2017B1A31749H - Pragya Sinha
2018A7PS0189H- Rishabh Baid
2018A7PS0498H- Akshat Bajpai

18th April 2021

BITS F464 - Machine Learning

Logistic Regression

Introduction:

We implemented Logistic Regression for the given dataset:

	attr1	attr2	attr3	attr4	class
1101	2.194300	4.5503	-4.97600	-2.72540	1
77	0.049175	6.1437	1.78280	-0.72113	0
8	3.203200	5.7588	-0.75345	-0.61251	0
713	4.096200	10.1891	-3.93230	-4.18270	0
110	3.992200	-4.4676	3.73040	-0.10950	0

We observe from the data that, there are two classes to label the data points into, class 1 and class 0. This makes the problem a binary logistic regression. We also observe that there are 4 features in the data:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1372 entries, 1101 to 681
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   attr1   1372 non-null   float64
 1   attr2   1372 non-null   float64
 2   attr3   1372 non-null   float64
 3   attr4   1372 non-null   float64
 4   class   1372 non-null   int64
```

Logistic regression Model and its Implementation:

It is a classification algorithm used for classifying data points into a discrete set of classes. Logistic regression model transforms the model using log sigmoid function to separate the training data into classes. The sigmoid function is used to map predicted values between 0 and 1 for binary Logistic Regression.

2017B1A31749H - Pragya Sinha
2018A7PS0189H- Rishabh Baid
2018A7PS0498H- Akshat Bajpai

18th April 2021

BITS F464 - Machine Learning

$$S(z) = \frac{1}{1 + e^{-z}}$$

A decision boundary is selected such that probability values above it are classified as 1 and below it are classified as 0, typically set to 0.5 (Also known as threshold). The closer the output probabilities are to the boundary, the less confident those predictions are.

We calculate probabilities using weights such that

$$z = W_0 + W_1 * x_1 + W_2 * x_2 + \ldots + W_n * x_n$$

