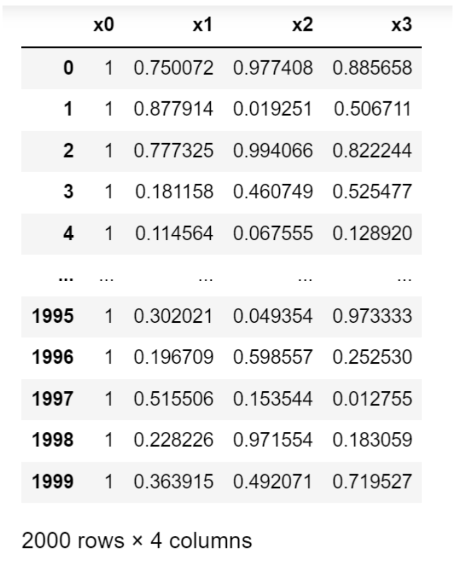
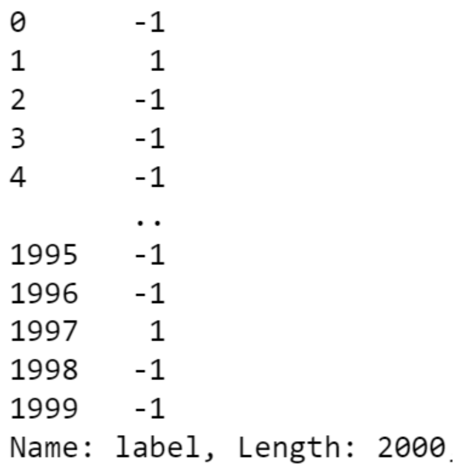
I. Data Structure

The data structure for the 'classification' txt file is a 2000 x 5 matrix with each row storing a data point's three independent variables and two category labels. Meanwhile, to calculate the threshold function's parameter w0, I add for each data point an x0 value of 1. Next, I divide the original data set into two parts: a data set X that stores all independent variables, and a one Y that stores each data point's label:



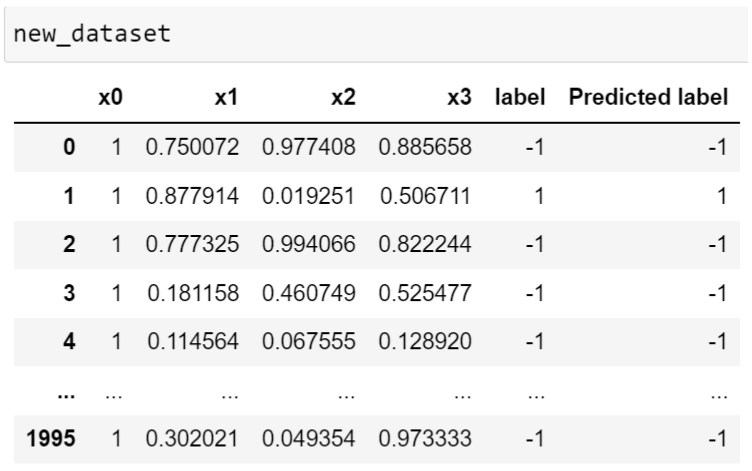
	x0	x1	x2	x3
0	1	0.750072	0.977408	0.885658
1	1	0.877914	0.019251	0.506711
2	1	0.777325	0.994066	0.822244
3	1	0.181158	0.460749	0.525477
4	1	0.114564	0.067555	0.128920
...	...	...	...	...
1995	1	0.302021	0.049354	0.973333
1996	1	0.196709	0.598557	0.252530
1997	1	0.515506	0.153544	0.012755
1998	1	0.228226	0.971554	0.183059
1999	1	0.363915	0.492071	0.719527

2000 rows × 4 columns



```
0       -1
1        1
2       -1
3       -1
4       -1
        ..
1995    -1
1996    -1
1997     1
1998    -1
1999    -1
Name: label, Length: 2000
```

Finally, when completing the PCA, Pocket algorithm, and logistic regression, I will attach a new column showing each data point's predicted label to the dataset:



```
new_dataset
```

	x0	x1	x2	x3	label	Predicted label
0	1	0.750072	0.977408	0.885658	-1	-1
1	1	0.877914	0.019251	0.506711	1	1
2	1	0.777325	0.994066	0.822244	-1	-1
3	1	0.181158	0.460749	0.525477	-1	-1
4	1	0.114564	0.067555	0.128920	-1	-1
...	...	...	...	...	...	...
1995	1	0.302021	0.049354	0.973333	-1	-1

The dataset for linear regression is a 3000 x 4 matrix. Like the pre-works done on 'classification' file, I add x0 of value 1 to each data point; and divide the dataset into two parts: one part stores independent variable x and the other contains dependent variable y.

