Ian Schweer

22514022

CS 177 - Naive Bayes

```
import numpy as np
import matplotlib.pyplot as plt

class params:
        _targets = []
        _data = []
        _classes = []
        _classProbs = []
        def __init__(self, Data, Classes, Targets):
                self._targets = np.array(Targets)
                self._data = np.array(Data)
                self._classes = np.array(Classes)
                self._classProbs = np.array([self.lenfrac(Data[Targ
ets==Classes[0]]) + 1, self.lenfrac(Data[Targets==Classes[1]]) +
1])

        # get frame
        def lenfrac(self, var):
                return 1. * len(var)
        def params(self, j, i):
                if (j < 0 or j >= self._data.shape[1]):
                        raise Exception("j out of bounds",j)
                return self._data[self._targets == i, j]

        def mprobs(self, j, k, i):
                var = self.params(j, i);
                return var[var == k]

        def getclass(self, c):
                return self._classProbs[self._classes==c][0]

        def classprobs(self, c):
                return self.getclass(c) / len(self._data)

        def pofx(self, x):
                # given a vector of x values
                # compute the probability of the value
                # then multiply them all together.
                px = 1.0
                for i, xj in enumerate(x):
                        X = self._data[:,i]
                        counter = 0
                        for y in X:
                                counter = counter + (y == xj)
                        px = (counter / self.lenfrac(X))
                return px

        def cprobs(self, j, k, i):
                denom = self.getclass(i)
                head = self.lenfrac(self.mprobs(j,k,i))
                # naive bayes assumes conditional independence. So
P[X|Y] = P(X)P(Y)
                return (head + .5) / self.getclass(i)
```

```
X = np.genfromtxt('data.txt')
Y = np.genfromtxt('labels.txt')

classes=[1,2]
p = params(X, classes, Y)
print "p(C = 1) =",p.getclass(classes[0]) / len(X)
print "p(C = 2) =",p.getclass(classes[1]) / len(X)

for c in classes:
        for i in range(0,X.shape[1]):
                ps = [p.cprobs(i, 1, c), p.cprobs(i, 2, c)]
                print "P(X =",i,"| C =",c,") [1, 2] = ", ps

#train using only a subset of data.
Xte,Yte = X[1500:], Y[1500:]
predictions = np.array([])
for i in [1500, 50, 10]:
        Xtr,Ytr = (X[0:i], Y[0:i])
        p = params(Xtr, classes, Ytr)
        for d in range(Xte.shape[0]):
                pxji = np.empty(2) # P(Xj|C=i)
                pxji.fill(1.0)
                for ci,c in enumerate(classes):
                        temp = 1.0
                        for ji,j in enumerate(Xte[d]):
                                temp = temp * p.cprobs(ji, j, c)
                        pxji[ci] = temp * p.classprobs(c)