Where z is the output of the prediction function and $W_0, W_1, \ldots, W_n$ are weights and x are features.

Then we plug z into the sigmoid function to get the probability and the prediction is assigned for each observation based on this probability and the threshold.

The weights are updated using a cost function, which calculates the error and penalizes confident wrong predictions more than rewarding confident correct predictions and using a slope descent function to minimize cost such that cost is reduced after every iteration. We implemented two functions for minimizing cost: Gradient Descent and Stochastic Gradient Descent.

The cost function typically looks like:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))]$$

Here, m is the number of features (or attributes), y is the class value and h(x) is the predicted value. The cost function we have used is log loss function.

The Gradient Descent means that we have to get the slope of our cost function, which makes the function look like this:

$$C' = x(s(z) - y)$$

Here, C' is derivative of cost function, x is the feature vector, y is the class label value and s(z) is the sigmoidal function.

2017B1A31749H - Pragya Sinha
2018A7PS0189H- Rishabh Baid
2018A7PS0498H- Akshat Bajpai

18th April 2021

BITS F464 - Machine Learning

The SGD mathematical formula that we implemented looks like this:

$$E(w,b) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, f(x_i)) + \alpha R(w)$$

Where, n is the number of data points, L is the cost function, y is the class value, f(x) is the predicted value, alpha is the hyperparameter that controls regularization strength through a penalty function, R(w).

After the model is trained, we plug in test data points and calculate the probabilities using sigmoid function and map observations to its class. We plotted the graphs for three different learning rates for the LR models implementing GD and SGD.
