CS109 Final Project

by Albert Wandui and Lawrence Lin Murata
Spring 2015

Written Problem

The Naive Bayesian Classifier works under the assumption that the variables X1, … Xm are conditionally independent, which is proven to be a very inaccurate assumption in this case since the variables are not only dependent, but in fact they are exactly the same as each other.

This bad assumption leads to significant accuracy in applying Naive Bayes to this case. Thus, let's say one of the Xi variables is zero, all the other X the entire result will end up being approximated as if they were conditionally independent variables.

Programming Part

1. **Naive Bayes**

   a. **Simple Train**

      MLE:
      Class 0: tested 2, correctly classified 2
      Class 1: tested 2, correctly classified 2
      Overall: tested 4, correctly classified 4
      Accuracy = 1.00

      MAP (Laplace):
      Class 0: tested 2, correctly classified 2
      Class 1: tested 2, correctly classified 2
      Overall: tested 4, correctly classified 4
      Accuracy = 1.00

   b. **Vote Train**

      MLE:
      Class 0: tested 52 correctly classified 48
      Class 1: tested 83 correctly classified 76
      Overall: tested 135, correctly classified 124
      Accuracy = 0.92

      MAP:
      Class 0: tested 52, correctly classified 48
      Class 1: tested 83, correctly classified 76
      Overall: tested 135, correctly classified 124

Accuracy = 0.92

c. **Heart Train**

MLE:
Class 0: tested 15, correctly classified 10
Class 1: tested 172, correctly classified 135
Overall: tested 187, correctly classified 145
Accuracy = 0.78

MAP:
Class 0: tested 15 correctly classified 10
Class 1: tested 172 correctly classified 130
Overall: tested 187, correctly 140
Accuracy = 0.75

d. There was no difference between MLE and Laplace Estimations in the Vote data set. However, Laplace Estimators outperformed MLE with the Heart dataset. Laplace Estimators work better than MLE when used on data sets in which one of the classes is underrepresented as is the case with the Heart data set.

e. The code is attached in the end of the document, in the "Code for Naive Bayes Classifier"

2. **Logistic Regression**

a. **Simple Train**

Learning Rate = 0.0001
Class 0: tested 2, correctly classified 2
Class 1: tested 2, correctly classified 2
Overall: tested 4, correctly classified 4
Accuracy = 1.00

b. **Vote Train**

Learning Rate = 0.0001
Class 0: tested 52 correctly classified 51
Class 1: tested 83 correctly classified 83
Overall: tested 135, correctly classified 134
Accuracy = 0.99

Here, we were able to achieve a higher classification accuracy than for Naive Bayesian using both MLE and Laplace Estimators. Since Logistic Regression optimizes for P(Y | X), the probability of seeing the y-value given the x data that is given, logistic regression is much more likely to correctly predict y-values if the training dataset and the testing dataset are similar as is the case here. Naive Bayes models P(Y, X) which is the probability of seeing both the y-value and the x data.

c. Heart Train

Learning Rate = 0.0001
Class 0: tested 15, correctly classified 12
Class 1: tested 172, correctly classified 133
Overall: tested 187, correctly classified 145
Accuracy = 0.78

Learning Rate = 0.00002
Class 0: tested 15, correctly classified 10
Class 1: tested 172, correctly classified 136
Overall: tested 187, correctly classified 146
Accuracy = 0.78

Learning Rate = 0.0005
Class 0: tested 15, correctly classified 12
Class 1: tested 172, correctly classified 127
Overall: tested 187, correctly classified 149
Accuracy = 0.74

The learning rate is important to determine the maximum classification accuracy. The algorithm we used in this section for Logistic Regression is standard (or "batch") gradient descent. Thus, we can think of the learning rate as the magnitude of the step that the gradient descent takes when it is trying to converge to the maxima.

Therefore, if the it is too high, the learning rate will make our algorithm overshoot the maxima and then oscillate around the maxima.

Because of that, we will be able to the maxima if we use smaller learning rates. That is why we get the highest accuracy when we use a smaller learning rate.

Thus, we end up getting the best accuracy with really small learning rates, such as 0.0000004, 0.0000005 or even a value as small as 0.000000001.