                predictions=np.append(predictions, classes[pxji.arg
max()])
        print np.mean(predictions.reshape(Yte.shape) != Yte)
        predictions = np.array([])
```

```
p(C = 1) = 0.60617257118
p(C = 2) = 0.394262116931
P(X = 0 | C = 1 ) [1, 2] =  [0.85209752599498023, 0.14790247400501
971]
P(X = 1 | C = 1 ) [1, 2] =  [0.90193617784152025, 0.09806382215847
9738]
P(X = 2 | C = 1 ) [1, 2] =  [0.72266045177482974, 0.27733954822517
032]
P(X = 3 | C = 1 ) [1, 2] =  [0.99695231265686624, 0.00304768734313
37396]
P(X = 4 | C = 1 ) [1, 2] =  [0.77967013266403729, 0.22032986733596
271]
P(X = 5 | C = 1 ) [1, 2] =  [0.88580136249551811, 0.11419863750448
189]
P(X = 6 | C = 1 ) [1, 2] =  [0.98440301183219792, 0.01559698816780
208]
P(X = 7 | C = 1 ) [1, 2] =  [0.92631767658659014, 0.07368232341340
9828]
P(X = 8 | C = 1 ) [1, 2] =  [0.92165650770885621, 0.07834349229114
3776]
P(X = 9 | C = 1 ) [1, 2] =  [0.8295087845105773, 0.170491215489422
73]
P(X = 10 | C = 1 ) [1, 2] =  [0.94890641807099319, 0.0510935819290
06815]
P(X = 11 | C = 1 ) [1, 2] =  [0.5821082825385443, 0.41789171746145
57]
P(X = 12 | C = 1 ) [1, 2] =  [0.8807816421656508, 0.11921835783434
923]
P(X = 13 | C = 1 ) [1, 2] =  [0.95464324130512723, 0.0453567586948
72712]
P(X = 14 | C = 1 ) [1, 2] =  [0.98225170311939758, 0.0177482968806
02366]
P(X = 15 | C = 1 ) [1, 2] =  [0.90946575833632126, 0.0905342416636
78738]
P(X = 16 | C = 1 ) [1, 2] =  [0.90444603800645396, 0.0955539619935
46072]
P(X = 17 | C = 1 ) [1, 2] =  [0.8743277160272499, 0.12567228397275
01]
P(X = 18 | C = 1 ) [1, 2] =  [0.64126927214055218, 0.3587307278594
4782]
P(X = 19 | C = 1 ) [1, 2] =  [0.98296880602366443, 0.0170311939763
35603]
P(X = 20 | C = 1 ) [1, 2] =  [0.69325923269989242, 0.3067407673001
0758]
P(X = 21 | C = 1 ) [1, 2] =  [0.99193259232699893, 0.0080674076730
010754]
P(X = 22 | C = 1 ) [1, 2] =  [0.97221226245966297, 0.0277877375403
37039]
P(X = 23 | C = 1 ) [1, 2] =  [0.98045894585873072, 0.0195410541412
69273]
P(X = 24 | C = 1 ) [1, 2] =  [0.6269272140552169, 0.37307278594478
31]
P(X = 25 | C = 1 ) [1, 2] =  [0.71871638580136255, 0.2812836141986
3751]
```

```
P(X = 26 | C = 1 ) [1, 2] =  [0.72301900322696311, 0.2769809967730
3695]
P(X = 27 | C = 1 ) [1, 2] =  [0.84456794550017933, 0.1554320544998
2073]
P(X = 28 | C = 1 ) [1, 2] =  [0.87074220150591608, 0.1292577984940
8389]
P(X = 29 | C = 1 ) [1, 2] =  [0.83811401936177843, 0.1618859806382
2157]
P(X = 30 | C = 1 ) [1, 2] =  [0.89584080315525283, 0.1041591968447
4723]
P(X = 31 | C = 1 ) [1, 2] =  [0.92703477949085689, 0.0729652205091
43058]
P(X = 32 | C = 1 ) [1, 2] =  [0.87647902474005024, 0.1235209752599
498]
P(X = 33 | C = 1 ) [1, 2] =  [0.92631767658659014, 0.0736823234134
09828]
P(X = 34 | C = 1 ) [1, 2] =  [0.84241663678737899, 0.1575833632126
2101]
P(X = 35 | C = 1 ) [1, 2] =  [0.82520616708497674, 0.1747938329150
2332]