One of the advantages of LogReg is that it performs well in a linearly separable dataset, and can be used to map multiclass classifications. The disadvantage of logistic regression is that it shouldn't be used for data where no. of attributes is greater than number of data points.
The above theory was implemented in python with 3 different learning rates of:

1) 0.1
2) 0.01
3) 0.0001

For both GD and SGD optimization. The dataset was split into 10 parts and a 70:30 split was performed on each fold to train the model.

Most important Attribute:
From examining the the optimized weights, we find out that the most important attribute for classifying data to be: **attr1**

2017B1A31749H - Pragya Sinha
2018A7PS0189H- Rishabh Baid
2018A7PS0498H- Akshat Bajpai

18th April 2021

BITS F464 - Machine Learning

Train Test Metrics

Observations

	Gradient Descent		Stochastic Gradient Descent	
Mean Values	**Train Data**	**Test Data**	**Train Data**	**Test Data**
Loss	**0.0991501297478 2758**	**-**	**0.0991501297478 2758**	**-**
Accuracy	**0.9583333333333 334**	**0.9577669902912 621**	**0.9537500000000 001**	**0.9521844660194 174**
Precision	**1.0**	**1.0**	**0.9959161420681 817**	**0.9936205601529 424**
Recall	**0.9065757013359 625**	**0.9043319967166 639**	**0.9000048523230 744**	**0.8975434929943 31**
F Measure	**0.9509832195096 986**	**0.9497146889457 865**	**0.9455099313804 072**	**0.9430293634349 17**

2017B1A31749H - Pragya Sinha
2018A7PS0189H- Rishabh Baid
2018A7PS0498H- Akshat Bajpai

18th April 2021

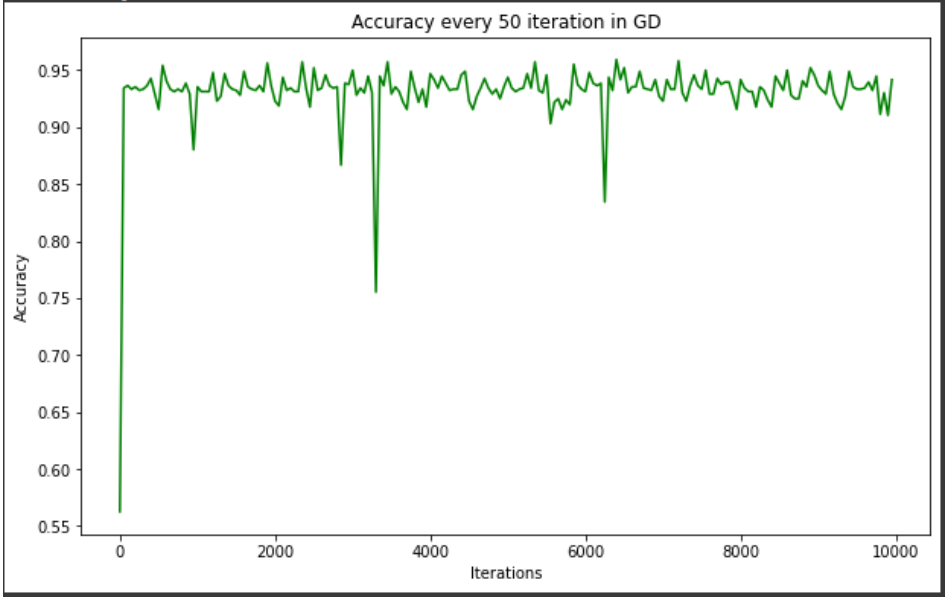
BITS F464 - Machine Learning

Graphs

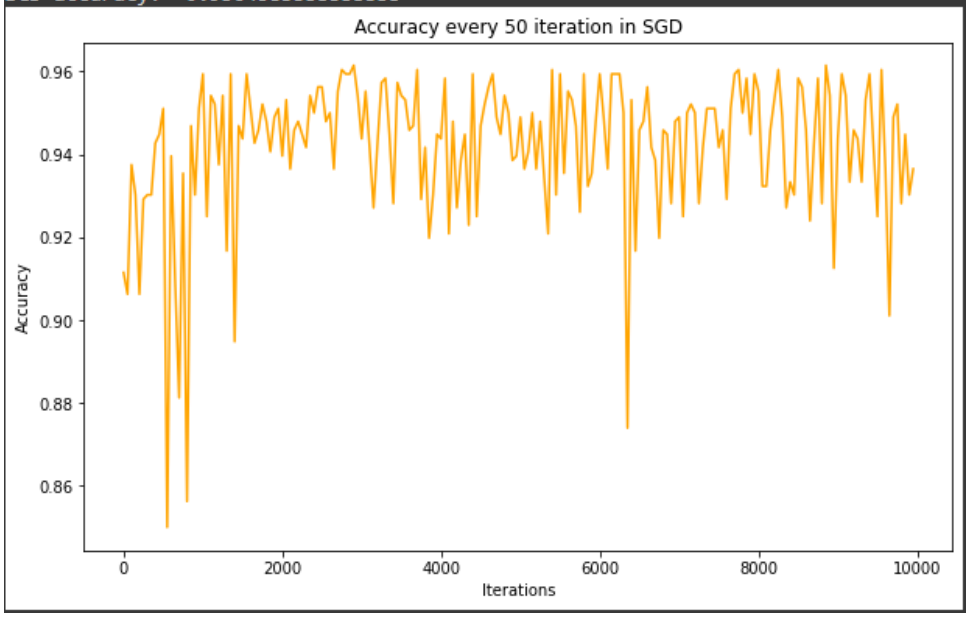
The graphs comparing the plot of the output accuracy every 50 iterations on GD and SGD on the same dataset split on three learning rates:

1. For Learning Rate = 0.1

