Homework 6 [100 pts]: Machine Learning

General Information

Due: 11:59 PM, Tuesday, December 13th, 2016

Notes:

- For the written questions put your answers in a pdf (with your name in the pdf) and submit to the corresponding assignment on Gradescope.
- For the coding questions, **rename** each provided file `code.py` to `firstname_lastname_code.py` and submit the file to the corresponding assignment on canvas. Also, **append the code** to the end of your pdf.
- You have **one late day** for this assignment. Late homework will be penalized 10% per day (each day starts at 11:59 PM on the due day).
- Homework turned in after one late day **will not be accepted**.

Exercise 1 [10 pts]: k-NN Regression

The traditional k-NN algorithm classifies new data points by the majority class of the k most similar observed points. But we can extend k-NN to perform regression. The k-NN regression algorithm predicts the value of new data points by averaging the values of the k nearest points in the observed set. “Nearest” in this case means shortest euclidean distance. The following table gives the observed values of function f for four points in the Cartesian plane:

p	**f[p]**
(0,0)	3
(2,0)	20
(0,2)	-5
(2,2)	30

1. (2 pts) Predict f[(3,2)] with 1-NN, 2-NN, 3-NN, and 4-NN.
2. (4 pts) Suppose you have *n* observed data points, all with different values. You are trying to predict the value of a point *p* that is closest to the observed point of maximum value. For each of the following prediction methods, state which method will predict the greater value for *p* and explain why. If there is not enough information to make a determination, say so and explain why.
    a. 1-NN vs. 2-NN
    b. 2-NN vs. 3-NN
    c. 1-NN vs. n-NN

   d. 2-NN vs. n-NN

3. (2 pts) A simple modification to the k-NN algorithm is to add weights to observed points based on how close they are to the point being predicted. If $x_1, \ldots, x_k$ are the *k* nearest neighbors of *p*, then the predicted function value of *p* is

$$f(p) = \sum_{i=1}^{k} w_i f(x_i)$$

where

$$w_i = 1 - \frac{d(x_i, p)}{\sum_{j=1}^{k} d(x_j, p)}$$

$d(\cdot, \cdot)$ being a distance function. Suppose you have only two observed points, (0,0) and (2,0), with values as described in the table. Describe the set of all points whose predicted values under weighted 2-NN are exactly the same as those for non-weighted 2-NN, assuming the standard Euclidean distance function.

4. (2 pts) Using the setup in problem 3, what values does non-weighted 2-NN predict for *p*=(0,0) and *p*=(2,0)? What values does weighted 2-NN predict? Note that it is perfectly fine to run k-NN on points that are already observed; it's just that the closest observed point has distance 0 from the point you're trying to predict.

Exercise 2 [10 pts]: Game Theory

1) (6 pts) Mary and John are deciding whether to watch movie A or movie B. Both of them secretly prefer movie B (each gains +5 utility from watching it). Movie A would also be enjoyable, but less so (+2 utility per person from watching it). However, due to social pressure, it is generally uncool to *say* that you like movie B. If, for example, John says that he wants to watch movie B, but Mary says that she wants to watch movie A, then John will gain -8 utility points from social embarrassment. Similarly, Mary will gain -8 utility points if she advocates movie B and John advocates movie A. If Mary and John advocate the same movie, they will both watch that movie; otherwise, they will watch movie A by default.
   a) Draw the payoff matrix for this social game.
   b) Is this a prisoner's dilemma? Why or why not?

2) (4 pts) Two firms, Firm 1 (F1) and Firm 2 (F2), are deciding which type of product to release on the market next. Both firms make the same three types of products, X, Y, and Z. However, depending on which product each firm releases, the subsequent profits of each will vary. The following is the payoff matrix for the firms. The positive numbers reflect profits made, and the negative numbers reflect money lost.

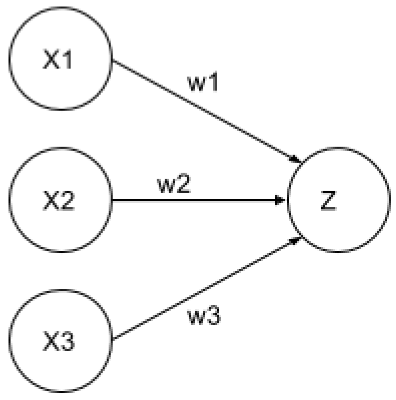
	F1:X	F1:Y	F1:Z
F2:X	F1 = +1, F2 = -2	F1 = -2, F2= -4	F1 = -2, F2 = -3
F2:Y	F1 = +2, F2 = 0	F1 = +1, F2 = +4	F1 = +1, F2 = +4
F2:Z	F1 = +4, F2 = -2	F1 = -2, F2 = -2	F1 = +3, F2= +3

a) List the Nash equilibrium of this game. Explain what this means for each firm.
b) For each square you listed in (a), is that square Pareto optimal? If not, which squares are Pareto optimal?

