Machine Learning (BITS F464)

Assignment-3 (Naive Bayes Classifier)

Submitted on-> 23/11/2016

Submitted to -> **Dr. N.L.Bhanu Murthy**

Submitted by->

JVN Swetha	2014A7PS128H
Wajeeha Fathima	2014A7PS131H
Ayushi Behl	2014A7PS145H
Soamya Agrawal	2014A7PS185H

Training of faces ->

Our naive Bayes model has several parameters to estimate. One parameter is the prior distribution over labels (face/not-face represented in binary, 1 being face and 0 being non-face in “facedatatrainlabels” file) **P(Y).**
We can estimate **P(Y)** directly from the training data:

$$\hat{P}(y) = \frac{c(y)}{n}$$

Where **c(y)** is the number of training instances with label y and n is the total number of training instances.
The other parameters to estimate are the conditional probabilities of our features given each label y: **$P(F_i \mid Y=y)$**. We do this for each possible feature value **(fi ∈ ‘#’,’ ’).**

$$\hat{P}(F_i = f_i|Y = y) \quad = \quad \frac{c(f_i, y)}{\sum_{f_i} c(f_i, y)}$$

Where **$c(f_i,y)$** is the number of times pixel **F_i** took value **f_i** in the training examples of class y.

Experimental Settings (Smoothing Constant) ->

Our current parameter estimates are unsmoothed, that is, we are using the empirical estimates for the parameters **$P(f_i \mid y)$** These estimates are rarely adequate in real systems. Minimally, we need to make sure that no parameter ever receives an estimate of zero, but good smoothing can boost accuracy quite a bit by reducing overfitting.
The basic smoothing method we'll use here is ***Laplace Smoothing*** which essentially adds k counts to every possible observation value:

$$P(F_i = f_i|Y = y) = \frac{c(F_i=f_i,Y=y)+k}{\sum_{f_i} c(F_i=f_i,Y=y)+k}$$

If k=0 the probabilities are unsmoothed, as k grows larger the probabilities are smoothed more and more.

Accuracy ->

Smoothing Constant Value	Accuracy
0.001	88.67%
0.01	88.67%
0.05	88%
0.1	88%
0.5	89.34%
1	**90.67%**
5	86.67%
10	85.34%
50	78%
100	86%

We are getting maximum accuracy when smoothing constant is 1. So value of smoothing constant is set to be 1.

