CS 6923: Machine Learning

Spring 2018

Homework 4

Submit on NYU Classes by Fri. April 26 at 11:55 PM (5 minutes BEFORE midnight). You may work together with one other person on this homework. If you do that, hand in JUST ONE homework for the two of you, with both of your names on it. You may *discuss* this homework with other students but YOU MAY NOT SHARE WRITTEN ANSWERS OR CODE WITH ANYONE BUT YOUR PARTNER.

IMPORTANT SUBMISSION INSTRUCTIONS: There is no separate programming part in this homework. Submit your solutions in ONE file.

This homework gives you practice with Decision Trees, Neural Nets, and PCA.

1. The entropy of a dataset S is defined as follows:

$$Entropy(S) = -\sum_{l \in L} \frac{N_l}{N} \log_2 \frac{N_l}{N}$$

where L is the set of class labels for the examples in S (e.g., + or -), N is the number of examples in S, and N_l is the number of examples in S that have label l.

(In lecture, we focused on the definition for the case where there are 2 classes. The above definition is the generalization to k class problems, for $k \geq 2$. Note that the logarithm is base 2, even with more than 2 classes.)

Let x_i be a discrete-valued attribute of the examples in dataset S, and let V be the set of possible values of attribute x_i. For $v \in V$, let S_v be the subset of examples in S satisfying $x_i = v$.

Consider a classification problem with categorical attributes. Applied to such a problem, the decision tree algorithm discussed in class builds a decision tree where each node is labeled by an attribute x_i. When building the tree, at each node, the algorithm chooses the decision (i.e., the x_i) that minimizes

$$\sum_{v \in V} \frac{|S_v|}{|S|} Entropy(S_v)$$

This is equivalent to saying that it chooses the x_i that maximizes the following quantity, which is sometimes called the Information Gain of x_i (with respect to S)

$$\text{Information-Gain}(S) = Entropy(S) - \sum_{v \in V} \frac{|S_v|}{|S|} Entropy(S_v)$$

It can be shown that Information-Gain(S) is always greater than or equal to 0. Recall that we consider $0 \log_2 0$ to be equal to 0.

(a) Answer the following question without using a calculator. Which dataset has higher entropy: A dataset with 3 positive examples and 5 negative examples, or a dataset with 6 positive examples and 2 negative examples?

(b) Calculate the Information Gain of x_1 in the following dataset. Show your work.

	x_1	x_2	x_3	r
$x^{(1)}$	T	F	F	+
$x^{(2)}$	F	T	F	+
$x^{(3)}$	T	T	F	-
$x^{(4)}$	T	F	T	+
$x^{(5)}$	F	F	T	-
$x^{(6)}$	F	F	F	-

(c) Run the decision tree algorithm described in class (without any pruning) to produce a tree with 0% training error on the dataset in the previous problem. Show the resulting tree.

(d) In information theory, the entropy of a discrete random variable taking values in $\{1, \ldots, n\}$ is $-\sum_{i=1}^{n} P[X = i] \log_2 P[X = i]$. This is the expected number of bits needed to encode a randomly drawn value of X, under the "most efficient"code. It is standard to use $H(X)$ to denote the entropy of X.

(In fact, the definition of entropy is not quite standardized. In some definitions of entropy, the logarithm has a base other than 2.)

Given two discrete random variable, X and Y, both taking values in a finite set, the conditional entropy of Y given X, written $H(Y|X)$ is defined to be

$$H(Y|X) = \sum_x P[X = x] * \left(\sum_y -P[Y = y|X = x] * \log_2 P[Y = y|X = x] \right)$$

Here $\sum_x$ and $\sum_y$ denote, respectively, the sum over all possible values x of X and y of Y. The quantity

$$H(Y) - H(Y|X)$$

is called the *mutual information* between X and Y. Interestingly, this quantity is symmetric with respect to X and Y, and is also equal to

$$H(X) - H(X|Y)$$

We can relate this quantity to our definition of Information Gain. Consider an example drawn uniformly at random from from dataset S. Let Y be the label of the example, and let X be the value of attribute x_i in the example. Using the definition of $H(Y|X)$, we see that the information gain of x_i in dataset S is $H(Y) - H(Y|X)$. This is why the Information Gain of x_i is sometimes called the mutual information betwen attribute x_i and the label.

Consider the dataset in part (b) again. Let Y be the label of an example drawn uniformly at random from the dataset. let X be the value of x_1 in the example. What are the values of $H(X)$ and $H(Y|X)$? What is the value of $H(X) - H(Y|X)$?

