Department of Computer Science - Rutgers University **Summer 2019**

CS 440: Mini-Project 3 - Machine Learning

01:198:440

Due August 14, 2019 at 11:55pm on Gradescope (PDF)

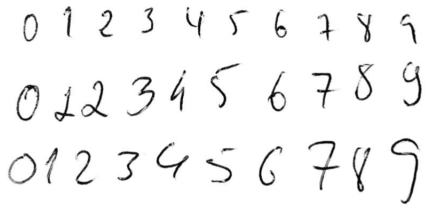
Acknowledgement: This project is based on the one created by Dan Klein and John DeNero that was given as part of the programming assignments of Berkeley's CS188 course.

In this project, you will design three classifiers: a naive Bayes classifier, a perceptron classifier and a neural network 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.

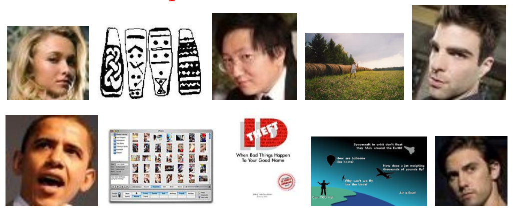
Optical character recognition (OCR) is the task of extracting text from image sources. The first data set on which you will run your classifiers is a collection of handwritten numerical digits (0-9). This is a very commercially useful technology, similar to the technique used by the US post office to 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.

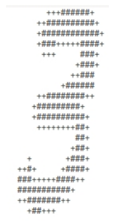
Data: The data is provided on Sakai under Resources. Look for the file data.zip.



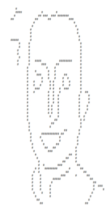
Which Digit?



Which are Faces?



Which Digit?



Face or not face?

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

What you should do:

1. Implement three classification algorithms for detecting faces and classifying digits:
   (a) Perceptron
   (b) Naive Bayes Classifier
   (c) Neural Network
2. Design the features for each problem, and write a program for extracting the features from each image.

3. Train the three 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 a function of the number of data points used for training.
4. Compare the performances of the three 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 of data points used for training.
   - The prediction error (and standard deviation) as a function of the number 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. The only things I want you to implement by yourselves are the core operations of the three algorithms.
- 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 70% 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. Once you hit that barrier of 70%, don't spend more time on the algorithm.
- It's OK if you have a bad performance (less than 70%) on a single type of data and algorithm. Example: everything is above 70% except Naive Bayes on Digits. I am fine with that, it's hard to get every algorithm work well on every dataset.
- 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.
- There is absolutely no excuse for being confused, having unanswered questions or feeling helpless! Please drop by our office hours, post on Piazza, e-mail us if you have any questions.

- I am not a big fun 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 I 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. Please let us know if you have any questions.

Have fun!