Homework 5

CMPSCI 370 Spring 2018, UMass Amherst
Due: April 25, 11:55 PM
Instructor: Subhransu Maji
TA: Tsung-Yu Lin

Guidelines

Submission. Submit your answers via Gradescope that includes a pdf with your solutions and source code. You have a total of 5 late days for all your homework assignments which you can use in any way you like. However note that delay by even one minute counts as a full late day and submissions beyond the late days will not be given *any* credit. Please contact the instructor and obtain permission ahead of time if you seek exceptions to the policy.

Plagiarism. We might reuse problem set questions from previous years, covered by papers and webpages, we expect the students not to copy, refer to, or look at the solutions in preparing their answers. We expect students to want to learn and not google for answers. Any instance of cheating will lead to zero credit for the homework, and possibly a failure grade for the entire course.

Collaboration. The homework must be done individually, except where otherwise noted in the assignments. 'Individually' means each student must hand in their own answers, and each student must write their own code in the programming part of the assignment. It is acceptable, however, for students to collaborate in figuring out answers and helping each other solve the problems. We will be assuming that you will be taking the responsibility to make sure you personally understand the solution to any work arising from such a collaboration.

Matlab requirements. The code should work any Matlab version ($\geq$ 2011a) and relies on the Image Processing Toolbox for some functions.

Python requirements. We will be using Python 2.7. The Python starter code requires `scipy`, `matplotlib` and `pillow` for loading images and `numpy` for matrix operations (at least v1.12). If you are not familiar with installing those libraries through some package manager (like `pip`), the easiest way of using them is installing Anaconda. Contact the course staff if you are having trouble installing these.

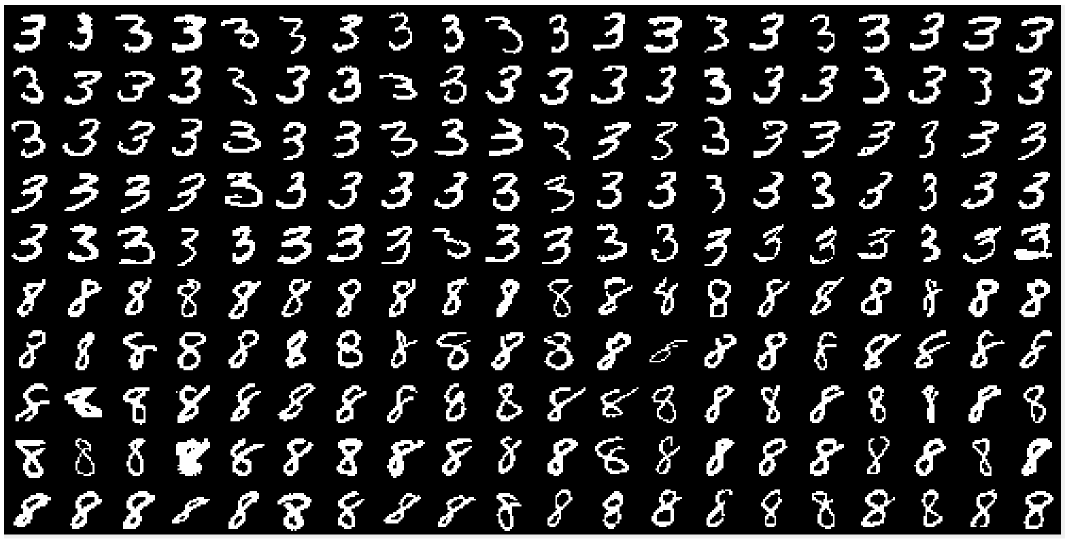
Using other programming languages. We made the starter code in Python and Matlab. You are free to use other languages such as Octave or Julia with the caveat that these will not be supported by the course staff.

The MNIST dataset

In this homework you will train a decision tree classifiers to predict whether an image is a 3 or a 8. The dataset for this homework can be loaded into Matlab by typing `load(data.mat)`. In Python, you can use the function `loadmat` in `utils.py` to store the data in a variable. E.g., `data = utils.loadmat('data.mat')`. The data is split into `train` and `test` sets.