Experimenting with other learning rates:

Best accuracy:

Learning Rate = 0.0000004
Class 0: tested 15 correctly classified 7
Class 1: tested 172 correctly classified 168
Overall: tested 187 correctly, classified 175
Accuracy = 0.94

Learning Rate = 0.000000001

Class 0: tested 15 correctly classified 7
Class 1: tested 172 correctly classified 168
Overall: tested 187 correctly, classified 175
Accuracy = 0.94

Learning Rate = 0.0000005
Class 0: tested 15 correctly classified 7
Class 1: tested 172 correctly classified 168
Overall: tested 187 correctly, classified 175
Accuracy = 0.94

Learning Rate = 100000000

Other results:

Learning Rate = 0.00009
Class 0: tested 15 correctly classified 12
Class 1: tested 172 correctly classified 134
Accuracy = 0.78

Learning Rate = 0.0002
Class 0: tested 15 correctly classified 12
Class 1: tested 172 correctly classified 127
Accuracy = 0.74

Learning Rate = 0.00015
Class 0: tested 15 correctly classified 12
Class 1: tested 172 correctly classified 131
Accuracy = 0.76

Learning Rate = 0.00017
Class 0: tested 15 correctly classified 12
Class 1: tested 172 correctly classified 129
Accuracy = 0.75

Learning Rate = 0.00013
Class 0: tested 15 correctly classified 12
Class 1: tested 172 correctly classified 131
Accuracy = 0.76

Learning Rate = 0.00012
Class 0: tested 15 correctly classified 12
Class 1: tested 172 correctly classified 131
Accuracy = 0.76

Learning Rate = 1.00
Class 0: tested 15 correctly classified 14

```
Class 1: tested 172 correctly classified 94
Accuracy = 0.58

Learning Rate = 0.50
Class 0: tested 15 correctly classified 11
Class 1: tested 172 correctly classified 123
Accuracy = 0.72

Learning Rate = 0.00011
Class 0: tested 15 correctly classified 12
Class 1: tested 172 correctly classified 131
Accuracy = 0.76

Learning Rate = 0.000105
Class 0: tested 15 correctly classified 2
Class 1: tested 172 correctly classified 172
Accuracy = 0.93

Learning Rate = 0.000107
Class 0: tested 15 correctly classified 2
Class 1: tested 172 correctly classified 172
Accuracy = 0.93

Learning Rate = 0.000103
Class 0: tested 15 correctly classified 2
Class 1: tested 172 correctly classified 172
Accuracy = 0.93

Learning Rate = 0.000109
Class 0: tested 15 correctly classified 2
Class 1: tested 172 correctly classified 172
Accuracy = 0.93

Learning Rate = 0.000101
Class 0: tested 15 correctly classified 2
Class 1: tested 172 correctly classified 172
Accuracy = 0.93
```

d. The required code is attached in the end of the document, in the section “Code for Logistics Regression”

Open-Ended Project: The Search for Humanity's Next Home

by Albert Wandui and Lawrence Lin Murata
Spring 2015

"Mankind was born on Earth. It was never meant to die here."
— Cooper from Interstellar (2014) [3]

1. The "Finding Home" Project

Our project aims at finding exoplanets that are habitable to human beings using machine learning and astrophysics. An exoplanet is is a planet that orbits a star other than the Sun, a stellar remnant, or a brown dwarf. [6] We used data sets from NASA to study how properties that would, in theory, be relevant to the habitability of an exoplanet by humans can help us predict whether certain exoplanets we know of would be habitable or not.

Using our astrophysics knowledge to select the properties, the best data and use the available data in the most relevant way, we train two variations of Naive Bayes (Maximum Likelihood Estimate and Laplace Estimate) and Logistic Regression to classify the data and predict whether each planet is habitable or not (Y = 0 for not habitable and Y = 1 for habitable).

A - Our results

Naive Bayes:

Using Maximum Likelihood Estimator

Class 0: tested 798, correctly classified 775
Class 1: tested 71, correctly classified 67
Accuracy = 0.97

Using Laplace Estimator

Class 0: tested 798, correctly classified 775
Class 1: tested 71, correctly classified 67
Accuracy = 0.97

Logistic Regression:

Class 0: tested 798, correctly classified 775
Class 1: tested 71, correctly classified 67
Accuracy = 0.97

