**Abstract**

Face detection is the first critical step in face recognition, face tracking, pose estimation, facial expression recognition. This project is aimed at giving you an opportunity to compare the predictive powers of both Naïve Bayes classifier and logistic regression classifier to distinguish between a face image and a non-face image. Since you have already implemented the logistic regression classifier in the previous assignment, here you are going to use that implementation. Besides that you need to implement the Naive Bayes classifier. Both of these algorithms should learn from the given training dataset, and you are going to evaluate the corresponding models on the given test dataset. Here below are some examples of face and non-face images.

Some face examples Some non-face examples:

**1 About the dataset**

The dataset that you are going to use is prepared by MIT Center For Biological and Computational Learning (CBCL) lab. The link to download the dataset is <https://goo.gl/gYacor>

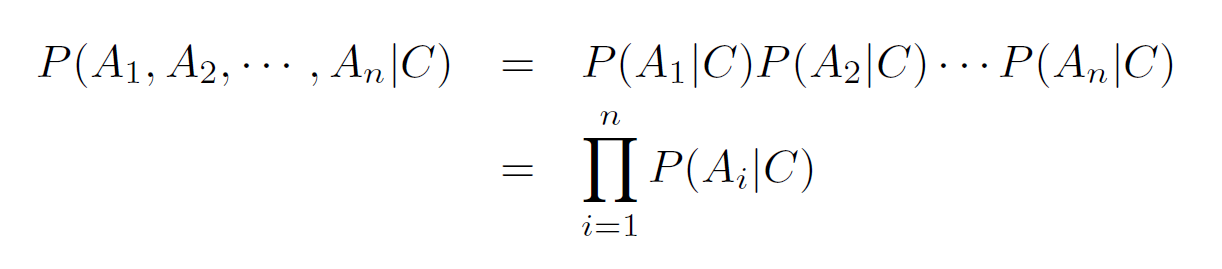
* Training set contains 2,429 faces, and 4,548 non-face images
* Test set contains 472 faces, and 23,573 non-face images
* Each image file is 19 X 19 grayscale image, and is stored as a pgm file.

That means

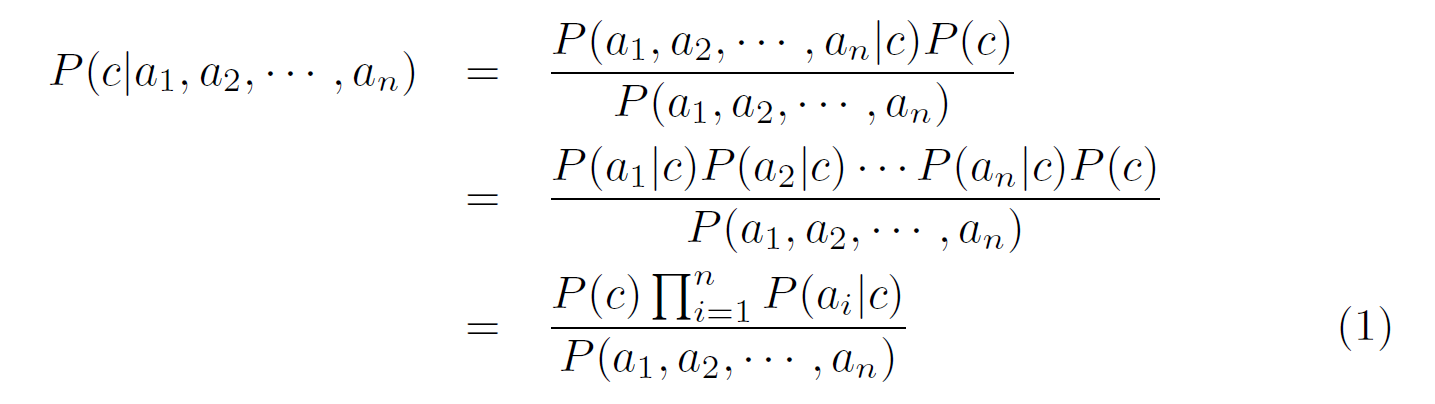
* Each image sample is 19 pixel wide and 19 pixel high.
* So, each image sample is a 361 dimensional vector, where each element in the vector contains the grayscale value of a pixel from the image.
* A grayscale value of a pixel takes any (discrete integer) value from the range {0, 1, … , 255}

**2 Naive Bayes Classifier : Theory revisited**

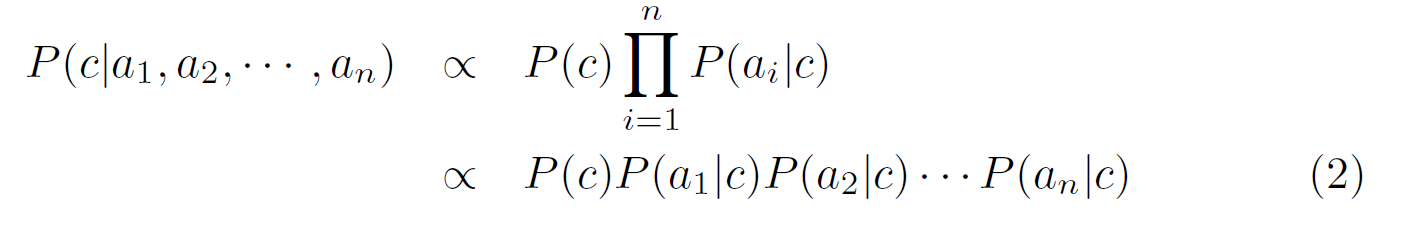
A naive Bayes Classifier models a joint probability distribution over a target label C and a set of input variables, or features (A1,A2, … ,An), using the assumption that the input variables are conditionally independent given the target label:



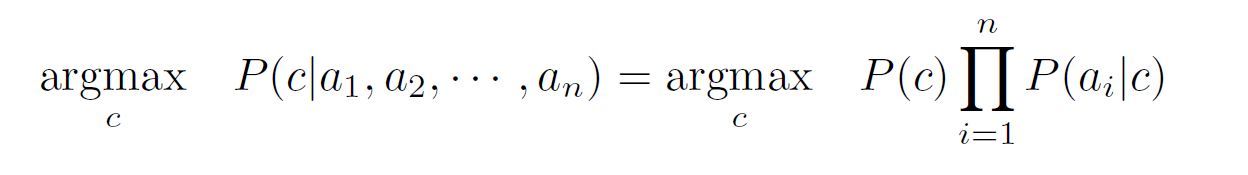
To classify a test sample, X = [a1, a2, … , an]T , we can find the most probable target label, C = c, given the feature values for each pixel, i.e., A1 = a1, A2 = a2, … , An = an, using the Bayes theorem below. Here, Ai is the name of the variable, ai is the value the variable Ai takes, and the C is the target variable, and c is one possible value of C:



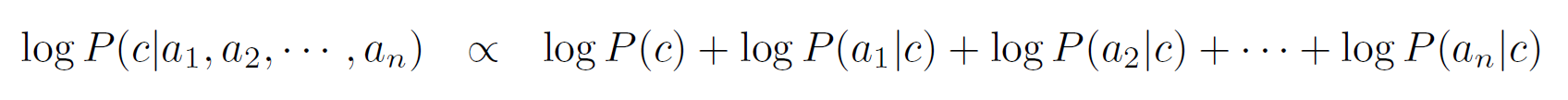
We need to find c for target variable C that maximizes the numerator of equation 1 only, as the denominator is common in all possible cases for c. So, we can write Equation 1 as:



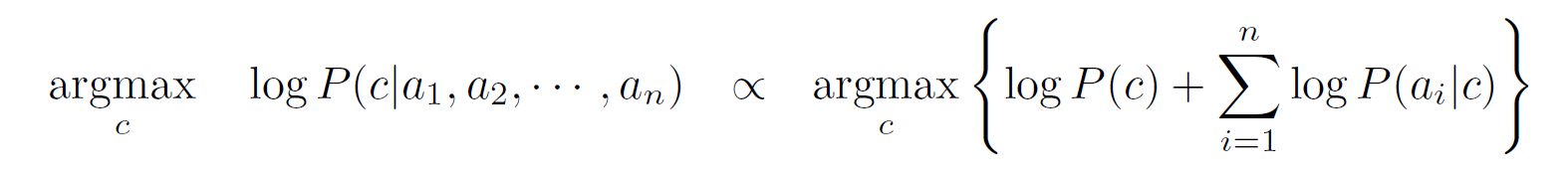
We need to find the best target value c, such that P(c|a1, a2, … , an) is comparatively maximum among all other targets. It can be written as the following:



Please note that the right hand side of this equation is a multiplication of several probabilities, value of each ranging from 0 to 1. As you increase the number of features (i.e., n), this product will decrease tremendously and this might result in an underflow of the floating point variable. Such an underflow problem can be avoided if we take the log on both sides of equation 2, which transforms the multiplication into summation of the log probabilities. And fortunately, this tweak preserves the comparison to predict the target label c:



Therefore, we are actually computing the following:

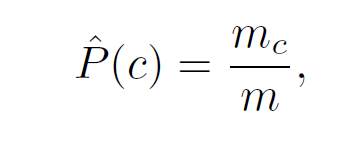


To compute logarithms, use natural log function (i.e., the e-base logarithm).

**2.1 Parameter estimation**

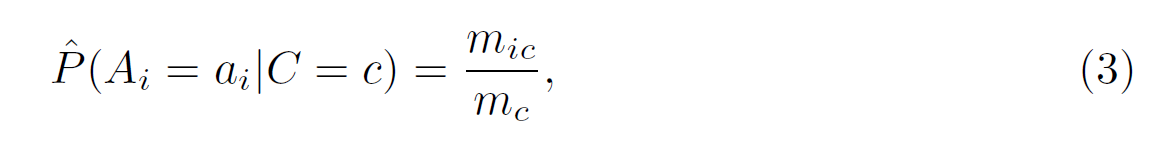
The naive Bayes model has several parameters to estimate. One parameter is the prior probability of each of the target labels, i.e., P(c).

We can estimate P(c) directly from the training data:



where ^ P(c) is the estimated prior probability of target label C = c, and m is the total number of samples, mc is the total number samples having class label C = c.

The other parameters to estimate are the conditional probabilities of our features Ai, given each target label C = c: P(Ai = ai|C = c). We can compute this from the training data as well:

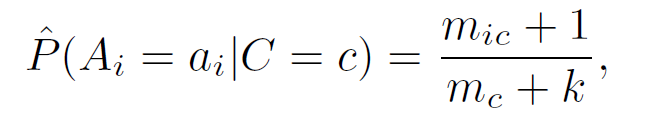


where, mic is the number of samples having feature value Ai = ai that belongs to class c.

**2.2 Handling the zero probability issues by smoothing the**

**conditional probabilities**

The current estimate of the parameters are unsmoothed, as some of the probabilities might be zero. You are going to use Laplace method to smooth the conditional probability estimate. So you are not going to use Equation 3. Instead use the following:



where k is the possible number of values the variable Ai can take.

**3 Problem statement**

**3.1 Meet the classifiers: Naive Bayes and Two versions of**

**Logistic Regression**

* Implement the Naive Bayes Classifier.
* Bring back your batch gradient descent based regularized logistic regression function from previous assignment that takes Xtrain, ytrain, nEpoch, alpha , lambda as input parameters, where Xtrain, ytrain are the training dataset, and training target labels respectively.
* Implement stochastic gradient descent based regularized logistic regression function that also takes Xtrain, ytrain, nEpoch, alpha , lambda as input parameters.

**3.2 Now do three experiments only**

* **Experiment 1 with Naive Bayes classifier (NB):**

🡪Train the naive bayes classifier with the training dataset. You do not need to perform any cross validation here. You use the entire training set for training. Also, no feature scaling is necessary.

🡪Then test the classifier with the test dataset, and report Accuracy, Precision, Recall, F1-measure, ROC and AUC. The last two performance measures will be introduced on Monday, so please stay tuned!

* **Experiment 2 with Batch gradient descent based logistic regression (BGD-LR):**

🡪Set the parameter values of logistic regression as : nEpoch = 100; alpha = 0.1; lamda = 1

🡪Train the classifier with the training dataset. Again, you do not need to perform any cross-validation here. You use the entire training set for training. Also no feature scaling is necessary.

🡪Then test the classifier with the test dataset, and report Accuracy, Precision, Recall, F1-measure, ROC and AUC.

* **Experiment 3 with Stochastic gradient descent based logistic regression (SGD-LR):**

🡪Set the parameter values of logistic regression as : nEpoch = 1000; alpha = 0.1; lambda = 1

🡪Train the classifier with the training dataset. Again, you do not need to perform any cross-validation here. You use the entire training set for training. Also no feature scaling is necessary.

🡪Then test the classifier with the test dataset, and report Accuracy, Precision, Recall, F1-measure, ROC and AUC.