P(X = 36 | C = 1 ) [1, 2] =  [0.73879526712083188, 0.2612047328791
6818]
P(X = 37 | C = 1 ) [1, 2] =  [0.98153460021513084, 0.0184653997848
69128]
P(X = 38 | C = 1 ) [1, 2] =  [0.88436715668698462, 0.1156328433130
1542]
P(X = 39 | C = 1 ) [1, 2] =  [0.90982430978845463, 0.0901756902115
45353]
P(X = 40 | C = 1 ) [1, 2] =  [0.94711366081032633, 0.0528863391896
73719]
P(X = 41 | C = 1 ) [1, 2] =  [0.88472570813911799, 0.1152742918608
8203]
P(X = 42 | C = 1 ) [1, 2] =  [0.89584080315525283, 0.1041591968447
4723]
P(X = 43 | C = 1 ) [1, 2] =  [0.89942631767658654, 0.1005736823234
134]
P(X = 44 | C = 1 ) [1, 2] =  [0.70437432771602726, 0.2956256722839
7274]
P(X = 45 | C = 1 ) [1, 2] =  [0.83883112226604517, 0.1611688777339
5483]
P(X = 46 | C = 1 ) [1, 2] =  [0.98404446038006455, 0.0159555396199
35462]
P(X = 47 | C = 1 ) [1, 2] =  [0.93277160272499104, 0.0672283972750
08969]
P(X = 48 | C = 1 ) [1, 2] =  [0.81373252061670853, 0.1862674793832
915]
P(X = 49 | C = 1 ) [1, 2] =  [0.50143420580853348, 0.4985657941914
6646]
P(X = 50 | C = 1 ) [1, 2] =  [0.85675869487271428, 0.1432413051272
8578]
P(X = 51 | C = 1 ) [1, 2] =  [0.73198278953029761, 0.2680172104697
0239]
P(X = 52 | C = 1 ) [1, 2] =  [0.89548225170311935, 0.1045177482968
806]
```

```
P(X = 53 | C = 1 ) [1, 2] =  [0.91771244173538902, 0.0822875582646
10969]
P(X = 54 | C = 1 ) [1, 2] =  [0.66959483685908927, 0.3304051631409
1073]
P(X = 55 | C = 1 ) [1, 2] =  [0.70437432771602726, 0.2956256722839
7274]
P(X = 56 | C = 1 ) [1, 2] =  [0.65740408748655432, 0.3425959125134
4568]
P(X = 0 | C = 2 ) [1, 2] =  [0.64636163175303196, 0.35363836824696
804]
P(X = 1 | C = 2 ) [1, 2] =  [0.65518191841234841, 0.34481808158765
159]
P(X = 2 | C = 2 ) [1, 2] =  [0.38506063947078278, 0.61493936052921
716]
P(X = 3 | C = 2 ) [1, 2] =  [0.97822491730981254, 0.02177508269018
7433]
P(X = 4 | C = 2 ) [1, 2] =  [0.37458654906284455, 0.62541345093715
55]
P(X = 5 | C = 2 ) [1, 2] =  [0.6243109151047409, 0.375689084895259
1]
P(X = 6 | C = 2 ) [1, 2] =  [0.57855567805953689, 0.42144432194046
305]
P(X = 7 | C = 2 ) [1, 2] =  [0.65848952590959209, 0.34151047409040
791]
P(X = 8 | C = 2 ) [1, 2] =  [0.69377067254685776, 0.30622932745314
224]
P(X = 9 | C = 2 ) [1, 2] =  [0.54382579933847852, 0.45617420066152
148]
P(X = 10 | C = 2 ) [1, 2] =  [0.6871554575523704, 0.31284454244762
955]
P(X = 11 | C = 2 ) [1, 2] =  [0.37458654906284455, 0.6254134509371
555]
P(X = 12 | C = 2 ) [1, 2] =  [0.71306504961411243, 0.2869349503858
8757]
P(X = 13 | C = 2 ) [1, 2] =  [0.87238147739801541, 0.1276185226019
8456]
P(X = 14 | C = 2 ) [1, 2] =  [0.84151047409040791, 0.1584895259095
9206]
P(X = 15 | C = 2 ) [1, 2] =  [0.45452039691289969, 0.5454796030871
0031]
P(X = 16 | C = 2 ) [1, 2] =  [0.61549062844542446, 0.3845093715545
7554]
P(X = 17 | C = 2 ) [1, 2] =  [0.62045203969128992, 0.3795479603087
1002]
P(X = 18 | C = 2 ) [1, 2] =  [0.29796030871003309, 0.7020396912899