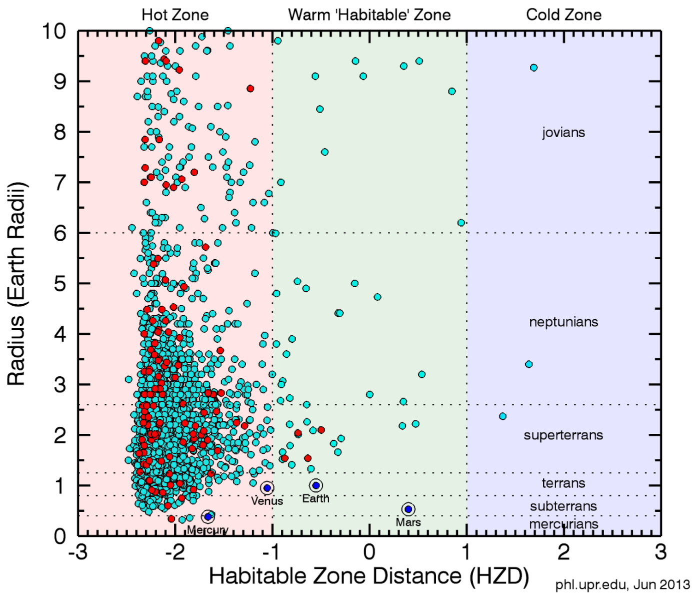
It was interesting to see how accurate Naive Bayes (with both Laplace and MLE) and Logistic Regression were in predicting whether an exoplanet is potentially habitable or not, even though we had to simplify some factors in our data, as described later in part 2 "Data Set". We go into more detail about the significance of the results observed and what we learned from them in part 3 "Significance."

B - Some of the libraries, tools and knowledge required

- numpy
  - python library that helped us model and use the data
- Astrophysics knowledge about exoplanets and about habitability of exoplanets
  - learned through PHYSICS 100: Introduction to Observational and Laboratory Astronomy
  - and by reading many papers on habitability, exoplanets and astrophysics
- Machine Learning
  - Naive Bayes
    - Laplace Estimate
    - Maximum Likelihood Estimate
  - Logistic Regression
  - Classifier
- Modelling the data
  - making sure all data is accurate, binary and relevant to our project

2. Data Set

Our data set comes from the **NASA Exoplanet Archive**, made available thanks to NASA Exoplanet Institute, the Infrared Processing and Analysis Center and Caltech. [4] We chose the NASA data set because it is the most comprehensive and detailed archive for exoplanets, with over 100 properties of 1852 confirmed exoplanets (over 7000 unconfirmed and confirmed exoplanets) available.
The data set about habitable planets is from the **Planetary Habitability Laboratory**.



One of the lists we researched was a list of potentially habitable planets made available by the Planetary Habitability Laboratory [5]. The methods used by the laboratory to find the most potentially habitable exoplanets is rigorous and includes factors like ESI (Earth Similarity Index), HZD (Habitable Zones Distance), HZC (Habitable Zone Composition), HZA (Habitable Zone Atmosphere), SPH

(Standard Primary Habitability), pClass (Planetary Class) and hClass (Habitable Class) [8].

The other list researched was the list of not habitable planets. It required much more work than predicted (predicted, get it?). The reason for that is that people are generally more curious about knowing which planets are habitable for humanity rather than which planets are not habitable. Thus, we had to generate our own data for this category, calculating an estimate of each exoplanet's ESI (Earth Similarity Index) [9] and also its HZD (Habitable Zone Distance), which were used as some of the main factors to determine which planets had the most extreme conditions and were not habitable at all. The HZD of a planet depends mostly on the mass and radius of the host star (the system's star), the star's radiation and between the distance and the planet to the star [7].

A - Modelling the data

A lot of the work required in this project involved understanding the data available, selecting the relevant information using our knowledge of Physics and astronomy, and being able to format the data sets in a way that allows us to properly use our machine learning algorithms on them. We also had to convert features to binary values, partition the data into test and train, and decide how to work with each feature.

1. Binary data:

We had to turn all the information available about the selected properties. First, we encoded the predicted value as Y=0 for exoplanets that are not habitable and Y=1 for habitable exoplanets.

For the properties/input variables we were analyzing, we chose Xi=0 if the measurement of the property was lower than Earth's and Xi=1 if it was higher. We describe which properties we selected as input variables and why in the "B - Properties selected" section under item "3. Significance"

2. Choosing the types of planets:

We had to make sure we were using planets that could meet our criteria, which included being a confirmed planet and having enough available information for us to calculate the ESI (Earth Similarity Index) and to compare the data among planets from the two different data sets (NASA Exoplanet Archive and Planetary Habitability Laboratory).