(e) Information gain has many nice properties, and is a reasonable quantity to use when choosing attributes to place in decision tree nodes. However, it has some drawbacks.

When used with categorical attributes, it is said that Information Gain favors attributes with many possible values. In other words, an attribute with, e.g., 100 possible values will usually have a higher information gain than an attribute with only 2 possible values, even if it is not better in terms of helping to classify examples.

As an extreme case, consider two attributes used in classifying drivers who have received a ticket, according to whether or not the driver will appeal the ticket and try to get it dismissed. One

attribute is a 9-digit driver's license number, which we are treating as a categorical attribute with 10^9 possible values. The other attribute has only 2 values, plus or minus, indicating whether the ticket was issued to the driver of a car or of a truck. Why is the first attribute likely to have higher gain than the second? Considering that our goal is to have a tree with good prediction accuracy, why wouldn't we want to place the first attribute into a node in our decision tree?

2. The backpropogation pseudocode we covered in lecture was for regression with 1 output node, using stochastic gradient descent.

   In some regression problems, we may want to predict more than one output value. For example, we might want to predict a person's location by specifying the longitude and latitude (2 numbers), or tomorrow's temperature and humidity. To use a single neural net for such problems, we add additional output nodes. Each of these output nodes is connected to the nodes in the previous layer.

   Pseudocode for backpropagation, training a neural net with K output nodes, is presented in Figure 11.11 in the textbook. This pseudocode assumes that the neural net has d input nodes (not counting $x_0 = 1$), H hidden nodes (not counting $z_0 = 1$), and K output nodes.

   Each hidden node h applies a sigmoid function to the weighted sum $w_h^T x$. So its output z_h is equal to $sigmoid(w_h^T x) = \frac{1}{1+e^{-w_h^T x}}$. We say that the *activation function* at hidden node h is the sigmoid function. More generally, if z_h was equal to $g(w_h^T x)$ for some other function g, we would say that g was the activation function at hidden node h.

   Each output node just outputs the linear function $v_i^T z$ directly (so $y_i = v_i^T z$). This is equivalent to saying that the activation function at the output nodes is the identity function, since $y_i = g(v_i^T z)$ where g is the identity function (the function $g(t) = t$ for all $t \in \mathcal{R}$).

   The backpropogation performs gradient descent to minimize the sum of the squared errors, over all of the output nodes, on the input x^t:
   $E(W, v|\mathcal{X}) = \sum_i (\frac{1}{2}(r_i - y_i)^2)$

   (a) The pseudocode in Figure 11.11 is performing stochastic gradient descent. Each iteration of the repeat loop (processing all examples in the training set) is called an "epoch". Suppose there are 500 training examples each with 10 attributes, $K = 3$, and the neural net has 4 nodes in the hidden layer. If we run the above psuedocode for 100 epochs, how many times is weight v_{ih} updated, for $i = 2$ and $h = 3$?

   (b) The pseudocode in Figure 11.11 implements the gradient descent update rules for the weights, which are computed by taking partial derivatives. The value $\Delta(v_{ih})$ for updating the weight from hidden input number h to the ith output node is

$$\begin{aligned}
\Delta(v_{ih}) &= -\eta \frac{\partial E(W, v|\mathcal{X})}{\partial v_{ih}} \\
&= -\eta \frac{\partial}{\partial v_{ih}} (\frac{1}{2}(r_i - y_i)^2) \\
&= -\eta (r_i - y_i)(-\frac{\partial y_i}{\partial v_{ih}}) \qquad \text{by the chain rule} \\
&= \eta (r_i - y_i) \frac{\partial}{\partial v_{ih}} (v_i^T z) \qquad \text{by the definition of } y_i \\
&= \eta (r_i - y_i) z_h
\end{aligned}$$

   This is the LMS (least mean square) rule, if we focus just on output node i and the H inputs z_{ih} into it.

Suppose that $K = 3$ and you are especially concerned about avoiding prediction error for the third output y_3. To address this concern, you define a new error function, which gives more weight to errors in the third output:
$E_{new}(W, v|\mathcal{X}) = \frac{1}{2}[(r_1 - y_1)^2 + (r_2 - y_2)^2 + 5(r_3 - y_3)^2]$.

i. With the new error function, we need to modify the update rules. What is the value of $\Delta v_{i,h}$ for $i \in \{1, 2\}$, with the new error function? What is the value of $\Delta v_{i,h}$ for $i = 3$, with the new error function? Be sure to answer both questions.

ii. In the textbook and lecture slides, there is a derivation of the value $\Delta w_{h,j}$, which is used for updating the weight from input node j to hidden node h. Look at this derivation and then answer the following question: What is the value of $\Delta w_{h,j}$ for the new error function?