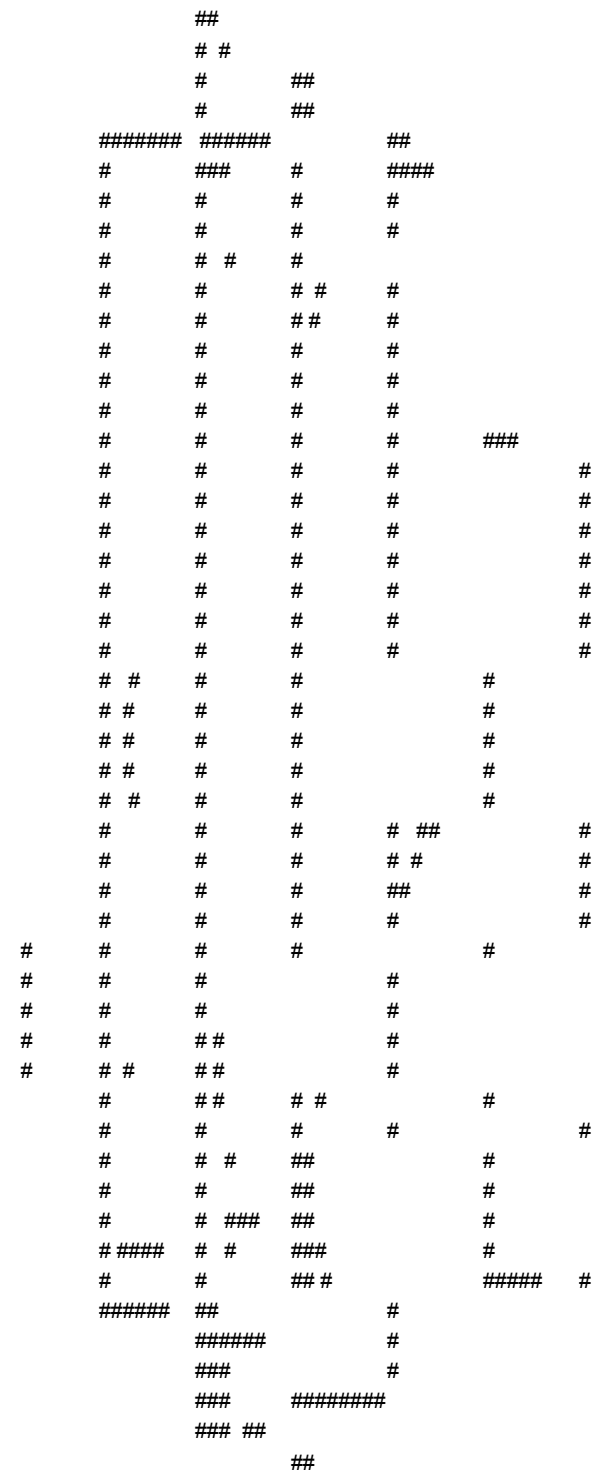
Confusion Matrix ->

For smoothing constant = 1

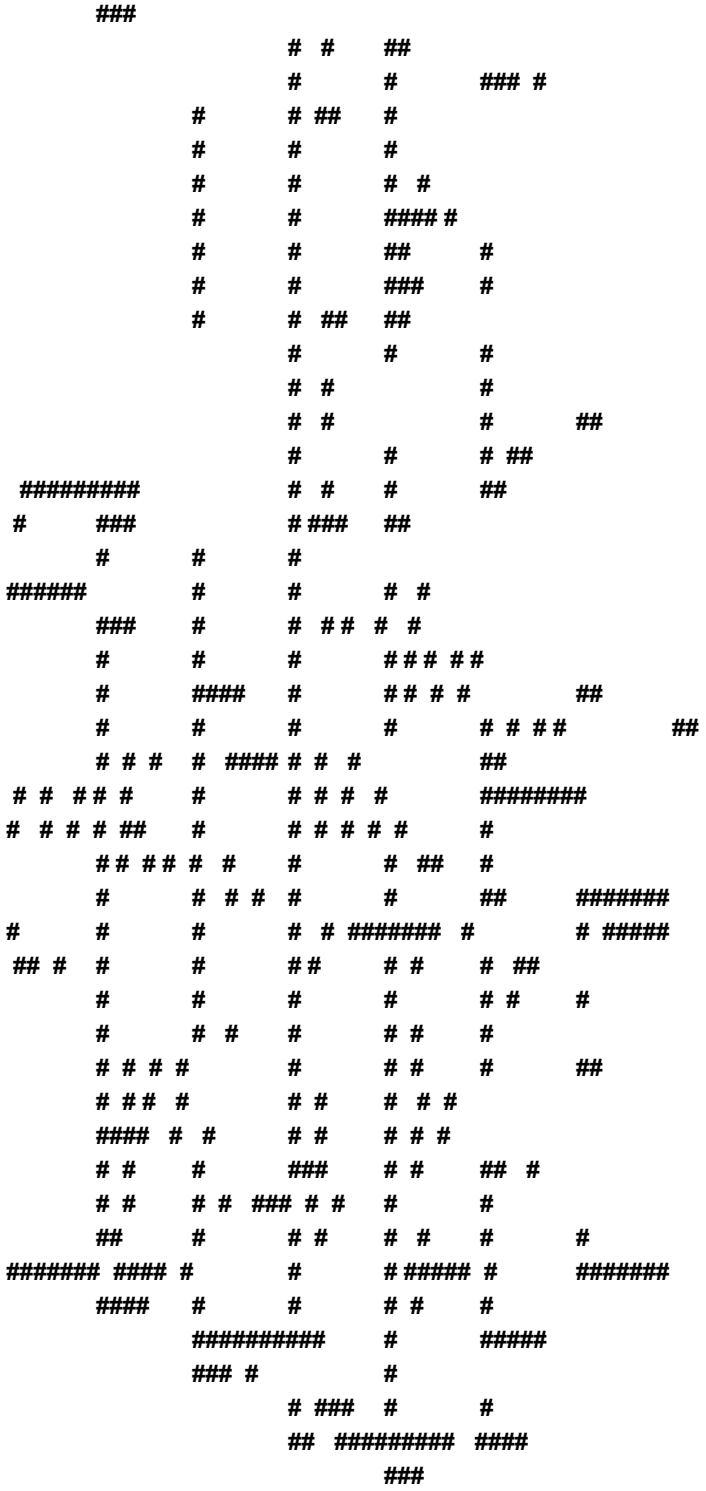
n=150	**Predicted No**	**Predicted Yes**	
Actual No	**TN=** 68	**FP=** 9	77
Actual Yes	**FN=** 5	**TP=** 68	73
	73	77	