3. Formatting the data:

We wrote code (available in section "Code for Modelling Data" in the end of the document) to format the data following these steps:

   a. Load the data into python from 2 distinct data sets. One from exodata.csv which contains the habitable planets and one from exodata1.csv which has non-habitable planets.
   b. Convert all the data into binary values by comparing them against the earth values. Now we can cast the new boolean values into 1 for True and 0 for False.

The names allow us to directly access the columns we want in the structured array.

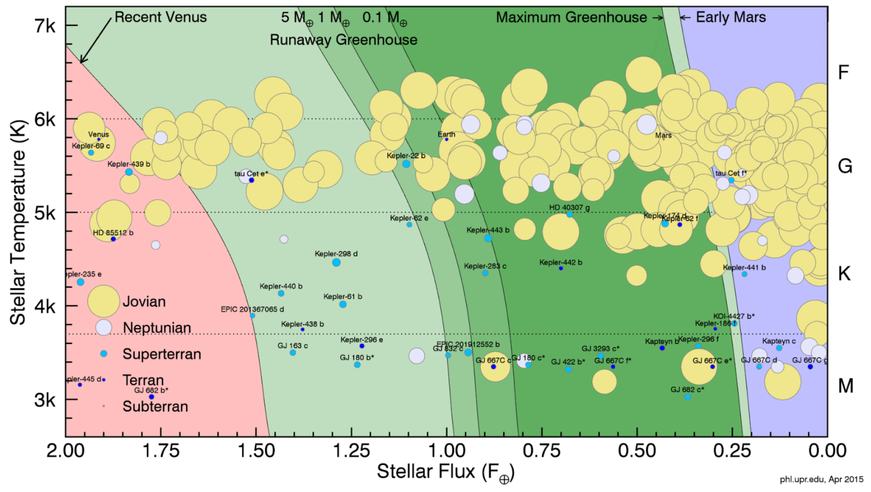
c. Make the training data set by combining half the data from habitable_file and 200 random non-habitable exoplanets from the non-habitable data set. All variables should be binary. The ESI is the measure of similarity of a planet to Earth. For our calculations, all planets in the habitable data set of habitable planets have ESI of 1.
d. Combine all the data into one train data array and create the test data array.
e. Write all the data into txt files.

3. Significance

A - Results Observed

Simply from the statistical distributions of exoplanets, we expect to have relatively few potentially habitable planets. The success of the classification thus shows that there any a number of key parameters that can be successfully used as predictors even in models as simplified as this. A routine such as this can help an astronomer or planetary scientist narrow down potential target planets and focus on selecting exoplanets that meet some specific criteria of interest such as habitability.

B - Properties selected



We selected relevant properties from our data set based on Planet Habitability and aspects we thought would be relevant for habitability. Another factor considered was whether the information about the properties was available in both major data sets (NASA Exoplanet Archive and Planetary Habitability Laboratory) we were working with. The following properties are the ones selected:

1. **Radius of exoplanet:** The radius of the planet is easily measurable for exoplanets discovered through transits. The radius of the planet (as well as the mass) affect the density of the planet which determines whether a planet is rocky like Earth and can possibly support life or a gas giant planet like Jupiter which doesn't support life.
2. **Insolation Flux:** This is the amount of radiation that the planet receives from its host star. Given that the sun is the chief source of energy for life on Earth, the amount of flux that a planet receives influences its capacity for hosting life.
3. **Equilibrium Temperature:** Most life on Earth can only survive and thrive when temperatures lie within a narrow range. If a planet has Earth-like temperatures on average, then it may be a good candidate for life forms similar to those seen on Earth.

Most exoplanets however, are vastly dissimilar from Earth in this respect. Perhaps, then life similar to that found in extreme environments such as geothermal vents may thrive in such vastly different conditions.

4. **Period:** Kepler's Third Law relates the period of orbit of a planet to its average distance from the star. $T^2 = k . a^3$ where T is the period, k is a constant and a is the average distance. Planets that have very short periods are very close to their host star and are constantly baked by radiation which is unconducive for life (think Mercury). Planets too far away, don't receive enough radiation hence are also unsuitable for life. Only Goldilock planets where the distance is just right can support life similar to what is seen here on Earth.

C - Humanity's Future and Survival: searching for our next home

The question we answered by predicting the potential habitability for each exoplanet is one that is relevant to all of us humans: "can this planet potentially become our new home?"

