An Introduction to Naïve Bayes

Introduction

In Part 1 and Part 2 of my series, we have created conclusions about data such as "This data instance is in this class!" The next question will be "What if we assign a probability to a data instance belonging to a given class?" The answer will be discussed in the this tip.

Background

Before jumping to Python code section, we need to glance a bit of knowledge about probability theory. You can refer to links list in my references section at the end of this tip. Here are summary of my reading.

In classification tasks, our job is to build a function

$$\hat{Y} = g(X)$$

that takes in a vector of features X (also called "inputs") and predicts a label Y (also called the "class" or "output"). A natural way to define this function is to predict the label with the highest conditional probability, that is choose:

$$g(\mathbf{X}) = \arg\max_y P(Y = y|\mathbf{X})$$

To help with computing the probabilities $P(Y = y|X)$, you are able to use training data consisting of examples of feature–label pairs (X,Y). You are given N of these pairs: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(N)}, y^{(N)})$, where $x^{(i)}$ is a vector of m discrete features for the ith training example and $y^{(i)}$ is the discrete label for the ith training example.

We assume that all labels are binary ($\forall i.\ y^{(i)} \in \{0,1\}$) and all features are binary ($\forall i,j.\ x_j^{(i)} \in \{0,1\}$).

Training

The objective in training is to estimate the probabilities $P(Y)$ and $P(X_j|Y)$ for all $2 \geq j \leq m$ features. Using an MLE estimate:

$$\hat{p}(X_j = x_j|Y = y) = \frac{(\#\text{ training examples where } X_j = x_j \text{ and } Y = y)}{(\text{training examples where } Y = y)}$$

Using a Laplace MAP estimate:

$$\hat{p}(X_j = x_j | Y = y) = \frac{(\#\text{ training examples where } X_j = x_j \text{ and } Y = y) + 1}{(\text{training examples where } Y = y) + 2}$$

Prediction

For an example with $x = [x_1, x_2, \ldots, x_m]$, estimate the value of y as:

$$\begin{aligned}
\hat{y} = g(\mathbf{X}) &= \arg\max_y \hat{P}(Y)\hat{P}(\mathbf{X}|Y) && \text{this is equal to } \arg\max_y \hat{P}(Y = y|\mathbf{X}) \\
&= \arg\max_y \hat{p}(Y = y) \prod_{j=1}^{m} \hat{p}(X_j = x_j | Y = y) && \text{Naïve Bayes assumption} \\
&= \arg\max_y \left(\log \hat{p}(Y = y) + \sum_{j=1}^{m} \log \hat{p}(X_j = x_j | Y = y) \right) && \text{log version for numerical stability}
\end{aligned}$$

Assume that we are building a filtering function for our website. The purpose of this function is to filter comments that are submitted by users in two classes: Abusive and NOTAbusive. That is, we are given a set of m comments x_i, denoted by $X := \{ x_1, \ldots , x_m\}$ and associated labels y_i, denoted by

$Y := \{ y_1; \ldots; y_m\}$. Here the labels satisfy $y_i \in$ {Abusive, NOTAbusive}.

We also notive that, comments are text. One way of converting text into a vector is by using the so-called **bag of words** representation. In its simplest version it works as follows: Assume we have a list of all possible words occurring in X, that is a dictionary, then we are able to assign a unique number with each of those words (e.g. the position in the dictionary). Now we may simply count for each document x_i the number of times a given word j is occurring. This is then used as the value of the j-th coordinate of x_i.

In the context of our filtering function, the actual text of the comment x corresponds to the test and the label y is equivalent to the diagnosis. Using Bayes rule:

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

we may now treat the occurrence of each word in a document as a separate test and combine the outcomes in a naive fashion by assuming that:

$$p(x|y) = \prod_{j=1}^{\#\text{ of words in } x} p(w^j|y)$$

Now, we are going to use Python code for our fitering function.

Using the code

The first our task is to prepare data set and vectors.

