Perceptron, Pocket, Logistic Regression and Linear Regression Implementation and Applications

Chaoyu Li, chaoyuli@usc.edu, 6641732094

DSCI-552: Machine Learning for Data Science

Instructor: Satish Kumar Thittamaranahalli

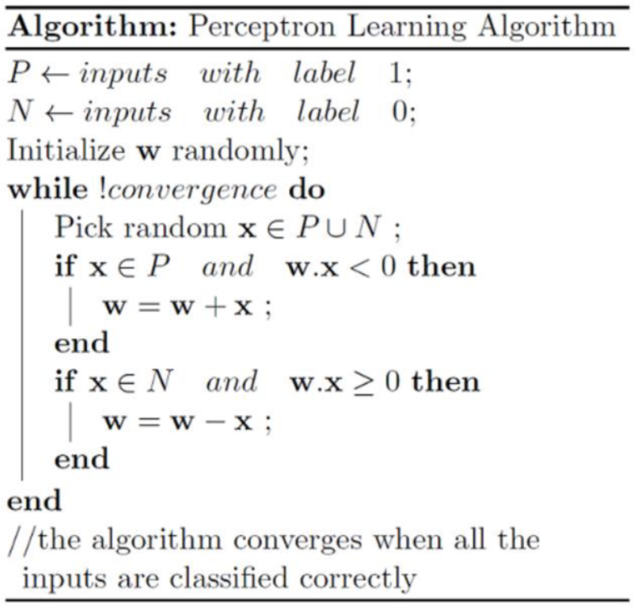
Mar. 5, 2022

Table of Contents

1 Basic Algorithm and Code Functions Description

1.1 Implementation of Perceptron Algorithm

1.1.1 Basic Algorithm of Perceptron



```
Algorithm: Perceptron Learning Algorithm
$P \leftarrow inputs \quad with \quad label \quad 1;$
$N \leftarrow inputs \quad with \quad label \quad 0;$
Initialize $\mathbf{w}$ randomly;
while $!convergence$ do
    Pick random $\mathbf{x} \in P \cup N$ ;
    if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ then
        $\mathbf{w} = \mathbf{w} + \mathbf{x}$ ;
    end
    if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ then
        $\mathbf{w} = \mathbf{w} - \mathbf{x}$ ;
    end
end
//the algorithm converges when all the
  inputs are classified correctly
```

1.1.2 Code Functions of Perceptron

- **train(X, Y_train, lr)**

It trains the Perceptron model to get the final weights. It accepts the dataset, labels and learning rate as input. It uses the algorithm in 1.1.1 to update the weights and errors value.

- **predict(X, weights)**

It predicts the new label for each point in the dataset. It accepts the dataset and trained weights as input.

1.2 Implementation of Pocket

1.2.1 Basic Algorithm of Pocket

The basic algorithm of Pocket is very similar to Perceptron.

The only difference between them is Pocket will try to keep track of the iteration with the least error number, only updating the weights if it lowers the error number.

1.2.2 Code Functions of Pocket

- **train(X, Y_train, lr, maxIteration)**

It trains the Pocket model to get the final weights. It accepts the dataset, labels and learning rate as input. It uses the algorithm in 1.1.1 to update the weights and errors value. The “maxIteration” here is set to 7000. It is the stop condition.

- **predict(X, weights)**

It predicts the new label for each point in the dataset. It accepts the dataset and trained weights as input.

1.3 Implementation of Logistic Regression

1.3.1 Basic Algorithm of Logistic Regression

```
Input: Training data
1. For i ← 1 to k
2. For each training data instance $d_i$:
3. Set the target value for the regression to
   $z_i \leftarrow \frac{y_j - P(1 \mid d_j)}{[P(1 \mid d_j) . (1 - P(1 \mid d_j))]}$
4. nitialize the weight of instance $d_j$ to $P(1 \mid d_j). (1 - P(1 \mid d_j)$
5. finalize a f(j) to the data with class value ($z_j$) & weights (wj)
   Classification Label Decision
6. Assign (class label:1) if $P(1|d_j) > 0.5$, otherwise (class label: 2)
```

1.3.2 Code Functions of Logistic Regression

- **calcSigmoid(z)**

It calculates the σ function. The formula is shown below:

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

- **calcGradient(weights, x, y)**

It calculates the incremental gradient of a single data point. It can be seen as a step of the formula below:

$$w_j := w_j + \eta \sum_{i=1}^{n} (y_i - \phi(z_i)) \cdot x_{ij}$$

- **train(X, Y_train, lr, maxIteration)**

It trains the Logistic Regression model to get the final weights. It accepts the dataset, labels and learning rate as input. It uses the algorithm in 1.3.1 to update the weights based on the gradient of the loss function. The "maxIteration" here is set to 7000. It is the stop condition.

- **predict(X, weights)**

It predicts the new label for each point in the dataset. It accepts the dataset and trained weights as input.

1.4 Implementation of Logistic Regression

1.4.1 Basic Algorithm of Linear Regression

$$\boldsymbol{w}^* = (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}$$

1.4.2 Code Functions of Linear Regression

- **train(X, Y_train)**

It trains the Linear Regression model to get the final weights. It accepts the dataset and labels as input. It uses the algorithm in 1.4.1 to update the weights.

2 Data Structure Used

2.1 Perceptron data structure

A. The data points are stored in the data structure of 2D NPArray. The format is:

[[x,y,z,label], [x,y,z,label], …, [x,y,z,label]]

B. The weight matrix is stored in the data structure of (dimension+1)*1 NPArray, here the dimension is 3. The format is:

[W_0, W_1, W_2, W_3]

C. The error number of each iteration is stored in the data structure of List. The length of the list is depended on the iteration. The format is:

[error in iter#1, error in iter#2, … , error in iter#k]

2.2 Pocket data structure

A. The data points are stored in the data structure of 2D NPArray. The format is:

[[x,y,z,label], [x,y,z,label], …, [x,y,z,label]]

B. The weight matrix is stored in the data structure of (dimension+1)*1 NPArray, here the dimension is 3. The format is:

[W_0, W_1, W_2, W_3]

C. The error number of each iteration is stored in the data structure of List. The length of the list is 7000 here. The format is:

[error in iter#1, error in iter#2, … , error in iter#7000]

2.3 Logistic Regression data structure

A. The data points are stored in the data structure of 2D NPArray. The format is:

[[x,y,z,label], [x,y,z,label], …, [x,y,z,label]]

B. The weight matrix is stored in the data structure of (dimension+1)*1 NPArray, here the dimension is 3. The format is:

[W_0, W_1, W_2, W_3]

C. The error number of each iteration is stored in the data structure of List. The length of the list is 7000 here. The format is:

[error in iter#1, error in iter#2, … , error in iter#7000]

2.4 Linear Regression data structure

A. The data points are stored in the data structure of 2D NPArray. The format is:

[[x,y,label], [x,y,label], …, [x,y,label]]

B. The weight matrix is stored in the data structure of (dimension)*1 NPArray, here the dimension is 2. The format is:

[W_0, W_1]

3 Challenges Faced and Optimizations

3.1 Challenges

A. When implementing the Perceptron algorithm, it was difficult to figure out what the number of iterations should be because I found that after a certain number of iterations I was sure to find a plane with 100% accuracy.

B. When implementing the Pocket algorithm and the Logistic Regression algorithm, I get results with low accuracy. At first, I thought that the algorithm I designed had a problem, but after comparing it with sklearn I found that the data itself is not separable.

C. If the covariance matrix is invertible, then we need to add a regularization term.

3.2 Optimizations

A. When implementing the Perceptron algorithm, after each misclassification instance, a weight update is performed and that instance is not visited again until a new iteration commences. After all misclassification examples are visited, the updated weights are checked by tracking the number of instances correctly classified.

B. Np.sign can recognize the class (+1 or -1) quickly.

C. When implementing the Logistic Regression algorithm, the gradient descent is cumulatively added after visiting each training instance then averaged out.

4 Execution of my algorithm and results

4.1 Execution of Perceptron program

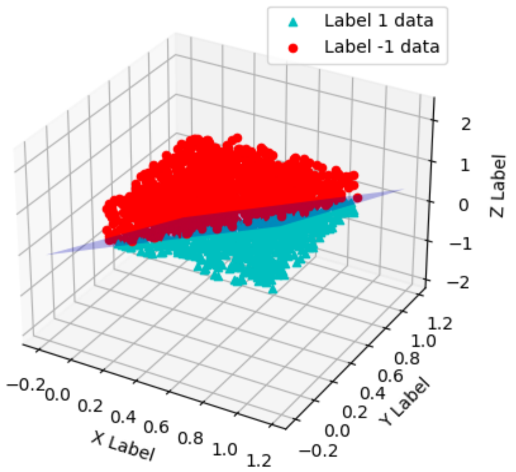
- Print of result:

Number of Iterations: 34255

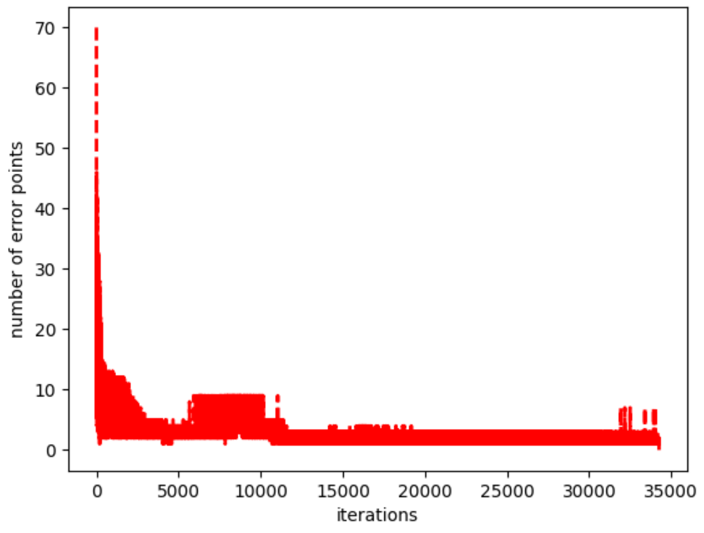
Accuracy: 1.0

Final Weights [W0, W1, W2, W3]: [-0.0065706   2.9824086 -2.38578088 -1.78239223]

- Image of 3D result:



- Image of error changes:



4.2 Execution of Pocket program

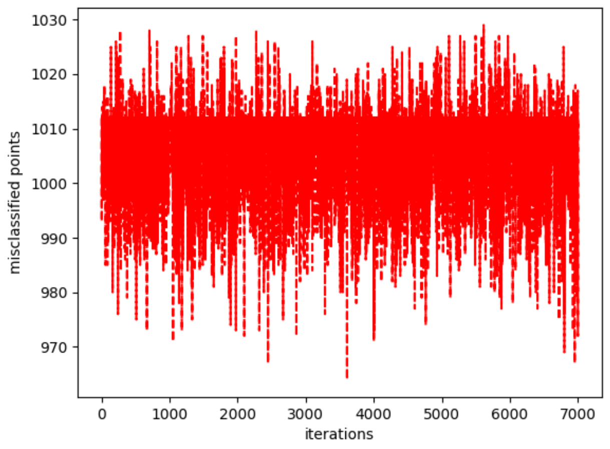
- Print of result:

Number of Iterations: 7000

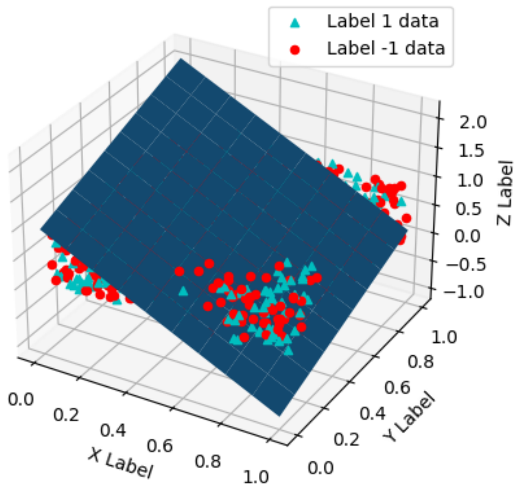
Accuracy: 0.518

Final Weights [W0, W1, W2, W3]: [0.0034294 -0.0068327   0.00379093 -0.0034939]

- Image of error changes:



- Image of 3D result:



4.3 Execution of Logistic Regression program

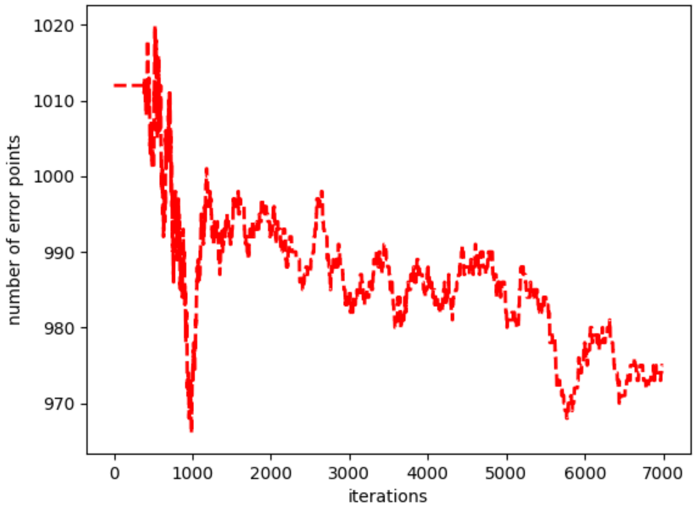
- Print of result:

  Number of Iterations: 7000

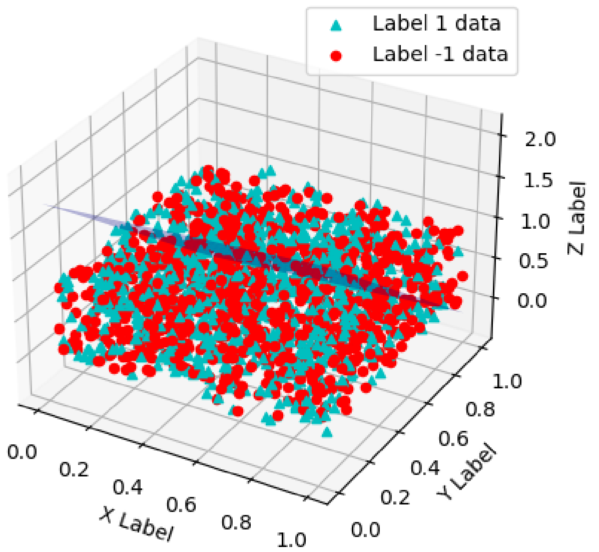
  Accuracy: 0.5125

  Final Weights [W0, W1, W2, W3]: [-0.15371782 -0.01663847 0.18210722 0.08226736]

- Image of error changes:



- Image of 3D result:

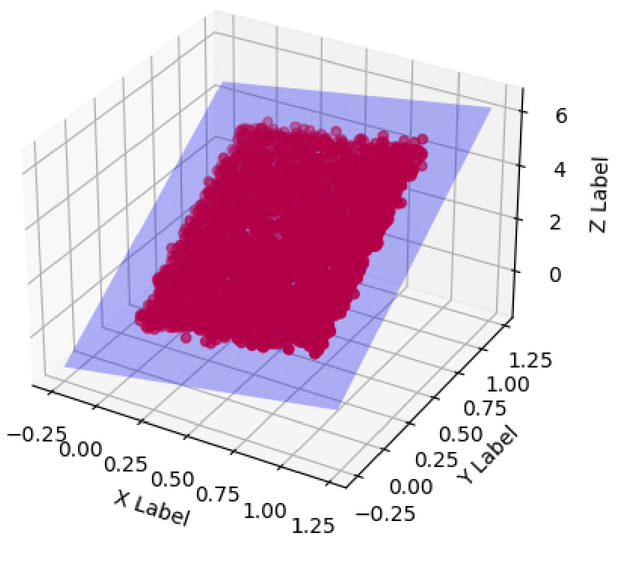


4.4 Execution of Linear Regression program

- Print of result:

```
Final Weights [W0, W1]: [1.09833772 4.00413136]
```

- Image of 3D result:



5 Comparison between my algorithm and scikit-learn

5.1 Perceptron Comparison

- The code of scikit-learn in Perceptron is:

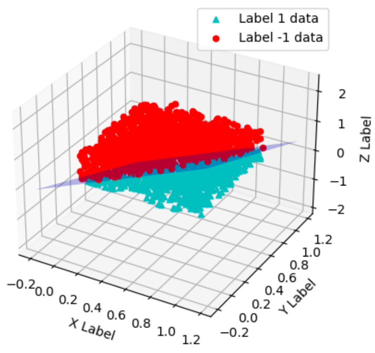
```
import matplotlib.pyplot as plt
from sklearn.linear_model import Perceptron
if __name__ == '__main__':
    # Perceptron
    X, Y = getInputData("classification.txt")
    X = np.c_[np.ones(len(X)), np.array(X)]
    pla = Perceptron()
    pla.n_iter = 1
    pla.max_iter = 40000
    pla.alpha = 0.01
    info = pla.get_params()
    print(info)
    pla = pla.fit(X, Y)
    score = pla.score(X, Y)
    W = pla.coef_
    print('Accuracy: {0}'.format(score))
    print('Final Weights [W0, W1, W2, W3]: {0}'.format(W))
```

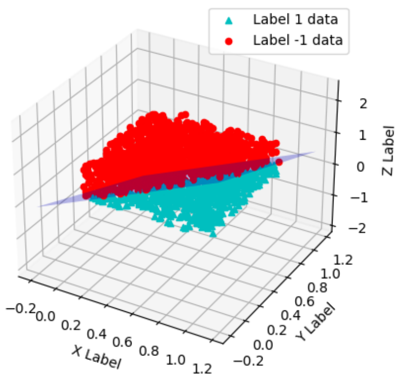
- The result is:

Accuracy: 0.98
Final Weights: [[0.   24.35173999 -18.56164074 -14.10260498]]

The accuracy is a bit lower than mine but it only executes 10000 times.

- The comparison of image between my program and scikit-learn is:





Mine **scikit-learn**

5.2 Pocket Comparison

- The code of scikit-learn in Pocket is:

```
import matplotlib.pyplot as plt
from sklearn.linear_model import Perceptron
if __name__ == '__main__':
# Pocket
    iterList = []
    numList = []
    best_score = 0
    W = None
    X, Y = getInputData1("classification.txt")  # Get column 5 as Y
    X = np.c_[np.ones(len(X)), np.array(X)]  # Coordinates vector of points, which is
added '1' at first column.
    pla = Perceptron()
    pla.n_iter = 1
    pla.warm_start = True
    info = pla.get_params()
    for i in range(0, 7000):
        pla = pla.fit(X, Y)
        score = pla.score(X, Y)
        ErrorNum = (1 - score) * 2000
        iterList.append(i)
        numList.append(ErrorNum)
        if (best_score <= score or i == 0):
            best_score = score
            W = pla.coef_
    print('Accuracy: {0}'.format(best_score))
    print('Final Weights [W0, W1, W2, W3]: {0}'.format(W))
```

- The result is:

  Accuracy: 0.531

  Final Weights [W0, W1, W2, W3]: [[0.   1.14929708 -0.2587861 -0.95413407]]

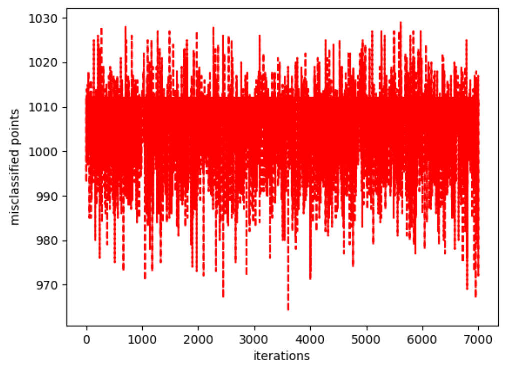
- My result is:

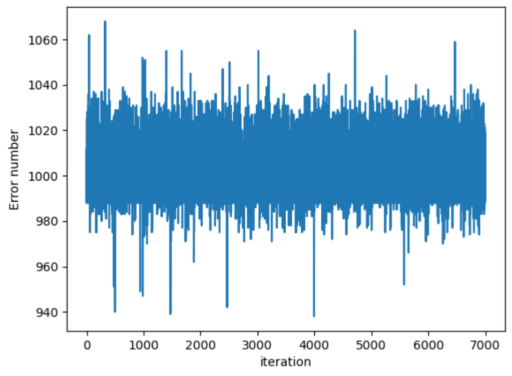
  Accuracy: 0.518

  Final Weights [W0, W1, W2, W3]: [0.0034294 -0.0068327   0.00379093 -0.0034939]

  The accuracy is a bit higher than mine, and it can be seen in the error plot.