3. In the previous question, we considered a neural net for a regression problem with multiple outputs. In this question, we will also discuss neural nets for three other types of problems.

To make it easier to refer to the four types of neural nets, we'll give them names. We'll call the neural net in the previous question NeuralNetRK (R for regression, K for the number of outputs). We now give the names of the other three types of neural nets, with their descriptions.

NeuralNetCB: NeuralNetCB is a standard neural net for binary classification, with one hidden layer. It has a single output node which outputs a value y, where y is the predicted value of $P[Class1|x]$. It uses sigmoid functions as the activation functions both for the hidden nodes and for the output nodes. [1] The goal of the backpropagation is to minimize cross-entropy error. The pseudocode for backpropagation in this case is the same as in Figure 11.11, if d is set to 1, except that the line $y_i = v_i^T z$ has to be replaced by $y_i = \frac{1}{1+e^{-v_i^T z}}$.

NeuralNetCK: This neural net is for classification with $K > 2$ classes. There are K output nodes i, and y_i is the predicted value of $P[Classi|x]$. In this case, each output y_i is equal to $\frac{e^{v_i^T z}}{\sum_{\ell=1}^{K} e^{v_\ell^T z}}$. Note that the sum of the y_i is 1, which is appropriate since the y_i are the estimated probabilities for the K labels of x. (In fact, the denominator in the expression for y_i is just a "normalizing factor" that causes the sum of the y_i to equal 1.) The error function is the generalization of cross-entropy to K classes: $-\sum_{i=1}^{K} r_i^t \log y_i^t$, where for each t, $r_i^t = 1$ if x^t is in Class i, and $r_i^t = 0$ otherwise. The pseudocode for backpropagation in this case is the same as in Figure 11.11, except that the line $y_i = v_i^T z$ has to be replaced by $y_i = \frac{e^{v_i^T z}}{\sum_{\ell=1}^{K} e^{v_\ell^T z}}$.

NeuralNetRZeroOne: This neural net is designed for regression problems with K outputs where each output is either an element in the set $\{0, 1\}$, or a real value in the interval $[0, 1]$. The error function is squared error, the same as for NeuralNetRK. This neural net has sigmoid activation functions in both the hidden nodes AND the output nodes. So each output node computes $y_i = \frac{1}{1+e^{-v_i^T z}}$. Changing the pseudocode for backpropagation in this case requires changing both the line $y_i = v_i^T z$ and the lines computing Δv_i and Δw_h (as computed in bppy.py and bp.m, described below).

(a) If we have a regression problems with K outputs, where each output is an element in the interval $[0, 1]$, why might it be better to use NeuralNetRZeroOne instead of NeuralNetRK?

(b) If we have a regression problem with K outputs, where each output is an element in $\{0, 1\}$ why don't we just use NeuralNetCK, instead of NeuralNetZeroOne? Note that in both neural nets, the outputs y_i will be elements in the interval $[0, 1]$.

[1] Another popular choice of activation function for such a network is *tanh*, the hyperbolic tangent function.

(c) On NYU Classes, with this homework, you will find a simple implementation of NeuralNetRZeroOne. The Python version is in bppy.py and sigmoidpy.py and the MATLAB version is in in bp.m and sigmoid.m. You may choose whether to use the Python or MATLAB version.

Read through the file bp.py or bpy.m including the comments.

In the comments at the beginning of the program, there are 3 examples on which the code can be run: XOR, autoassociation, and function approximation. The XOR example trains the net to compute the XOR function. The autoassociation example trains the net to take as input a vector of length 8 with exactly one 1 (and all other entries 0), and to produce the same vector as output – which sounds easy until you look at the number of hidden units in the net (hint, it's not 8). The function approximation example trains the neural net to compute a scaled version of the function $f(x) = cos(x)$.

(i) Run the XOR example and do the following: (a) Look at the output and report the error for epochs 10, 20, 30, and 50. Is the error increasing or decreasing? (b) Report the error from the final epoch. (c) Modify the code so that you can print out the values $y_1, \ldots, y_4$ for all 4 input examples (0 0, 0 1, 1 0, 1 1) in the final epoch. Report those values.

(ii) Run the autoassociation example. Report the error in the first epoch and in the final epoch. In the last epoch, report the values of $y_1, \ldots, y_8$ for the input $x = [1, 0, 1, 0, 1, 0, 1, 0]$ (modify the code to do this).