Asume that, we have collected comments that are submitted by users and have put them into two files: **NOTAbusive.txt** and **Abusive.txt**. The **NOTAbusive.txt** file contains comments are not abusive as follows:

```
My dog has flea problems, help please!

My dalmation is so cute. I love him!

Mr licks ate my steak. How to stop him?
```

The **Abusive.txt** file contains comments are abusive as follows:

```
May be not take him to dog park stupid.

Stop posting stupid worthless garbage!

Quit buying worthless dog food stupid.
```

If we have a text string, we can split it using the Python **split**() method and use regular expressions to split up the sentence on anything that isn't a word or number. We also can count the length of each string, return only the items greater than 0 and convert strings to all lowercase by using **lower**(). The following **textParse** function is created for all of tasks above:

```
def textParse(bigString):

        listOfTokens = re.split(r'\W*', bigString)

        return [tok.lower() for tok in listOfTokens if len(tok) >0]
```

We create a function to return two variables. The first variable (docList) is a set of documents and the text has been broken up into a set of words (also called is **tokens**). The second variable (classList) is a set of class labels. Here you have two classes, Abusive – assigned the 1 label and NOTAbusive assigned the label 0. The following loadDataSet is our function:

```
def loadDataSet():

        docList=[]; classList = [];

        wordList = textParse(open('NOTabusive.txt').read())

        docList.append(wordList)

        classList.append(0)

        wordList = textParse(open('Abusive.txt').read())

        docList.append(wordList)

        classList.append(1)
```

```
    return docList,classList
```

If we implement the following statement:

```
listOPosts, listClasses = loadDataSet()

print(listOPosts)

print(listClasses)
```

listOPosts can look like this:

```
[['my', 'dog', 'has', 'flea', 'problems', 'help', 'please', 'my', 'dalmation', 'is', 'so', 'cute', 'i', 'love', 'him', 'mr',
'licks', 'ate', 'my', 'steak', 'how', 'to', 'stop', 'him'], ['may', 'be', 'not', 'take', 'him', 'to', 'dog', 'park',
'stupid', 'stop', 'posting', 'stupid', 'worthless', 'garbage', 'quit', 'buying', 'worthless', 'dog', 'food', 'stupid']]
```

Notes that, we have two lists: the first list is the NOT Abusive list and the second one is the Abusive list.

listClasses can look like this:

```
[0, 1]
```

Next, we create the function **createVocabList**() that will create a list of all the unique words in all

of our documents:

```
def createVocabList(dataSet):

    # Create an empty set

    vocabSet = set([])

    for document in dataSet:

        # Create the union of two sets

        vocabSet = vocabSet | set(document)

    return list(vocabSet)
```

if you run the following code:

```
listOPosts,listClasses = loadDataSet()

myVocabList = createVocabList(listOPosts)

print(myVocabList )
```

the result will look like this:

```
['garbage', 'to', 'posting', 'i', 'him', 'mr', 'problems', 'help', 'stop', 'dog', 'has', 'my', 'food', 'ate',
'dalmation', 'be', 'may', 'so', 'take', 'worthless', 'steak', 'love', 'how', 'park', 'not', 'cute', 'stupid', 'is', 'licks',
'flea', 'please', 'quit', 'buying']
```

We also need to create the **bagOfWords2Vec** function. This function will create **a bag of words** can have multiple occurrences of each word (whereas a set of words can have only one occurrence of each word). The lines of Python code for the **bagOfWords2Vec** function:

```
def bagOfWords2Vec(vocabList, inputSet):
    returnVec = [0]*len(vocabList)
    for word in inputSet:
        if word in vocabList:
            returnVec[vocabList.index(word)] += 1
    return returnVec
```

we can test this function by using the following lines of Python code:

```
listOPosts,listClasses = loadDataSet()
myVocabList = createVocabList(listOPosts)
testEntry = ['help','cute']
thisDoc = bagOfWords2VecMN(myVocabList, testEntry)
print(thisDoc)
```

the result is a vector that will look like this:

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

We can see that, both the word **help** and the word **cute** appear once in our vocabulary.

