



**Data Optimization**

**Titanic: Machine Learning from Disaster**

**Description:**

* The ship Titanic sank in 1912 with the loss of most of its passengers. Details can be obtained of 1309 passengers and crew on board in the Titanic ship.
* The main use of this dataset is logistic regression with survival as the key dependent variable
* It describes the survival status of individual passengers on the Titanic. The principal source for data about Titanic passengers is the Encyclopaedia Titanic
* The variables on our extracted dataset are PassengerId, Survival, Pclass, Name, Sex, Age,

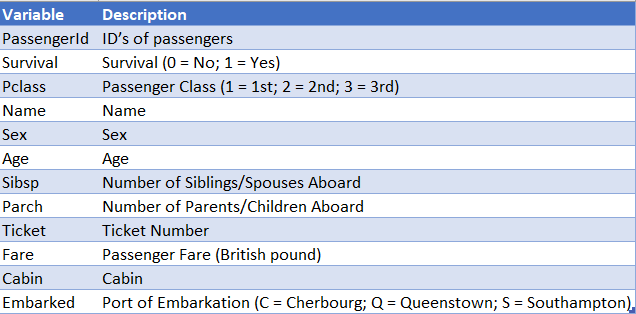
Sibsp, Parch Ticket, Fare, Cabin & Embarked

* Pclass refers to passenger class (1st, 2nd, 3rd), and is a proxy for socio-economic class
* Age in years is represented as float
* The variables are pclass, age, sex, survived
* These data frames are useful for demonstrating many of the functions, as well as demonstrating binary logistic regression analysis

**Dataset Specification**

**Name**: titanic(training dataset)

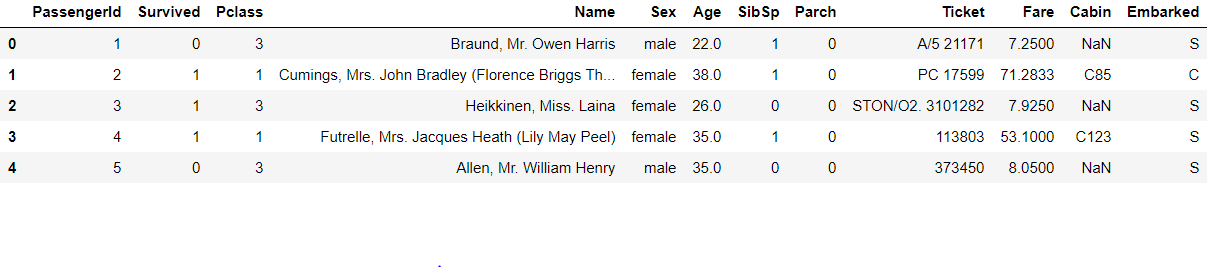
**Size**: 891 Passengers, 12 Variables

**Variable Descriptions** 

1. **Import python packages and dataset**

Import the necessary python packages and load the dataset. The dataset is loaded using pandas read\_csv method

train\_df=pd.read\_csv('train.csv')



1. **Data pre-processing**
   1. Handling missing data

Check for the missing data in training and test dataset. If the number of missing values is too high, we can ignore the column. Else we can impute the missing values with mean, median or mode of that feature based on its distribution.

# To check missing values in the train dataset

train\_df.isnull().sum()

PassengerId 0

Survived 0

Pclass 0

Name 0

Sex 0

Age 177

SibSp 0

Parch 0

Ticket 0

Fare 0

Cabin 687

Embarked 2

dtype: int64

Percentage of missing values in Age column 19.87%

Percentage of missing values in Cabin column 77.10%

Percentage of missing values in Embarked column 0.22%

Since the percentage of missing values in the Cabin column is 77%, imputing values won’t be appropriate. Hence, we can ignore the column.