Unfortunately, we have been spending the resources Earth has in a prejudicial way. Since the industrial revolution, the rate at which humanity consumes resources can been increasing astoundingly year over year, making many of us worry if the next generations will have the resources to live properly. [2]

At the same time, NASA and other space traveling research have received considerable budget cuts in the past few years due to a combination of shift in focus for the government, lack of optimism and few results. [1]

In times like these, it is crucial to answer the question of whether we can habitate in exoplanets or not, how far they are and how many potentially habitable exoplanets are out there. By using this data the way did in order to predict the habitability of all the known exoplanets we can help not only astrophysicists and aero engineers answer these questions but we can also find out relevant statistics that quantify questions that have fueled human imagination for thousands of years. The search for the unknown and the question of whether we can live in a planet or not have inspired astronomers in Ancient Greek to spend their lives studying the sky or inspired directors to create mind-blowing movies like Kubrick's 2001: Space Odyssey or Nolan's Interstellar.

We want to make sure humanity will be able to thrive and survive despite all the limitations faced on Earth and, to do so, we need to first know whether we can habitate othe planets and where/which these planets are. Machine learning, used together with astronomy, can help us search for humanity's next home.

"We've always defined ourselves by the ability to overcome the impossible. And we count these moments. These moments when we dare to aim higher, to break barriers, to reach for the stars, to make the unknown known. We count these moments as our proudest achievements. But we lost all that. Or perhaps we've just forgotten that we are still pioneers. And we've barely begun. And that our greatest accomplishments cannot be behind us, because our destiny lies above us."

— Cooper from Interstellar (2014)

Bibliography:

[1] Nasa budgets: US spending on space travel since 1958 UPDATED
http://www.theguardian.com/news/datablog/2010/feb/01/nasa-budgets-us-spending-space-travel
[2] Humankind/nature Interaction: Past, Present and Future
https://books.google.com/books?id=s5loBk4dMzsC&pg=PA40&dq=humans+destroying+nature&hl=en&sa=X&ei=PA11VdWoCcjWoATH-IPACQ&ved=0CCMQ6AEwAQ#v=onepage&q=humans%20destroying%20nature&f=false
[3] Nolan, Christopher. Interstellar (2006)
[4] NASA Exoplanet Archive
http://exoplanetarchive.ipac.caltech.edu/cgi-bin/TblView/nph-tblView?app=ExoTbls&config=planets
[5] Potentially Habitable Exoplanets http://phl.upr.edu/projects/habitable-exoplanets-catalog
[6] Schneider, J. "Interactive Extra-solar Planets Catalog".*The Extrasolar Planets Encyclopedia*.
[7] What's New in the NASA Kepler Data
http://phl.upr.edu/press-releases/whatsnewinthenasakeplerdata
[8] What Makes a World Habitable?
http://www.lpi.usra.edu/education/explore/our_place/hab_ref_table.pdf
[9] Earth Similarity Index (ESI) http://phl.upr.edu/projects/earth-similarity-index-esi