After preparing data set and vectors, the next task is to create a training function called **trainNB**. This function will calculate probability of a word w_i in the 0 class (NOTAbusive class), $p(w_i|0)$, and in the 1 class (Abusive class), $p(w_i|1)$. This function is created based on the following formula (log version):

$$\hat{p}(X_j = x_j | Y = y) = \frac{(\#\text{ training examples where } X_j = x_j \text{ and } Y = y) + 1}{(\text{training examples where } Y = y) + 2}$$

The Python code of the **trainNB** looks like this:

```
def trainNB(trainMatrix,trainCategory):
    numTrainDocs = len(trainMatrix)
    numWords = len(trainMatrix[0])
    pAbusive = sum(trainCategory)/float(numTrainDocs)
    p0Num = ones(numWords)
    p1Num = ones(numWords)
```

```
        p0Denom = 2.0
        p1Denom = 2.0
        for i in range(numTrainDocs):
                if trainCategory[i] == 1:
                        p1Num += trainMatrix[i]
                        p1Denom += sum(trainMatrix[i])
                else:
                        p0Num += trainMatrix[i]
                        p0Denom += sum(trainMatrix[i])
        p1Vect = log(p1Num/p1Denom) #change to log()
        p0Vect = log(p0Num/p0Denom) #change to log()
        return p0Vect,p1Vect,pAbusive
```

As we can see above, the **trainNB** function takes a matrix of documents (abusive document and not abusive document), **trainMatrix**, and a vector with the class labels for each of the documents, **trainCategory**. The trainNB returns p0Vect – the vector contains probability values of words in a NOT Abusive document ($p(w_i|0)$s), p1Vect - the vector contains probability values of words in an Abusive document ($p(w_i|1)$s), and pAbusive or p(1). Becauce we have two classes, we can get pNOTAbusive or p(0) by 1 – p(1).

So far, we can test this function by using the following lines of code:

```
listOPosts,listClasses = loadDataSet()
myVocabList = createVocabList(listOPosts)
trainMat=[]
for postinDoc in listOPosts:
        trainMat.append(bagOfWords2Vec(myVocabList, postinDoc))
p0V,p1V,pAb = trainNB(array(trainMat),array(listClasses))
print("p0V = ", p0V)
print("p1V = ", p1V)
print("pAb = ", pAb)
```

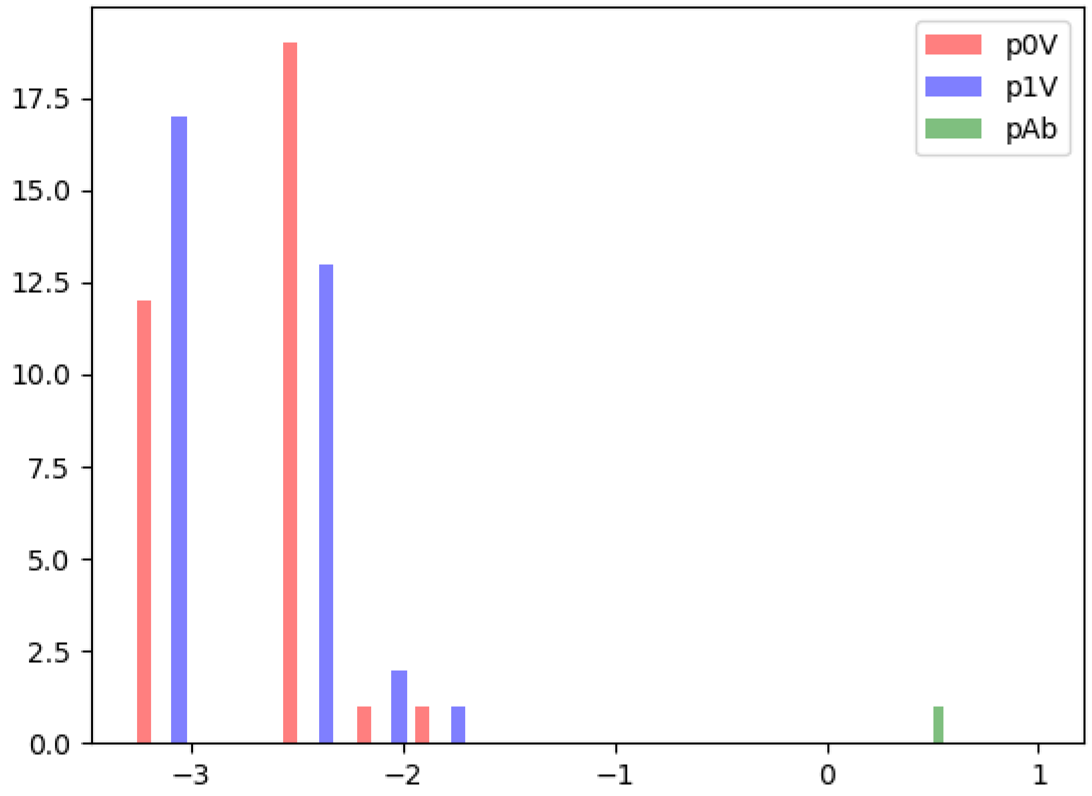
The result:

```
p0V =  [-2.56494936 -2.56494936 -3.25809654 -3.25809654 -2.56494936 -3.25809654
 -2.56494936 -2.56494936 -2.56494936 -2.56494936 -2.56494936 -2.56494936
 -2.56494936 -3.25809654 -2.56494936 -2.56494936 -2.56494936 -2.56494936
 -2.15948425 -2.56494936 -2.56494936 -3.25809654 -3.25809654 -2.56494936
 -3.25809654 -3.25809654 -1.87180218 -3.25809654 -3.25809654 -2.56494936
 -3.25809654 -3.25809654 -2.56494936]
p1V =  [-3.09104245 -3.09104245 -2.39789527 -2.39789527 -2.39789527 -2.39789527
 -3.09104245 -3.09104245 -3.09104245 -3.09104245 -2.39789527 -3.09104245
 -1.99243016 -1.70474809 -3.09104245 -3.09104245 -3.09104245 -3.09104245
 -2.39789527 -3.09104245 -3.09104245 -2.39789527 -2.39789527 -3.09104245
 -2.39789527 -2.39789527 -3.09104245 -2.39789527 -1.99243016 -3.09104245
 -2.39789527 -2.39789527 -3.09104245]
pAb =  0.5
```

