Robin Mehta (robimeht)
AI Homework 6

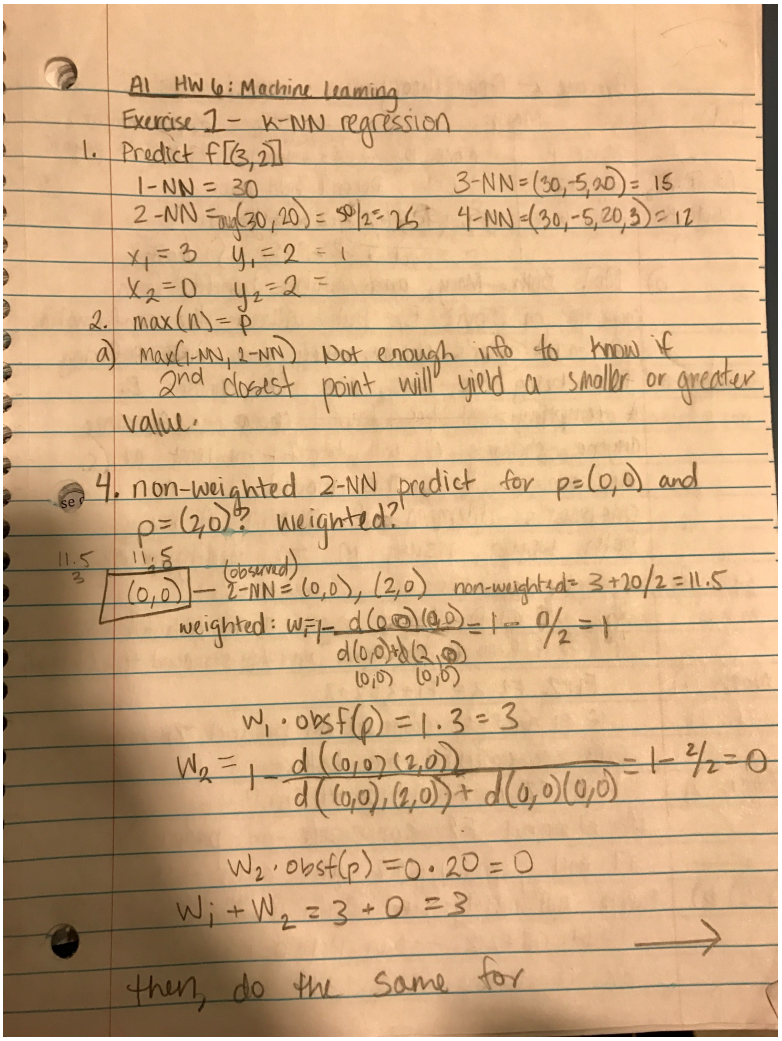
***Answers = highlighted
***Work shown = images

Exercise 1: k-NN Regression

1. Predict f[(3,2)]

1-NN = 30
2-NN = 25
3-NN = 15
4-NN = 12

Work:



2. Which predicts the greater value for p?

a) 1-NN or 2-NN?
1-NN
b) 2-NN or 3-NN?
Not enough information because we don't know the values of the 2nd and 3rd predicted points.
c) 1-NN or n-NN (n>1)?
1-NN
d) 2-NN or n-NN (n>2)?
Not enough information because we don't know the values of the 2nd or next n predicted points.

3. Describe the set of all points whose predicted values under weighted 2-NN are exactly the same as those for non-weighted 2-NN. Observed points: (0,0) and (2,0).

All points on line x=1

4. What values does non-weighted 2-NN predict for p=(0,0) and p=(2,0)? What values does weighted 2-NN predict?

Non-weighted(0,0): 11.5
Weighted(0,0): 1 – 2/2 = 3

Non-weighted(2,0) = 11.5
Weighted(2,0) = 20

Exercise 2: Game Theory

1. Mary & John deciding on Movie A or B.

a) Payoff matrix:

	Movie A	Movie B
Movie A	M(+2) J(+2)	M(-6) J(+2)
Movie B	M(+2) J(-6)	M(+5) J(+5)

b) No, this is not a prisoner's dilemma.
T = trade-off; R = reward; P = punishment; S = sucker's payoff
T > R > P > S must hold true to be a prisoner's dilemma. +2 > +2 > +5 > -6 is false.
Both Mary and John benefit most from cooperating on Movie B, but without the confirmation of each other's choice, they each risk suffering from being the only

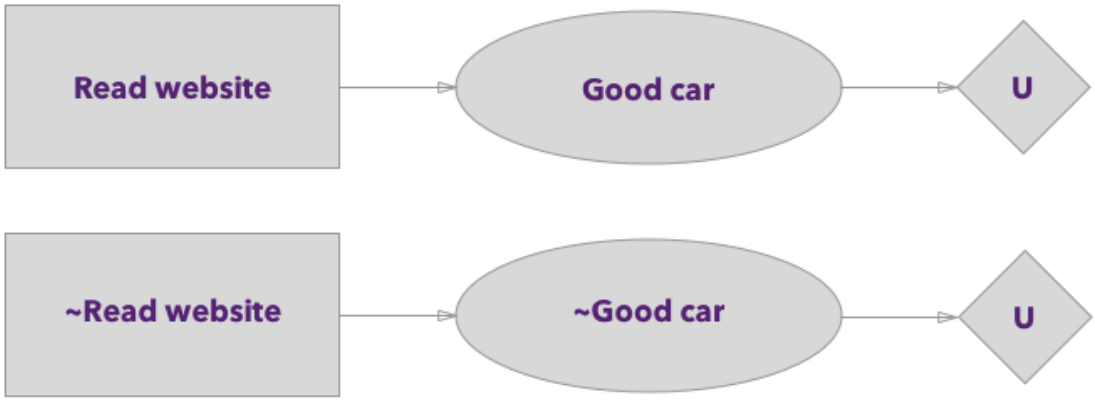
one to choose Movie B. A tempting choice is Movie A, since anyone choosing A will benefit at least at the minimum (+2). This could be a prisoner's dilemma if choosing A by both would result in negative utility points.

2. Two firms, product releases

a) Nash equilibria: (F1X, F2Y)
b) Is that square Pareto optimal? If not, which squares are Pareto optimal?
(F1X, F2Y) is not Pareto optimal
(F1Z, F2Y), (F1Z, F2Z), and (F1Y, F2Y) are Pareto optimal.

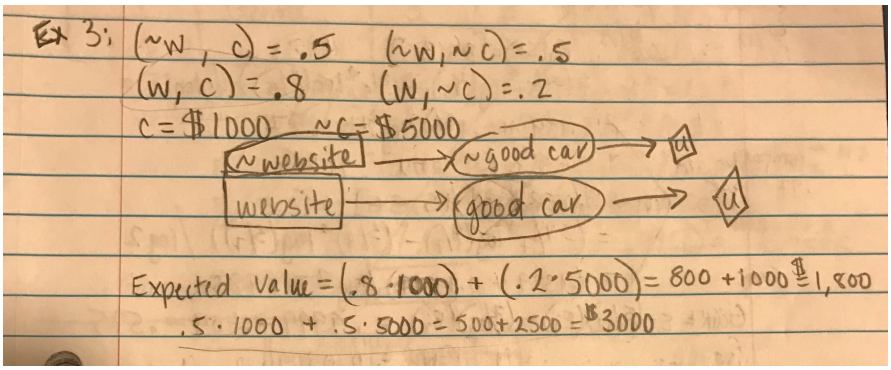
Exercise 3: Utility & Decision Theory

1. Draw a decision network. How much money is the information worth in expectation?



Expected Value(website) = (.8 * 1000) + (.2 * 5000) = $1800
Expected Value(no website) = (.5 * 1000) + (.5 * 5000) = $3000
The information is worth ($3000 - $1800) = **$1200** in expectation.