```
Final GD weights:  [-97.66898388 -64.23757178 -74.4542047  -39.77332253]
GD accuracy:  0.9416666666666667
```




```
Final SGD weights:  [-4.16167844 -2.80196127 -3.20677633 -1.69920476]
SGD accuracy:  0.9364583333333333
```



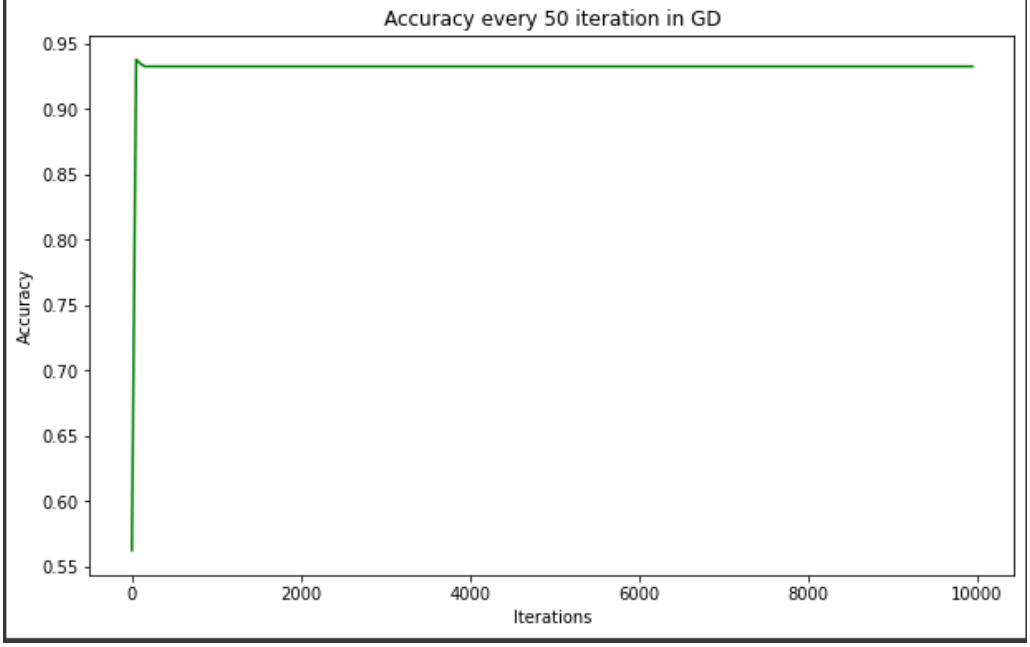
2017B1A31749H - Pragya Sinha
2018A7PS0189H- Rishabh Baid
2018A7PS0498H- Akshat Bajpai

18th April 2021

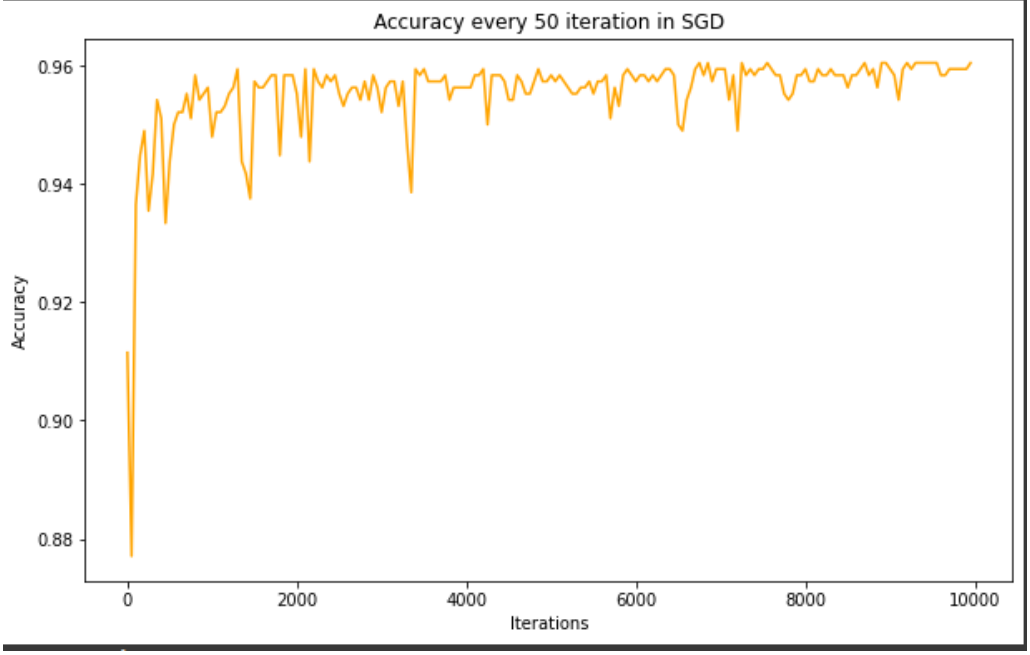
BITS F464 - Machine Learning

2. For Learning Rate = 0.001