Examples of False Positives (non-faces classified as faces) ->

Non Face Number 15

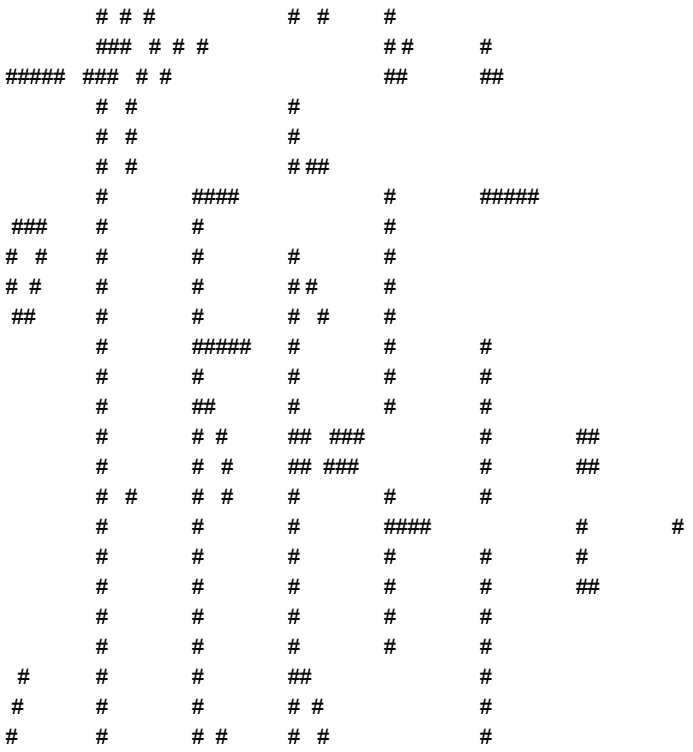


Non Face Number 20

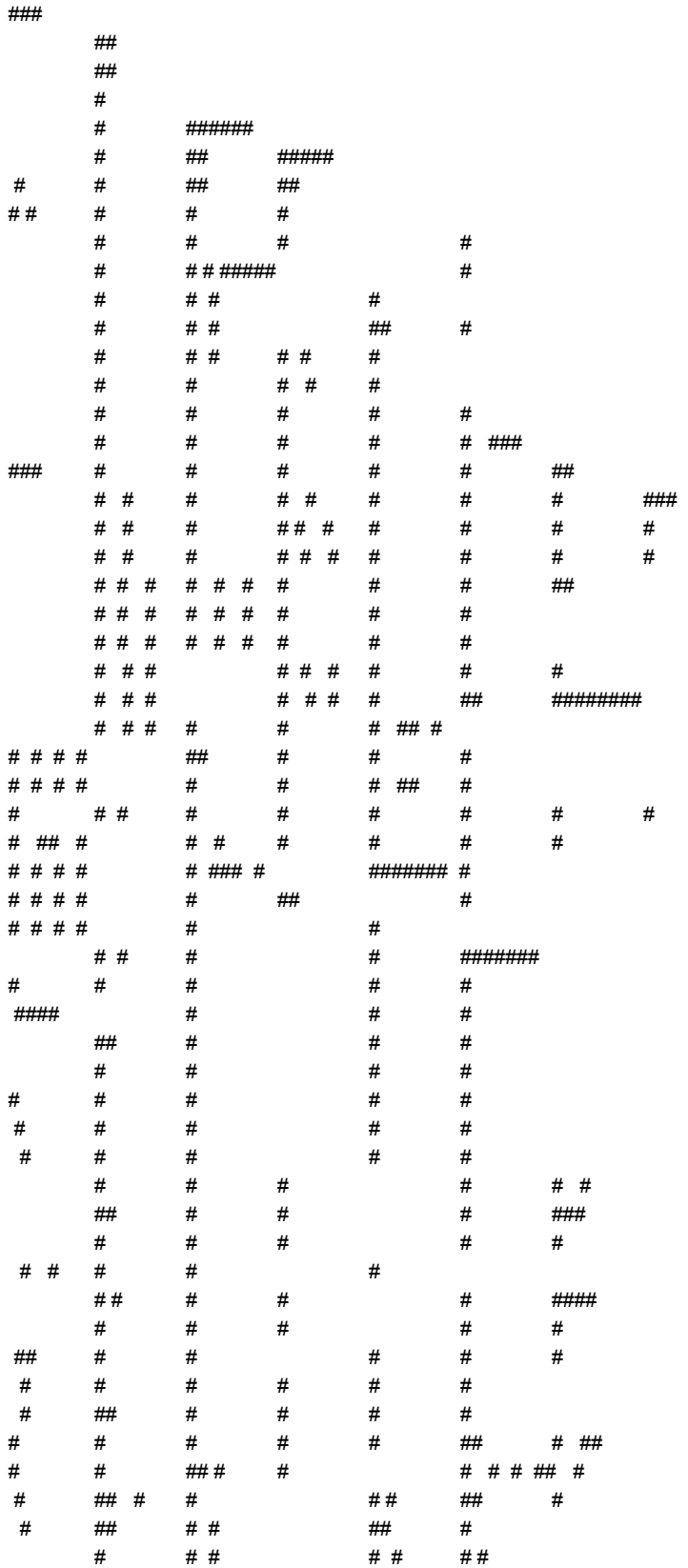


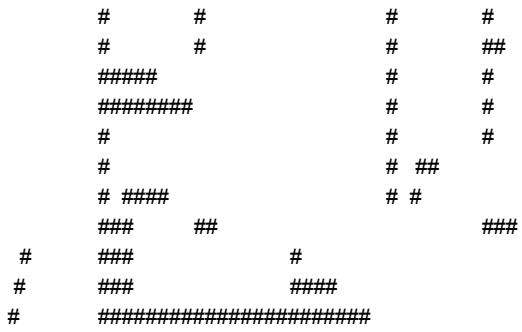
Examples of False Negatives -> (faces classified as non-faces) ->

