INF-552
MACHINE LEARNING FOR DATA SCIENCE

PROGRAMMING ASSIGNMENT 4:
Linear Classification,
Linear Regression &
Logistic Regression

NAME	USC-ID	EMAIL
Amitabh Rajkumar Saini	7972003272	amitabhr@usc.edu
Shilpa Jain	4569044628	shilpaj@usc.edu
Sushumna Khandelwal	7458911214	sushumna@usc.edu

PART 1: IMPLEMENTATION

- **Data Structures Defined/Used**
  - Dataset represented as numpy 2-D Array
  - Weights represented as numpy 1D array
  - Classes represented as numpy 1D array

Linear Classification

Perceptron Learning Algorithm

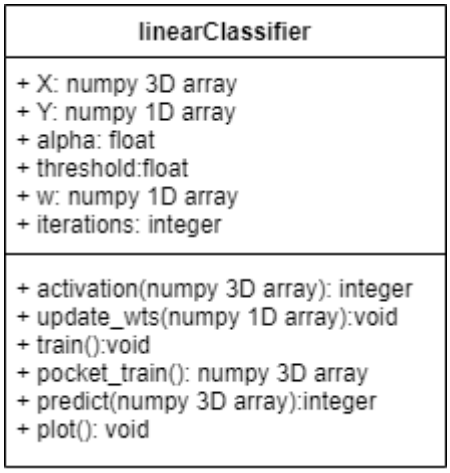
A linear classifier achieves objects separation by making a classification decision based on the value of a linear combination of the characteristics. An object's characteristics are also known as feature values and are typically presented to the machine in a vector called a feature vector. A linear classifier achieves this by making a classification decision based on the value of a linear combination of the characteristics. An object's characteristics are also known as feature values and are typically presented to the machine in a vector called a feature vector.

Algorithm: Perceptron Learning Algorithm

```
P ← inputs with label 1;
N ← inputs with label 0;
Initialize w randomly;
while !convergence do
    Pick random x ∈ P ∪ N ;
    if x ∈ P and w.x < 0 then
        w = w + x ;
    end
    if x ∈ N and w.x ≥ 0 then
        w = w − x ;
    end
end
//the algorithm converges when all the
 inputs are classified correctly
```

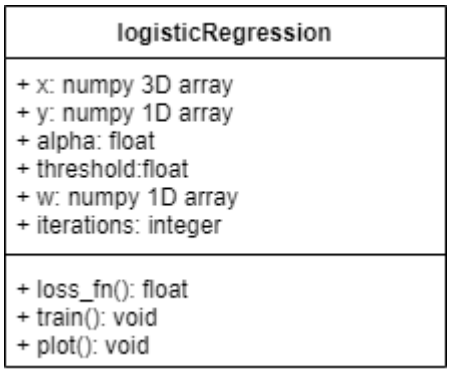
Pocket Learning Algorithm

1. Initialize the pocket weight vector, W_{pocket}, to 0 or small random numbers and use this weight vector as the initialized weight vector, W_0 of Perceptron Learning Algorithm.
2. For each training iteration, perform the following sub-steps:
3. Run the training step of Perceptron Learning Algorithm to obtain the updated weight vector, W_t, where t indicates the current iteration.
4. Evaluate W_t by comparing the number of misclassifications on the entire sample set with the number of misclassifications performed by W_{pocket}.
5. If W_t is better than W_{pocket}, replace W_{pocket} to W_t.
6. Return W_{pocket} when the training iteration terminates.



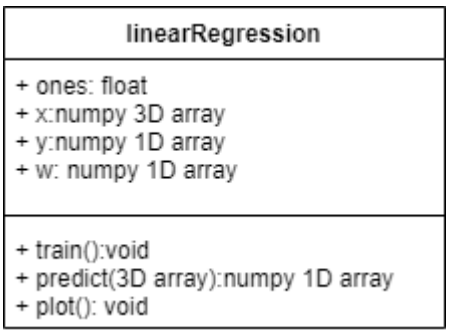
Logistic Regression:

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Some of the examples of classification problems are Email spam or not spam, Online transactions Fraud or not Fraud, Tumor Malignant or Benign. Logistic regression transforms its output using the logistic sigmoid function to return a probability value.



Linear Regression:

Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression. If we plot the independent variable (x) on the x-axis and dependent variable (y) on the y-axis, linear regression gives us a straight line that best fits the data points



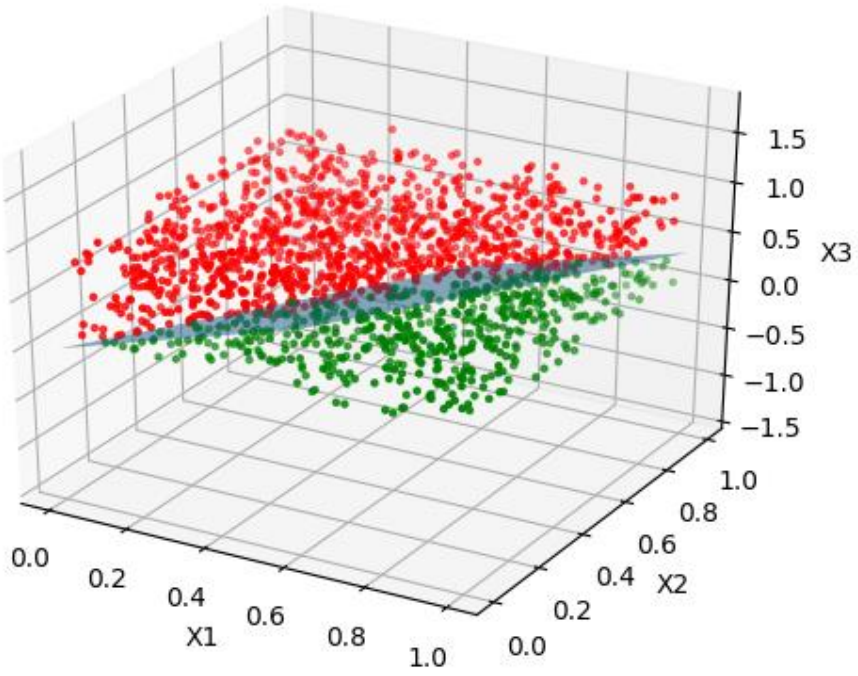
Run and Output the result (Our Implementation)

- **Perceptron Learning**

Output:

```
Perceptron Learning Algorithm
Number of Training Data 1600
Number of Test Data 400
Misclassification Threshold: 0.0
Weights: [[ 5.84546450e-04  6.76537260e-01 -5.44569979e-01 -4.04850199e-01]]
Misclassifications: 0
Accuracy:  0.9975
Recall:  1.0
Precision:  0.991869918699187
Fmeasure:  0.9959183673469388
=======================
['', 'P1', 'N0']
['P1', 122, 0]
['N0', 1, 277]
TP:  122
FP:  0
FN:  1
TN:  277
Equation: 0.00 + 0.68x1 + -0.54x2 + -0.40x3 = 0
```

Visualisation:

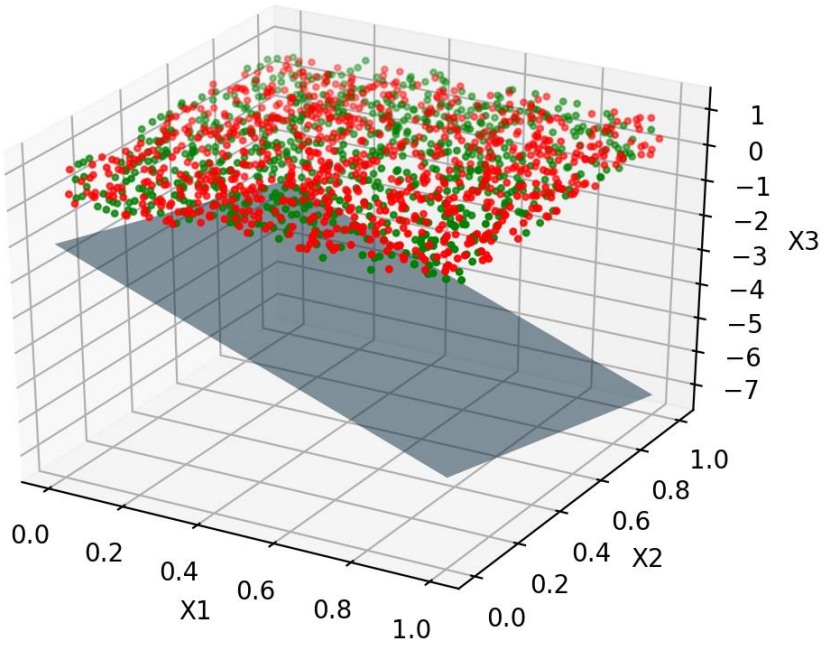


- **Pocket Algorithm**

Output:

```
Pocket Algorithm
Number of Training Data 1600
Number of Test Data 400
Iterations: 7000
Epoch: 6999        Min Misclassification: 758
Weights: [[0.01213509 0.04846756 0.02423028 0.01
Misclassification: 758
Accuracy:  0.465
Recall:  0.2843601895734597
Precision:  0.4878048780487805
Fmeasure:  0.3592814371257485
=======================
['', 'P1', 'N0']
['P1', 60, 151]
['N0', 63, 126]
TP:  60
FP:  151
FN:  63
TN:  126
Equation: 0.01 + 0.05x1 + 0.02x2 + 0.01x3 = 0
```

Visualization

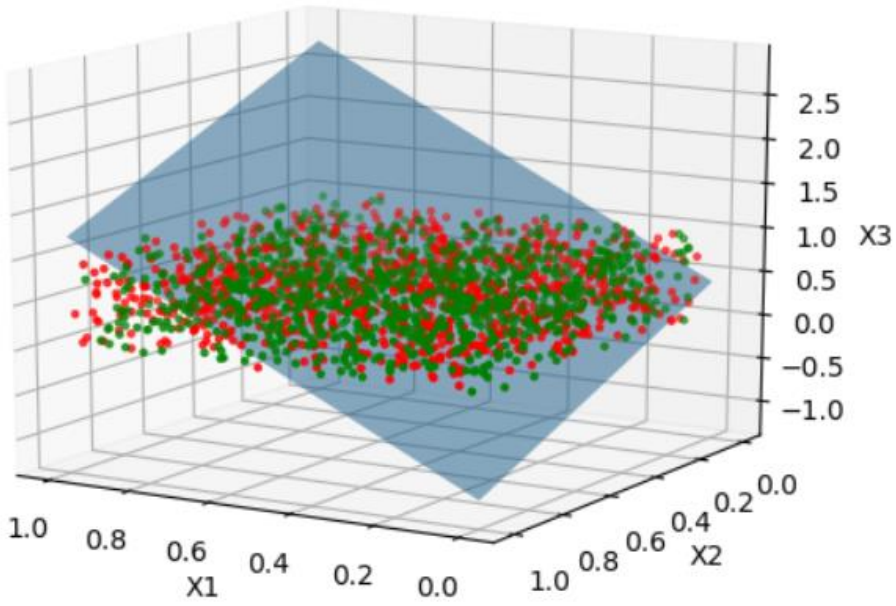


- **Logistic regression**

Output:

```
Logistic Regression
Number of Training Data 1600
Number of Test Data 400
Loss Value: [0.69314718]
Iterations: 7000
Loss Value: [0.69257597]
Weights: [[-0.08631761 -0.06490397  0.10089485  0.02172832]]
Accuracy:  0.4925
Recall:  0.08530805687203792
Precision:  0.6428571428571429
Fmeasure:  0.1506276150627615
=======================
['', 'P1', 'N0']
['P1', 18, 193]
['N0', 10, 179]
TP:  18
FP:  193
FN:  10
TN:  179
Equation: -0.09 + -0.06x1 + 0.10x2 + 0.02x3 = 0
```

Visualization:

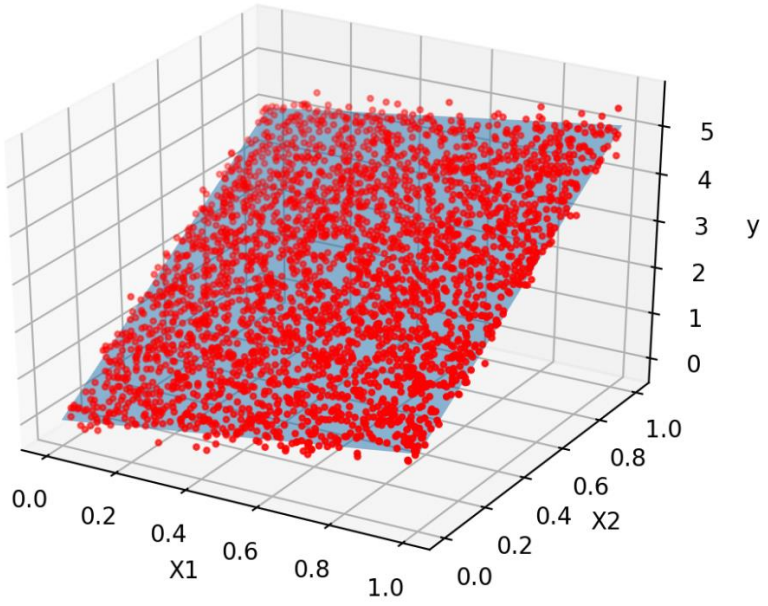


- **Linear Regression**

Output:

```
Linear Regression
Number of Training Data 1600
Number of Test Data 400
Weights: [0.01523535 1.08546357 3.99068855]
MAE: 0.15670441572187646
RMSE: 0.1875840278253885
Equation: y = 0.02 + 1.09x1 + 3.99x2
```

Visualization:



CHALLENGES

Challenges: Perceptron & Pocket Algorithm

- Fixing iteration required testing for a wide range of values 2.
- The learning rate was fixed at 0.01, though other values were tried and tested.
- While implementing the Pocket Algorithm, Since the given dataset is not linearly separable, we cannot classify the data properly.

Challenges: Logistic Regression

- More of a black box unless you learn the specifics.
- Fixing the rate of learning. The final value of weights changes with the rate of learning. Therefore, fixing this value required lot of trials.
- Since the given dataset is not linearly separable, we cannot classify the data properly.

Challenges: Linear Regression

- Finding the right function in the NumPy library that offers a good implementation of matrix operations.
- Finding the right function for computing inverse.

CODE LEVEL OPTIMIZATION

Optimization: Perceptron & Pocket Algorithm

1. Iteration was fixed for this input as 7000. This value was the most optimum one. Any value greater than 7000 gave no change in weights.
2. The learning coefficient alpha was fixed to 0.01.
3. Built in functions from the libraries is used.

Optimization: Logistic Regression

1. Learning rate optimally chosen as 0.01.
2. Built in functions from the libraries are used which makes our program efficient.
3. Vectorized algorithm for replacing iterations.

Optimization: Linear Regression

1. Built in functions from the numpy library for computing inverse of matrices.
2. Vectorized algorithm for replacing iterations.

PART 2: SOFTWARE FAMILIARIZATION

Library Function:

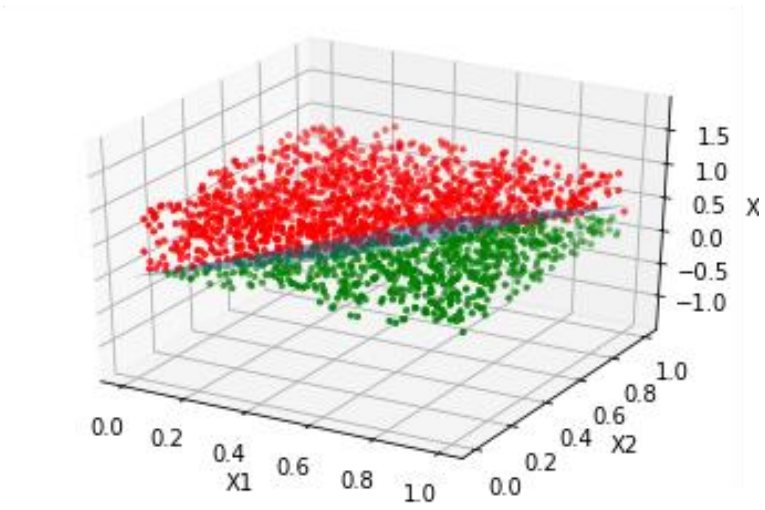
SGDClassifier

class sklearn.linear_model.**SGDClassifier**(*loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000, tol=0.001, shuffle=True, verbose=0, epsilon=0.1, n_jobs=None, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, class_weight=None, warm_start=False, average=False*

Output:

```
Weights:- [[ 17.66076251 -13.84208715 -10.37347967]]
Intercept:- [-0.01398779]


Accuracy on test dataset:- 0.9906666666666667
Accuracy on test dataset:- 0.988
              precision    recall  f1-score   support

        -1.0       1.00      0.98      0.99       338
         1.0       0.96      1.00      0.98       162

    accuracy                           0.99       500
   macro avg       0.98      0.99      0.99       500
weighted avg       0.99      0.99      0.99       500

Confusion Metrics
[[332   6]
 [  0 162]]
Equation: -0.01 + 17.66x1 + -13.84x2 + -10.37x3 = 0
```

Visualization:



Improvements:

Sklearn employs a function sparsify() to convert the matrix in to a sparse matrix. Converts the coef_ member to a scipy.sparse matrix, which for L1-regularized models can be much more memory- and storage-efficient than the usual numpy.ndarray representation. We can consider this in our program to improve the runtime.

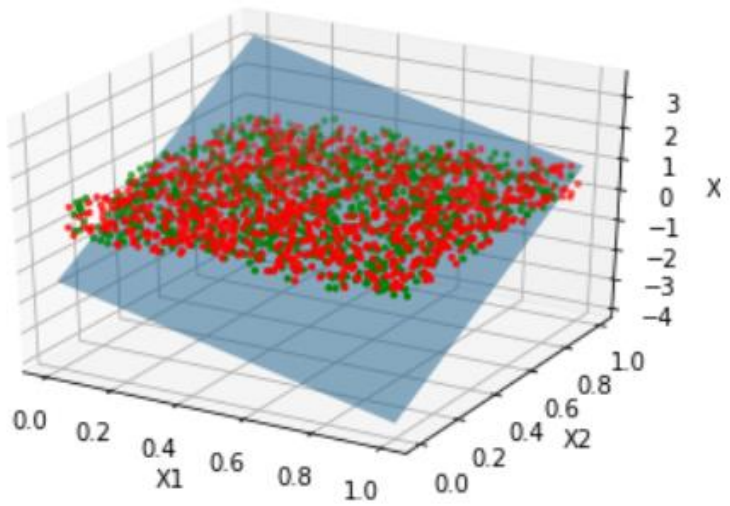
Logistic Regression

class sklearn.linear_model.**LogisticRegression**(*penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None*)

Output:

```
Accuracy of Logistic regression classifier on training set: 0.52
Accuracy of Logistic regression classifier on test set: 0.47
Accuracy of logistic regression classifier on test set: 0.47
[[155  91]
 [172  82]]
              precision    recall  f1-score   support

        -1.0       0.47      0.63      0.54       246
         1.0       0.47      0.32      0.38       254

    accuracy                           0.47       500
   macro avg       0.47      0.48      0.46       500
weighted avg       0.47      0.47      0.46       500

Weights
[[-0.14163105  0.27614164 -0.05919198]]
[-0.14163105  0.27614164 -0.05919198]
[-0.0822644]
Equation: -0.08 + -0.14x1 + 0.28x2 + -0.06x3 = 0
```

Visualization:



Improvement:

Sklearn employs a function sparsify() to convert the matrix in to a sparse matrix. Converts the coef_ member to a scipy.sparse matrix, which for L1-regularized models can be much more memory- and storage-efficient than the usual numpy.ndarray representation. We can consider this in our program to improve the runtime