For each split, the features and labels are in variables `x` and `y` respectively. E.g., `data.train.x` is an array of size $28 \times 28 \times 1 \times 200$ containing 200 digits. In Python, data is organized as a dictionary (i.e. `data["train"]["x"]`) and the array has size $28 \times 28 \times 200$. There are 100 digits each from classes 3 and 8, each of size 28×28. Class labels are given by the variable `data.train.y`. Thus, pixel `(i,j)` in the the `k'th` training example is `data.train.x(i,j,1,k)` with label `data.train.y(k)`. In Python, the same pixel can be accessed thorugh `data["train"]["x"][i, j, k]` with label `data["train"]["y"][k]`. The `test` set contains 100 examples from the two class in the same format. This data is a subset of the much larger MNIST dataset [1].

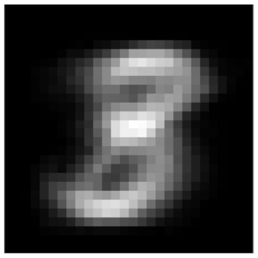
The features are simply the binary pixel values, i.e., each pixel is either 0 or 1. You can visualize the dataset using `montageDigits(x)` function. Below is the output of `montageDigits(data.train.x)`. *Tip: `montageDigits(x)` internally uses the `montage` command in Matlab which has a `size` option controlling the number of rows and columns of the output.*

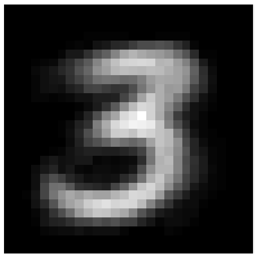


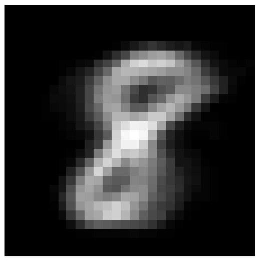
The function `avgImg = montageDigits(x)` additionally returns the average of all images in x. You can visualize it using `imagesc(avgImg); colormap gray`. This is useful for debugging your feature selection algorithms in decision trees. Below are the outputs of:

- `avgImg = montageDigits(data.train.x)`, i.e., all training images
- `avgImg = montageDigits(data.train.x(:,:,:, data.train.y==3)`, i.e., only the 3s
- `avgImg = montageDigits(data.train.x(:,:,:, data.train.y==8)`, i.e., only the 8s

[1] `http://yann.lecun.com/exdb/mnist/`


all


3s


8s

1 Decision trees

Answer the following about decision trees.

1. (4 points) What is the accuracy of an empty decision tree on the training set? Accuracy is the fraction of the examples correctly classified. What about its accuracy on the test set? Briefly explain your answer.

2. Suppose you are allowed to ask the value of one pixel for predicting the label of the image. Which pixel should it be? This is a decision tree of depth 1.

   (a) (15 points) To answer this question write a function `score = scoreFeatures(x, y)`. The `score` is a matrix of size 28×28 that corresponds to the number of examples `x` that can be correctly classified if the value of that pixel was known. From this find the pixel that has the highest score. Include the code, the score visualized as an image using `imagesc(score); colormap gray; axis image`. Also write down the (x,y) coordinates of pixel with the highest score and the value of the score.

   (b) (5 points) Write down the decision tree that obtains the best classification in the form of an if-then-else statement. For example, ”if pixel(5,10) == 1 then predict 3, else predict 8”. The exact syntax is not important.

   (c) (5 points) Apply this rule on the test set. What accuracy does this rule obtain?

3. Compute a decision tree of depth 2 to classify the data. Suppose (x,y) was the pixel selected in the previous step. This divides the training data into two subsets, one for which pixel(x,y) = 1, and another for which pixel(x,y) = 0. The best pixel in the next level of the tree can be obtained by repeating the previous step for each subset of the data.