(iii) Run the function approximation example. Report the error in the first epoch and in the final epoch.

(iv) Look at the code. Is the code implementing batch gradient descent or stochastic gradient descent?

(v) In the function approximation example, the training examples are for a scaled version of the function $f(x) = cos(x)$. The unscaled inputs to the cos function are in the domain of the cos function; they are values between $-\pi$ and π. They are scaled to be between -1 and 1.
It is common to scale the inputs to neural nets to be in this range (we probably should have done this in examples 1 and 2). Such scaling is done for a number of reasons, including to improve the speed of training.
Look at the call to bp in the function approximation example. What arithmetic expression is being used to scale the inputs?
The outputs are also being scaled. What arithmetic expression is being used to scale the outputs?
Why were the output values also scaled here? Why would this neural net have difficulty achieving small error with the unscaled output $cos(x)$?

4. This problem will give you practice with some of the linear algebra involved in PCA. For a quick review the basics of eigenvalues and eigenvectors, see the following short webpage: `http://math.oregonstate.edu/home/programs/undergrad/CalculusQuestStudyGuides/vcalc/eigen/eigen.html`.

(a) By hand, find the characteristic polynomial of the matrix $A =$

$$\begin{bmatrix} 0 & 14 \\ 2 & 3 \end{bmatrix}$$

(b) By hand, solve for the eigenvalues of A using the characteristic polynomial you just computed.

(c) Solve for the eigenvectors of A using those eigenvalues. The L_2 norm of your eigenvectors should be equal to 1.

(d) Validate your results in Python (numpy) using linalg.eig(A) or in matlab using the command [V,D]=eig(A).

Give the returned eigenvalues and eigenvectors.

5. (a) Consider the following training set, consisting of 4 unlabeled examples, with real-valued attributes x_1, x_2, and x_3. The training set is represented by a matrix X, where each row of the matrix corresponds to an example, and column i corresponds to the ith attribute.

$$\begin{bmatrix} 4 & 1 & 3 \\ 8 & 5 & 3 \\ 6 & 0 & 1 \\ 1 & 4 & 5 \end{bmatrix}$$

Begin by centering the data, causing the sample mean in each dimension to be 0, as follows. Calculate the sample means for each column $[s_1, s_2, s_3]$ by hand, and then subtract these values from the entries in each column (e.g. s_3 is subtracted from all values in column 3). Call the resulting training set (matrix) B. Show the matrix B.

(b) By hand, calculate $s_{1,2}$, the sample covariance of x_1 and x_2 in B. Show your result.

Recall that the sample covariance of x_i and x_j for dataset a dataset with N examples is

$$\frac{\sum_t (x_i^t - s_i) * (x_j^t - s_j)}{N - 1}$$

(c) For the same B as in the previous question, use the Python command np.cov(B) or the MATLAB command cov(B) to produce the sample covariance matrix of X. Entry $1, 2$ should be equal to the value you computed in the previous question (ignoring round-off errors).

Sample covariance matrices are symmetric positive semi-definite. (By definition, a symmetric matrix is positive semi-definite if all of its eigenvalues are non-negative.)

What is the largest eigenvalue of the sample covariance matrix of B? Use MATLAB or Python to compute this answer.

(d) Again using the same B as in the previous question, run:

```
MATLAB
% PCA(X) returns [W,Z,eigvals]
% The columns of W are the eigenvectors of the covariance matrix of X.

[W,Z,eigvals]=pca(X)

PYTHON
import sklearn.decomposition as skd
import numpy as np
# .fit computes the principal components  (n_components of them)
# The columns of W are the eigenvectors of the covariance matrix of X
pca = skd.PCA(n_components = 3)
skd.PCA.fit(pca,X)
W1 = pca.components_
W = W1.transpose()
Z = pca.transform(X)
```

The first column of the matrix W is a vector w of norm 1, such that the projection of the data in X in the direction of w has the largest variance possible. That is, it is the solution to the problem of maximizing the variance of the entries of w'X' subject to the constraint $||w|| = 1$.

Z is a new representation of the data in matrix X, using the attributes produced by PCA, rather than the original attributes. The first column of Z corresponds to the attribute associated with the principal (largest) eigenvalue of the sample covariance matrix. The second column of Z corresponds to the attribute associated with the second largest eigenvalue of the sample covariance matrix, etc.

If we wanted to do dimension reduction on X, and reduce the dimension of the data to 2 (using new, transformed features), we could replace X by the first 2 columns of Z.

Show the first two columns of Z.