Code: Naive Bayes, Logistic Regression, Data Modelling

1. Code for Naive Bayes Classifier:

```
import sys
import sys

def load_data(file_name):

    data = list(open(file_name, 'r'))

    m = int(data[0]) #number of input variables
    N = int(data[1]) #number of data vectors

    x = []
    y = []
    for vec in data[2:]:
        instr, outstr = vec.split(':')
        invec = [int(_) for _ in instr.split()]
        x.append(invec)
        y.append(int(outstr))

    return m, N, x, y
```

```
#Training through Laplace Estimators or MAP
# lect 24, p. 52

def laplace_estimate(m, N, x, y):

    # prob_y[0] = P(Y = 0)
    # prob_y[1] = P(Y = 1)

    # prob_x_y[0][0] = P(Xi = 0, Y = 0)/P(Y = 0)
    # prob_x_y[1][0] = P(Xi = 1, Y = 0)/P(Y = 0)

    #instance_x_y[n][m] = # of instances where Xi = n and Y = m

    #inst = instances in class 0

    prob_y = []
    prob_x_y = dict()
    #instance_x_y = []
    zero_instances = 0
    total_instances = N

    for cl in y:
        # cl = class (0 or 1)
        if cl == 0:
            zero_instances+=1

    one_instances = total_instances - zero_instances
    prob_y.append(float(zero_instances + 1)/(total_instances + 2))
    prob_y.append(float(one_instances + 1)/(total_instances + 2)) # = 1 - prob_y[0]

    for col in xrange(m):



        instances_zero_zero = 0.0
        instances_zero_one = 0.0
        instances_one_zero = 0.0
        instances_one_one = 0.0

        for row in xrange(N):

            instance = x[row][col]
            classY = y[row]
            #instances_zero_zero = instances where Xi = 0 and class Y = 0

            if(instance == 0 and classY == 0):
                instances_zero_zero += 1
            if(instance == 0 and classY == 1):
                instances_zero_one += 1
            if(instance == 1 and classY == 0):
                instances_one_zero += 1
            if(instance == 1 and classY == 1):
                instances_one_one += 1

        # N = number of total instances
        # P(Xi = 0, Y = 1) = instances_zero_one/N
        # prob_x_y[col][0][1] = P(Xi = 0| Y = 1) = P(Xi = 0, Y = 1)/P(Y = 1)
        # col = number of the column

        prob_x_y[col] = [[(instances_zero_zero + 1)/(zero_instances + 2),\
                           (instances_zero_one + 1)/(one_instances + 2)],\
                          [(instances_one_zero + 1)/(zero_instances + 2), \
                             (instances_one_one + 1)/(one_instances + 2)]]
        #print (instances_one_one + 1)/(one_instances + 2)
    return prob_x_y, prob_y



#Training through Maximum Likelihood Estimators

def mle_estimate(m, N, x, y):

    # prob_y[0] = P(Y = 0)
    # prob_y[1] = P(Y = 1)

    # prob_x_y[col][0][0] = P(Xi = 0, Y = 0)/P(Y = 0)
    # prob_x_y[col][1][0] = P(Xi = 1, Y = 0)/P(Y = 0)

    #instance_x_y[n][m] = # of instances where Xi = n and Y = m

    #inst = instances in class 0

    prob_y = []
    prob_x_y = dict()
    #instance_x_y = []
    zero_instances = 0
    total_instances = N

    for cl in y:
        # cl = class (0 or 1)
        if cl == 0:
            zero_instances+=1

    prob_y.append(float(zero_instances)/(total_instances))
    prob_y.append(float(total_instances - zero_instances)/total_instances) # = 1 - prob_y[0]
    for col in xrange(m):

        instances_zero_zero = 0.0
        instances_zero_one = 0.0
        instances_one_zero = 0.0
        instances_one_one = 0.0

        for row in xrange(N):

            instance = x[row][col]
            classY = y[row]
            #instances_zero_zero = instances where Xi = 0 and class Y = 0

            if(instance == 0 and classY == 0):
                instances_zero_zero += 1
            if(instance == 0 and classY == 1):
                instances_zero_one += 1
            if(instance == 1 and classY == 0):
                instances_one_zero += 1
            if(instance == 1 and classY == 1):
                instances_one_one += 1

        # N = number of total instances
        # P(Xi = 0, Y = 1) = instances_zero_one/N
        # prob_x_y[col][0][1] = P(Xi = 0| Y = 1) = P(Xi = 0, Y = 1)/P(Y = 1)
        # col = number of the column

        prob_x_y[col] = [[(instances_zero_zero/N)/prob_y[0],\
                          (instances_zero_one/N)/prob_y[1]],\
                         [(instances_one_zero/N)/prob_y[0], \
                           (instances_one_one/N)/prob_y[1]]]

    return prob_x_y, prob_y

#Classifying
#lect 14, p. 26

def bayes_predictor(x, m, N, prob_x_y, prob_y):

    pred_y = []

    for x_row in xrange(N):

        # pred_y_one = Y hat for Y = 1

        pred_y_one = 1
        pred_y_zero = 1
        for x_col in xrange(m):
            x_value = x[x_row][x_col]
            pred_y_zero *= prob_x_y[x_col][x_value][0]
            pred_y_one *= prob_x_y[x_col][x_value][1]

        pred_y_zero *= prob_y[0]
        pred_y_one *= prob_y[1]

        #y_hat is the value of y that has higher likelihood
        y_hat = 0 if pred_y_zero > pred_y_one else 1

        pred_y.append(y_hat)

    return pred_y

# pred_y = values of y we predicted
# y = actual values of y

def calculate_accuracy(pred_y, y):
    total_zero = 0
    total_one = 0
    correct_zero = 0
    correct_one = 0

    for i in xrange(len(y)):

        if y[i] == 0:
            total_zero+=1

        if y[i] == 1:
            total_one+=1

        if y[i] == pred_y[i]:
            if y[i] == 0:
                correct_zero += 1
            if y[i] == 1:
                correct_one += 1
    accuracy = float(correct_zero + correct_one)/(total_zero + total_one)

    print "Class 0: tested %d, correctly classified %d" %(total_zero, correct_zero)
    print "Class 1: tested %d, correctly classified %d" %(total_one, correct_one)
    print "Accuracy = %1.2f" %(accuracy)


if __name__ == '__main__' :
    train_file = sys.argv[1]
    test_file = sys.argv[2]
```

```
  #lect 24 p. 11

  # m = number of input variables (num of columns)
  # N = number of data vectors (num of rows)
  # x = input variables/observations (matrix)
      #eg.: [[0,0],[0,1],[1,0]]
  # y = output

  m, N, x, y = load_data(train_file)

  # lect 24 p. 24
  #Training:

  # Y = argmax P(X,Y) = argmax P(X|Y)P(Y)

  #mle_x_probs = P(X|Y)
  #mle_y_probs = P(Y)

  print "\nUsing Maximum Likelihood Estimator\n"

  prob_x_y, prob_y = mle_estimate(m, N, x, y);

#  print prob_x_y
#  print prob_y

  m, N, x, y = load_data(test_file)

  #pred_y is an array of predicted values of Y for each vector
  pred_y = bayes_predictor(x, m, N, prob_x_y, prob_y)

#  print pred_y

  calculate_accuracy(pred_y, y);

  #Now using Laplace Estimators instead of MLE

  print "\nUsing Laplace Estimate\n"

  m, N, x, y = load_data(train_file)

  prob_x_y, prob_y = laplace_estimate(m, N, x, y);

#  print prob_x_y
#  print prob_y

  m, N, x, y = load_data(test_file)
```

```
    #pred_y is an array of predicted values of Y for each vector
    pred_y = bayes_predictor(x, m, N, prob_x_y, prob_y)

  #  print pred_y

    calculate_accuracy(pred_y, y);
```

2. Code for Logistic Regression:

```
#! /usr/bin/env python

import sys
from math import exp

def load_data(file_name):

    data = list(open(file_name, 'r'))

    m = int(data[0]) #number of input variables
    N = int(data[1]) #number of data vectors

    x = []
    y = []
    for vec in data[2:]:
        instr, outstr = vec.split(':')
        invec = [1] + [int(_) for _ in instr.split()]
        x.append(invec)  #insert a 1 at index 0 of the row
        y.append(int(outstr))

    return m, N, x, y


def calculate_z(beta, x, m, i):
    z = sum([beta[j]*x[i - 1][j] for j in xrange(m + 1)])
    return z

#Training for Logistic Regression

def train_logistic_reg(m, N, x, y, learning_rate):
    beta = [0]*(m + 1) #list of size m that is initialized with all zeros
    epochs = 10000 #number of passes over data during learning (constant)
    #learning_rate = learning rate "mi"

    for e in xrange(epochs):

        gradient = [0]*(m + 1)
        z = []

        for i in xrange(1,N + 1): # i goes from 1 to N (both included)
            z.append(calculate_z(beta, x, m, i))

        for k in xrange(m + 1):
            for i in xrange(1,N + 1): # i goes from 1 to N (both included)
                gradient[k] += x[i - 1][k]*(y[i - 1] - 1/(1+ exp(-z[i - 1])))

        for k in xrange(m + 1):
            beta[k] += learning_rate*gradient[k]

    return beta

# lect 24, p. 35

def classifier_logistic_reg(beta, x, m, N):
    pred_y = []
    for x_row in xrange(1, N + 1): #goes from 1 to N (both included)
        z = calculate_z(beta, x, m, x_row)
        # p = P(Y = 1|X)
        p = 1/(1+exp(-z))

        if p > 0.5:
            pred_y.append(1)
        else:
            pred_y.append(0)
    return pred_y

# pred_y = values of y we predicted
# y = actual values of y

def calculate_accuracy(pred_y, y):
    total_zero = 0
    total_one = 0
    correct_zero = 0
    correct_one = 0

    for i in xrange(len(y)):

        if y[i] == 0:
            total_zero+=1

        if y[i] == 1:
            total_one+=1

        if y[i] == pred_y[i]:
            if y[i] == 0:
                correct_zero += 1
            if y[i] == 1:
                correct_one += 1
    accuracy = float(correct_zero + correct_one)/(total_zero + total_one)

    print "Class 0: tested %d, correctly classified %d" %(total_zero, correct_zero)
    print "Class 1: tested %d, correctly classified %d" %(total_one, correct_one)
    print "Accuracy = %1.2f" %(accuracy)


if __name__ == '__main__' :
    train_file = sys.argv[1]
    test_file = sys.argv[2]

    m, N, x, y = load_data(train_file)

    n = 0.0001

    beta = train_logistic_reg(m, N, x, y, n)

    print beta

    m, N, x, y = load_data(test_file)

    pred_y = classifier_logistic_reg(beta, x, m, N)

    print pred_y

    calculate_accuracy(pred_y, y)
```

