probabilities for question class

prob_q = []
for count in freq_list_q:
    prob_q.append((count+a)/(sum(freq_list_q)+len(freq_list_q)*
a))
prob_q.append(a/(sum(freq_list_q)+len(freq_list_q)*a))


print('Probability of words for "statement" class \n')
print(dict(zip(word_list_s, prob_s)))
print('------------------------------------------ \n')
print('Probability of words for "question" class \n')
print(dict(zip(word_list_q, prob_q)))
```

```
Probability of words for "statement" class

{'are': 0.043478260869565216, 'author': 0.043478260869565216, 'bo
ok': 0.08695652173913043, 'bought': 0.043478260869565216, 'charac
ters': 0.043478260869565216, 'fictional': 0.043478260869565216, '
has': 0.043478260869565216, 'how': 0.043478260869565216, 'i': 0.0
6521739130434782, 'is': 0.06521739130434782, 'like': 0.0434782608
69565216, 'more': 0.043478260869565216, 'my': 0.04347826086956521
6, 'novel': 0.043478260869565216, 'novels': 0.043478260869565216,
'one': 0.043478260869565216, 'than': 0.043478260869565216, 'the':
0.043478260869565216, 'they': 0.043478260869565216, 'this': 0.086
95652173913043}
------------------------------------------

Probability of words for "question" class

{'are': 0.05128205128205128, 'author': 0.05128205128205128, 'book
': 0.10256410256410256, 'characters': 0.05128205128205128, 'favor
ite': 0.05128205128205128, 'have': 0.05128205128205128, 'is': 0.1
0256410256410256, 'my': 0.05128205128205128, 'novels': 0.05128205
128205128, 'read': 0.05128205128205128, 'the': 0.0769230769230769
3, 'this': 0.07692307692307693, 'what': 0.07692307692307693, 'who
': 0.05128205128205128, 'you': 0.05128205128205128, 'your': 0.051
28205128205128}
```

In [37]:
```
# Calculate prior for one class

def prior(className):
    denominator = len(stmt_docs) + len(question_docs)

    if className == 'statement':
        numerator =  len(stmt_docs)
    else:
        numerator =  len(question_docs)

    return np.divide(numerator,denominator)

# Calculate class conditional probability for a sentence

def classCondProb(sentence, className):
    words = get_words(sentence)
    prob = 1
    for word in words:
        if className == 'statement':
            idx = np.where(word_list_s == word)
            prob = prob * prob_s[np.array(idx)[0,0]]
        else:
            idx = np.where(word_list_q == word)
            prob = prob * prob_q[np.array(idx)[0,0]]

    return prob

# Predict class of a sentence

def predict(sentence):
    prob_statement = classCondProb(sentence, 'statement') * prior
('statement')
    prob_question = classCondProb(sentence, 'question') * prior('
question')
    if  prob_statement > prob_question:
        return 'statement'
    else:
        return 'question'
```

In-lab exercise: Laplace smoothing

Run the code below and figure out why it fails.

When a word does not appear with a specific class in the training data, its class-conditional probability is 0, and we are unable to get a reasonable probability for that class.

Research Laplace smoothing, and modify the code above to implement Laplace smoothing (setting the frequency of all words with frequency 0 to a frequency of 1). Run the modified code on the test set.

In [38]:
```
test_docs = list([test['sentence'] for index,test in testing_dat
a.iterrows()])
print('Getting prediction for %s"' % test_docs[0])
predict(test_docs[0])
```

```
Getting prediction for this is the book"

---------------------------------------------------------------------
----------
IndexError                                Traceback (most recent
call last)
<ipython-input-38-1f9ebe5cd5e4> in <module>
      1 test_docs = list([test['sentence'] for index,test in test
ing_data.iterrows()])
      2 print('Getting prediction for %s"' % test_docs[0])
----> 3 predict(test_docs[0])

<ipython-input-37-1e60a1384353> in predict(sentence)
     29
     30 def predict(sentence):
---> 31     prob_statement = classCondProb(sentence, 'statement')
* prior('statement')
     32     prob_question = classCondProb(sentence, 'question') *
prior('question')
     33     if  prob_statement > prob_question:

<ipython-input-37-1e60a1384353> in classCondProb(sentence, classN
ame)
     19         if className == 'statement':
     20             idx = np.where(word_list_s == word)
---> 21             prob = prob * prob_s[np.array(idx)[0,0]]
     22         else:
     23             idx = np.where(word_list_q == word)

IndexError: index 0 is out of bounds for axis 1 with size 0
```

Exercise 1.1 (10 points)

Explain Why it failed and explain how to solve the problem.

Explanation here! (Double click to explain)

Exercise 1.2 (20 points)

Modify your code and make it works.

In [39]:
```
### BEGIN SOLUTION
# Calculate prior for one class
def prior(className):
    denominator = len(stmt_docs) + len(question_docs)

    if className == 'statement':
        numerator =  len(stmt_docs)
    else:
        numerator =  len(question_docs)

    return np.divide(numerator,denominator)

# Calculate class conditional probability for a sentence

def classCondProb(sentence, className):
    words = get_words([sentence])
    prob = 1
    for word in words:
        if className == 'statement':
            try:
                idx = np.argwhere(word_list_s == word)
                prob = prob * prob_s[idx[0,0]]
            except:
                prob = prob * prob_s[-1]
        else:
            try:
                idx = np.argwhere(word_list_q == word)
                prob = prob * prob_q[idx[0,0]]
            except:
                prob = prob * prob_q[-1]
    return prob

# Predict class of a sentence

def predict(sentence):
    prob_statement = classCondProb(sentence, 'statement') * prior
('statement')
    prob_question = classCondProb(sentence, 'question') * prior('
question')
    if  prob_statement > prob_question:
        return 'statement'
    else:
        return 'question'
### END SOLUTION
```

In [40]:
```
# Test function: Do not remove
test_docs = list([test['sentence'] for index,test in testing_dat
a.iterrows()])

for sentence in test_docs:
    print('Getting prediction for %s"' % sentence)
    print(predict(sentence))

print("success!")
# End Test function
```

```
Getting prediction for this is the book"
question
Getting prediction for who are the novels characters"
question
Getting prediction for is this the author"
question
Getting prediction for I like apples"
statement
success!
```

Expect result:\ Getting prediction for this is the book"\ question\ Getting prediction for who are the novels characters"\ question\ Getting prediction for is this the author"\ question

Take home exercise

Find a more substantial text classification dataset, clean up the documents, and build your NB classifier. Write a brief report on your in-lab and take home exercises and results.

In [ ]:
```

```