- The comparison of error plot between my program and scikit-learn is:





Mine **scikit-learn**

5.3 Logistic Regression Comparison

- The code of scikit-learn in Logistic Regression is:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
lr = 0.01
numIteration = 7000
filename = "classification.txt"
np.random.seed(8)

if __name__ == '__main__':
    trainData = np.loadtxt(filename, delimiter=',', dtype='float', usecols=(0, 1, 2, 3, 4))
    X = trainData[:, :-1]
    y_train = trainData[:, -1]
    model = LogisticRegression()
    model.max_iter = numIteration
    model.fit(X, y_train)
    # predict
    y_predicted = model.predict(X)
    # print out results
    print('Number of Iterations: {0}'.format(model.n_iter_))
    print('Best Accuracy: {0}'.format(model.score(X,y_train)))
    print('Logistic Regression Weights: {0}'.format(model.coef_))
```

- The result is:
  Best Accuracy: 0.5295
  Logistic Regression Weights: [[-0.17403597 0.11202419 0.07513373 0.00029175]]
- My result is:

Accuracy: 0.5125

Final Weights [W0, W1, W2, W3]: [-0.15371782 -0.01663847 0.18210722 0.08226736]

The accuracy is a bit higher than mine.

5.4 Linear Regression Comparison

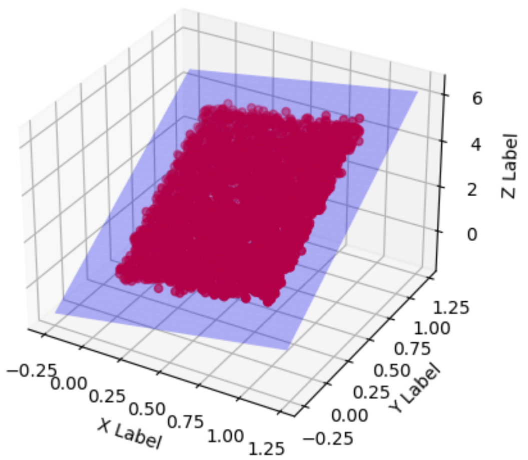
- The code of scikit-learn in Linear Regression is:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
if __name__ == '__main__':
    X, Z = getInputData("linear-regression.txt")
    lr = LinearRegression()
    lr.fit(X, Z)
    #print(str(lr.get_params))
    #print(lr.score(X,Z))
    print('Linear Regression Weights: {0}'.format(lr.coef_))
    plot3D(X, Z, lr)
```

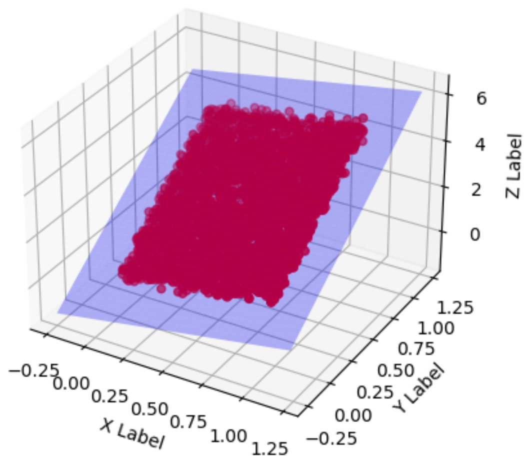
- The result is:

  Linear Regression Weights: [1.08546357 3.99068855]
- My result is:

  Final Weights [W0, W1]: [1.09833772 4.00413136]
- The comparison of image between my program and scikit-learn is:



Mine



scikit-learn

6 Application of Perceptron, Pocket, Logistic Regression and Linear Regression

6.1 Application of Perceptron and Pocket

- In the field of artificial intelligence, perceptron algorithms are mostly multi-layered. Multilayer perceptron classical neural networks are used for basic operations like data visualization, data compression, and encryption. Example: CNN.

6.2 Application of Logistic Regression

- **Credit scoring:** Data preprocessing for credit score modeling includes steps such as reducing correlated variables. With logistic regression, it's easy to find out which variables have more and less impact on the final outcome of the prediction. It is also possible to use methods such as recursive feature elimination to find the optimal number of features and eliminate redundant variables.
- **Medicine:** When we need to predict a binary answer, logistic regression is a good fit for this data type. The accuracy of the rapid blood test was improved by applying a logistic regression model.
- **Gaming:** Game companies use the logistic regression model to build a recommendation system that can respond quickly because fast speed is an advantage of the logistic regression algorithm. By analyzing user behavior, game companies can recommend equipment to players in a targeted manner.

6.3 Application of Linear Regression

- **Business:** Linear regression can be used in business to assess trends and make estimates or forecasts. For example, if a company's sales have grown steadily each month over the past few years, by performing a linear analysis of the sales data for monthly sales, the company can predict sales for the next few months.

- **Risk prediction:** Linear regression can be used to assess risk in financial services or insurance. For example, a car insurance company could use linear regression to assess risk based on a car's attributes, driver information, or demographics.

7 Contributions

All works were completed by Chaoyu Li.