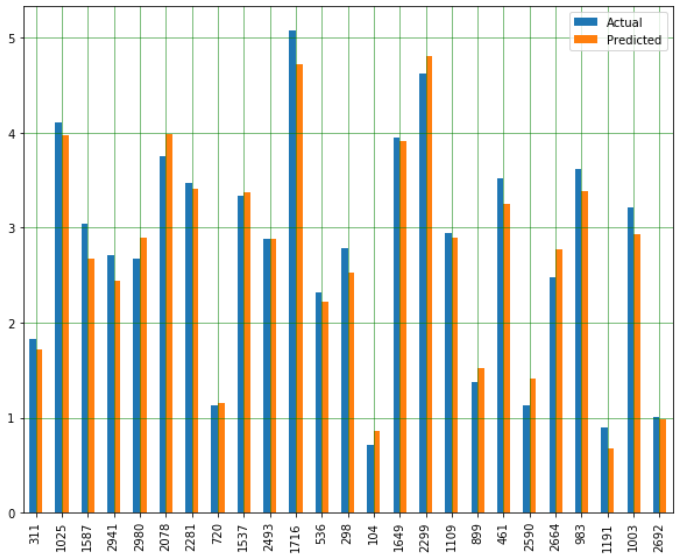
Linear Regression

class `sklearn.linear_model.`**`LinearRegression`**(*fit_intercept=True, normalize=False, copy_X=True, n_jobs=None*)

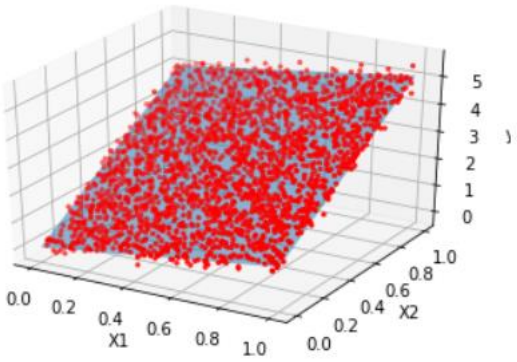
Output:

```
Coefs: [[1.08764022 3.99248301]]
Intercept: [0.0139561]
```

Visualization




```
MAE: 0.16038356316846566
RMSE: 0.19910844755974205
```



Improvements:

fit_intercept if set to false the input is not normalized. If set to true the input is normalized. Normalizing is a linear transformation of data to better the accuracy of the algorithm sometimes. We can add a configurable code to do normalization on a need basis depending on the dataset.

Comparison between our Implementation and sklearn library:

Sklearn employs a function sparsify() to convert the matrix in to a sparse matrix which for L1-regularized models can be much more memory and storage-efficient than the usual numpy.ndarray representation. We can consider this in our program to improve the runtime.

Most of the algorithms in Sklearn are implemented using Vectorization and thus they are faster than our program.

Observation:

After executing the program, we observed that the data set with column 5 as the category/classes is not linearly separable because the dataset is overlapping data set and it is not possible to apply classification algorithms on such data set.

PART 3: APPLICATIONS

Application: Perceptron Learning Algorithm

- Handwritten digit recognition
- Application of perceptron algorithm in classifying response of an individual to a particular dosage of a drug

Application: Pocket Algorithm

- Can be applied to credit screening: whether an applicant will be approved a credit card or not based on certain features like income, credit history.

Application: Logistic Regression

- Crime Data Mining: Predicting the crime rate of a state based on drug usage, number of gangs, human trafficking, and Killings.
- Assessing the risk of cardiovascular diseases based on current habits and health

Application: Linear Regression

- Predicting house prices with increasing size of houses.
- Predicting crop yields on rainfall

PART 4: CONTRIBUTION

The project was planned and implemented by all group members.

1. We all discussed the design of the project.
2. The code was built in group together with discussion and peer reviews within the group.
3. Library function was studied by each member individually and collaborated to compare and analyse the difference between our results and library functions output on Linear Regression, Linear classification and Logistic Regression
4. Design of data set, linear separability of data and misclassification calculations were all discussed and then coded.