We also make a histogram by using matplotlib (https://plot.ly/matplotlib/histograms/):

```
import matplotlib.pyplot as plt
…
legend = ['p0V','p1V','pAb']
plt.hist(p0V,20,color='red',alpha=0.5)
plt.hist(p1V,20,color='blue',alpha=0.5)
plt.hist(pAb,20,color='green',alpha=0.5)
plt.legend(legend)
plt.show()
```

The result:



After training, we can predict easily what class a word belongs to by using formula:

$$\hat{y} = g(\mathbf{X}) = \arg\max_{y} \hat{P}(Y)\hat{P}(\mathbf{X}|Y) \qquad \text{this is equal to } \arg\max_{y} \hat{P}(Y = y|\mathbf{X})$$

$$= \arg\max_{y} \hat{p}(Y = y) \prod_{j=1}^{m} \hat{p}(X_j = x_j|Y = y) \qquad \text{Naïve Bayes assumption}$$

$$= \arg\max_{y} \left(\log \hat{p}(Y = y) + \sum_{j=1}^{m} \log \hat{p}(X_j = x_j|Y = y) \right) \qquad \text{log version for numerical stability}$$

In short, we have a comment as follows:

Stop him, please!

We can easily calculate p(stop|0), p(him|0), p(please|0), p(stop|1), p(him|1) p(please|1) by using the **trainNB** function. And then, we can calculate:

Probability of the comment in a NOT Abusive document (P0):

P0 = log(pNOTAbusive) + log (p(stop|0)) + log (p(him|0)) + log (p(please|0))

Probability of the comment in an Abusive document (P1):

P1 = log(pAbusive) + log (p(stop|1)) + log (p(him|1)) + log (p(please|1))

When we have P1 and P0, We can use the following rules:

- If P1 > P0 then the comment belongs to 1 class
- Else the comment belongs to 0 class

From all of above, we can create a predicting function called classifyNB as follows:

```
def classifyNB(vec2Classify, p0Vec, p1Vec, pClass1):
    p1 = sum(vec2Classify * p1Vec) + log(pClass1)
    p0 = sum(vec2Classify * p0Vec) + log(1.0 - pClass1)
    if p1 > p0:
        return 1
    else:
        return 0
```

If you want to know values of p0 and p1, we can create another version of the classifyNB:

```
def classifyNB1(vec2Classify, p0Vec, p1Vec, pClass1):
    p1 = sum(vec2Classify * p1Vec) + log(pClass1)
    p0 = sum(vec2Classify * p0Vec) + log(1.0 - pClass1)
    return p0, p1
```