For Age column, we can fill the missing values with the median of the column since the data is not uniformly distributed. For Embarked column, we can impute with the most frequent values in that column.

train\_data["Age"].fillna(train\_df["Age"].median(skipna=True), inplace=True)

train\_data["Embarked"].fillna(train\_df['Embarked'].mode()[0], inplace=True)

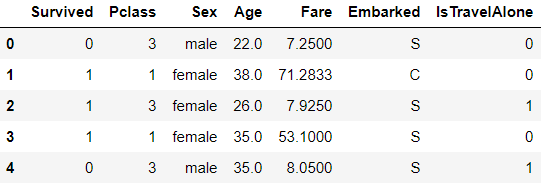
* 1. Combining SibSp and Parch column

SibSp and Parch are the informations about number of sibling/spouse abroad and number of parent/child abroad. These two columns can be combined into a single column consisting of boolean values depicting whether a passenger travelled alone or not.

train\_data["IsTravelAlone"] = np.where((train\_data['SibSp'] + train\_data['Parch']) >0,0,1)

train\_data.drop(['SibSp','Parch'], axis=1, inplace=True)

We can ignore the columns ['PassengerId','Cabin', 'Ticket','Name']



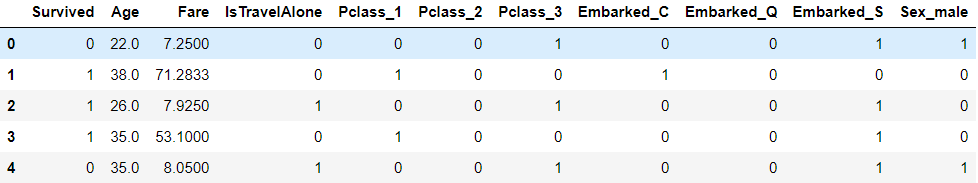
* 1. Handling categorical data

There are 3 columns that have categorical values ["Pclass","Embarked","Sex"]

We can use pandas.getdummies() method to create dummy columns for each categorical value. If the column has ‘n’ values, the method generates ‘n’ columns.

To avoid dummy variable trap, we should consider n-1 columns.

Cleaned data:



1. **Applying Classification algorithms**

The given problem is a classification model where we must predict a binary outcome whether the passenger survived or not.

* 1. Splitting data

Split the dataset into array of features and actual outcome using sklean.model\_selection train\_test\_split() method

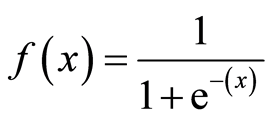
* 1. Feature scaling

Feature scaling is a method used to normalize the range of independent variables or features of data. This can be done using StandardScaler class from sklearn.preprocessing

* 1. Building Classification model using Logistic Regression

Logistic regression is a ML algorithm used to predict the probability of an outcome. It uses sigmoid activation function. The hypothesis of logistic regression tends it to limit the cost function between 0 and 1. Therefore linear functions fail to represent it as it can have a value greater than 1 or less than 0 which is not possible as per the hypothesis of logistic regression.

Sigmoid function

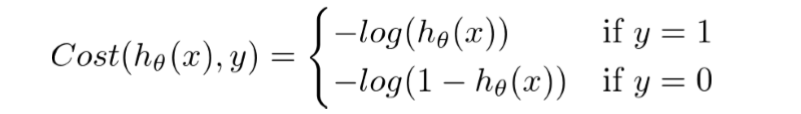


The function maps any real value into another value between 0 and 1. It maps prediction to probabilities.

Cost function

For logistic regression, the Cost function is defined as:

−log(hθ(x)) if y = 1 and −log(1−hθ(x)) if y = 0



Optimizer

It is used to minimize the cost function. There are several optimization algorithms that can be used in logistic regression model. We shall consider three optimizers

* Batch/Vanilla Gradient Descent

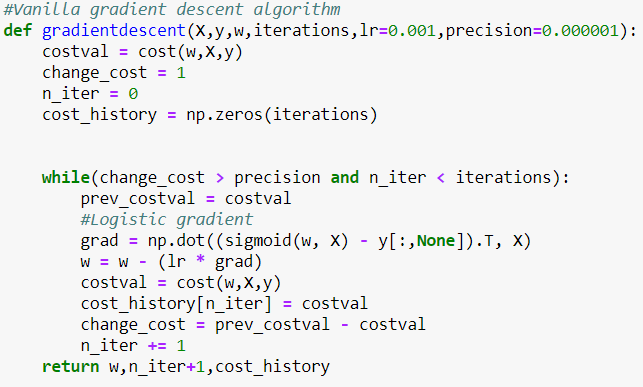
It is the simplest form of the gradient descent algorithm. Its main feature is that we take small steps in the direction of the minima by taking gradient of the cost function.