6691]
P(X = 19 | C = 2 ) [1, 2] =  [0.79189636163175303, 0.2081036383682
4697]
P(X = 20 | C = 2 ) [1, 2] =  [0.20424476295479604, 0.7957552370452
0401]
P(X = 21 | C = 2 ) [1, 2] =  [0.94735391400220503, 0.0526460859977
94925]
P(X = 22 | C = 2 ) [1, 2] =  [0.66786108048511572, 0.3321389195148
8422]
```

```
P(X = 23 | C = 2 ) [1, 2] =  [0.6243109151047409, 0.37568908489525
91]
P(X = 24 | C = 2 ) [1, 2] =  [0.97216097023153247, 0.0278390297684
67475]
P(X = 25 | C = 2 ) [1, 2] =  [0.9848401323042999, 0.01515986769570
0111]
P(X = 26 | C = 2 ) [1, 2] =  [0.99531422271223813, 0.0046857772877
61852]
P(X = 27 | C = 2 ) [1, 2] =  [0.98318632855567811, 0.0168136714443
21939]
P(X = 28 | C = 2 ) [1, 2] =  [0.99310915104740904, 0.0068908489525
90959]
P(X = 29 | C = 2 ) [1, 2] =  [0.98980154355016536, 0.0101984564498
34619]
P(X = 30 | C = 2 ) [1, 2] =  [0.99807056229327451, 0.0019294377067
254685]
P(X = 31 | C = 2 ) [1, 2] =  [0.99862183020948181, 0.0013781697905
181918]
P(X = 32 | C = 2 ) [1, 2] =  [0.96609702315325252, 0.0339029768467
47521]
P(X = 33 | C = 2 ) [1, 2] =  [0.99421168687982364, 0.0057883131201
764059]
P(X = 34 | C = 2 ) [1, 2] =  [0.97436604189636167, 0.0256339581036
38367]
P(X = 35 | C = 2 ) [1, 2] =  [0.9379823594266814, 0.06201764057331
8635]
P(X = 36 | C = 2 ) [1, 2] =  [0.94404630650496146, 0.0559536934950
38585]
P(X = 37 | C = 2 ) [1, 2] =  [0.98208379272326352, 0.0179162072767
36495]
P(X = 38 | C = 2 ) [1, 2] =  [0.96554575523704522, 0.0344542447629
54798]
P(X = 39 | C = 2 ) [1, 2] =  [0.88836824696802641, 0.1116317530319
7354]
P(X = 40 | C = 2 ) [1, 2] =  [0.99917309812568911, 0.0008269018743
1091512]
P(X = 41 | C = 2 ) [1, 2] =  [0.98869900771775088, 0.0113009922822
49173]
P(X = 42 | C = 2 ) [1, 2] =  [0.9528665931642778, 0.04713340683572
2161]
P(X = 43 | C = 2 ) [1, 2] =  [0.97381477398015437, 0.0261852260198
45645]
P(X = 44 | C = 2 ) [1, 2] =  [0.73125689084895262, 0.2687431091510
4743]
P(X = 45 | C = 2 ) [1, 2] =  [0.96223814773980154, 0.0377618522601
98459]
P(X = 46 | C = 2 ) [1, 2] =  [0.98925027563395806, 0.0107497243660
41897]
P(X = 47 | C = 2 ) [1, 2] =  [0.99090407938257996, 0.0090959206174
200669]
P(X = 48 | C = 2 ) [1, 2] =  [0.85033076074972436, 0.1496692392502
7564]
P(X = 49 | C = 2 ) [1, 2] =  [0.50303197353913998, 0.4969680264608
5997]
```

```
P(X = 50 | C = 2 ) [1, 2] =  [0.92805953693495036, 0.0719404630650
49615]
P(X = 51 | C = 2 ) [1, 2] =  [0.16675854465270121, 0.8332414553472
9877]
P(X = 52 | C = 2 ) [1, 2] =  [0.38836824696802646, 0.6116317530319
7359]
P(X = 53 | C = 2 ) [1, 2] =  [0.71251378169790514, 0.2874862183020
9481]
P(X = 54 | C = 2 ) [1, 2] =  [0.24007717750826901, 0.7599228224917
3093]
P(X = 55 | C = 2 ) [1, 2] =  [0.25275633958103638, 0.7472436604189
6362]
P(X = 56 | C = 2 ) [1, 2] =  [0.26488423373759645, 0.7351157662624
0349]
0.10899709771
0.128345694937
0.192841019026
```