II. Perceptron Learning Algorithm

Step 1. Import the packages and the txt file ‘classification’.

Step 2. Divide the dataset into two subsets: X (independent variables) and Y (label).

Step 3. Randomly select the parameters (weights) of threshold function:

```
#Randomize the initial weights:
weights = np.random.random(4)
print('The initial weights are',weights)
```

```
The initial weights are [0.62579261 0.3006867  0.1148084  0.25580977]
```

Step 4. Define the Perceptron Learning Algorithm function:

Function	Input	Output
perceptron_learning_algo	***Data***: dataset ***x***: independent variables ***y***: dependent variables(labels) ***w***: initial weights ***n_iteration***: times of iteration ***alpha***: learning rate	Updated weights (parameters for threshold function)

This function inserts each data point into threshold function with initial weights; next, for those data points whose threshold function value do not meet its label, then update the weights by moving it along its normal vector one alpha unit. Alpha unit is our learning rate, normally be set at 0.01:

```
for i in range(len(data)):
    thershold = np.dot(w,(np.array(x.iloc[i]))) #Calculate the thershold function
    if thershold < 0 and y.iloc[i] == 1: #Pick the violated data points and update weights
      w = w + alpha *np.array(x.iloc[i])
```

Step 5. Run the perceptron learning algorithm with 7000 times iteration and learning rate 0.01:

```
#update the weights 7000 tims with alpha 0.01:
weights_updated=perceptron_learning_algo(dataset,X,Y,weights,7000,0.01)
```

```
The weights after PLA iterates 7000 times are [ 0.00579261  0.65346804 -0.5161071  -0.37992947]
```

Step 6. Show the predicted results and the model accuracy rate:

	x0	x1	x2	x3	label	Predicted label
0	1	0.750072	0.977408	0.885658	-1	-1
1	1	0.877914	0.019251	0.506711	1	1
2	1	0.777325	0.994066	0.822244	-1	-1
3	1	0.181158	0.460749	0.525477	-1	-1
4	1	0.114564	0.067555	0.128920	-1	-1
...	...	...	...	...	...	...
1995	1	0.302021	0.049354	0.973333	-1	-1
1996	1	0.196709	0.598557	0.252530	-1	-1
1997	1	0.515506	0.153544	0.012755	1	1
1998	1	0.228226	0.971554	0.183059	-1	-1
1999	1	0.363915	0.492071	0.719527	-1	-1

2000 rows × 6 columns

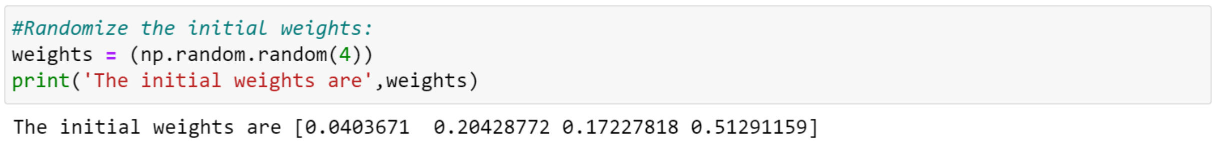
```
The accuracy of PLA after model iterates 7000 times is 0.982
```

III. Pocket Algorithm

Step 1. Import the packages and the txt file ‘classification’.

Step 2. Divide the dataset into two subsets: X (independent variables) and Y (label).

Step 3. Randomly select the parameters (weights) of threshold function:



```
#Randomize the initial weights:
weights = (np.random.random(4))
print('The initial weights are',weights)
```

```
The initial weights are [0.0403671  0.20428772 0.17227818 0.51291159]
```

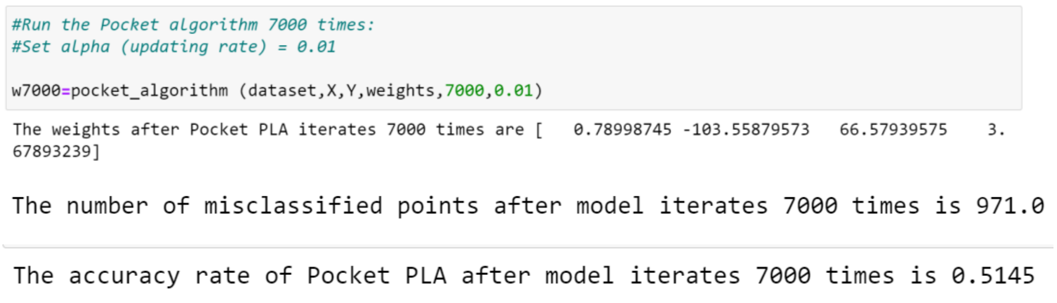
Step 4. Define the function to calculate the error rate of threshold function after each iteration finished. This function will return the percentage of wrong classified data points against the entire data set.