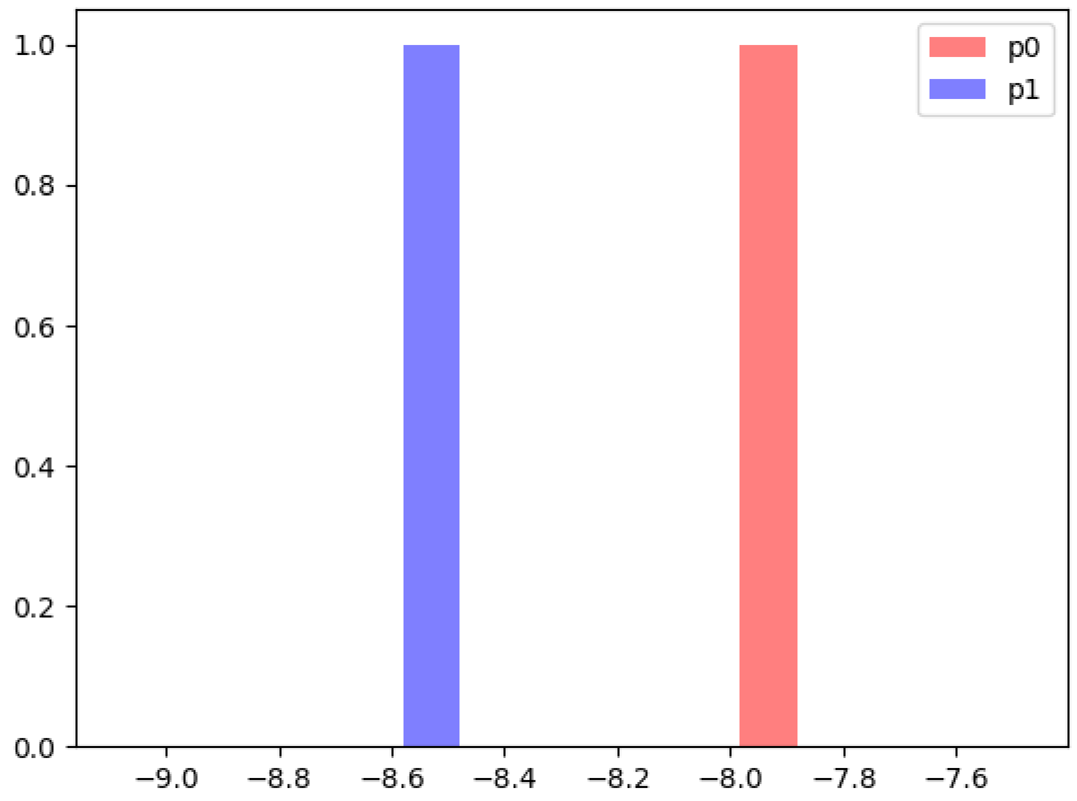
We can test this function:

```
listOPosts,listClasses = loadDataSet()
myVocabList = createVocabList(listOPosts)
trainMat=[]
for postinDoc in listOPosts:
    trainMat.append(bagOfWords2Vec(myVocabList, postinDoc))
p0V,p1V,pAb = trainNB(array(trainMat),array(listClasses))
testEntry = ['stop', 'him', 'please']
thisDoc = array(bagOfWords2Vec(myVocabList, testEntry))
print('{0} belongs to:{1} class '.format(testEntry,classifyNB(thisDoc,p0V,p1V,pAb)))
```

The result:

```
['stop', 'him', 'please'] belongs to:0 class
```

And we can see histogram of p0 and p1:



Clearly, we can see p0 > p1, so our comment will belongs to 0 class.

Points of interest

In this tip, we have used probability for our classification tasks. Using probabilities can sometimes be more effective than using hard rules for classification. Bayesian probability and Bayes' rule gives us a way to estimate unknown probabilities from known values. You can view all of source code in this tip here (https://github.com/TranNgocMinh/Machine-Learning-and-Tensorflow/tree/master/naiveBayes).

References

http://www.inf.u-szeged.hu/~ormandi/ai2/06-naiveBayes-example.pdf

https://web.stanford.edu/class/cs124/lec/naivebayes.pdf

https://web.stanford.edu/class/archive/cs/cs109/cs109.1178/lectureHandouts/210-naive-bayes.pdf

https://www.python-course.eu/naive_bayes_classifier_introduction.php