5. Consider the matlab command

[W,Z,eigvals]=pca(X)

or the Python commands above.

PCA does not just project the data, it also centers it around the origin (which doesn't affect the variance). This is accomplished by computing the sample mean m of the examples (rows) in X, and subtracting it from each row of X, prior to computing the projection. If x is the first row (example) of X, and z is the corresponding first row of z, then its first entry, z_1, is computed as follows: $z_1 = w^T * (x^T - m^T)$ or equivalently, $z_1 = (x - m) * w$.

More generally, if N is the number of examples (rows) in X, and M is the $N \times N$ matrix whose rows are all equal to m, we get $Z = (X - M) * W$.

Because the columns of W are orthogonal, and have unit length, $X = ZW^T + M$.

If we use PCA to reduce the dimension to k attributes, we want to use only the first k principal components, corresponding to the first k columns of W and Z. Let W_k and Z_k denote the matrices consisting of the first k columns of W and Z. In this case, we can use the same calculation, $X = ZW^T + M$, but with W_k and Z_k, to get an approximation to X.

Follow the instructions below for either MATLAB or Python, and then answer the questions. [Note: You will be using different image datasets depending on whether you follow the instructions for MATLAB or for Python.]

MATLAB

Download the file Yale_64x64.mat which is on NYU Classes with this assignment. This dataset is taken from a webpage set up by Deng Cai, `http://www.cad.zju.edu.cn/home/dengcai/Data/FaceData.html`. The webpage also contains additional related datasets and references to related research papers.

The file you are downloading is a dataset containing images of faces. Each image is a 64x64 pixel array. The images are contained in a matrix called fea. The rows of the matrix fea are the images (examples). The features are the pixels. Each example is represented by a vector of real numbers of length 4096, listing the pixels from left to right, row by row, from top to bottom.

Load the file into MATLAB. Run the following code which displays some of the images in the dataset:

```
faceW = 64;
faceH = 64;
numPerLine = 5;
ShowLine = 5;

Y = zeros(faceH*ShowLine,faceW*numPerLine);
for i=0:ShowLine-1
   for j=0:numPerLine-1
     Y(i*faceH+1:(i+1)*faceH,j*faceW+1:(j+1)*faceW) =
         reshape(fea(i*numPerLine+j+1,:),[faceH,faceW]);
   end
end

imagesc(Y);colormap(gray);
```

You can display the face corresponding to Example t in the dataset by executing the following commands:

```
imagesc(reshape(fea(t,:),[faceH,faceW]));colormap(gray)
```

PYTHON

The code below loads a dataset containing images of faces and displays some of them. You will need to run the code to answer the given questions. (You may need to install pillow or something similar.)

Each image is a 62x47 pixel array. The images are read into a matrix called fea. The rows of the matrix fea are the images (examples). The features (columns) are the pixels. Each example is represented by a vector of real numbers of length 2914, listing the pixels from left to right, row by row, from top to bottom.

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd

from sklearn.datasets import fetch_lfw_people
lfw_people = fetch_lfw_people(min_faces_per_person=70)
n_samples, h, w = lfw_people.images.shape
npix = h*w
fea = lfw_people.data

def plt_face(x):
    global h,w
    plt.imshow(x.reshape((h, w)), cmap=plt.cm.gray)
    plt.xticks([])

plt.figure(figsize=(10,20))
nplt = 4
for i in range(nplt):
    plt.subplot(1,nplt,i+1)
    plt_face(fea[i])

plt.show()
```

You can display the face corresponding to Example t in the dataset by executing the following commands:

```
plt_face(fea[t])
plt.show()
```

(a) Display the fifth face in the dataset using the appropriate command above. Print the resulting display.

(b) Compute the mean of all the examples in the dataset fea. (That is, compute an image such that each pixel i of the image is the mean of pixel i in all the images in fea.) Display it using a modification of the above command. Give the Matlab of Python commands you used and show the resulting display.

(c) Let's do dimensionality reduction with pca. Using MATLAB or Python, compute the 6 top principal components of the data matrix fea. Give the MATLAB or Python commands you used. What are the values of the associated 5 attributes of the fifth image in the dataset?

(d) Using the reconstruction equation $X = W^T Z + M$ described above, but with just the first 6 columns of Z and W (the attributes associated with the first 6 principal components), approximate the fifth image in the dataset. Give the MATLAB or Python commands you used. Display the resulting approximate image and print the resulting display. Repeat with the first 100 columns of Z and W.

(These are representations of the fifth image, based on 6 or 100 features instead of the original 4096 (MATLAB) or 2914 (Python) pixel features.)