Assignment № 1

Bishnu Sarker, Khulna Univesity of Engineering Technology 26/10/2016

Naive Bayes Classifier

Naive Bayes classifier is one of the simplest and popular probabilistic machine learning techniques. Naive Bayes classifier works based on Bayesian Theorem.

According to Bayes theorem, $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$

To use this theorem in machine learning, let's define a classification problem: Given, a data point d, and a set of classes $C = c_1, c_2, c_3, .., c_n$, the likelihood of d being belongs to the any of the classes is P(C|d).
To expand this using Bayes theorem, we have:

$$P(C|d) = \frac{P(d|C)P(C)}{P(d)}$$

P(d) is same for all of the classes. Therefore, we can omit the this and we have following equation:

$$P(C|d) = P(d|C)P(C)$$

To decide on appropriate class, we have to take class for which P(C|d) is maximum. So the problem can be reduced to the follwoing form:

$$argmax_C P(C|d)$$
$$= argmax_C P(d|C)P(C)$$

Lets split the document in it's constituent tokens such as $d = w_1 w_2 w_3 w_4 w_5 .. w_n$

$$= argmax_C P(d = w_1 w_2 w_3 w_4 w_5 .. w_n|C)P(C)$$

Assuming terms are independent, we have:

$$= argmax_C P(C)P(w_1|C)P(w_2|C)P(w_3|C)..P(w_n|C)$$

$= argmax_C P(C) \prod_{w_i \in V} P(w_i|C)$, V is the Vocabulary

$= argmax_C \log P(C) + \sum_{w_i \in V} \log P(w_i|C)$, V is the Vocabulary

We can compute the class probability using following formula:

$$P(C) = \frac{N_c}{N}$$
$$P(w_i|C) = \frac{count(w_i, c) + 1}{V + \sum_{w_i \in V} count(w_i, C)}$$

Here is the a python code for working with Naive Bayes:

Implementation

Listing 1: Sample Python code – Naive Bayes for Product Review Analysis.

```python
'''
Author: Bishnu Sarker, dept. of Computer Sciecne and Engineering, Khulna University of Engineering & Technology
'''
import random
import math

def write_object(object,filename):
    '''
    To write a python list object in external file
    :param object: Object prefereably a List object to be written in a file
    :param filename: Filename in which data will be written
    :return: NA
    '''
    with open(filename,'w') as W:
        for item in object:
            W.write(str(item)+'\n')

def read_review(filename):
    '''
    Reading raw review and prepraring dataset seperating pos and neg data
    :param filename: Dataset file name
    :return:tuple, positive dataset and negative dataset
    '''
    pos_dataset=[]
    neg_dataset=[]
    with open(filename) as R:
        for line in R:
            #print(line)
            line=line.split()
            if line[0]=="Neg":
                neg_dataset.append(list(line[1:]))
            else:
                pos_dataset.append(list(line[1:]))
    write_object(pos_dataset,"positve.txt")
    write_object(neg_dataset,"negative.txt")
    return (pos_dataset,neg_dataset)

def split_test_train(pos, neg):
    '''
    Splits the datasets into train and test set
```

```
    :param pos: positive dataset
    :param neg:  negative dataset
    :return:  postive training, negative training, positive testing and negative training
    '''
    pl=len(pos)
    nl=len(neg)

    pSample_train=random.sample(pos,int(pl*0.80))
    nSample_train=random.sample(neg,int(nl*0.80))
    pSample_test=random.sample(pos,int(pl*0.20))
    nSample_test=random.sample(neg,int(nl*0.20))

    return (pSample_train,nSample_train,pSample_test,nSample_test)

def build_Vocab(pTrain,nTrain):
    '''
    Building a vocabulary  V from taining dataset
    :param pTrain: positive training dataset
    :param nTrain: negative training dataset
    :return: Vocabulary of the form {"Token":(p, n)} p for positve frequency and n for negative frequency
    '''
    Vocabulary={}
    for sample in pTrain:
        for word in sample:

            word=word.lower().strip("'!.:)(?-")
            if word == "":
                continue
            if Vocabulary.__contains__(word):
                freq,T=Vocabulary[word]
                freq=freq+1
                Vocabulary[word]=(freq,T)
            else:
                Vocabulary.__setitem__(word,(1,0))
    for sample in nTrain:

        for word in sample:
            word = word.lower().strip("'!.:)(?-[]+")
            if word=="":
                continue
            if Vocabulary.__contains__(word):
                T,freq=Vocabulary[word]
                freq=freq+1
                Vocabulary[word]=(T,freq)
            else:
```

```
                Vocabulary.__setitem__(word,(0,1))
    print_dic(Vocabulary)
    return Vocabulary

def Maxmimum_likelihood_Estimation(pTrain,nTrain):
    '''
    Compute the required probabilities using maximum likelihood estimation.
    :param pTrain:
    :param nTrain:
    :return:
    '''

    pN=len(pTrain)
    nN=len(nTrain)
    N=pN+nN
    pCount=0
    nCount=0
    vocab=build_Vocab(pTrain,nTrain)
    V=len(vocab)
    for words in vocab:
        pCount=pCount+vocab[words][0]
        nCount=nCount+vocab[words][1]
    return {"vocab":vocab,"pN":pN,"nN":nN,"N":N,"V":V,"pC":pCount,"nC":nCount}


def NB_classifier(test_sample,Vocab,pN,nN,N,V,pCount,nCount):
    '''
    Naive bayes classifier
    :param test_sample:
    :param Vocab:
    :param pN:
    :param nN:
    :param N:
    :param V:
    :param pCount:
    :param nCount:
    :return:
    '''
    #tokens=test_sample.strip().split()
    N=float(N)
    prob_pos=pN/N
    prob_neg=nN/N
    pos_prob=0.0
    neg_prob=0.0
    for token in test_sample:
```

```python
        token=token.lower().strip("'!.:)(?-[]+")
        #computing p(token|pos)
        if token=="":
            continue
        if Vocab.__contains__(token):
            pos_prob=pos_prob+math.log10(Vocab[token][0]+1.0/(pCount+V))
            neg_prob=neg_prob+math.log10(Vocab[token][1]+1.0/(nCount+V))
        else:
            pos_prob = pos_prob + math.log10( 1.0 / (pCount + V))
            neg_prob = neg_prob + math.log10( 1.0 / (nCount + V))

    pP=math.log10(prob_pos)+pos_prob
    nP=math.log10(prob_neg)+neg_prob
    return (pP,nP)

def testing(pTest,nTest,Param):
    '''
    Find the probabale class for a test sample
    :param pTest:
    :param nTest:
    :param Param:
    :return:
    '''
    with open("Rsult.txt",'w') as W:
        for item in pTest:
            pP,nP=NB_classifier(item,Param["vocab"],Param["pN"],Param["nN"],Param["N"],Param["V"],Param["pC"],Param["nC"])
            print("Prediction-->\tpos:"+str(pP)+" \tNeg:"+str(nP)+" \tActual-->Positive")
            W.write("Prediction--> pos:"+str(pP)+" \tNeg:"+str(nP)+" \tActual-->Positive\t"+str(item)+'\n')
        for item in nTest:
            pP, nP = NB_classifier(item,Param["vocab"],Param["pN"],Param["nN"],Param["N"],Param["V"],Param["pC"],Param["nC"])
            print("Prediction--> pos:" + str(pP) + " Neg:" + str(nP)+" Actual-->Negative")
            W.write("Prediction--> pos:" + str(pP) + " \tNeg:" + str(nP) +" \tActual-->Negative\t"+str(item)+"\n")
def print_dic(D):
    with open("Vocab.txt",'w') as W:
        for d in D.keys():
            W.write(d+"-->"+" Pos:"+str(D[d][0])+" Neg:"+str(D[d][1])+'\n')

if __name__ == '__main__':
    pos,neg=read_review("AppleReview.txt")
    pos_train,neg_train,pos_test,neg_test=split_test_train(pos,neg)
```

```
    Params=Maxmimum_likelihood_Estimation(pos_train,neg_train)
    print (Params)
    testing(pos_test,neg_test,Params)
```

Code 1... shows a typical implementation of naive Bayes following the formula described above.

Tasks to be done

- Select A data set from following sources:
  - Twitter Data: http://www.sananalytics.com/lab/twitter-sentiment/
  - Movie Review Data https://www.cs.cornell.edu/people/pabo/movie-review-data/
  - Kaggle Competition: https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews
  - https://inclass.kaggle.com/c/si650winter11/data
- Experiemnt with given datasets and document the classification error to evaluate the model performance. The model performance is computed using confusion Matrix.

$$Error = \frac{NoofMissclassifiedexamples}{TotalNoofExamplesinthetestset}$$

Deliverable

A short report (max 3 page docx and pdf) and Code. Things that should be stated in your report:

- General Introduction
- Description of the datasets you have tried
- Experiment setup and Result Analysis
- Conclusion and discussion
- References