How does the accuracy vary across experiements?

Our error rate **increases**. This is because our naive bayes classifer can only use data it has seen too determine it's prediction. As we lower the amount of data feed into the classifer, the less information it has to determine the output of a feature vector is hasn't seen before.

Suppose a classifier randomly guessed the class labels with equal probabilities of 0.5. How accurate would you expect it to be?

For this classifier, P(C=i | Xj) = .5, which means we would need a way to choose the class C. Assume that C is binary [Spam, Not Spam], and we always pick a class based off some random method (e.g. random number generator). That means at best, ever item in our dataset happens to be class as our guess, giving us 0% error. That is unlikely. It's also unlikely that we get every single item wrong. Given that, I would assume this classifier would get about **50% incorrect**, with a 10% fluctuation on each side, depending on the "luck" of our random number generator.

How accurate would a classifier be that always predicted the class label which occurred most commonly in the training set (i.e., if the majority of examples 1–1500 were class i, it would always report class i regardless of input)?

This classifer could predict as well as the previous classifer, with about a 50% error. However, this classifer is susceptible to bad sampling. An example, what if only 33% of our data is class 1 and that happens to be our training data? Then we would 100% on the testing set, and probably 50% error on the remaining. If this was a rather representative sample, then we would have an error close to ratio of the minority element in the dataset. On average, I suspect this classifer to get roughly **50%** incorrect.

How do these two simple strategies compare to the performance of your system measured in parts 1-3?

This two classifers are definietly easier to implement then the one I built. However, they would do really bad in practice. The reason for this is neither classifer learns from our dataset. Our classifer attempts to understand the class labeling from a cooresponding feature vector.

Write a third function which returns the probability of the class label C being spam and non-spam. Compute these probabilities for the first 20 examples in the data file. You do not have to turn in code for this function, just the table of values. You will probably want to use the code you have already written for nbayes predict.m as a template and modify it slightly to return the class probabilities rather than just the most likely class.

In [7]:
```
Xte,Yte = X[20:], Y[20:]
predictions = np.array([])
for i in [20]:
        Xtr,Ytr = (X[0:i], Y[0:i])
        p = params(Xtr, classes, Ytr)
        for d in range(Xtr.shape[0]):
                pxji = np.empty(2) # P(Xj|C=i)
                pxji.fill(1.0)
                for ci,c in enumerate(classes):
                        temp = 1.0
                        for ji,j in enumerate(Xtr[d]):
                                temp = temp * p.cprobs(ji, j, c)
                        pxji[ci] = (temp * p.classprobs(c)) / p.pof
x(Xtr[d])
                print pxji
```

```
[  1.41543637e-13   1.51357769e-07]
[  1.17329313e-08   3.75416247e-13]
[  9.95858565e-09   2.48213923e-08]
[  5.03947933e-14   1.57412080e-08]
[  3.55752516e-08   8.49358025e-15]
[  8.20289353e-08   1.79510398e-10]
[  2.41290270e-12   6.31817258e-10]
[  6.50762886e-07   2.19680557e-10]
[  6.04516570e-19   2.48213923e-08]
[  4.34270834e-08   3.35982028e-11]
[  2.44451735e-18   9.20677170e-29]
[  2.91144988e-12   2.90042214e-26]
[  3.20521467e-14   2.03283244e-14]
[  5.06840779e-14   5.38748322e-26]
[  2.07990511e-10   2.24627985e-11]
[  2.60305155e-08   2.19680557e-10]
[  3.44521528e-07   1.29223857e-11]
[  6.84459274e-28   2.99085093e-14]
[  2.86841849e-10   1.87708124e-12]
[  3.62629200e-17   1.81342137e-09]
```

In [ ]: