University of Houston - Clear Lake [Spring 2023]

CSCI 4336 - Introduction to Machine Learning

Logistic Regression using Java and Kaggle Titanic Dataset

Student Name: Brandon E Ramirez

Date: 4/17/2023

Assignment: Term Project

Due: 4/24/2023 @ 11:59PM

Student ID: 1952649

Contents

List of Figures

1 Introduction

The aim of this project is to implemnt the logistic regression machine learning algorithm using the Kaggle ”Titanic - Machine Learning from Disaster” dataset. The model should be able to identify who survived the disaster by categorizing their likelyhood of survival as a 0 or 1. This will be based on the cummulative probability determined by the provided features. Surpassing a threshold value of 0.5 or 50% will output a logical 1 (survived), 0 (died) otherwise.

2 Project Scope/Requirements

CSCI 4336– Introduction to Machine Learning
Term Project.
Due: Monday April 24, 2023 by 11:59pm

Requirements:

1. Pick your favorite data set (Kaggle, UCI, etc.) and figure out how you want to analyze it. You may choose either linear regression, gradient descent, or a logistic regression gradient descent algorithm to be used with the data set. Once you decide what you want to do, you need to email the instructor and the TA by April 5th your plan of action on what you are going to implement and what type of information are you going to extract and conclude out of the data.

   This step is your first deliverable in writing via email to the instructor and the TA.

2. You must implement the code for this project using a high-level programming language: Java, C, C++. That means the machine learning algorithms must be implemented from scratch using your choice of language. Keep in mind that your results must be clear and concise to be able to graph them using your favorite graphing tool(using matplotlib). Plotting the data created from the logistic regression algorithm is not required. Your code must also be interactive as much as possible to be able to allow the user to interact with it (it should be able to be run on any machine with ease).
3. You must be creative. Creativity and usefulness of your project will be part of the grade. This means also that you should find data that can and needs to be cleaned.
4. Visualization and analysis of the data is important, you may use Jupyter Notebooks to do so to help you visualize the data. You also need to compare your results to those of Scikit-Learn(run logistic regression using sklearn tools from book).
5. You need to provide the code, and a completely and cleanly written report to the instructor by the due date as well as schedule an appointment to demo your code and your project to both the instructor and the TA. This is also part of the grade(optional, may not be necessary if the documentation is detailed enough).
6. When in doubt ask the instructor about any details or if you have any questions.

Best of Luck

3 Email recap

```
---------------------------------------|proposal.txt|---------------------------------------

Send to:
Instructor: Dr. Ahmed Abukmail; Email: Abukmail@uhcl.edu
(TA): Vinay Kopuri; Email: KopuriV6277@uhcl.edu

1. Pick your favorite data set (Kaggle, UCI, etc.) and figure out how you want to analyze it. You
may choose either linear regression, gradient descent, or a logistic regression gradient descent

algorithm to be used with the data set. Once you decide what you want to do, you need to email the
instructor and the TA by April 5th your plan of action on what you are going to implement and what
type of information are you going to extract and conclude out of the data.

This step is your first deliverable in writing via email to the instructor and the TA.

---------------------------------------------------------------------------------------------

-The data: Kaggle Titanic Survivor Dataset ("https://www.kaggle.com/competitions/titanic/data")
-Intended Algorithm: Logistic Regression / Sigmoid Function
-Technologies: Java(implementing model algorithm), Mathplotlib/awt+swing(visualization),
Jupyter/Excel(analyzing & cleaning data).

The available data is composed of 2 data sets, a training and test set which are both csv files.
And a file indicating their passenger ID and whether they survived or not(1 or 0). All the available
features are:

Variable:     Definition:                                 Key:
survival      Survival                                    0 = No, 1 = Yes
pclass        Ticket class                                1 = 1st, 2 = 2nd, 3 = 3rd
sex           Sex
Age           Age in years
sibsp         # of siblings / spouses aboard the Titanic
parch         # of parents / children aboard the Titanic
ticket        Ticket number
fare          Passenger fare
cabin         Cabin number
embarked      Port of Embarkation C = Cherbourg, Q = Queenstown, S = Southampton

Note*: "embarked" will need a numeric value for it to be useful. I will figure out how to determine
these values later.

I will try to find the probability that a passenger from the Titanic disaster survived given relevant
characteristics. I intend (most likely) to use: age, pclass, sex, embarked, parch, & sibsp. I want to
use these features but they may vary as I may include more or less features as my model evolves in
requirements and scope. The model should provide a probability of someone surviving (1 or 0 and ##%)
given these parameters.

Sources: https://onix-systems.com/blog/top-10-java-machine-learning-tools-and-libraries
```

4 Analyzing the data

First of all, the training data was imported using Anaconda which was used to import dependencies and initialize Jupyter Notebook by using the cmd.exe prompt ("Jupyter Notebook"). The data was analyzed using a dataframe and was visualized using mathplotlib at this stage. I will use other tools to graph the Sigmoid function and the data's outputs later on. Here is my analysis of the data before cleaning it:

Note* Further analytics about the data can be found at the source. Other files/data we will use are "gender_submission.csv" which has a list of passenger ids and their actual survival outcome; and test.csv which we will use to verify the model later.

In [4]:

```
import sys
assert sys.version_info >= (3, 7)
from packaging import version
import sklearn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [5]:

```
def load_training_data():
    # reading csv files
    return pd.read_csv('train.csv', sep=",") #the mighty dataframe

passengers = load_training_data()
passengers.head()
```

Out[5]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

In [6]:

```
passengers.info() #get the info of the resulting data frame
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

In [8]:

```
passengers["Survived"].value_counts()
```

Out[8]:

```
0    549
1    342
Name: Survived, dtype: int64
```

In [9]:

```
passengers["Pclass"].value_counts()
```

Out[9]:

```
3    491
1    216
2    184
Name: Pclass, dtype: int64
```

In [10]:

```
passengers["Sex"].value_counts()
```

Out[10]:

```
male      577
female    314
Name: Sex, dtype: int64
```

In [11]:

```
passengers["Embarked"].value_counts()
```

Out[11]:

```
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

In [12]:

```
passengers["Fare"].value_counts()
```

Out[12]:

```
8.0500     43
13.0000    42
7.8958     38
7.7500     34
26.0000    31
           ..
35.0000     1
28.5000     1
6.2375      1
14.0000     1
10.5167     1
Name: Fare, Length: 248, dtype: int64
```

In [13]:

```
passengers["SibSp"].value_counts()
```

Out[13]:

```
0    608
1    209
2     28
4     18
3     16
8      7
5      5
Name: SibSp, dtype: int64
```

In [14]:

```
passengers["Parch"].value_counts()
```

Out[14]:

```
0    678
1    118
2     80
5      5
3      5
4      4
6      1
Name: Parch, dtype: int64
```

In [15]:

```
passengers["Age"].value_counts()
```

Out[15]:

```
24.00    30
22.00    27
18.00    26
19.00    25
28.00    25
         ..
36.50     1
55.50     1
0.92      1
23.50     1
74.00     1
Name: Age, Length: 88, dtype: int64
```

In [16]:

```
passengers["PassengerId"].value_counts()
```

Out[16]:

```
1      1
599    1
588    1
589    1
590    1
      ..
301    1
302    1
303    1
304    1
891    1
Name: PassengerId, Length: 891, dtype: int64
```

In [18]:

```
passengers["Cabin"].value_counts()
```

Out[18]:

```
B96 B98        4
G6             4
C23 C25 C27    4
C22 C26        3
F33            3
              ..
E34            1
C7             1
C54            1
E36            1
C148           1
Name: Cabin, Length: 147, dtype: int64
```

In [20]:

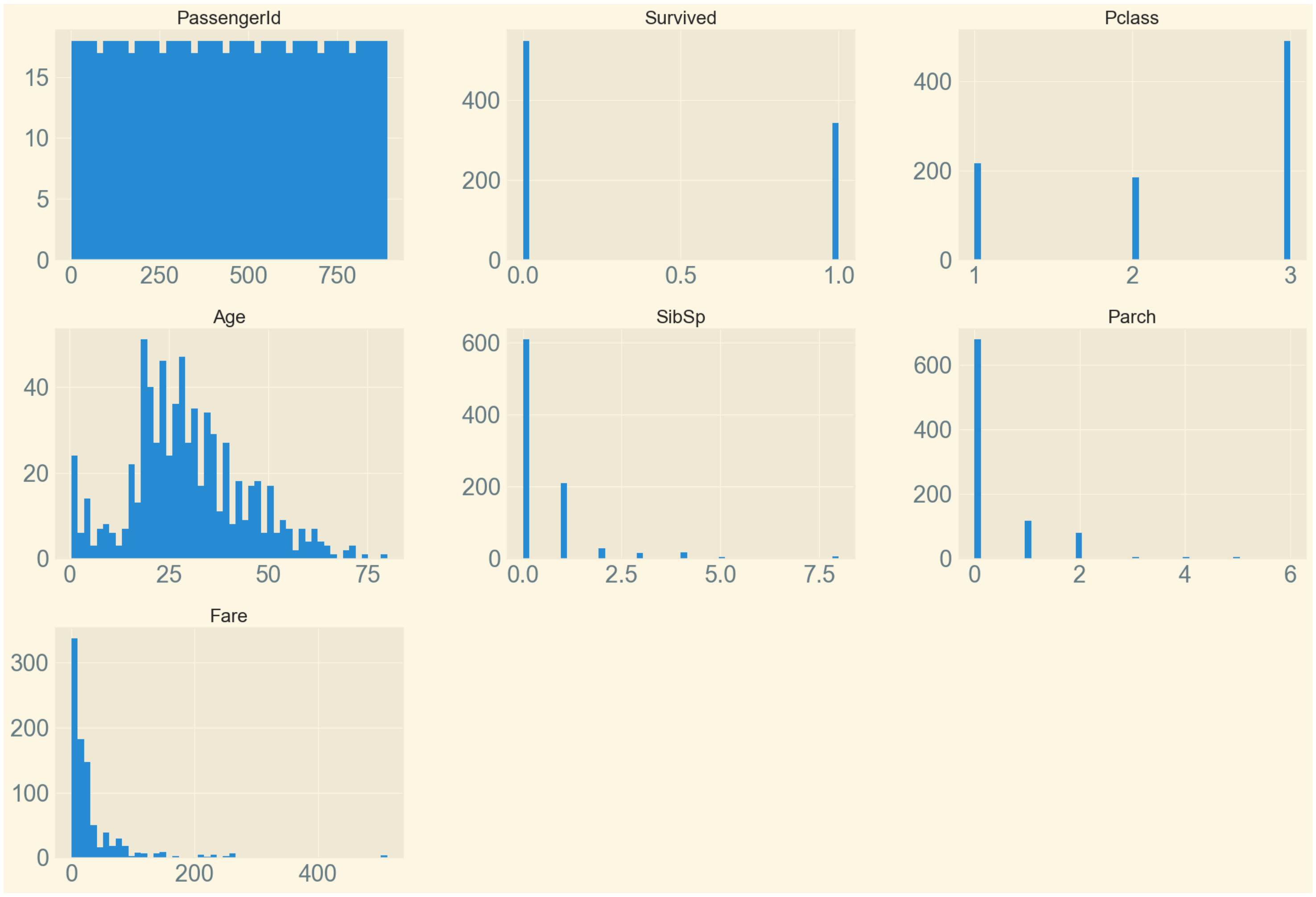
```
passengers.describe()
```

Out[20]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

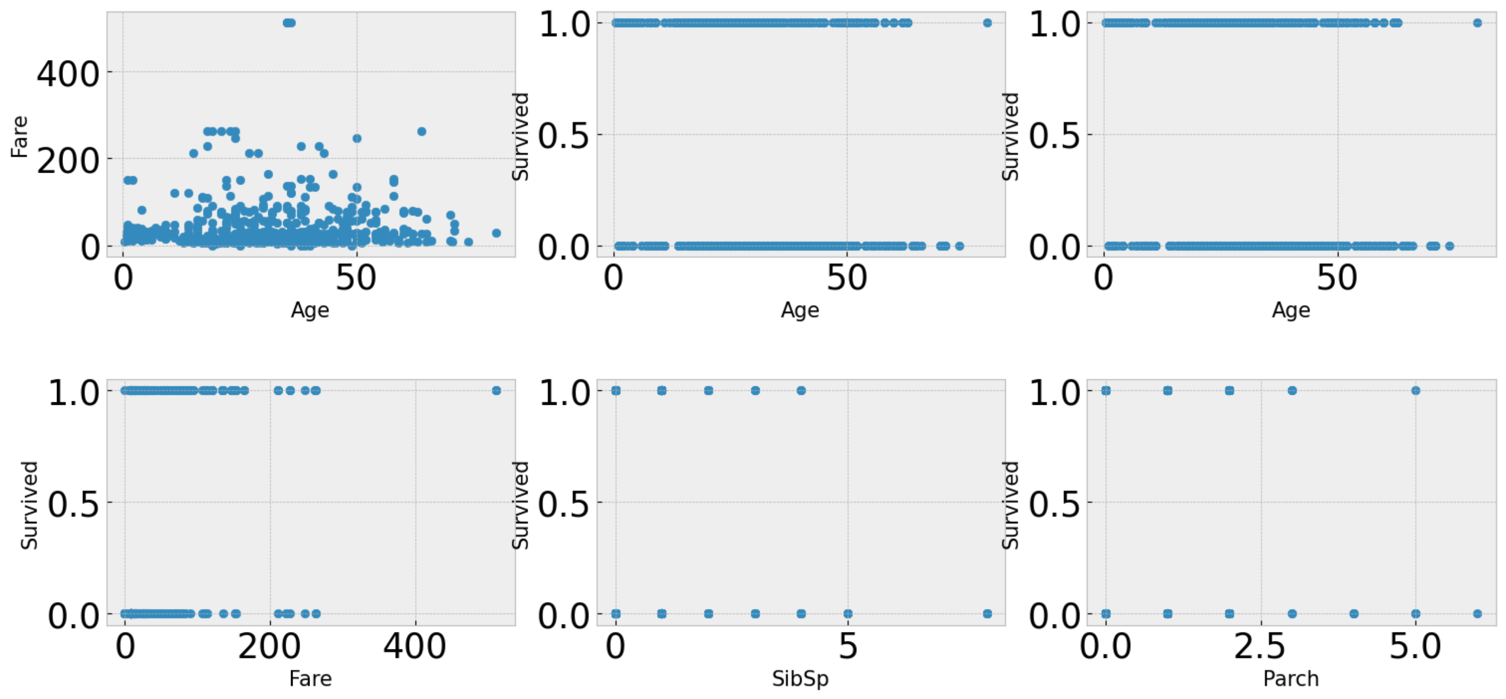
In [17]:

```
import matplotlib.pyplot as plt

#naturally, only the numerical data can be plotted
# extra code – the next 5 lines define the default font sizes
plt.style.use('Solarize_Light2')
plt.rc('font', size=20)
plt.rc('axes', labelsize=30, titlesize=20)
plt.rc('legend', fontsize=30)
plt.rc('xtick', labelsize=25)
plt.rc('ytick', labelsize=25)
# plt.plot(a, b, color="red")
passengers.hist(bins=50, figsize=(24, 16)) # one type of visualization chart
plt.show()
```



In [9]:

```python
import sys
assert sys.version_info >= (3, 7)
from packaging import version
import sklearn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#naturally, only the numerical data can be plotted
# extra code – the next 5 lines define the default font sizes
plt.style.use('bmh')
plt.rc('font', size=20)
plt.rc('axes', labelsize=15, titlesize=20)
plt.rc('legend', fontsize=30)
plt.rc('xtick', labelsize=25)
plt.rc('ytick', labelsize=25)

def load_training_data():
    # reading csv files
    return pd.read_csv('train.csv', sep=",") #the mighty dataframe

passengers = load_training_data()

fig, axs = plt.subplots(2, 3, figsize=(18, 8))

axs[0, 0].scatter(passengers['Age'], passengers['Fare'])
axs[0, 1].scatter(passengers['Age'], passengers['Survived'])
axs[0, 2].scatter(passengers['Age'], passengers['Survived'])
axs[1, 0].scatter(passengers['Fare'], passengers['Survived'])
axs[1, 1].scatter(passengers['SibSp'], passengers['Survived'])
axs[1, 2].scatter(passengers['Parch'], passengers['Survived'])

axs[0, 0].set_xlabel('Age')
axs[0, 0].set_ylabel('Fare')
axs[0, 1].set_xlabel('Age')
axs[0, 1].set_ylabel('Survived')
axs[0, 2].set_xlabel('Age')
axs[0, 2].set_ylabel('Survived')
axs[1, 0].set_xlabel('Fare')
axs[1, 0].set_ylabel('Survived')
axs[1, 1].set_xlabel('SibSp')
axs[1, 1].set_ylabel('Survived')
axs[1, 2].set_xlabel('Parch')
axs[1, 2].set_ylabel('Survived')

fig.subplots_adjust(hspace=0.5)
plt.show()
```



5 Cleaning the Data

A machine learning model needs a good dataset in order to provide useful information. There are many ways to "prepare" or "clean" data, there may be fields that are incorrect, incomplete, irrelevant, duplicated, or improperly formatted.

5.1 Missing Values

While analyzing the excel data I noticed that there are missing values. A good approach to fix this is combing through each column and find which ones contain any missing values: PassengerId{null},Survived{null}, Pclass{null}, Name{null}, Sex{null}, SibSp{null}, Parch{null}, Ticket{null}, Fare{null}(some of the fares were 0 or free).

Age{177 rows}, Cabin{206 rows}, Embarked{2 rows}.

Some features don't matter or have data that a little to complex to interpret like cabin and ticket type, but the others must be free of error and have values to work with later on. The age column has the most missing values so I will concentrate on it. I can fill in these values by deleting the columns, inputting the mean as a placeholder value, or make an educated guess as to what the passengers likely paid. I will do a combination of these three. For a third of the passengers I will assign them an age based on the price of their ticket in the following way:

Avg Ticket cost	Range	Projected Age
$\leq$ \$35(£7)	0-20	10
\$35 $\leq$ \$65(£12)	20-45	30
\$65 $\leq$ \$400(£30)	45+	55

I will assign 50 missing ages using this method, assign 50 the mean age ($29.699118 \approx 30$), and remove the remaining rows with missing values. These will be colored differently in the excel training data file. We now have 814 rows to train the model with. Source: https://rmstitanic1912.weebly.com/the-cost-of-tickets.html

5.2 Interpreting the Features

The "embarked" feature has 3 letters denoting the ports of embarkation, C = Cherbourg, Q = Queenstown, S = Southampton. The embarked column only has 2 missing values so I will just make an educated guess with those based on ticket price and port. Finally, I will be marking them as 0,1, or 2 based on the relative GDP of each town in 1912 (C=0,S=1,Q=2) where "3" was the wealthiest port at the time. This feature may not be useful enough as the port cities might not hold sufficient applicable information to get any practical results. The sex of each passenger can be interpreted as 0 for male and 1 for female.

5.3 Useful features

We need to make an assessment of which features are quantifiable and relevant to our model. We have a decent collection of features to work with but we need to choose the features that will grant us useful results. The columns that are seemingly unnecessary are cabin, name, and ticket number; these features should have no effect on our results. There is also the matter of extrapolating useful information from the remaining columns. PassengerId will be used to identify a passenger (1 to 814), and "survived" are essential to managing and testing the model. We need to scrutinize the features and possibly omit/include them to revise the model as we see fit. The features we will definitely use are: sex, age, fare, Pclass, SibSp, Parch, & embarked.

6 How Binomial Logistic (logit) Regression Works

Logistic regression uses the sigmoid function which is a mathematical function used to assign predicted values to probabilities by mapping any real value into another value within a range of 0 to 1. It is defined by a characteristic S-shaped curve or sigmoid curve which is produced as y values asymptotically approach 0 and 1 as the x values approach $-\infty$ and ∞ respectively. The input weights/parameters must be trained to this curve best-fits our dataset. Once the curve reasonably approximates our dataset we can start making useful predictions. Some assumptions about the data are:

1. The dependent variable must be categorical in nature
2. Only relevant variables should be included
3. Large sample size is required
4. The independent variable should not have multi-collinearity
5. Given output $\hat{y}$, if $\hat{y} \geq 0.5$ then output $= O(1)$, else $\hat{y} < 0.5 = O(0)$

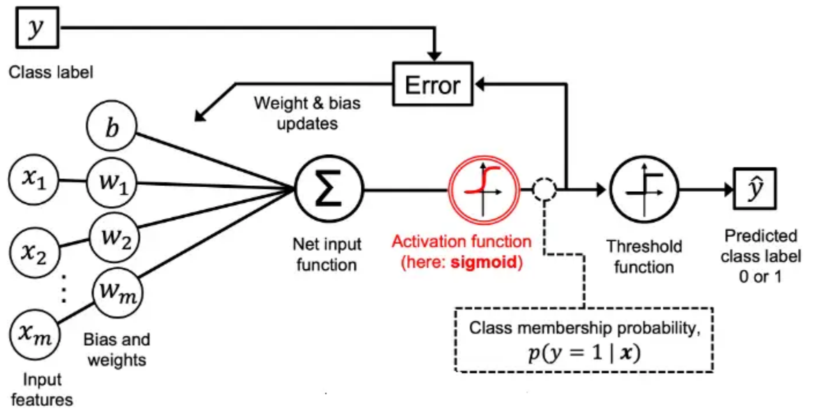


Figure 1: High Level Logistic Regression Schematic

The above process can be described mathematically as:
"$y = w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + \ldots + w_{n-1} \cdot x_{n-1} + w_n \cdot x_n + b$" or $y = w^T x + b$". If $\hat{y} \geq 0.5 = O(1)$, else $\hat{y} < 0.5 = O(0)$

This is another way of writing it:

$$\begin{bmatrix} x_1, & x_2, & x_3, & \ldots, & x_{n-1}, & x_n \end{bmatrix}_{1\times n} \cdot \begin{bmatrix} w_1, \\ w_2, \\ w_3, \\ \ldots, \\ w_{n-1}, \\ w_n \end{bmatrix}_{n\times 1} + b \rightarrow N \rightarrow A \rightarrow U \rightarrow O(0,1)$$

Where b = bias term, N = Net Input Function, A = Activation Function, U = Unit Step Function, O = Output.

The x row vector are the input features, the w column vector are the weights which are initialized to 0 when we begin training the model.

The logistic regression algorithm needs several components, these include:

1. Sigmoid function (activation function)

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$

where $\sigma(x)$ = S(x) = sigmoid function, e = Euler's Number (≈ 2.71828)
Domain: $(-\infty, \infty)$, Range: (0, +1), & S(0) = 0.5

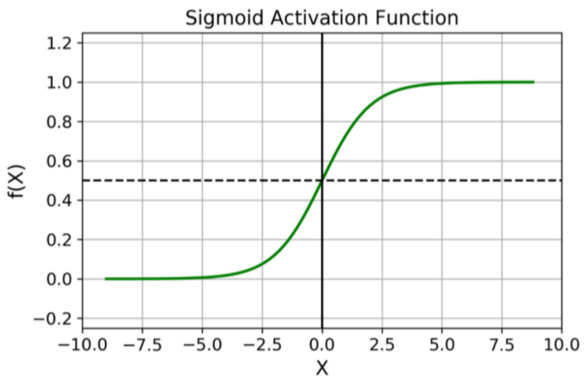


Figure 2: Sigmoid function

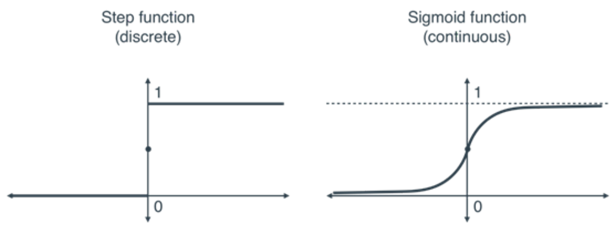


Figure 3: The sigmoid function can give us a continous value (probability) as well as a discrete value.

2. Cost function
   The cost function is a representation of the error in our predictions, we need to decrease its value in order to get accurate results.

$$\hat{y} = \sigma(w^T x + b) = \frac{1}{1 + e^{-w^T x + b}}$$

$$cost = \frac{-1}{m} \cdot \sum_{i=o}^{m} [y \cdot log(\hat{y}) + (1 - y) log(1 - \hat{y})]$$

   Where m = number of values in the data-set, $\hat{y}$ is our predicted value, and y is the actual value. We use this function because it only allows for 0 or 1 (or any range in between) to be accepted values. Lets see what happens when y equals 0 and 1 for a single observation:

$$\text{if } y = \begin{cases} 1, & \text{error} = -log(\hat{y}); \text{because: } -[y \cdot log(\hat{y}) + \cancelto{0}{(1-y)log(1} - \hat{y})] \\ 0, & \text{else, error} = -log(1-\hat{y}); \text{because: } -[\cancelto{0}{(y \cdot log(\hat{y}))} + (1-y) \cdot log(1-\hat{y})] \end{cases} \tag{1}$$

   If the true value $y = 1$, and our predicted value $\hat{y}$ is close or equal to 1, then the error rate will be lower because $-log(1) = 0$. Otherwise, if the true value $y = 0$, and our predicted value $\hat{y}$ is close or equal to 0, then the error rate will be lower because $-log(1) = 0$.

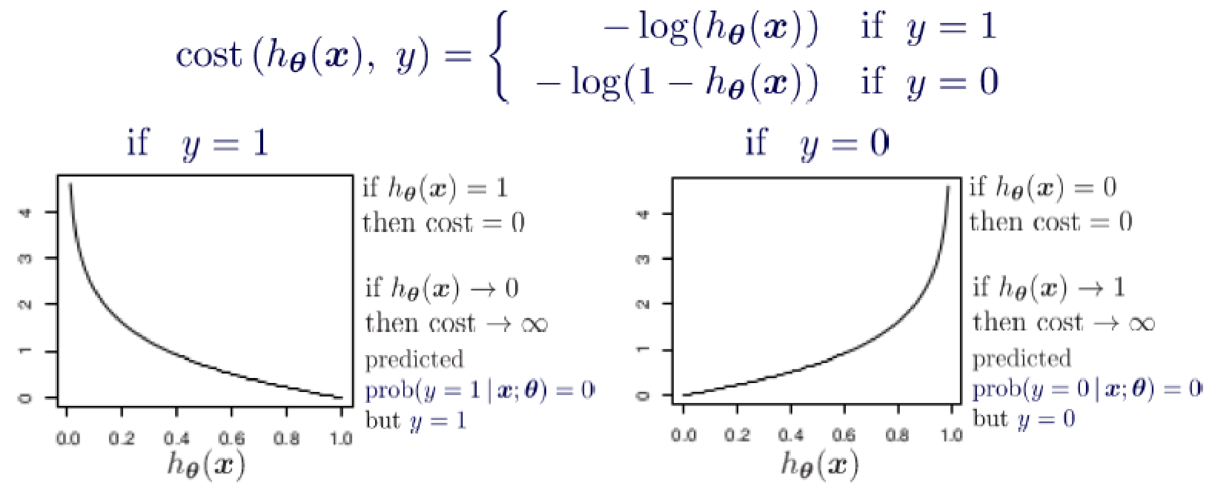


Figure 4: Cost function behavior when y = 1, & y = 0

3. Gradient descent
   Gradient descent is an iterative optimization algorithm which will help us reduce the cost function value by modifying the weight(s) in our model so that we can approach a global minima. This part of the algorithm will be repeated using a for loop. This algorithms components are:

$$\hat{W} = W - \alpha \cdot \frac{\partial_{cost}}{\partial_W}$$

$$\hat{b} = b - \alpha \cdot \frac{\partial_{cost}}{\partial_b}$$

The first order derivatives can also be written as:

$$\frac{\partial_{cost}}{\partial_W} = (\hat{y} - y) \cdot x$$

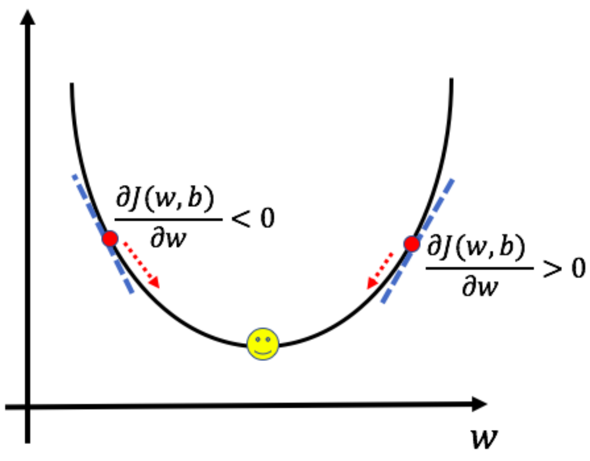
$$\frac{\partial_{cost}}{\partial_b} = (\hat{y} - y)$$

$$\therefore \hat{W} = W - \alpha \cdot \boldsymbol{(\hat{y} - y) \cdot x}$$

$$\hat{b} = b - \alpha \cdot \boldsymbol{(\hat{y} - y)}$$

s.t. α = learning rate, $\hat{W}$ = the new weight, $\hat{b}$ = is the updated y intercept. The bold $\hat{y}, y, x$ represent the matrix form of all utilized observations in data-set.

The type of gradient descent I will use is batch gradient descent, which calculates the error for each example within the training data-set. The model is not revised until every training sample has been assessed. Every cycle is referred to as a cycle or a training epoch. The learning rate hyper-parameter must be chosen to avoid surpassing the global minima along the x-axis but not be too small so that it doesn't take too many iterations to find it.



$\frac{\partial J(w,b)}{\partial w} < 0$
$\frac{\partial J(w,b)}{\partial w} > 0$
w

Figure 5: The rate of error is minimized as the slope approaches 0

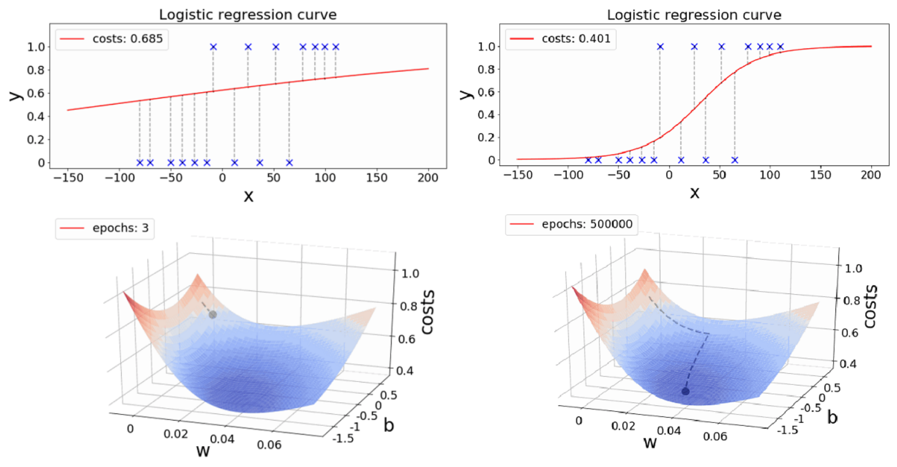


Figure 6: Optimization of logistic curve

7 Source Code & Technical Challenges

7.1 My code

The code for this project is in the folder X, just open the file from any IDE including the data-sets files to run the program.

7.2 Explanation

The program should be able to generate a list of outputs including the cost function value after a certain number of epochs as well as the rate of success once the model has been fully trained.

7.3 Processing the Data

The data needs to be stored in a way such that all the relevant data can be called when needed and to run analysis on it (margin of error, etc.). The data must be able to read the train-X(gives us id and other features) and train-Y data (tells us who survived as well as id). The way in which the algorithm is implemented should be easy to replicate using Python and Jupyter Notebooks like in the SciKit Learn book. I deleted the header rows for each file to get them out of the way of calculations. I then took the transpose of the input matrix so that each feature could be read as an individual line into Java. I had difficulty importing the csv data so I converted them to txt files and used the following code to read from the files at specific lines so I could give them the appropriate feature names and save them in their easy to access array format.

```java
import java.nio.file.Files;
import java.nio.file.Paths;
//          ...
String test_X_embarked = Files.readAllLines(Paths.get(test_X)).get(7);
```

Listing 1: Importing the rows from external files

I then had to parse these string arrays to either integer or float arrays. After I had done this for all the files I could finally start training the model

7.4 Tech Stack Used

The project needed the use Java, my IDE of choice was IntelliJ IDEA. I planned using the Java 2D Graphics API provided by Oracle. I realized awt + swing is used for creating GUI components rather than for plotting and visualizing data.

7.5 Visualizing the data (optional)

The probability(x) should be mapped to a function σ(x) which will yield a y-axis value, this can be used to visualize the coordinate points. Ideally, by pressing the coordinate point it should display the calculated probability and whether or not they survived as well as age, sex, name, & whether or not they actually survived. I decided to do this later on and not visualize the data this time. I plan to do it later.

7.6 Results

Our rate of success was should be in the program output. I'm sure we could improve our results by including observations from the other data-sets, using other techniques (stochastic gradient descent, etc.) or experimenting with the "iterations" and "learning rate" parameters further. Feel free to experiment with the parameters and try to find the sweet spot yourself.

8 Running My Logistic Regression Algorithm (Instructions)

Just make sure to read the "README".txt file that came int the Java source folder. It should be simple since running the program doesn't require setting up any external dependencies. There shouldn't be any setting up on your part to make it work, everything is set to just run.

9 SciKitLearn Textbook Implementation

The book "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow" talks about logistic regression on page 268 using the famous Iris data-set. It primarily uses the sklearn "LogisticRegression" and "train-test-split" python classes from the model selection and linear modules to train and process the model.

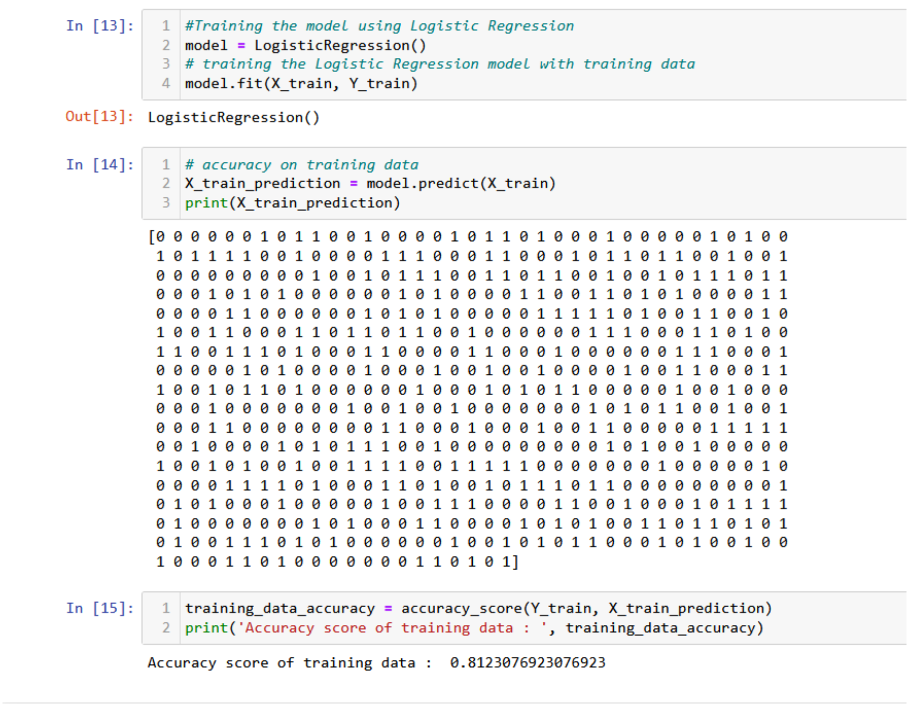
9.1 SciKit Source Code

```python
#!/usr/bin/env python
# coding: utf-8
# In[1]:
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
titanic_data = pd.read_csv('train.csv')
# In[2]:
# check the number of missing values in each column
titanic_data.isnull().sum()
# In[3]:
# drop the "Cabin" column from the dataframe
titanic_data = titanic_data.drop(columns='Cabin', axis=1)
# In[4]:
# replacing the missing values in "Age" column with mean value
titanic_data['Age'].fillna(titanic_data['Age'].mean(), inplace=True)
# In[5]:
# replacing the missing values in "Embarked" column with mode value
titanic_data['Embarked'].fillna(titanic_data['Embarked'].mode()[0], inplace=True)
# In[6]:
```

```python
# converting categorical Columns
titanic_data.replace({'Sex':{'male':0,'female':1}, 'Embarked':{'S':0,'C':1,'Q':2}},
    inplace=True)
# In[7]:
X = titanic_data.drop(columns = ['PassengerId','Name','Ticket','Survived'],axis=1)
Y = titanic_data['Survived']
print(X)
# In[8]:
print(Y)
# In[9]:
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.2, random_state=2)
# In[10]:
print(X.shape, X_train.shape, X_test.shape)
# In[11]:
#Training the model using Logistic Regression
model = LogisticRegression()
# training the Logistic Regression model with training data
model.fit(X_train, Y_train)
# In[12]:
# accuracy on training data
X_train_prediction = model.predict(X_train)
print(X_train_prediction)
# In[13]:
training_data_accuracy = accuracy_score(Y_train, X_train_prediction)
print('Accuracy score of training data : ', training_data_accuracy)
```

Listing 2: Code used in book using Titanic dataset

9.2 Results & Comparison



```
In [13]: #Training the model using Logistic Regression
         model = LogisticRegression()
         # training the Logistic Regression model with training data
         model.fit(X_train, Y_train)

Out[13]: LogisticRegression()

In [14]: # accuracy on training data
         X_train_prediction = model.predict(X_train)
         print(X_train_prediction)

         [0 0 0 0 0 0 1 0 1 1 0 0 1 0 0 0 0 1 0 1 1 0 1 0 0 0 1 0 0 0 0 0 1 0 1 0 0
          1 0 1 1 1 1 0 0 1 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0 1 0 1 1 0 1 1 0 0 1 0 0 1
          0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 1 0 0 1 1 0 1 1 0 0 1 0 0 1 0 1 1 1 0 1 1
          0 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 0 0 1 1 0 1 0 1 0 0 0 0 1 1
          0 0 0 0 1 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 1 1 1 1 1 0 1 0 0 1 1 0 0 1 0
          1 0 0 1 1 0 0 0 1 1 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 1 1 1 0 0 0 1 1 0 1 0 0
          1 1 0 0 1 1 1 0 1 0 0 0 1 1 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 1 1 1 0 0 0 1
          0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 1 0 0 1 0 0 0 0 1 0 0 1 1 0 0 0 1 1
          1 0 0 1 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 0 0 0 0 1 0 0 1 0 0 0
          0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 1 0 0 1
          0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 1 1 1 1 1
          0 0 1 0 0 0 0 1 0 1 0 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0
          1 0 0 1 0 1 0 0 1 0 0 1 1 1 1 0 0 1 1 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0
          0 0 0 0 1 1 1 1 0 1 0 0 0 1 1 0 1 0 0 1 0 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 1
          0 1 0 1 0 0 0 1 0 0 0 0 0 1 0 0 1 1 1 0 0 0 0 1 1 0 0 1 0 0 0 1 0 1 1 1 1
          0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 1 1 0 0 0 0 1 0 1 0 1 0 0 1 1 0 1 1 0 1 0 1
          0 1 0 0 1 1 1 0 1 0 1 0 0 0 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 1 0 1 0 0 1 0 0
          1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 1 1 0 1 0 1]

In [15]: training_data_accuracy = accuracy_score(Y_train, X_train_prediction)
         print('Accuracy score of training data : ', training_data_accuracy)

         Accuracy score of training data :  0.8123076923076923
```

10 Conclusion

Logistic regression is a statistical technique used to model the probability of a binary response variable based on one or more predictor variables. It is a type of regression analysis that uses a logistic function to transform the output of a linear regression model into a probability value between 0 and 1. The model estimates the coefficients of the predictor variables, which are used to calculate the odds of the response variable being in one category over the other. Logistic regression is widely used in various fields, including medicine, economics, and social sciences, to predict the probability of outcomes such as disease diagnosis or customer behavior.

11 Appendices

N/A

12 Sources

Data

1. https://www.kaggle.com/competitions/titanic/data

Math and Algorithm Reference

1. https://www.analyticsvidhya.com/blog/2022/07/gradient-descent-and-its-types/
2. https://www.javatpoint.com/logistic-regression-in-machine-learning

General LaTeX reference:

1. https://www.math.uci.edu/~xiangwen/pdf/LaTeX-Math-Symbols.pdf
2. https://www.overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes
3. https://www.youtube.com/watch?v=ydOTMQC7np0

Excel Reference:

1. https://www.youtube.com/watch?v=YiC-z_FH7SU
2. https://www.youtube.com/watch?v=fSLBEPJglaU

Code formatting and colors reference: https://www.overleaf.com/learn/latex/Code_listing
Colors reference: https://www.overleaf.com/learn/latex/Using_colours_in_LaTeX
Java 2D Graphics API Documentation: https://docs.oracle.com/javase/tutorial/2d/index.html
Other Documentation:

1. https://nbconvert.readthedocs.io/en/latest/install.html
2. https://www.dunderdata.com/blog/view-all-available-matplotlib-styles
3. https://texdoc.org/serve/pdfpages/0
4. https://www.overleaf.com/learn/latex/Matrices

SciKit Learn:

1. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression

Miscellaneous Sources:

1. https://stackoverflow.com/questions/2739159/inserting-a-pdf-file-in-latex
2. https://tex.stackexchange.com/questions/85200/include-data-from-a-txt-verbatim
3. https://www.overleaf.com/learn/latex/Headers_and_footers
4. https://stackoverflow.com/questions/4027363/two-statements-next-to-curly-brace-in-an-equation
5. https://jansoehlke.com/2010/06/strikethrough-in-latex/

Tools:

1. (ChatGPT,personalcommunication,April22,2023)
2. https://www.tablesgenerator.com/
3. https://htmtopdf.herokuapp.com/ipynbviewer/
4. https://phrasefix.com/tools/remove-tabs-from-text/

Other:

1. https://en.wikivoyage.org/wiki/RMS_Titanic