FACE AND DIGIT CLASSIFICATION

Erica Cai, Boning Ding, and Shreya Jahagirdar

December 7, 2019

1 File Structure

Our project folder contains six files and two folders with images and labels. The six files are:

a. util.py

b. prepareFeatures.py

c. NaiveBayes.py

d. Perceptron.py

e. kNN.py

f. featureFuncLib.py

We wrote prepareFeatures.py, featureFuncLib.py, NaiveBayes.py, Perceptron.py and kNN.py. We used util.py to store all of the code from the Berkeley website that helps us parse each image and find black and white pixels in each image.

In NaiveBayes.py, Perceptron.py, and kNN.py, we implement the Naïve Bayes, Perceptron, and kNN algorithms respectively.

In featureFuncLib.py, we have several feature functions.

In prepareFeatures.py, we have a main program to run each machine learning algorithm on the dataset and to calculate the accuracy of the machine learning predictions.

For this project, there are six machine learning algorithms:

a. Naïve Bayes for predicting if an image is a face

b. Naïve Bayes for predicting which digit an image is

c. Perceptron for predicting if an image is a face

d. Perceptron for predicting which digit an image is

e. kNN for predicting if an image is a face

f. kNN for predicting which digit an image is

2 How to Test the Algorithms

We perform the same set of steps to test each algorithm.

1. Store the train images, test images, train labels, and test labels in respective arrays.

2. Apply the feature function on the images to get a train features and test features array.

3. Train the machine learning algorithm on the train features and train labels.

4. Predict the labels for the test features.

5. Calculate the percentage of labels that the machine learning algorithm guessed correctly.

Only the Naïve Bayes and Perceptron algorithms do step 3; the kNN algorithm does not have a train function and is discussed more in section 7.

3 How to Store Images

The images that we use in our train set and test set belong to the folders

a. digitdata, in the files

   digitdatatest

   digitdatatrain

b. facedata, in the files

   facedatatest

   facedatatrain

We extract each image and convert it into a "datum" object, which stores information about whether pixels in the image are black, white, or gray.

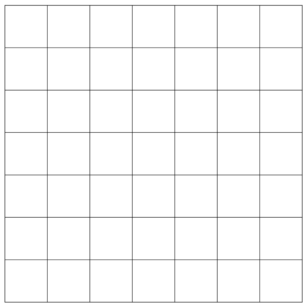
We store an array of "datum" objects and create features based on each "datum" object.

4 How to generate and store features

We convert each datum into a list of features.

Our feature function for digits splits the datum into 49 parts and counts the number of black pixels in each part. The regions of the data look like below.

Our feature function for faces splits the datum into 42 parts and counts the



number of black pixels in each part. The regions of the data look like below.

To count the number of black pixels in each part, we traverse through each

pixel of a part and update the number of black pixels in that part.

5 Naive Bayes Algorithm

5.1 Overview for Face Detection

We want to calculate $L(x) = \dfrac{p(y = true \mid x)}{p(y = false \mid x)} = \dfrac{p(x \mid y = true)p(y = true)}{p(x \mid y = false)p(y = false)}$.
We return true if$L(x) \geq 1$ and false otherwise.

Our train function prepares all of the values necessary for calculating $p(x \mid$

$y = true$), $p(y = true)$, $p(x \mid y = false)$, $p(y = false)$ based on information in the train set.

Our predict function calculates L(x) for each item in the test set. Based on the L(x) value of an item, the predict function assigns a label of 0 or 1 to that item. A 0 represents that the item is not a face and 1 represents that the item is a face.

5.2 Data Structures for Face Detection

We have one probability array containing information necessary to calculate $p(x \mid y = true)$ and one probability array containing information necessary to calculate $p(x \mid y = false)$.

Each probability array is an array of dictionaries.

We use dictionaries because searching is O(1) time while searching in an array takes O(trainingsize) time.

The probability array has the following form:

$[dictionarycontaining(value, probability)pairsforfeature1]$,

$[dictionarycontaining(value, probability)pairsforfeature2]$,

$[dictionarycontaining(value, probability)pairsforfeature3], \ldots$

5.3 Method Definitions for Face Detection

TRAIN

Input: train features, train labels

Output: P(face=true), P(face=not true), array1 as described below, array2 as described below

Where array1 is an array containing probabilities given that the image is a face. We organized the array to be an array of dictionaries; each dictionary represents a feature and contains (value, probability) pairs for the likelihood that the feature has that value, given that the image is a face.

Where array2 is an array containing probabilities given that the image is NOT a face. We organized the array to be an array of dictionaries; each dictionary represents a feature and contains (value, probability) pairs for the likelihood that the feature has that value, given that the image is not a face.

5.4 Train Algorithm for Face Detection

Part 0: This part initializes variables
numberOfFaces=0
numberOfNotFaces=0
Array1=[, , . . .]
Array2=[, , . . .]

Part 1: This part finds counts
For each feature array in the training set
    If the label shows that the image IS A FACE
        numberOfFaces++
        for feature i in the array of features
            in array1, add feature i to the dictionary that corresponds to feature i and update the count that corresponds to feature i
    If the label shows the image IS NOT A FACE
        numberOfNotFaces++
        for feature i in the array of features
            in array2, add feature i to the dictionary that corresponds to feature i and update the count that corresponds to feature i

Part 2: This part turns counts into probabilities
For every count in array1, count=count/numberOfFaces
For every count in array2, count=count/numberOfNotFaces
pFace=numberOfFaces/total
pNotFace=numberOfNotFaces/total
return pFace, pNotFace, array1, array2

5.5 Predict Algorithm for Face Detection

Input: test features, pFace, pNotFace, array1, array2
Output: predicted labels
Need to calculate $p(x \mid y = true)$, $p(y = true)$, $p(x \mid y = false)$, $p(y = false)$

For each item in test features
    Use the formulas below to compute $p(x \mid y = true)$ from array1 and $p(x \mid y = false)$ from array2
    Then compute L(x)
    Then add label based on the L(x) value into the predict array

$$p(x|y = true) = \prod_{j=1}^{l} p(\phi_j(x)|y = true)$$
$$p(x|y = false) = \prod_{j=1}^{l} p(\phi_j(x)|y = false)$$

5.6 Overview for Digit Identification

Instead of calculating L(x) as for face, calculate $p(x \mid y = true) * p(y = true)$ for each digit and predict the digit with the maximum result in this calculation.

The train and predict algorithm are very similar to that for Face.

5.7 Method Definitions for Digit Identification

Train

Input: train features, train labels
Output: P(digit 0=true), P(digit 1=true), P(digit 2=true), P(digit 3=true), P(digit 4=true), P(digit 5=true), P(digit 6=true), P(digit 7=true), P(digit 8=true), P(digit 9=true), array0, array1, array2, array3, array4, array5, array6, array7, array8, array9

Predict

Input: test features, test labels, P(digit 0=true), P(digit 1=true), P(digit 2=true), P(digit 3=true), P(digit 4=true), P(digit 5=true), P(digit 6=true), P(digit 7=true), P(digit 8=true), P(digit 9=true), array0, array1, array2, array3, array4, array5, array6, array7, array8, array9
Output: predict

6 Perceptron

6.1 Overview for Face Detection

FACE
We want to calculate $f(x_i, w) = w_0 + w_1\Phi1(x_i) + w_2\Phi2(x_i) + w_3\Phi3(x_i) + ... + w_l\Phi l(x_i)$. If ≥ 0, we predict that it is a face; otherwise, we predict that it is not a face.

Our train function prepares all of the weights necessary for calculating $f(x_i, w) = w_0 + w_1\Phi1(x_i) + w_2\Phi2(x_i) + w_3\Phi3(x_i) + ... + w_l\Phi l(x_i)$ based on information in the train set.

Our predict function calculates $f(x_i, w)$ for each item in test set assigns a label of 0 or 1 to that item; a 0 represents that the item is not a face and 1 represents that the item is a face.

6.2 Method Definitions for Face Detection

TRAIN

Input: train features, train labels
Output: array of weights

Initialize weights as [0.0001]
Iterate through the train features and labels until you don't have to adjust the weights anymore, or until a specific time has passed
Return weights

PREDICT
Input: test features, array of weights
Output: predict labels

6.3 Algorithm for Face Detection

For each test feature array, calculate f(x)
Based on f(x), append 0 or 1 to the predict array, where 0 represents that the item is not a face and 1 represents that the item is a face.

6.4 Overview for Digit Identification

Instead of calculating one $f(x_i, w)$ function as in face indentification, calculate 10 of the them and choose the digit that corresponds to the maximum f value as the digit that the algorithm predicts for an image

6.5 Method Definitions for Digit Identification

TRAIN
Input: train features, train labels
Output: array of weights for digit 0, array of weights for digit 1, array of weights for digit 2, array of weights for digit 3, array of weights for digit 4, array of weights for digit 5, array of weights for digit 6, array of weights for digit 7, array of weights for digit 8, array of weights for digit 9

PREDICT
Input: test features, array of weights for digit 0, array of weights for digit 1, array of weights for digit 2, array of weights for digit 3, array of weights for digit 4, array of weights for digit 5, array of weights for digit 6, array of weights for digit 7, array of weights for digit 8, array of weights for digit 9
Output: predict labels

7 kNN

Unlike the Naïve Bayes and Perceptron algorithms, kNN only has a predict function. The kNN algorithm takes in training features, training labels, test features, test labels, and a k value and returns an array of predicted labels for the test features.

For each array of features in the test set, the kNN algorithm looks through all of the arrays of features in the training set and chooses the k images that are most similar to the image in the test set. Then it uses the most frequent label of the k images to predict whether the current image in the test set is a face or not.

8 How We Compared the Algorithms

In total, we wanted to compare the performace for 6 algorithms: Naive Bayes for face detection and digit identification, Perceptron for face detection and digit identification, and kNN for face detection and digit identification. We also wanted to compare the performance of these algorithms when using 10%, 20%... 100% of the training set.

We ran each algorithm five times on each training set size. When the training set size was 40% of the entire training data, we implemented an algorithm to randomly select 40% of the entire training data during each of the five trials. As a result of the randomness, our algorithm has slightly different results in accuracy and training time.

Therefore, for each algorithm and training set size, we performed five trials. For each trial, we recorded the accuracy and the time necessary to train the algorithm. kNN does not have a training function; we could not calculate the time necessary to train the algorithm so we recorded the amount of time that the algorithm spent to predict one label for one image.

For each algorithm and training set size, we recorded the average accuracy, the standard deviation of the accuracy, and the average time needed to train the algorithm over all five trials.

9 Comparison of Training Times For Each Algorithm on Each Training Set Size

10 Accuracy, Prediction Error, and Standard Deviation

11 Analysis