```
Final GD weights:  [-9.23555545 -5.81380939 -6.8567428  -3.93726641]
GD accuracy:  0.9322916666666666
```




```
Final SGD weights:  [-2.21293508 -1.37925561 -1.34754745 -0.60677458]
SGD accuracy:  0.9604166666666667
```



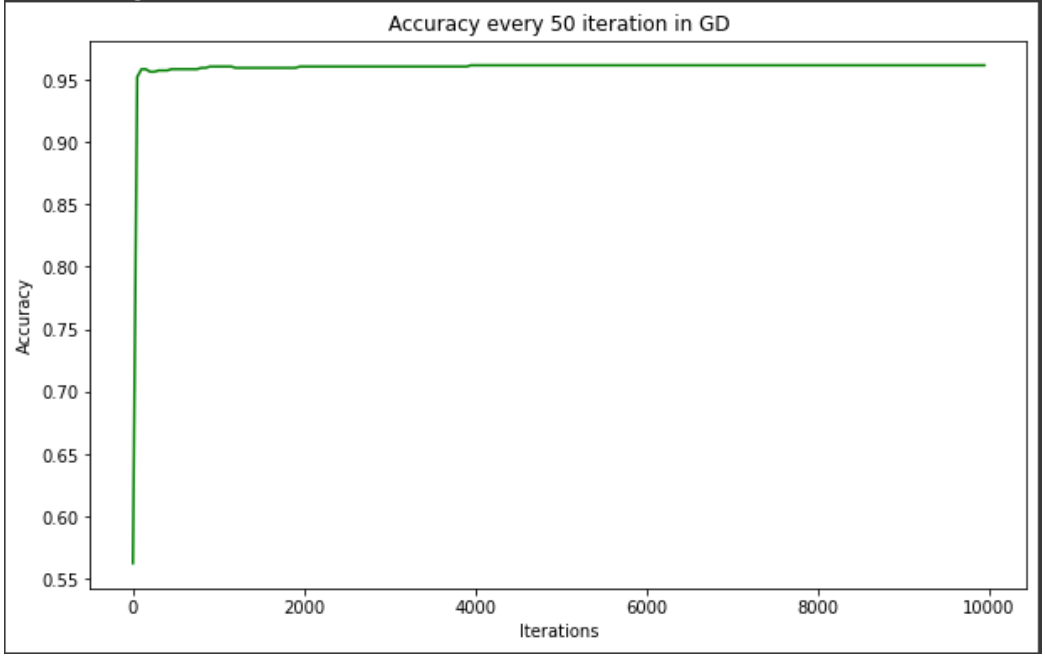
2017B1A31749H - Pragya Sinha
2018A7PS0189H- Rishabh Baid
2018A7PS0498H- Akshat Bajpai

18th April 2021

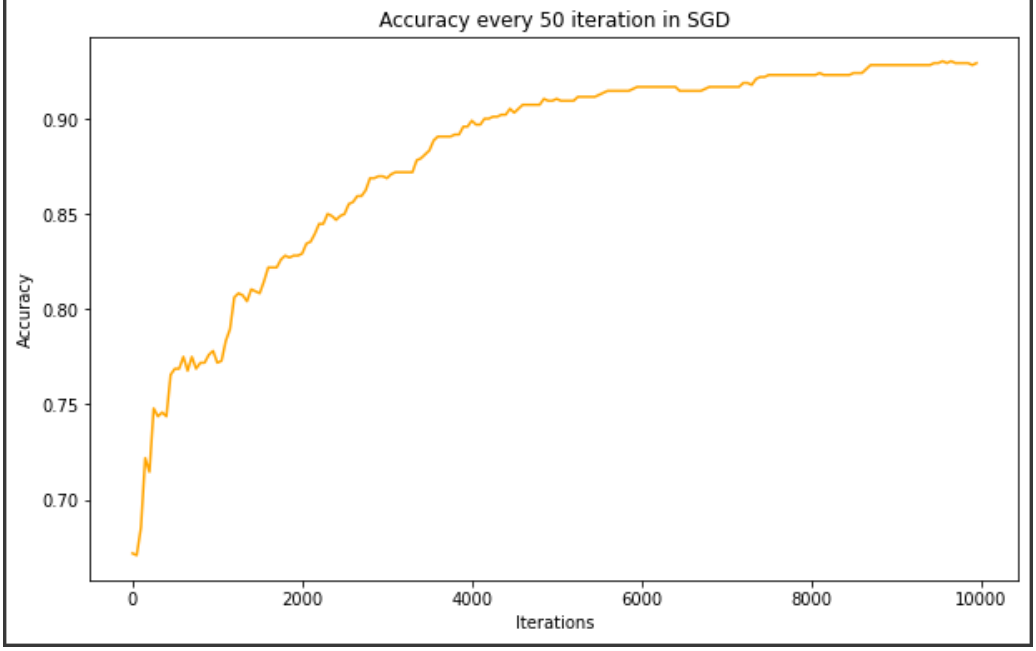
BITS F464 - Machine Learning

3. For Learning Rate = 0.0001

```
For Learning Rate : 0.0001
Final GD weights:  [-2.75916701 -1.67651633 -1.7905977  -0.79499356]
GD accuracy:  0.9614583333333333
```




