Project : Face and Digit Classification

Deadline: August 8th, 11:55pm.
Deadline for partner adjustment: July 29th, 12:00pm.
Deadline for one-day extension: August 9th, 11:55pm.
Perfect score: 100.

Project Instructions:

Teams: The project should be completed by teams of one or two students. No additional credit will be given for students who complete the project individually. Please indicate any partner change in a timely manner and be aware of the deadline for changing the partner.

Submission Rules: Submit your reports electronically as a PDF document through Sakai (`sakai.rutgers.edu`). For programming questions, you need to also submit a compressed file via Sakai, which contains your code. Do not submit Word documents, raw text, or hardcopies etc. Make sure to generate and submit a PDF instead. Each team of students should submit only a single copy of your solutions and indicate all team members on their submission. Failure to follow these rules may result in lower grade for the project.

Project Demonstrations: You will need to demonstrate your program to the TAs on a date after the deadline. The schedule will be coordinated by the TAs. During the demonstration you have to use the file submitted on Sakai before the deadline and execute it either on a university machine at CORE/HILL center (one of the CS labs) or on your laptop computer. You will also be asked to describe the architecture of your implementation and key algorithmic aspects of the project. You need to make sure that you are able to complete the demonstration and answer the TAs' questions within the allotted 15 minutes of time for each team. If your program is not directly running on the computer you are using and you have to spend time to configure your computer, this counts against your allotted time.

Late Submissions: You are supposed to finish your work before the deadline specified at the top. However, if you really cannot finish your work on time, at maximum one-day extension will be given at the cost of getting 70% of the scores you own in this project.

Extra Credit for LaTeX: You will receive 5 extra credit points if you submit your answers as a typeset PDF (using LaTeX, in which case you should also submit electronically your latex source code). Please keep in mind that we will not accept hardcopies and scanned document (handwritten then scanned).

Collusion, Plagiarism, etc.: Each team must prepare its solutions independently from other teams, i.e., without using common notes, code or worksheets with other students or trying to solve problems in collaboration with other teams. You must indicate any external sources you have used in the preparation of your solution. Do not plagiarize online sources and in general make sure you do not violate any of the academic standards of the department or the university. Failure to follow these rules may result in failure in this project.

Other rules: For other grading rules, please refer to the homepage of our course website on sakai. Thanks.

Project Description:

Acknowledgement: This project is based on the one created by Dan Klein and John DeNero that was given as part of the programming assignments of Berkeley's CS188 course.

In this project, you will design two classifiers: a naive Bayes classifier and a perceptron classifier. You will test your classifiers on two image data sets: a set of scanned handwritten digit images and a set of face images in which edges have already been detected. Even with simple features, your classifiers will be able to do quite well on these tasks when given enough training data.

Optical character recognition (OCR) is the task of extracting text from image sources. The first data set on which you will run your classifiers is a collection of handwritten numerical digits (0-9). This is a very commercially useful technology, similar to the technique used by the US post office to route mail by zip codes. There are systems that can perform with over 99% classification accuracy (see LeNet-5 for an example system in action).

Face detection is the task of localizing faces within video or still images. The faces can be at any location and vary in size. There are many applications for face detection, including human computer interaction and surveillance. You will attempt a simplified face detection task in which your system is presented with an image that has been pre-processed by an edge detection algorithm. The task is to determine whether the edge image is a face or not.

Please refer to `https://inst.eecs.berkeley.edu/~cs188/sp11/projects/classification/classification.html` for a brief description of the Perceptron and Naive Bayes classifiers.

Data: The data are attached to the project as a zip file. (You can find it with this pdf) There are three types of data: (1) **training data** The data you use for training your algorithms (2) **validation data** The data you use to validate your algorithms so that you can see how good they perform and make corresponding adjustments. (3) **test data** The data you use to test your algorithms, calculate and report the accuracy of learning on your report. You statistics for accuracy should come from the experiments on these test data.