Work:

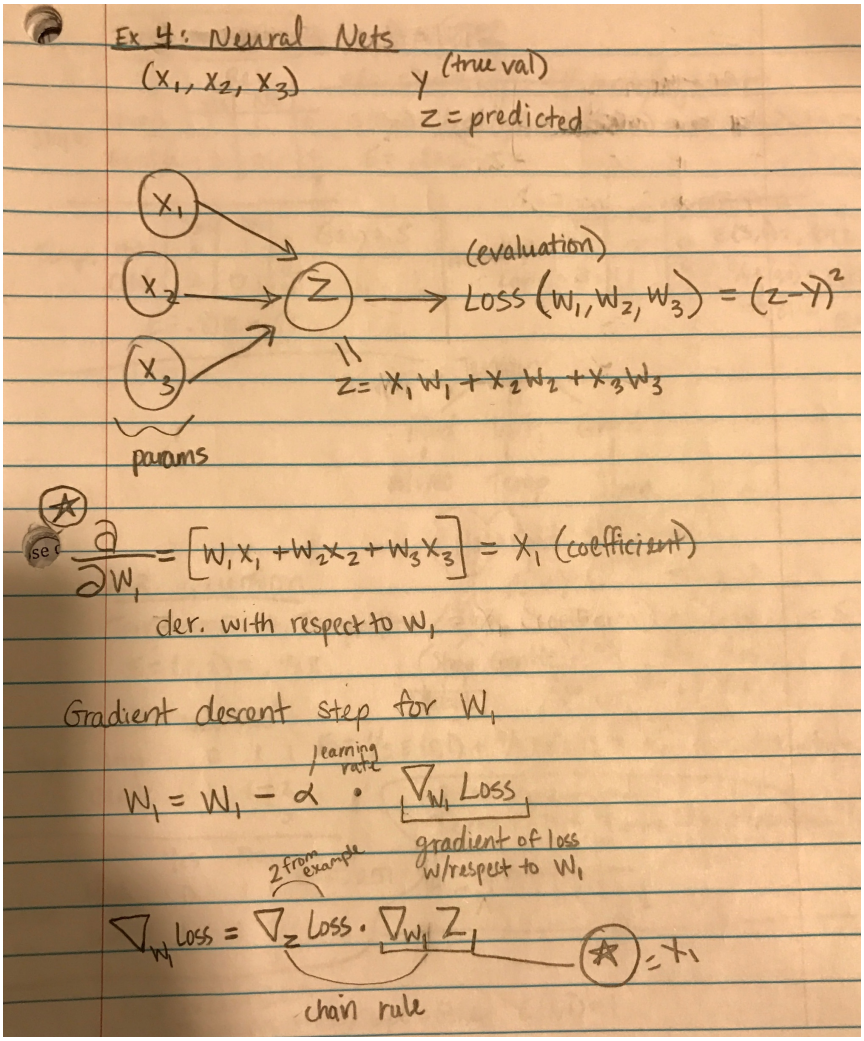


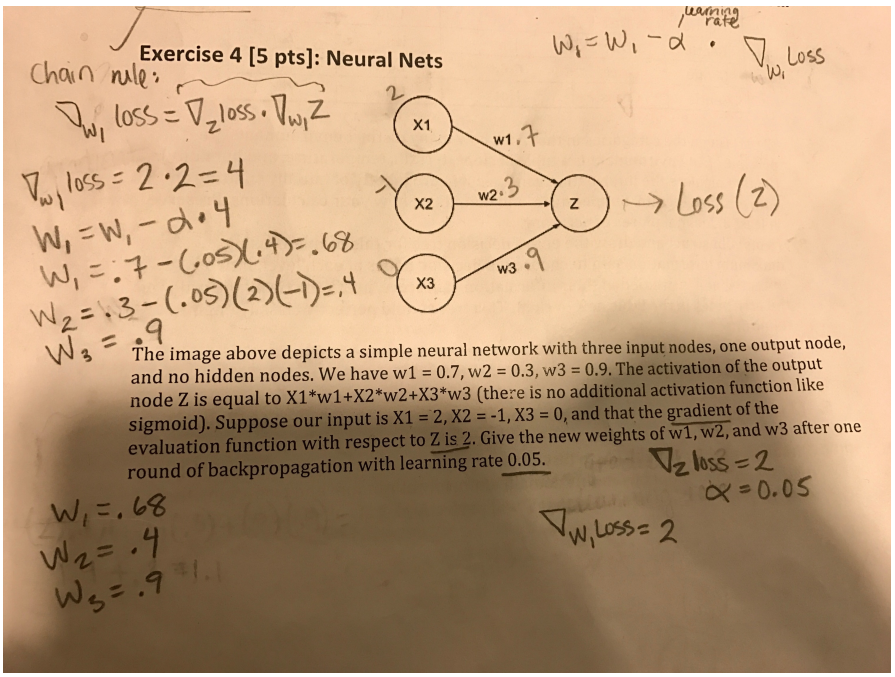
Exercise 4: Neural Nets

1. Give the new weights of w1, w2, and w3 after one round of backpropagation with learning rate 0.05.

w1 = .68
w2 = .4
w3 = .9

Work:





Exercise 5: Decision Trees

1. How many distinct environments are possible?

2 slopes * 3 terrains * 2 temps * 2 humidity = 24 distinct environments.

2. Calculate the information gain on engine reliability of initially splitting on each slope, terrain, temperature, and humidity.

Terrain: 0.37625
Humidity: 0.23674
Temperature: 0.072
Slope = 0.0215

Work:

Exc:5 : #2

target Entropy:

Ex: H(Loaded) = $-(0.99 * \log_2 0.99 + 0.01 \cdot \log_2 0.01)$ ≈ 0.08

Gain(ER) = : $B\left(\frac{5}{5+7}\right)$ =

target Entropy = $-\left[\frac{5}{12} \cdot \log(\frac{5}{12}) + \frac{7}{12} \cdot \log(\frac{7}{12})\right]$

$-(.417) \cdot \log(.417) + (.583) \cdot \log(.583)$

$= -[(.417)(-1.262) + (.583) \cdot (-.778)] =$

$-[(-.526) - .454] = -(-.98) = .98$

features: slope, terrain, tem, humidity →

Gain = E(target) − E(target, f I)

		Reliable?		
		Yes	No	
Slope	Steep	2	4	6
	Gentle	3	3	6
				12

E (ER, Slope) = P(steep) * E(2,4) + P(Gentle) * E(3,3)

$= (6/12) * E(2,4) + (6/12) * E(3,3)$

$E(2,4) = -2/6 \log(2/6) - 4/6 \log(4/6)/\log 2 = .917$ .159 + .117

$E(3,3) = -3/6 \log(.5) + 3/6 \log(.5)/\log 2 = .9966$ 1

$= (6/12)(.917) + (6/12)(.9966) =$ .4585 + .5 = .9585

Gain = .98 − .9585 = **.0215**

		Reliable		
		Yes	No	
Terrain	Mud	0	4	4
	Dirt	2	2	4
	Grass	3	1	4
				12

E(ER, terrain) = P(Mud) * E(0,4) + P(Dirt) * E(2,2) + P(Grass) * E(3,1)

P(mud) = 4/12 P(Dirt) = 4/12 P(Grass) = 4/12

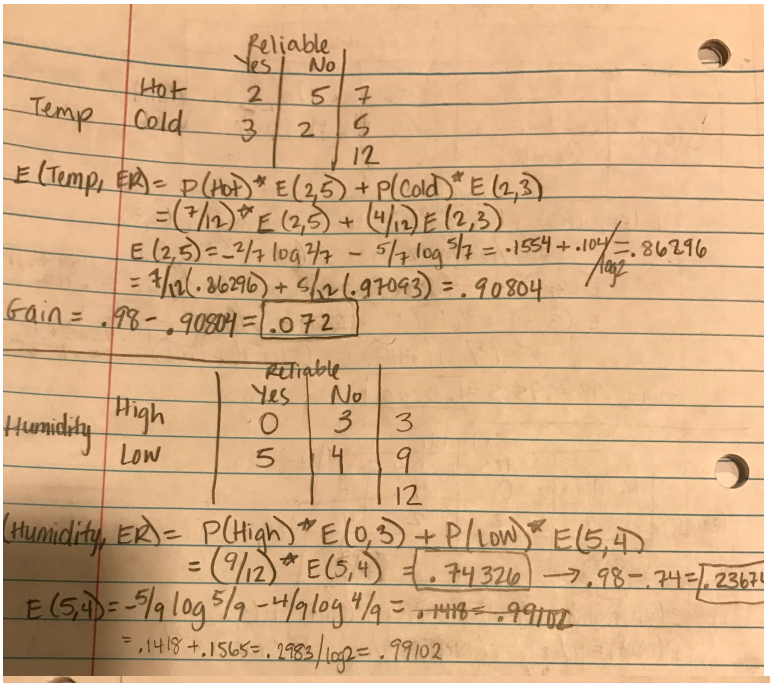
$E(0,4) = -4/4 \log(4/4) = 0$

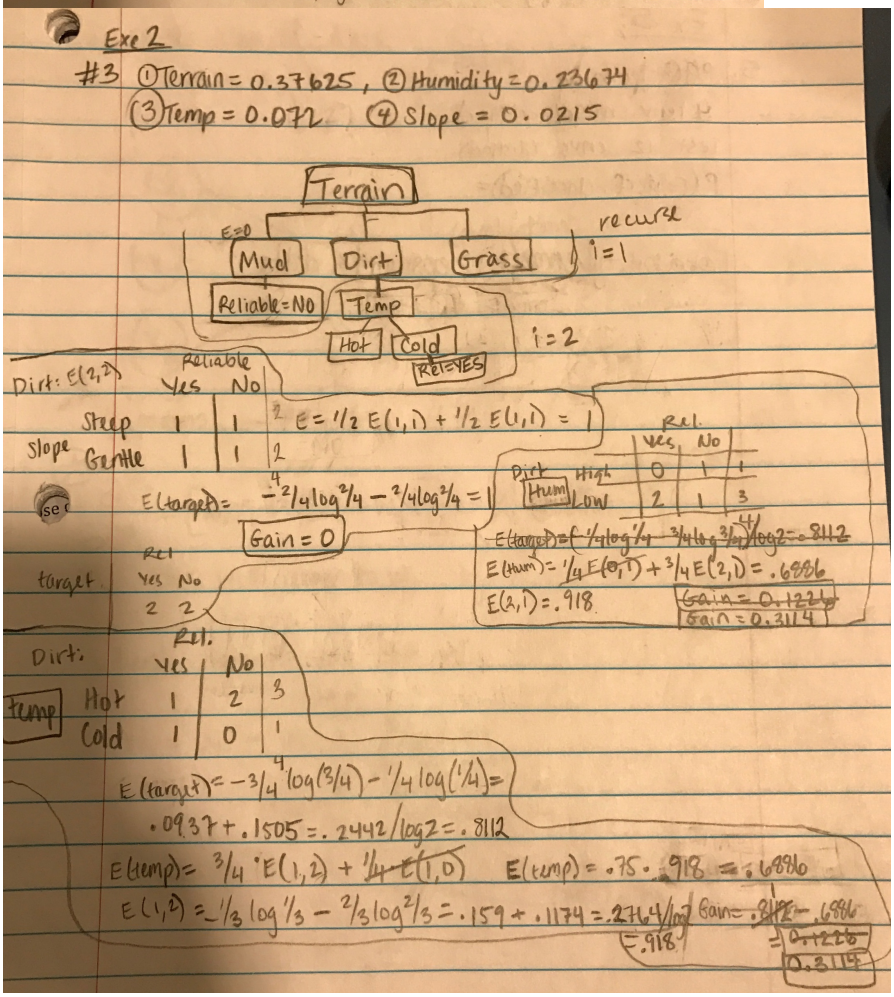
$E(2,2) = -.5 \log .5 - .5 \log .5 = .301/\log 2 = 1$

$E(3,1) = -3/4 \log 3/4 - 1/4 \log 1/4 / \log 2 = .50937 + .1505 = .811260$

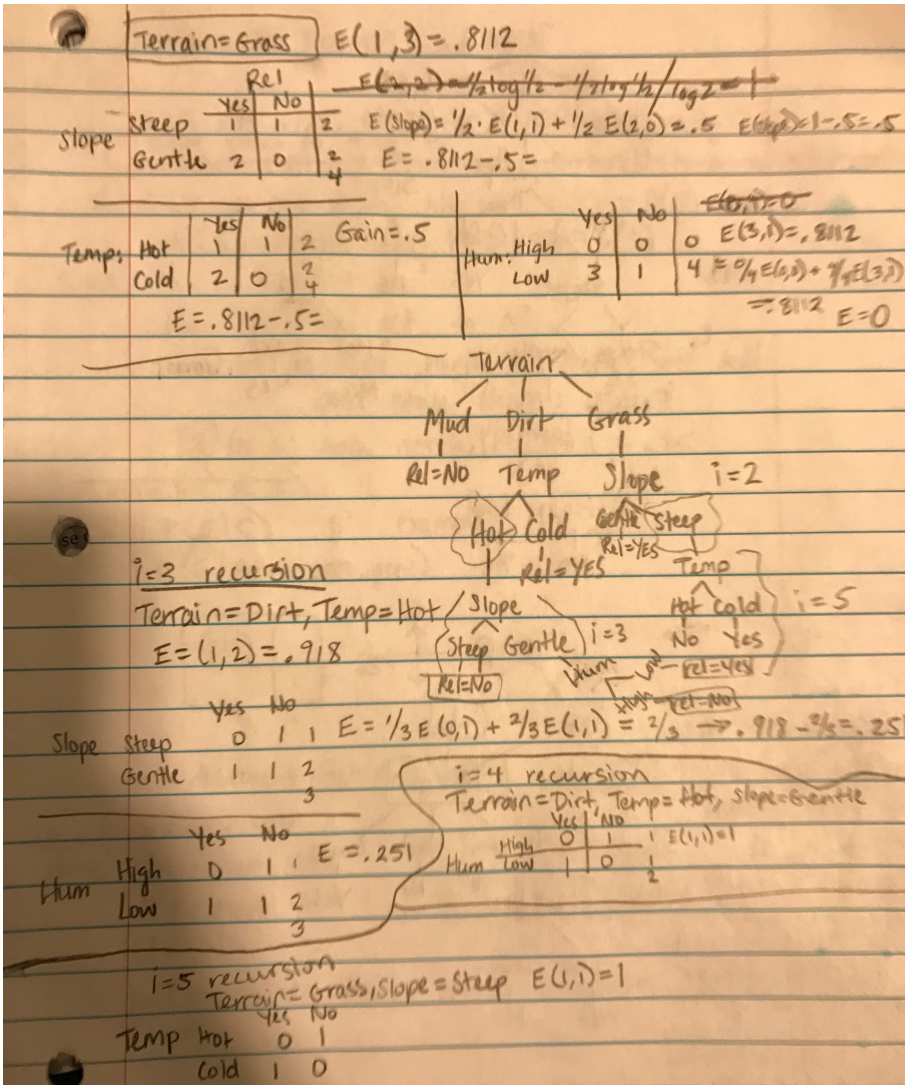
$E(ER, terrain) = 1/3(0) + 1/3(1) + 1/3(.81126) = .60375$

Gain = .98 − .60375 = **.37625**

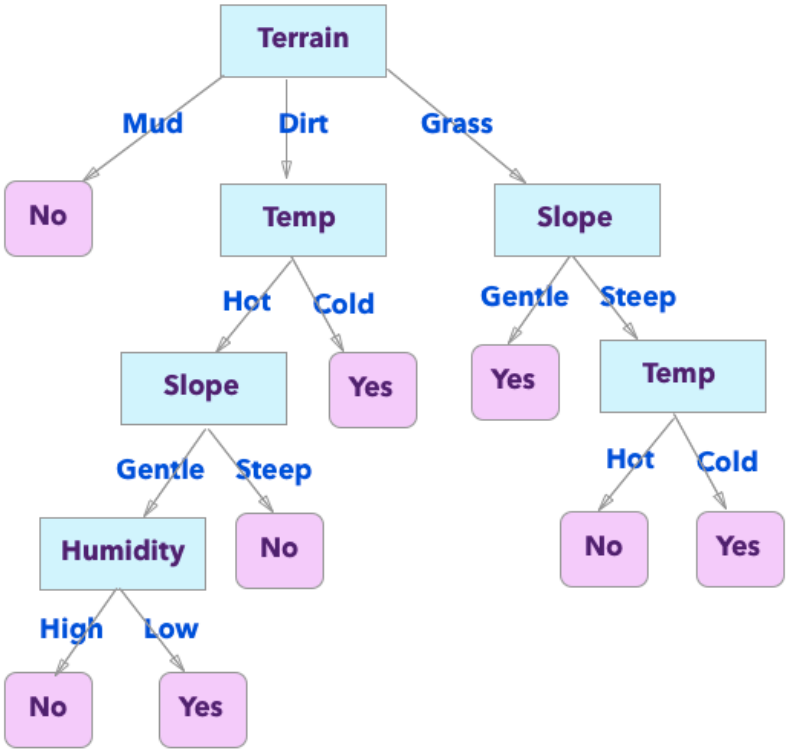




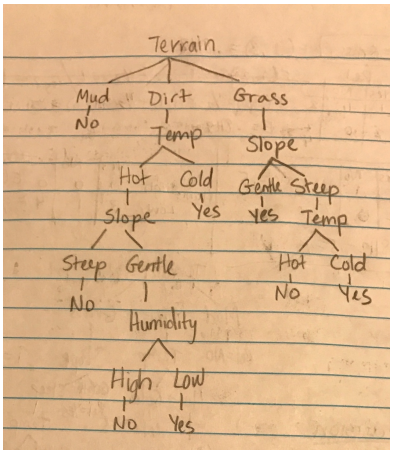
Recursive Step:



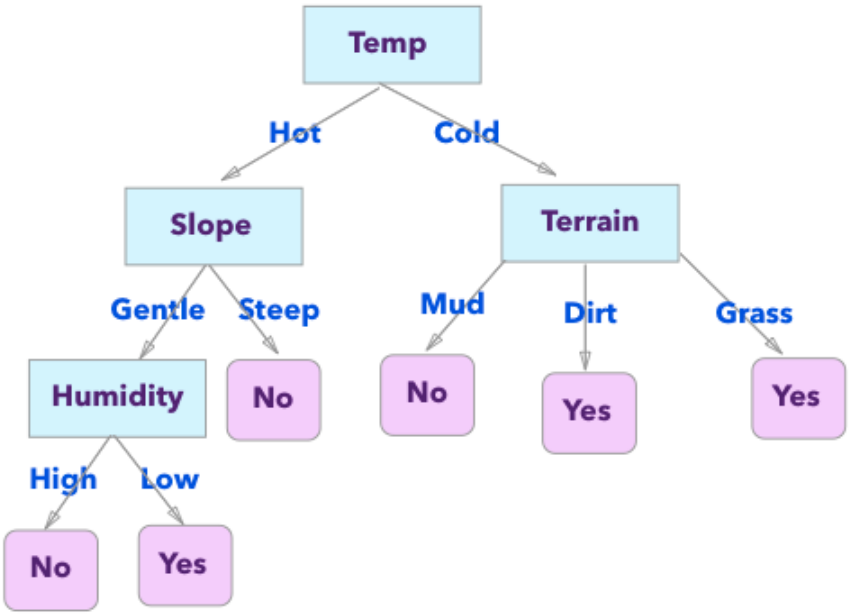
3. Construct and draw the entire decision tree using max information gain to choose nodes at each level.



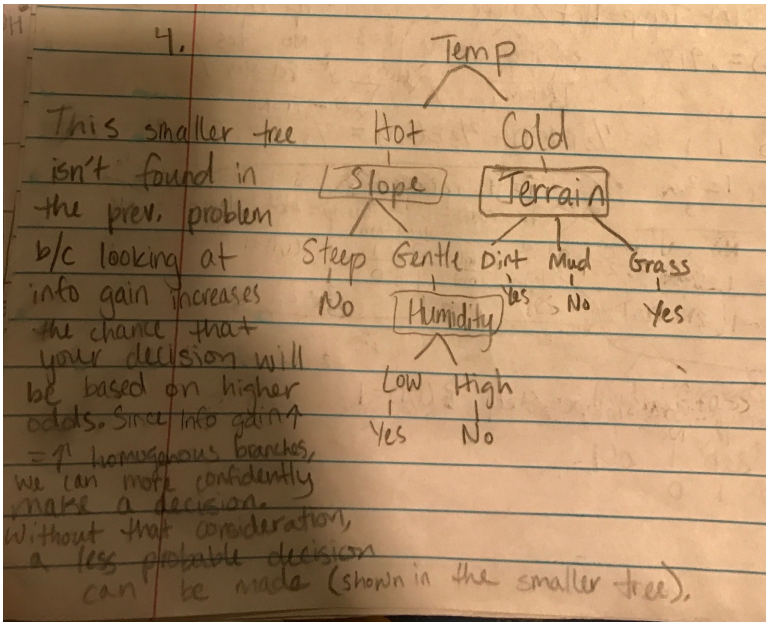
Work:



4. Draw a different tree with 4 decision nodes that perfectly classifies engine reliability



Work:



5. What is the probability that T will perfectly classify all of the selected examples?

Probability = Choose(20, 12) / Choose(24, 12)
= 125970 / 2704156
= 0.046584

Exercise 6: POMDPs and MDPs

1. For each state, a, b, c, d, list the possible resulting states of going north starting in that state.

State a:
a, c

State b:
b, c

State c:
a, b, c, d

State d:
d, c

2. Calculate belief state after going north:

Goal a:
Start a:
0.5*[north(0.7 * 1 * 0.2) + east(.1 * 1 * .2) + west(.1 * 1 * .2) + stay(.05 * 1 * .9)] = .1125
Start c:
0.25 [north(0.7 * 1 * .9)] = .1575
= 0.1125 + 0.1575 = 0.27

Goal b:
Start b:
0.125*[north(0.7 * 1 * .2) + south(0.05 * 1 * .2) + west(0.1 * 1 * 0.2) + stay(0.05 * 1 * .9)] = .026875
Start c:
0.25*[east(0.1 * 0.9)] = 0.0225
= 0.026875 + 0.0225 = 0.049375

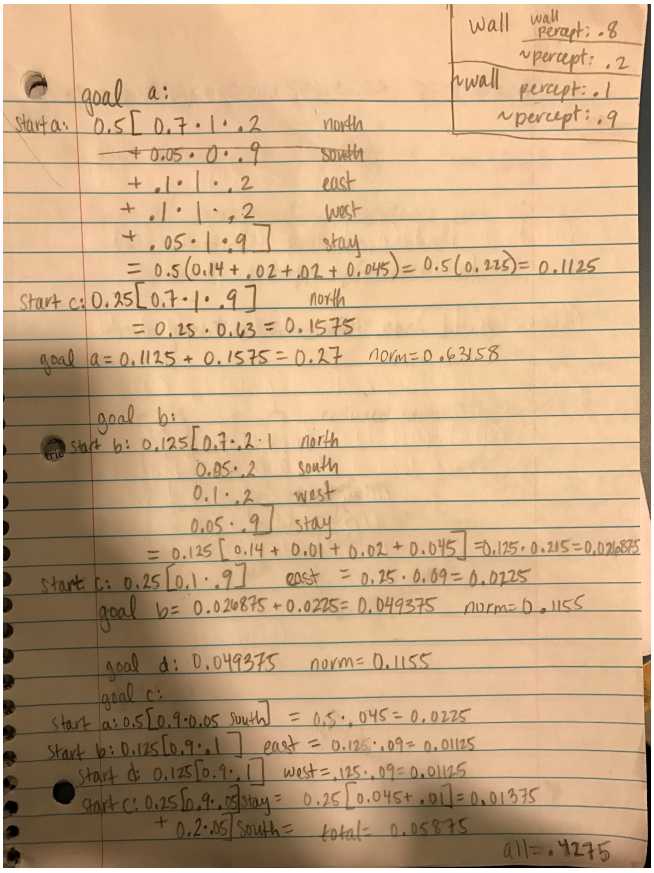
Goal c:
Start a:
0.5*[south(0.9 * 0.05)] = 0.0225
Start b:
0.125*[east(0.9 * .1)] = 0.01125
Start d:
0.125*[west(0.9 * .1)] = 0.01125
Start c:
0.25*[stay(0.9 * 0.05)] + south(0.2 * 0.05)] = 0.05875

Goal d:

Start d:
0.125*[north(0.7 * 1 * .2) + south(0.05 * 1 * .2) + east(0.1 * 1 * 0.2) + stay(0.05 * 1 * .9)] = .026875
Start c:
0.25*[west(0.1 * 0.9)] = 0.0225
= 0.026875 + 0.0225 = 0.049375

alpha = 0.4275
Initial belief state: <0.27, 0.049375, 0.05875, 0.049375>
Normalized belief state: <0.6316, 0.1155, 0.1375, 0.1155>

Work:



3. Coding MDPs

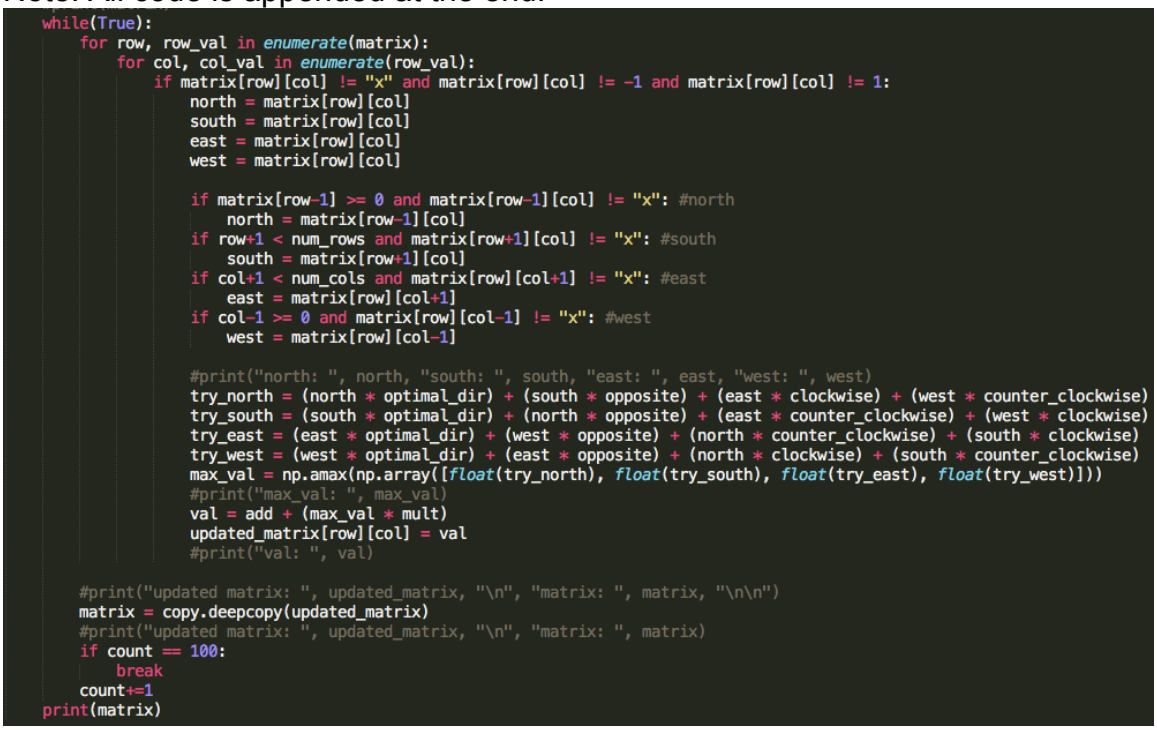
Run python robin_mehta_mdp.py testfile.txt

My code produces incorrect output. However, the logic seems sound.
Zhefan helped debug it, and Ryen looked over it, and I still couldn't figure out the bug.

Output:
```
[[0.7297, 0.7992, 0.8774, 1.0],
[0.6797, 'x', 0.6244, -1.0],
[0.6234, 0.5734, 0.5502, 0.3335]]
```

If anyone can figure out why this code isn't converging to the correct numbers, please let me know! I'm still curious.

Screenshot of main algorithm:
**Gamma = mult, reward = add, all other variables self-explanatory.
**Removed convergence code because code infinite loops. Not sure why!
Note: All code is appended at the end.

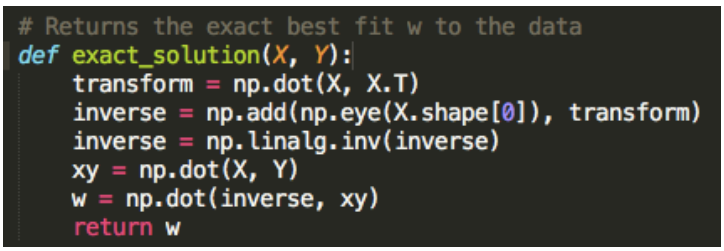


```
while(True):
    for row, row_val in enumerate(matrix):
        for col, col_val in enumerate(row_val):
            if matrix[row][col] != "x" and matrix[row][col] != -1 and matrix[row][col] != 1:
                north = matrix[row][col]
                south = matrix[row][col]
                east = matrix[row][col]
                west = matrix[row][col]

                if matrix[row-1] >= 0 and matrix[row-1][col] != "x": #north
                    north = matrix[row-1][col]
                if row+1 < num_rows and matrix[row+1][col] != "x": #south
                    south = matrix[row+1][col]
                if col+1 < num_cols and matrix[row][col+1] != "x": #east
                    east = matrix[row][col+1]
                if col-1 >= 0 and matrix[row][col-1] != "x": #west
                    west = matrix[row][col-1]

                #print("north: ", north, "south: ", south, "east: ", east, "west: ", west)
                try_north = (north * optimal_dir) + (south * opposite) + (east * clockwise) + (west * counter_clockwise)
                try_south = (south * optimal_dir) + (north * opposite) + (east * counter_clockwise) + (west * clockwise)
                try_east = (east * optimal_dir) + (west * opposite) + (north * counter_clockwise) + (south * clockwise)
                try_west = (west * optimal_dir) + (east * opposite) + (north * clockwise) + (south * counter_clockwise)
                max_val = np.amax(np.array([float(try_north), float(try_south), float(try_east), float(try_west)]))
                #print("max_val: ", max_val)
                val = add + (max_val * mult)
                updated_matrix[row][col] = val
                #print("val: ", val)

    #print("updated matrix: ", updated_matrix, "\n", "matrix: ", matrix, "\n\n")
    matrix = copy.deepcopy(updated_matrix)
    #print("updated matrix: ", updated_matrix, "\n", "matrix: ", matrix)
    if count == 100:
        break
    count+=1
print(matrix)
```

Exercise 7: Linear Regression & Gradient Descent

1. Implement function exact_solution in regression.py

Screenshot:



```
# Returns the exact best fit w to the data
def exact_solution(X, Y):
    transform = np.dot(X, X.T)
    inverse = np.add(np.eye(X.shape[0]), transform)
    inverse = np.linalg.inv(inverse)
    xy = np.dot(X, Y)
    w = np.dot(inverse, xy)
    return w
```

2. Implement function gradient_descent in regression.py

Screenshot:

```python
# Starts with an initial guess for w
# and performs gradient descent until it converges
def gradient_descent(X, Y, alpha=0.00001):
    w = np.zeros(X.shape[0])
    while(True):
        transform = np.dot(X.T, w)
        paren = np.subtract(Y, transform)
        mult = np.dot(X, paren)
        loss = np.subtract(w, mult)
        w = np.subtract(w, np.dot(alpha, loss))
        if np.linalg.norm(loss) <= 0.001:
            break
    return w
```

3. Learn a model to predict the number of shares of an article given the features. What is the root mean squared error (rmse) and standard deviation? Does the linear model fit this data well? Why or why not? Which 5 features were most important to the model, according to the weights?

```
Root mean squared error =  11594.7010699
Standard deviation 11626.8041057
-2.48910335443 , kw_min_min
-1.77846562306 , kw_avg_min
0.32035583568 , kw_max_avg
-0.30075797078 , kw_avg_avg
0.272368649134 , kw_max_min
0.175270323298 , num_hrefs
-0.158175770771 , num_self_hrefs
0.158108386742 , kw_min_avg
0.101698375506 , data_channel_is_lifestyle
0.101060927585 , self_reference_avg_sharess
0.097577356609 , data_channel_is_tech
0.0948516932657 , abs_title_subjectivity
0.0928313121419 , is_weekend
0.0927966335932 , self_reference_min_shares
0.0923656677283 , weekday_is_monday
0.0919024861678 , weekday_is_saturday
0.0914995751564 , LDA_03
0.0912512014273 , LDA_00
0.0907743661781 , LDA_04
0.0906044821084 , min_positive_polarity
0.0885161500676 , weekday_is_sunday
0.0879602322256 , global_subjectivity
0.0877267363812 , avg_positive_polarity
0.0876606180654 , max_positive_polarity
0.0875736816102 , global_rate_negative_words
```

```
0.0872578574587 , global_sentiment_polarity
0.0872040739461 , global_rate_positive_words
0.0871441332529 , min_negative_polarity
0.0868476853433 , rate_negative_words
0.0866901266358 , abs_title_sentiment_polarity
0.0855219651989 , weekday_is_friday
0.0854297711135 , n_non_stop_unique_tokens
0.0853247104195 , weekday_is_wednesday
0.0852146306404 , data_channel_is_socmed
0.0850249547405 , n_unique_tokens
0.0846668276774 , avg_negative_polarity
0.0832714243239 , weekday_is_thursday
0.0829714817466 , n_non_stop_words
0.0829150569661 , num_imgs
0.0828132567471 , n_tokens_title
0.0827224869284 , title_subjectivity
0.0819297303999 , rate_positive_words
0.0816141768271 , title_sentiment_polarity
0.0813487196037 , max_negative_polarity
0.0807475255885 , weekday_is_tuesday
0.0800795639694 , LDA_02
0.0789331651878 , num_keywords
0.0788688578602 , LDA_01
0.078303627987 , data_channel_is_world
0.0760345077338 , data_channel_is_entertainment
0.0733746202279 , num_videos
0.0728959932186 , data_channel_is_bus
0.0670116303407 , average_token_length
0.051552145364 , kw_avg_max
-0.0476890289303 , n_tokens_content
-0.0429924046456 , kw_min_max
-0.0187928937104 , kw_max_max
-0.00249235172393 , self_reference_max_shares
```

RMSE is within 1 standard deviation, so the linear model does not fit this data well. `kw_min_min`, `kw_avg_min`, `kw_max_avg`, `kw_avg_avg`, `kw_max_min` are the 5 features with the absval highest weights.