Face Number 20



Face Number 28





Code ->

```
import java.io.*;

public class NaiveBayesClassifier
{
   public static void main(String[] args) throws IOException
    {
        // reading training data file
        int i = 0, j = 0;
        int face = 0, f = 0;
        // declaring array containing information about pixels of each face
        char ft[][] = new char[451][4200];
        String sCurrentLine;
BufferedReader br = new BufferedReader(new FileReader("E:/3-1/machine learning/ML
Assignment/facedatatrain"));
        while ((sCurrentLine = br.readLine()) != null) {
                if (f == 4200) {
                        f = 0;
                        face++;
                }
                for (j = 0; j < sCurrentLine.length(); j++) {
                        ft[face][f] = sCurrentLine.charAt(j);
                        f++;
                }
        }
        br.close();
        // reading label file to calculate probability of being a face and not
        BufferedReader br1 = new BufferedReader(
                        new FileReader("E:/3-1/machine learning/ML
Assignment/facedatatrainlabels"));
        int train[] = new int[451];
        int facecount = 0, nonfacecount = 0;
        float pf, pnf;
        // setting of smoothconstant
        float smoothk = 1f;
        for (i = 0; i < 451; i++) {
                if (br1.read() == '0') {
                        train[i] = 0;
```

```
                        nonfacecount++;
                } else {
                        train[i] = 1;
                        facecount++;
                }
                br1.readLine();
        }
        br1.close();
        pf = (float) facecount / (facecount + nonfacecount);
        pnf = (float) nonfacecount / (facecount + nonfacecount);

        // now calculating conditional probabilities

        float condhashfaceprob[] = new float[4200];
        float condspacefaceprob[] = new float[4200];
        int hashcount = 0, spacecount = 0;
        for (i = 0; i < 4200; i++) {
                hashcount = 0;
                spacecount = 0;
                for (j = 0; j < 451; j++) {
                        if (ft[j][i] == '#' && train[j] == 1) {
                                hashcount++;
                        } else if (ft[j][i] == ' ' && train[j] == 1)
                                spacecount++;
                }
                condhashfaceprob[i] = (hashcount + smoothk) / (hashcount + smoothk +
spacecount + smoothk);
                condspacefaceprob[i] = (spacecount + smoothk) / (hashcount + smoothk +
spacecount + smoothk);
        }

        float condhashnonprob[] = new float[4200];
        float condspacenonprob[] = new float[4200];

        for (i = 0; i < 4200; i++) {
                hashcount = 0;
                spacecount = 0;
                for (j = 0; j < 451; j++) {
                        if (ft[j][i] == '#' && train[j] == 0) {
                                hashcount++;
                        } else if (ft[j][i] == ' ' && train[j] == 0)
                                spacecount++;
                }
```

```
                condhashnonprob[i] = (float) (hashcount + smoothk) / (hashcount + smoothk +
spacecount + smoothk);
                condspacenonprob[i] = (float) (spacecount + smoothk) / (hashcount + smoothk +
spacecount + smoothk);
        }

        // for testing data

        // reading data from testing file
        i = 0;
        j = 0;
        face = 0;
        f = 0;
        char ftest[][] = new char[150][4200];

        BufferedReader br2 = new BufferedReader(new FileReader("E:/3-1/machine
learning/ML Assignment/facedatatest"));
        while ((sCurrentLine = br2.readLine()) != null) {
                if (f == 4200) {
                        f = 0;
                        face++;
                }
                // System.out.println(f);
                for (j = 0; j < sCurrentLine.length(); j++) {
                        ftest[face][f] = sCurrentLine.charAt(j);
                        f++;
                }
        }
        br2.close();

        // test labels
        BufferedReader br3 = new BufferedReader(
                        new FileReader("E:/3-1/machine learning/ML
Assignment/facedatatestlabels"));
        int actualOutput[] = new int[150];
        int predictedOutput[] = new int[150];
        int tn = 0, fp = 0, fn = 0, tp = 0;
        for (i = 0; i < 150; i++) {
                if (br3.read() == '0') {
                        actualOutput[i] = 0;
                } else {
                        actualOutput[i] = 1;
```

```
		}
		br3.readLine();
	}

	br3.close();
	// printing of test labels
		// confusion matrix
	double max1[] = new double[150];
	double max2[] = new double[150];
	for (i = 0; i < 150; i++) {
		max1[i] = Math.log(pf);
		for (j = 0; j < 4200; j++) {
			if (ftest[i][j] == ' ') {
				max1[i] = max1[i] + Math.log(condspacefaceprob[j]);
			} else if (ftest[i][j] == '#') {
				max1[i] = max1[i] + Math.log(condhashfaceprob[j]);
			}
		}
	}
	for (i = 0; i < 150; i++) {
		max2[i] = Math.log(pnf);
		for (j = 0; j < 4200; j++) {
			if (ftest[i][j] == ' ') {
				max2[i] = max2[i] + Math.log(condspacenonprob[j]);
			} else if (ftest[i][j] == '#') {
				max2[i] = max2[i] + Math.log(condhashnonprob[j]);
			}
		}
	}
	for (i = 0; i < 150; i++) {
		if (max2[i] > max1[i]) {
			predictedOutput[i] = 0;
		} else
			predictedOutput[i] = 1;
	}

	for (i = 0; i < 150; i++) {
		if (actualOutput[i] == 0 && predictedOutput[i] == 0) {
			tn++;
		} else if (actualOutput[i] == 0 && predictedOutput[i] == 1) {
			fp++;
			System.out.println("false positive occuring at i= " + i);
		} else if (actualOutput[i] == 1 && predictedOutput[i] == 0) {
```

```
                        fn++;
                        System.out.println("false negative occuring at i= " + i);
                } else if (actualOutput[i] == 1 && predictedOutput[i] == 1) {
                        tp++;
                }
        }
        System.out.println(
                        "false positive " + fp + " true positive " + tp + " false negative " + fn + "
true negative" + tn);
        double accuracy = (double) (tp + tn) / (double) (fp + tp + fn + tn);
        System.out.println("smoothing constant " + smoothk + " accuracy " + accuracy);

    }
}
```