   (a) (15 points) Apply the above to construct the next level of the tree. Write down the resulting classifier using if-then-else statements. For example,

```
if pixel(5,10) == 1
    if pixel(14,22) == 1
        predict 3;
    else
        predict 8;
    end
else
    if pixel(11,6) == 1
        predict 8;
    else
        predict 3;
    end
end
```

   (b) (5 points) Apply this rule on the test set. What accuracy does this rule obtain?

2 Linear classifier

The codebase includes a function: `model=linearTrain(x, y)` that trains a linear classifier given features `x` and labels `y`. The features `x` are $d \times N$ dimensional and labels `y` are $1 \times N$ dimensional. To use the function you have to reshape the images to a matrix where each column corresponds to a training example. The returned model has two fields: `model.w` is the weight vector of size $d + 1$ corresponding to weights on features and a bias term and `model.classLabels` corresponding to the labels. In Python model is structured as a dictionary, so you should use `model["weights"]` and `model["classLabels"]`.

Once the model is trained you can obtain predictions using `ypred = linearPredict(model, x)`. This function expects the input `x` in the same format as the training data.

- `(10 points)` What accuracy does the linear model obtain on the test set? Include the code for training and evaluating the model.
- `(5 points)` Visualize the positive and negative parts of the weights by reshaping the first d dimensions (i.e., ignoring the bias term) as an image. You compute the positive and negative components as `wp=w.*(w > 0)` and `wn=-w.*(w < 0)` and using `imagesc(reshape(wp, [28 28]))` to display them. In Python, you can use `plt.imshow(wp.reshape((28, 28))`.

3 Nearest neighbor classifier

`(15 points)` Using Euclidean distance between pixels as the similarity between images plot the accuracy of a k nearest-neighbor classifier for `k=1,3,5` on the test set. Include the code for your implementation.

4 Bag-of-visual-words representations

In this part you will implement this Bag-of-visual representation by the following two steps: 1) constructing a dictionary of visual words, and 2) represent each image as a histogram over the words. The second part is counted as extra credits and you need to report the accuracy with linear classifiers to obtain the full extra credits.

- `[15 points]`: Implement the function `C=constructDictionary(x, K, ps)`. The function takes the entire training set `x` as input and outputs a dictionary `C` of size `ps x ps x K` where `K` is the number of visual words and each word is a patch of size `ps x ps`. To construct a dictionary, randomly draw 20 patches from each images in the training data to obtain a set of sampled patches and group all the sampled patches into `K` clusters by k-means clustering. In MATLAB, you can call the function `kmeans` for clustering. For Python users, you need the `scikit-learn` package for `sklearn.cluster.KMeans`. Take a look at the first example in the documentation[2]. Use `K=100` and `ps=9` and visualize the dictionary by `montageDigits` as shown in the following figure:

Take a look at the start code `evalBoVW.m`. Include your implementation and the visualization of dictionary in your submission.

[2]`https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html`

- **Extra credits:** `[15 points]`: Implement the function `y=encodeImage(im, C)`. The function encodes the image `im` with the dictionary `C` as a histogram over the visual words. For each pixel in the image, consider the patch of size `ps x ps` centered at the pixel. The patch is assigned to the visual word with smallest sum-of-squared distance. You can ignore the pixels at the boundary if the patch is not entirely in the image. For each visual word, compute the number of patches that are assigned to it. This gives you a K-dimensional vector `y` where each entry is the number of patches that belong to a visual word. Encode all the images with the dictionary and train a linear classifier to classify 3 and 8. You can find more details in `evalBoVW.m`. In your submission, include your implementation and report the accuracy on test set. You should be able to get the accuracy around 97%.
- *Hint:* To find the nearest visual word, you will find the function `pdist2` useful in MATLAB. The equivalent function in Python is `scipy.spatial.distance.cdist`.

5 What to submit?

In addition to what is required for each problem, include all the relevant code, and intermediate results needed for explaining your implementation. **Finally, congratulations on completing all the homework assignments!**

6 Submission and rubric

- Follow the outline on Gradescope for your submission.
- A fraction of the points for each coding assignment will be for readability and efficiency of your code. Thus it is important to properly document parts of the code you implemented.