Extra Credit:

Code:

```python
# classvals is a numpy size n array of class values
# class values here are Y values thresholded into quintiles
# Return the entropy of classvals
def entropy(classvals):
    entropy = 0
    uniqueDict = {}
    for x in classvals:
        if x not in uniqueDict:
            uniqueDict[x] = 1
        else:
            uniqueDict[x] += 1

    probabilities = {}
    for key in uniqueDict:
        prob = float(uniqueDict[key]) / float(len(classvals))
        probabilities[key] = prob

    for key in probabilities:
        if probabilities[key] > 0:
            entropy += - (probabilities[key] * math.log10(probabilities[key]) / math.log10(5))
    return entropy
```

```python
# # colvals is a numpy size n array of column values
# # That is, a feature column with the features thresholded into quintiles
# # colvals indices correspond to those in the size n classvals array
# # Return expected entropy of classvals after splitting on colvals
def expected_entropy(colvals, classvals):
    uniqueDict = dict()
    for x in colvals:
        if x not in uniqueDict:
            uniqueDict[x] = 1
        else:
            uniqueDict[x] += 1

    probabilities = dict()
    for key in uniqueDict:
        prob = float(uniqueDict[key]) / float(colvals.shape[0])
        probabilities[key] = prob

    indices_dict = dict()
    for key in probabilities:
        #find all indexes where key in colvals occurs
        indices = []
        index = 0
        for num in colvals:
            if num == key:
                indices.append(index)
            index+=1
        indices_dict[key] = indices
    total = 0
    for key in indices_dict:
        total += probabilities[key] * entropy(indices_dict[key])
    return total
```

Bin all numerical features into quintiles. Provide a table with the information gains from initially splitting on each feature.

Entropy: 2.3219280792819137

LDA_02 , -4.57962448926
kw_avg_avg , -4.57962448926
global_sentiment_polarity , -4.57962448926
LDA_04 , -4.57962448926
global_subjectivity , -4.57962448926
LDA_01 , -4.57962448926
LDA_00 , -4.57962448926
LDA_03 , -4.57962448926
n_unique_tokens , -4.57962448926
kw_avg_max , -4.57962449124
kw_avg_min , -4.57962449519
average_token_length , -4.57962449519
global_rate_negative_words , -4.57962449519
n_non_stop_unique_tokens , -4.5796245031
global_rate_positive_words , -4.57962483121
kw_max_avg , -4.5796248846
avg_positive_polarity , -4.57962691067
n_tokens_content , -4.57963156729
self_reference_min_shares , -4.57963196391
avg_negative_polarity , -4.57964885228
self_reference_avg_sharess , -4.57965072364
self_reference_max_shares , -4.57967854536
n_non_stop_words , -4.57973163736
kw_max_min , -4.58127296161
rate_positive_words , -4.58178898762
rate_negative_words , -4.58207342018
min_negative_polarity , -4.58324609334
num_hrefs , -4.58831444075
num_self_hrefs , -4.60195699531
n_tokens_title , -4.61774941519
num_keywords , -4.62732825138
min_positive_polarity , -4.65878485708
max_negative_polarity , -4.6630827909
max_positive_polarity , -4.74297707232
kw_min_max , -4.76757163853
kw_min_avg , -4.76786673403
title_subjectivity , -4.78216625654
abs_title_sentiment_polarity , -4.81938559323
abs_title_subjectivity , -4.98447075017
kw_min_min , -5.00184019051
num_imgs , -5.01376477077
num_videos , -5.02209940492
title_sentiment_polarity , -5.02724778993
kw_max_max , -5.23857229788
data_channel_is_world , -5.25818447127
weekday_is_wednesday , -5.2797419746
weekday_is_tuesday , -5.28077857779

```
data_channel_is_tech , -5.28179724919
weekday_is_thursday , -5.28363896651
data_channel_is_entertainment , -5.28861518617
weekday_is_monday , -5.29832544473
data_channel_is_bus , -5.30866302754
weekday_is_friday , -5.32375208647
is_weekend , -5.3384703186
weekday_is_sunday , -5.42357796639
weekday_is_saturday , -5.43541409818
data_channel_is_socmed , -5.44101201147
data_channel_is_lifestyle , -5.45094571701
```

Which 5 features provided the most information about the number of shares?

```
LDA_02 , -4.57962448926
kw_avg_avg , -4.57962448926
global_sentiment_polarity , -4.57962448926
LDA_04 , -4.57962448926
global_subjectivity , -4.57962448926
```

Are any of these the same as the 5 most important features in linear regression?

`kw_avg_avg` is one of the same 5 most important features.

APPENDED CODE:

MDP PROBLEM:

robin_mehta_mdp.py

```python
from __future__ import print_function
import numpy as np
import sys
import math
import copy

def parse_file(path):
    mult = 0
    add = 0
    optimal_dir = 0
    clockwise = 0
    opposite = 0
    counter_clockwise = 0
    num_rows = 0
    num_cols = 0

    line_num = 0
    with open(path,'r') as data:
        for line in data.read().split('\n'):
            if line_num == 0:
                mult = float(line)
            if line_num == 1:
                add = float(line)
            if line_num == 2:

                word_num = 0
                for word in line.split(" "):
                    if word_num == 0:
                        optimal_dir = float(word)
                    if word_num == 1:
                        clockwise = float(word)
                    if word_num == 2:
                        opposite = float(word)
                    if word_num == 3:
                        counter_clockwise = float(word)
                    word_num += 1

            if line_num == 3:
                for point in line.split(" "):
                    num_cols += 1
            line_num += 1
        num_rows = line_num - 3
```

robin_mehta_mdp.py

```
43
44      Matrix = [[0 for x in range(num_cols)] for y in range(num_rows)]
45      with open(path, 'r') as data:
46          line_num = 0
47          for line in data.read().split('\n'):
48              if line_num >= 3:
49                  row_count = line_num - 3
50                  col_count = 0
51
52                  for word in line.split(" "):
53                      if word == "*":
54                          word = 0
55                      if word != "x":
56                          Matrix[row_count][col_count] = float(word)
57                      else:
58                          Matrix[row_count][col_count] = word
59
60                      col_count += 1
61                  #row_count += 1
62              line_num += 1
63
64      inputs = [mult, add, optimal_dir, clockwise, opposite, counter_clockwise, num_rows, num_cols]
65      return Matrix, inputs
66
67  def algorithm(matrix, inputs):
68      mult = inputs[0]
69      add = inputs[1]
70      optimal_dir = inputs[2]
71      clockwise = inputs[3]
72      opposite = inputs[4]
73      counter_clockwise = inputs[5]
74      num_rows = inputs[6]
75      num_cols = inputs[7]
76      count = 0
77      updated_matrix = copy.deepcopy(matrix)
78      #print("mult: ", mult, "add: ", add, "optimal: ", optimal_dir, "clockwise: ", clockwise, "opposite: "
79      #print(matrix)
80      while(True):
81          for row, row_val in enumerate(matrix):
82              for col, col_val in enumerate(row_val):
83                  if matrix[row][col] != "x" and matrix[row][col] != -1 and matrix[row][col] != 1:
84                      north = matrix[row][col]
```