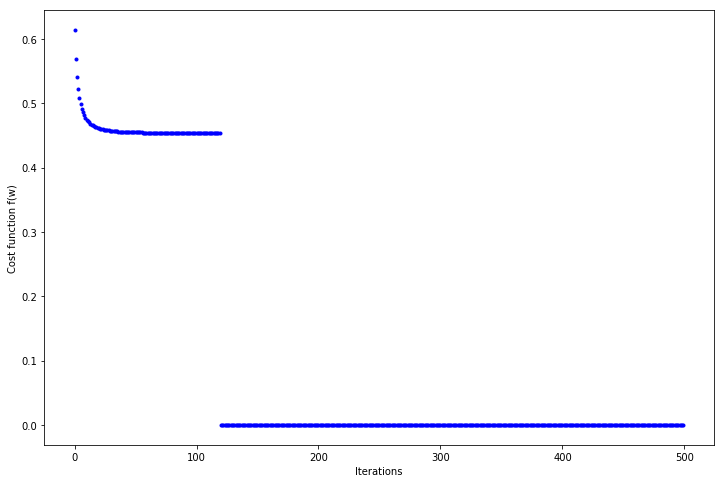
Pseudocode:

update = learning\_rate \* gradient\_of\_parameters

parameters = parameters - update

Gradient Descent makes an update to the parameters by taking gradient of the parameters. And multiplying it by a learning rate, which is essentially a constant number suggesting how fast we want to go the minimum. Learning rate is a hyper-parameter and should be treated with care when choosing its value. It takes all the records in the training set to make one update of the weights. For each iteration it considers all the training samples to update the theta value.

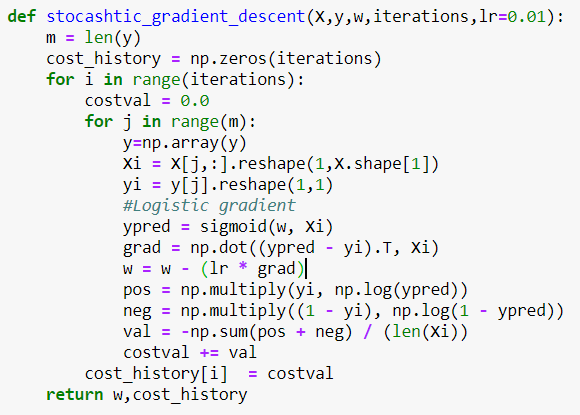


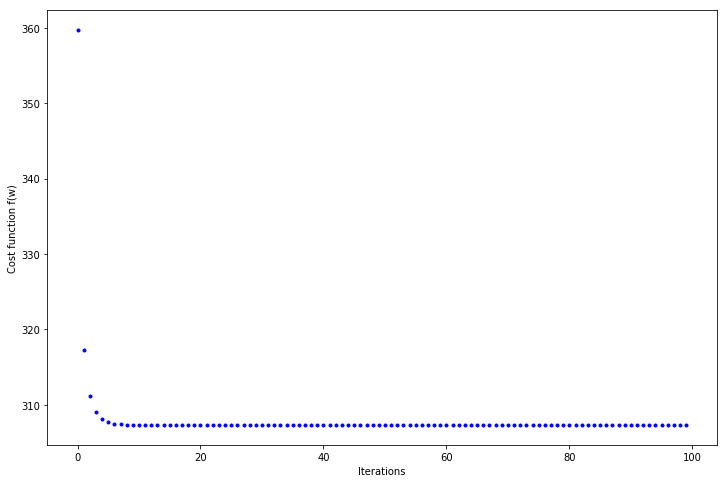


Since entire training data is considered before taking a step in the direction of gradient, therefore it takes a lot of time for making a single update. It makes smooth updates in the model parameters

* Stochastic Gradient Descent

In Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration. In Gradient Descent, we use batch which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration. In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset. Although, using the whole dataset is useful for getting to the minima in a less noisy or less random manner, but the problem arises when our datasets get huge.