Exercise 3 [5 pts]: Utility and Decision Theory

Suppose you are looking to buy a used car. You can either go to the dealer immediately and make a decision, or you can first pay a website for more information about the quality of each car. If you don't pay the website, you'll buy a good quality used car with probability 0.5 and a poor quality car with probability 0.5. If you pay for information, however, you'll buy a good quality car with probability 0.8 and a poor quality car with probability 0.2. A good quality car will cost $1000 dollars in maintenance over the next year, but a poor quality car will cost $5000 in maintenance. Draw the decision network for this problem (see Figures 16.6 and 16.7 for examples). How much money is the information worth in expectation?

Exercise 4 [5 pts]: Neural Nets



The image above depicts a simple neural network with three input nodes, one output node, and no hidden nodes. We have w1 = 0.7, w2 = 0.3, w3 = 0.9. The activation of the output node Z is equal to X1*w1+X2*w2+X3*w3 (there is no additional activation function like sigmoid). Suppose our input is X1 = 2, X2 = -1, X3 = 0, and that the gradient of the evaluation function with respect to Z is 2. Give the new weights of w1, w2, and w3 after one round of backpropagation with learning rate 0.05.

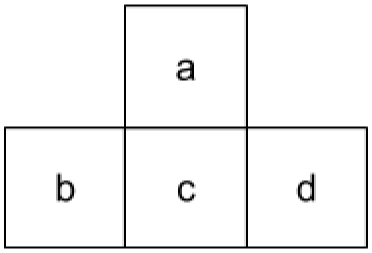
Exercise 5 [25 pts]: Decision Trees

A company is trying to learn the set of conditions under which its new off-road vehicle can operate reliably. In certain conditions, the engine has a tendency to sputter and struggle, but in other conditions it operates well. The company has collected the following data:

Slope	**Terrain**	**Temperature**	**Humidity**	**Reliable?**
Steep	Mud	Hot	High	No
Steep	Mud	Hot	Low	No
Steep	Dirt	Hot	Low	No
Steep	Dirt	Cold	Low	Yes
Steep	Grass	Hot	Low	No
Steep	Grass	Cold	Low	Yes
Gentle	Mud	Cold	High	No
Gentle	Mud	Cold	Low	No
Gentle	Dirt	Hot	Low	Yes
Gentle	Dirt	Hot	High	No
Gentle	Grass	Cold	Low	Yes
Gentle	Grass	Hot	Low	Yes

1. (2 pts) Given the categories in the table, how many distinct environments are possible? (An environment is a tuple of slope, terrain, temperature, and humidity).
2. (8 pts) Calculate the information gain on engine reliability of initially splitting on each of slope, terrain, temperature, and humidity. Show your calculations. Preserve at least 2 decimal places of accuracy.
3. (8 pts) Construct and draw the entire decision tree for this problem, using maximum information gain to choose the decision nodes at each level. If two potential nodes are tied in their information gain score, break the tie by the order of the attributes in the table, left to right. This tree should perfectly classify engine reliability.
4. (5 pts) Draw a different tree with 4 decision nodes that perfectly classifies engine reliability (hint: split on temperature first. No need to use information gain here). Why didn't we find this smaller tree in the previous problem?
5. (2 pts) Suppose someone gives you a pre-made decision tree T for this problem, and that this tree misclassifies exactly 4 environments out of all possible environments. Unaware of this, you decide to test the accuracy of T by selecting a subset of 12 <u>distinct</u> environments from this domain uniformly at random and classifying them with T. What is the probability that T will perfectly classify all of the selected examples? [This sort of calculation is useful in PAC learning, which you can read about in section 18.5.]

Exercise 6 [30 pts]: POMDPs and MDPs



Consider the simple 4-state environment above. You are not sure exactly where you are, but your belief state is $< p(a), p(b), p(c), p(d) > = < 0.5, 0.125, 0.25, 0.125 >$. In any state you can attempt to go north, south, east, or west. If you hit a wall, you will receive a "wall" percept with probability 0.8. If you don't hit a wall, you receive a wall percept anyway with probability 0.1. For any action, there is a

- 0.7 chance that you move in the given direction.
- 0.1 chance that you move in a 90 degrees clockwise direction instead
- 0.1 chance that you move in a 90 degrees counterclockwise direction instead
- 0.05 chance that you move in the opposite direction
- 0.05 chance that you don't move at all

1. (2 pts) For each state a, b, c, d, list the possible resulting states of going north starting in that state.
2. (8 pts) Suppose you take a single action, "north," and do <u>not</u> receive a wall percept. Calculate the resulting belief state. Show all work.

Coding MDPs (20 pts)

Write a python file *firstname_lastname_mdp.py* that takes an environment file as an argument and prints out the utilities of each state, accurate to at least 3 decimal places. Use the value iteration algorithm to calculate the utilities (Figure 17.4 in the book).