robin_mehta_mdp.py

```
                if matrix[row][col] != "x" and matrix[row][col] != -1 and matrix[row][col] != 1:
                    north = matrix[row][col]
                    south = matrix[row][col]
                    east = matrix[row][col]
                    west = matrix[row][col]

                    if matrix[row-1] >= 0 and matrix[row-1][col] != "x": #north
                        north = matrix[row-1][col]
                    if row+1 < num_rows and matrix[row+1][col] != "x": #south
                        south = matrix[row+1][col]
                    if col+1 < num_cols and matrix[row][col+1] != "x": #east
                        east = matrix[row][col+1]
                    if col-1 >= 0 and matrix[row][col-1] != "x": #west
                        west = matrix[row][col-1]

                    #print("north: ", north, "south: ", south, "east: ", east, "west: ", west)
                    try_north = (north * optimal_dir) + (south * opposite) + (east * clockwise) + (west * counter_clockwise)
                    try_south = (south * optimal_dir) + (north * opposite) + (east * counter_clockwise) + (west * clockwise)
                    try_east = (east * optimal_dir) + (west * opposite) + (north * counter_clockwise) + (south * clockwise)
                    try_west = (west * optimal_dir) + (east * opposite) + (north * clockwise) + (south * counter_clockwise)
                    max_val = np.amax(np.array([float(try_north), float(try_south), float(try_east), float(try_west)]))
                    #print("max_val: ", max_val)
                    val = add + (max_val * mult)
                    updated_matrix[row][col] = val
                    #print("val: ", val)

        #print("updated matrix: ", updated_matrix, "\n", "matrix: ", matrix, "\n\n")
        matrix = copy.deepcopy(updated_matrix)
        #print("updated matrix: ", updated_matrix, "\n", "matrix: ", matrix)
        if count == 100:
            break
        count+=1
    print(matrix)

def main():
    matrix, inputs = parse_file(sys.argv[1])
    #runIterations(matrix, inputs)
    algorithm(matrix, inputs)

if __name__ == '__main__':
    main()
```

LINEAR REGRESSION / GRADIENT DESCENT:

robin_mehta_regression.py

```python
from __future__ import print_function
import numpy as np
import sys

# Given coefficients w,
# generate noisy random data points X, Y
# Where Y is approximately (X transpose)*w
# This is used to test gradient_descent
def generate_data(npoints, w):
    dim = np.size(w)
    X = np.random.uniform(-10, 10, (dim, npoints))
    Y = np.dot(X.T, w)
    err = np.random.normal(0, 1, (npoints))
    return X, Y+err


# Returns the exact best fit w to the data
def exact_solution(X, Y):
    transform = np.dot(X, X.T)
    inverse = np.add(np.eye(X.shape[0]), transform)
    inverse = np.linalg.inv(inverse)
    xy = np.dot(X, Y)
    w = np.dot(inverse, xy)
    return w

# Starts with an initial guess for w
# and performs gradient descent until it converges
def gradient_descent(X, Y, alpha=0.00001):
    w = np.zeros(X.shape[0])
    while(True):
        transform = np.dot(X.T, w)
        paren = np.subtract(Y, transform)
        mult = np.dot(X, paren)
        loss = np.subtract(w, mult)
        w = np.subtract(w, np.dot(alpha, loss))
        if np.linalg.norm(loss) <= 0.001:
            break
    return w

```

EXTRA CREDIT:

robin_mehta_extra_credit.py

```python
from __future__ import division
import numpy as np
import sys
from collections import defaultdict
import math

# classvals is a numpy size n array of class values
# class values here are Y values thresholded into quintiles
# Return the entropy of classvals
def entropy(classvals):
    entropy = 0
    uniqueDict = {}
    for x in classvals:
        if x not in uniqueDict:
            uniqueDict[x] = 1
        else:
            uniqueDict[x] += 1

    probabilities = {}
    for key in uniqueDict:
        prob = float(uniqueDict[key]) / float(len(classvals))
        probabilities[key] = prob

    for key in probabilities:
        if probabilities[key] > 0:
            entropy += - (probabilities[key] * math.log10(probabilities[key]) / math.log10(5))
    return entropy

# # colvals is a numpy size n array of column values
# # That is, a feature column with the features thresholded into quintiles
# # colvals indices correspond to those in the size n classvals array
# # Return expected entropy of classvals after splitting on colvals
def expected_entropy(colvals, classvals):
    uniqueDict = dict()
    for x in colvals:
        if x not in uniqueDict:
            uniqueDict[x] = 1
        else:
            uniqueDict[x] += 1

```

```
    probabilities = dict()
    for key in uniqueDict:
        prob = float(uniqueDict[key]) / float(colvals.shape[0])
        probabilities[key] = prob

    indices_dict = dict()
    for key in probabilities:
        #find all indexes where key in colvals occurs
        indices = []
        index = 0
        for num in colvals:
            if num == key:
                indices.append(index)
            index+=1
        indices_dict[key] = indices
    total = 0
    for key in indices_dict:
        total += probabilities[key] * entropy(indices_dict[key])
    return total

if __name__ == '__main__':
    # Extract online_shares dataset
    fp = open(sys.argv[1], 'r')
    lines = fp.readlines()
    features = [st.strip() for st in lines[0].split(',')]
    features.pop() # Get rid of shares label
    data = np.genfromtxt(sys.argv[1], delimiter=',')

    X = data[1:, :data.shape[1]-1]
    Y = data[1:, data.shape[1]-1]

    # Get quintiles
    Xpercs = np.array(np.percentile(X,[20,40,60,80,100], axis=0))
    Ypercs = np.array(np.percentile(Y,[20,40,60,80,100]))

    # Threshold everything into quintiles
    for i in range(X.shape[0]):
        for k in range(Xpercs.shape[1]):
            for j in range(Xpercs.shape[0]):
                if X[i,k] <= Xpercs[j,k]:
                    X[i,k] = Xpercs[j,k]
                    break


    for i in range(len(Y)):
        for j in range(len(Ypercs)):
            if Y[i] <= Ypercs[j]:
                Y[i] = Ypercs[j]
                break

    # Compute all information gains
    y_entropy = entropy(Y)
    infogains = np.zeros((X.shape[1]))
    for i in range(len(infogains)):
        infogains[i] = y_entropy-expected_entropy(X[:,i],Y)

    # Sort infogains descending
    sorted_inds = np.argsort(infogains)[::-1]

    assert(len(infogains)==len(features))
    for i in range(len(sorted_inds)):
        print features[sorted_inds[i]], ',', infogains[sorted_inds[i]]
```