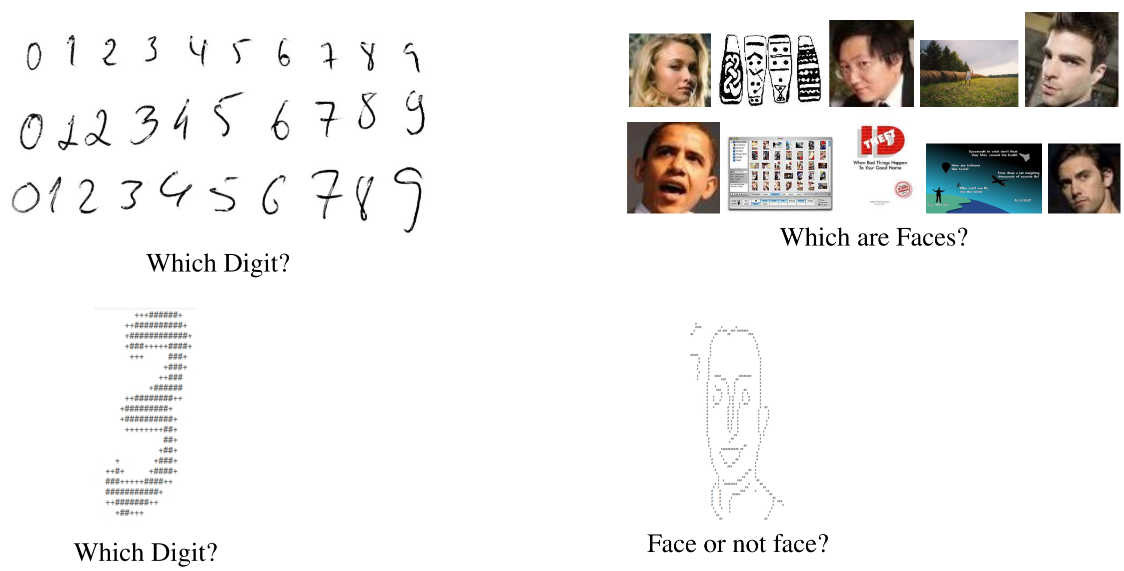

Figure 1: Examples of the data points in the data set.

What you should do:

1. Implement two classification algorithms for detecting faces and classifying digits:
   (a) Perceptron
   (b) Naive Bayes Classifier
2. Design the features for each of the two problems, and write a program for extracting the features from each image.
3. Train the two algorithms on the part of the data set that is reserved for training. First, use only 10% of the data points that are reserved for training, then 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, and finally 100%. All the results should be the output of a function of with input the percentage of data points used for training.
4. Compare the performances of the two algorithms using the part of the data set that is reserved for testing, and report:

- The time needed for training as a function of the number (or percentage) of data points used for training.
- The prediction error (and standard deviation) as a function of the number (or percentage) of data points used for training.

5. Write a report describing the implemented algorithms and discussing the results and the learned lessons.

Please keep in mind that:

- You can use existing libraries, **but not for the learning algorithms**. You should implement yourself the learning algorithms as well as the feature extraction.
- It's OK to share ideas, but not code or writing.
- Part of your score will depend on the accuracy of the predictions made by your program.
- Your algorithm should not look at the testing data before the training is over. If you use any testing data point for training, that would be considered as cheating.

Additional information:

- You are allowed to use any external resources you like, including the code on the Berkeley page. The only things I want you to implement by yourselves are the core operations of Naive Bayes and Perceptron (proba calculation and weight updates).
- How will you be graded? a) Show the TAs that you correctly implemented and understood Naive Bayes and Perceptron, b) Show that the code runs without issues (reads an image from the test data file, and returns a predicted label), c) Write a short report describing what you did, how the different algorithms performed (time and accuracy), and how did they improve as you used more and more of the training data (10%, 20%, ..., 100%).
- There is no standard definition of "good enough" here. Typically, if you have less than 60% accuracy even when you use 100% of your training data then that means that you could do a better job on the features or that something was wrong. For our project, **as long as you have 60% accuracy for digit data and 70% accuracy for face data, your algorithms are considered good enough**.
- Regarding the features, **don't spend too much time on designing complicated features that require a lot of coding**. You would be surprised that the simplest features could be great predictors. For instance, you could just use the pixels directly as features (i.e. one binary feature per pixel in the image). You could also just divide the image into a regular grid (say 10x10). Each square in the grid defines a binary feature that indicates whether there is anything marked inside the square.
- I am not a big fan of rigid structure, because it kills creativity and the spirit of taking initiatives. So, anything I didn't say or specify is open to your own interpretation and I wouldn't tell you that you did it wrong. For instance, I didn't tell you how to write the report (except for the learning curves we expect to see), so write whatever you think is a decent report that you would be proud of. I didn't say how to read the text files into images, but you could easily figure out the size of each image in the files and parse the files accordingly.
- How to compute the average prediction error (or accuracy) and its standard deviation? What we want is a metric for how consistent our prediction accuracy will be if we use a certain number of training points. Thus, we need to vary which training points are used and see the spread of results. The following pseudocode would accomplish this:
  For i=1:5 (you can use more iterations, if time permits)
      Train on 10% of randomly selected data points
      Test on test data and save the accuracy in acc[i]
  End
  Return mean(acc) and std(acc).
  Repeat the same for 20%, 30%, etc.
  You may find the random module in Python to be especially convenient in choosing your random samples.

Have fun!