3. Code for Modelling Data

```
#! /usr/bin/env python

import numpy as np

#This produces a correctly formatted textfile that can be passed to
#the machine learning routines.
def datatofile(xdata, y, name):
    m = len(xdata[0])
    N = len(y)
    f = open(name, 'w')
    f.write('{0}\n'.format(m))
    f.write('{0}\n'.format(N))
    for row in xrange(N):
```

```
    #     # print xdata[row]
        outstr = '%d %d %d %d' %tuple(xdata[row])
        outstr += ': %d\n' %y[row]
        f.write(outstr)
    f.close()
    pass

#1. Load the data into python
# There are two datasets in use here exodata.csv which contains potentially habitable planets
# and exodata1.csv which contains non-classified planets
habitable_file = 'exoplanets/exodata.csv'
nonhabitable_file = 'exoplanets/exodata1.csv'
names = "radius, flux, temp, period"
habitable = np.genfromtxt(habitable_file, delimiter=',',\
    missing_values='',filling_values=np.nan, usecols=(1,2,3,4), names=names)
nonhabitable = np.genfromtxt(nonhabitable_file, delimiter=',',\
    missing_values='',filling_values=np.nan, usecols=(2,3,4,5), names=names)

#Find any missing values in radius, flux, temp and period for later removal
selection1 = np.logical_or(np.isnan(nonhabitable['radius']),np.isnan(nonhabitable['flux']))
selection2 = np.logical_or(np.isnan(nonhabitable['temp']), np.isnan(nonhabitable['period']))
sel = np.logical_not(np.logical_or(selection1, selection2))

#Remove all the gaps in the current data
nonhabitable = nonhabitable[sel]

#2. Convert all the data into binary values by comparing them against the earth values
habitable['radius'] = habitable['radius'] > 1.0
nonhabitable['radius'] = nonhabitable['radius'] > 1.0
habitable['flux'] = habitable['flux'] > 1.0
nonhabitable['flux'] = nonhabitable['flux'] > 1.0
habitable['temp'] = habitable['temp'] > 288
nonhabitable['temp'] = nonhabitable['temp'] > 288
habitable['period'] = habitable['period'] > 1.0
nonhabitable['period'] = nonhabitable['period'] > 1.0

#Now we can cast the new boolean values into 1 for True and 0 for False. The names allow us
to
#directly access the columns we want in the structured array.
habitable = habitable.astype([('radius', int), ('flux', int), ('temp', int), ('period', int)])
nonhabitable = nonhabitable.astype([('radius', int), ('flux', int), ('temp', int), ('period', int)])

#3. Make the training data set by combining half the data from habitable_file and 200 random
non_habitable
# exoplanets from the nonhabitable dataset.
# All variables are binary. The esi is the measure of similarity of a planet to earth. All planets
in
# the habitable data set have an esi of 1.
```

```
num_habitable = len(habitable['radius'])/2
num_nonhabitable = 200

# Combine all the data into one train data array
train_y = np.array([1]*num_habitable + [0]* 200) #esi values
train_xdata = np.hstack((habitable[:num_habitable], nonhabitable[:200]))

#4. Lets create the test data array
test_y = np.array([1]*(len(habitable) - num_habitable) + [0]*(len(nonhabitable)-200))
test_xdata = np.hstack((habitable[num_habitable:], nonhabitable[200:]))

#5. Now we need to write all the data into txt files
datatofile(train_xdata, train_y, 'exoplanet-train.txt')
datatofile(test_xdata, test_y, 'exoplanet-test.txt')
```