```
Final SGD weights:  [-0.48391316 -0.24048127 -0.17647779 -0.09266914]
SGD accuracy:  0.9291666666666667
```

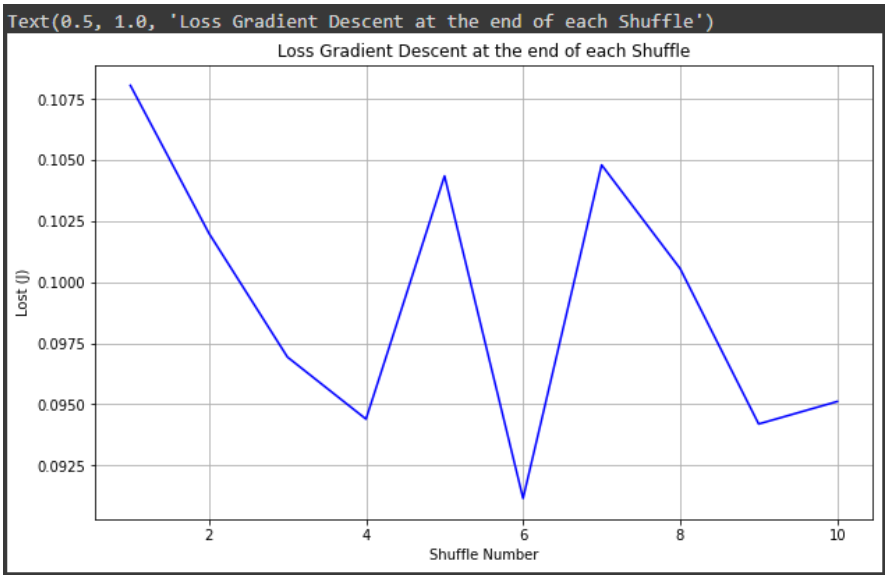


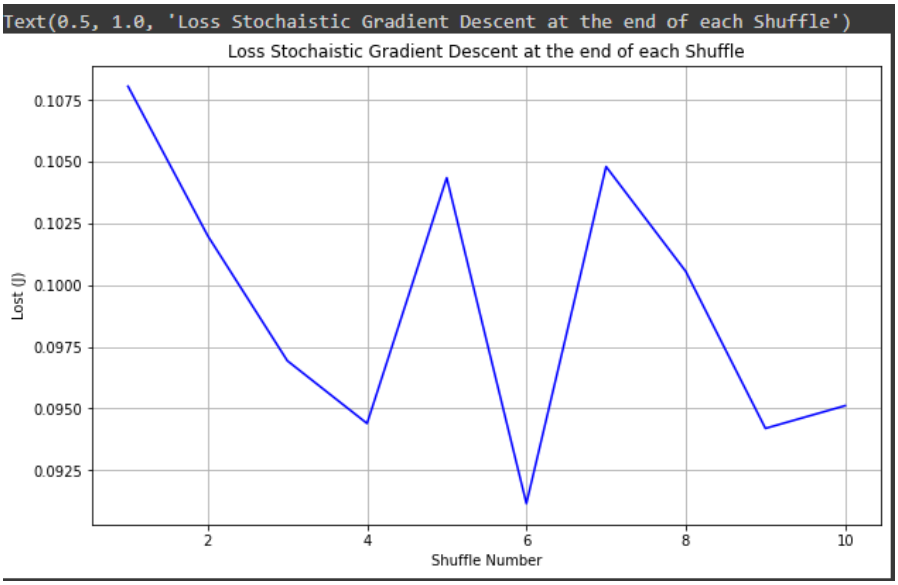
2017B1A31749H - Pragya Sinha
2018A7PS0189H- Rishabh Baid
2018A7PS0498H- Akshat Bajpai

18th April 2021

BITS F464 - Machine Learning

Furthermore we checked on the plot of the loss incurred after the end of each shuffle in the GD and SGD models :





2017B1A31749H - Pragya Sinha
2018A7PS0189H- Rishabh Baid
2018A7PS0498H- Akshat Bajpai

18th April 2021

BITS F464 - Machine Learning

We also looked at the convergence of the loss function in GD which wasn't the case of SGD