The available actions in any environment are moving north, south, east, and west. Any action can be taken from any non-terminal state, but hitting a wall causes you to remain in the same place. Moving into a terminal state ends a run.

File format is as follows: the first line specifies the discount factor gamma. The second line specifies the "cost of living" for all non-terminal states. The third line specifies the transition probabilities in the following order: going in the intended direction, going 90 degrees clockwise, going the opposite direction as intended, and going 90 degrees counterclockwise of intended direction (there is no possibility of being stationary here). The rest of the file contains the grid (with no trailing newlines). Star '*' represents a state,

‘x’ represents a wall, and numbers represent the values of terminal states. Columns are separated by spaces, and rows are separated by newlines. All grids are rectangular.
Example file (from figure 17.1 in the book):

```
1
-0.04
0.8 0.1 0 0.1
* * * 1
* x * -1
* * * *
```

When the command `python firstname_lastname_mdp.py testfile.txt` is run, your program should print out a grid in the same format as the input grid, but with the stars replaced by utilities. The output from the above file (see Figure 17.3 in the book) should be

```
0.812 0.868 0.918 1
0.762 x 0.660 -1
0.705 0.655 0.611 0.388
```

We will run your code on several test files to ensure accuracy.

Exercise 7 [15 pts]: Linear Regression and Gradient Descent

In machine learning, *regression* is the problem of learning a model to approximate an unknown function. In order to learn a model, we must have training data, consisting of points $x \in \mathbb{R}^k$ and a known function $y = f(x) \in \mathbb{R}$. More precisely, suppose we have n pairs of k-dimensional inputs and their corresponding outputs. Let X be a $k \times n$ matrix whose columns are the data vectors $x_1, x_2, \ldots, x_n$, and let Y be an $n \times 1$ vector whose entries correspond to the true function outputs of the data vectors. A simple *linear model* for this scenario is

$$Y = X^{\mathrm{T}}w$$

where w is a $k \times 1$ vector of model weights, and T is the transpose operator.

Our goal is to learn a best-fit set of weights w for given X and Y, so that $Y \approx X^{\mathrm{T}}w$. To measure fitness, we use the following *regularized loss function*:

$$\mathrm{Loss}(w) = \frac{1}{2}||w||^2 + \frac{1}{2}||Y - X^{\mathrm{T}}w||^2 = \frac{1}{2}\sum_{i=1}^{k} w_i^2 + \frac{1}{2}\sum_{j=1}^{n}(y_j - x_j^{\mathrm{T}}w)^2$$

Note that $||\cdot||$ is the Euclidean norm. Our objective is to find a w such that this loss function is minimized.

The gradient of the loss function with respect to w is

$$\nabla_w \mathrm{Loss}(w) = w - X(Y - X^{\mathrm{T}}w)$$

Setting this equal to the zero vector, the closed-form solution for the optimal w can be obtained:

$$w = (I + XX^{\mathrm{T}})^{-1}XY$$

where *I* is the identity matrix.

1. (3 pts) Implement the function `exact_solution` in the provided python file `regression.py` (which you should of course rename in the usual way). The function takes numpy matrices X and Y and returns the best fit w. Use built-in numpy functions to perform the matrix calculations. (You can use numpy.linalg.inv for matrix inverses, numpy.eye for identity matrices, and numpy.dot for ordinary matrix multiplication, as well as other numpy functions).
2. (5 pts) Implement the function `gradient_descent`. Update w step-by-step until convergence, with learning rate $\eta = 10^{-5}$. Use $||\nabla_w \mathrm{Loss}(w)|| < 0.001$ as the stopping criterion. The update rule is $w = w - \eta \nabla_w \mathrm{Loss}(w)$. You can test your implementation against your exact solution by typing
   ```
   python regression.py
   ```
3. (7 pts) You are provided with `online_shares.csv`, which contains numerical attributes (features) of various online articles and how often each was shared. The task is to learn a model to predict the number of shares of an article given the features. Run your your exact solution function on the provided `online_shares` dataset by typing
   ```
   python regression.py online_shares.csv.
   ```
   Report the root mean squared error (rmse) and standard deviation. Does the linear model fit this data well? Why or why not? Provide a table of the learned weights w_i associated with each feature. Which 5 features were most important to the model, according to the weights?

Extra Credit [8 pts]

Given the data from exercise 7, bin all numerical features into quintiles. Provide a table with the information gains from initially splitting on each feature. Which 5 features provided the most information about the number of shares? Are any of these the same as the 5 most important features in linear regression? To help with this, we have provided a skeleton file `extra_credit.py` (which you should rename in the usual way). Run

```
python extra_credit.py online_shares.csv
```

to print out the information gains once you have implemented the indicated functions.