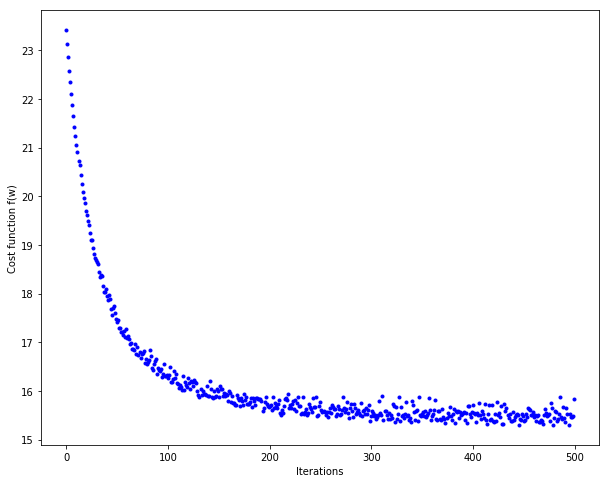
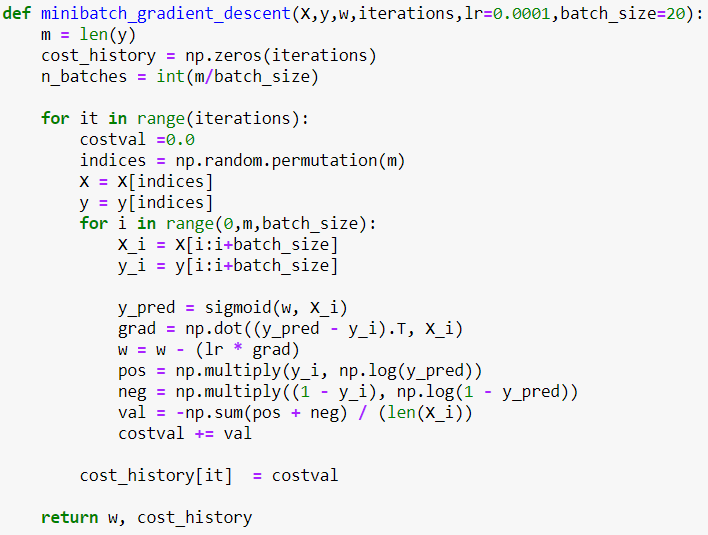




Since only a single training example is considered before taking a step in the direction of gradient, we are forced to loop over the training set and thus cannot exploit the speed associated with vectorizing the code. It makes very noisy updates in the parameters.

* Mini batch gradient descent

Mini-batch gradient descent makes a compromise between the speedy convergence and the noise associated with gradient update which makes it a more flexible and robust algorithm. Mini-batch gradient descent is a variation of the gradient descent algorithm that splits the training dataset into small batches that are used to calculate model error and update model coefficients. Implementations may choose to sum the gradient over the mini-batch which further reduces the variance of the gradient.



Since a subset of training examples is considered, it can make quick updates in the model parameters and can also exploit the speed associated with vectorizing the code. Depending upon the batch size, the updates can be made less noisy – greater the batch size less noisy is the update.

Stochastic gradient descent takes fewer number of iterations to converge when compared to the other optimizers. Hence it is the suitable optimizer for this dataset.

1. **Model Evaluation**

Confusion Matrix is used to find the number of True positives, True Negatives, False Positives and False Negatives

Vanilla Gradient descent Estimated regression coefficients :

[[-0.62940384 -0.42185745 0.01606745 0.03340444 0.89104112 0.52823296 0.03062518 -0.2128761 -1.22152831]]

Stochastic Gradient descent Estimated regression coefficients:

[[-0.58956742 -0.43644498 0.02953795 0.00284464 0.92863398 0.49671354 0.00866862 -0.25243295 -1.25132783]]

Mini batch Gradient Descent Estimated regression coefficients:

[[-0.60283804 -0.35919676 0.0752176 0.02067377 0.76465329 0.47130667 0.07199048 -0.16483538 -1.18779339]]