Function	Input	Output
error_rate	***Data***: dataset ***w***: initial weights ***x***: independent variables ***y***: dependent variables(labels)	wrong classified data points/ all data points

Step 5. Define the pocket algorithm function. This function update the weights the same way PLA has done but only leaves the updated weights that lower total error rate of classification from last iteration.

Function	Input	Output
pocket_algorithm	***Data***: dataset ***x***: independent variables ***y***: dependent variables(labels) ***w***: initial weights ***n_iteration***: times of iteration ***alpha***: learning rate	Updated weights (parameters for threshold function)

Step 6. Run the pocket algorithm 7000 times and calculate its accuracy:



```
#Run the Pocket algorithm 7000 times:
#Set alpha (updating rate) = 0.01

w7000=pocket_algorithm (dataset,X,Y,weights,7000,0.01)
```

```
The weights after Pocket PLA iterates 7000 times are [   0.78998745 -103.55879573   66.57939575    3.
67893239]

The number of misclassified points after model iterates 7000 times is 971.0

The accuracy rate of Pocket PLA after model iterates 7000 times is 0.5145
```

IV. Logistic Regression

Step 1. Import the packages and dataset ‘classification’.

Step 2. Divide the dataset into two subsets: X (independent variables) and Y (label).

Step 3. Randomly select the parameters (weights) of threshold function:

```
#Randomize the initial weights:
weights = np.random.random(4)
print('The initial weights are',weights)
```

```
The initial weights are [0.24551677 0.08817705 0.72030823 0.61341939]
```

Step 4. Define a function to calculate sigmoid function

Step 5. Calculate the initial cost by cost function:

$$\frac{1}{-2000}\sum_{i=1}^{2000}\left[y^i log\left(g\left(W^T x^i\right)\right)+\left(1-y^i\right)log\left(1-g\left(W^T x^i\right)\right)\right]$$

g() is a sigmoid function, W is a matrix for weights (parameters of regression line)

```
The initial cost of Logistic Regression is 1.294724992792918
```

Step 6. Update the weights 7000 times by:

$$W_{update}=W-(learning\ \ rate)\cdot\frac{1}{2000}X^T\cdot(g(X\cdot W)-y)$$

get the updated weights:

```
The weights of logistic regression after 7000 times iteration is [-3.60139425 -2.17464746 -0.84369358
-1.01531598]
```

Step 7. Calculate the updated cost and accuracy rate (the ratio of correct classification points against all data points):

```
The cost of Logistic Regression after 7000 times iteration is -20.03304834748808
```

```
The accuracy of Logistic regression after model iterates 7000 times is 0.506
```

V. Linear Regression

Step 1. Import the packages and the txt file ‘linear-regression’ as our dataset

Step 2. Divide the dataset into two subsets: X (independent variables) and Y (dependent variable).

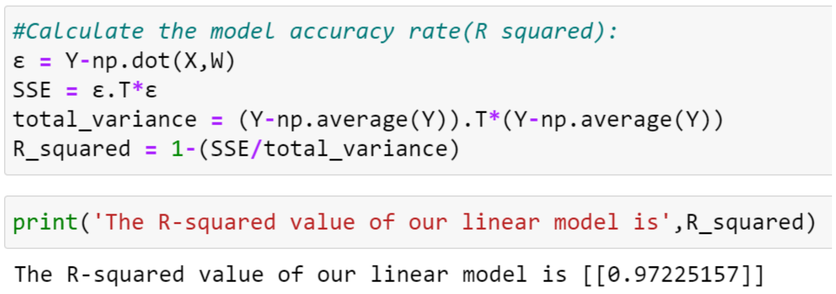
Step 3. Calculate the weights (regression line’s parameters) by least square method:

```
#Calculate the weights(parameters of the regression line):
W = (np.linalg.inv(X.T*X))*(X.T)*Y
```

```
W
```

```
matrix([[0.01523535],
        [1.08546357],
        [3.99068855]])
```

Step 4. Calculate the R-squared value of linear regression to check its accuracy:



```
#Calculate the model accuracy rate(R squared):
ε = Y-np.dot(X,W)
SSE = ε.T*ε
total_variance = (Y-np.average(Y)).T*(Y-np.average(Y))
R_squared = 1-(SSE/total_variance)
```

```
print('The R-squared value of our linear model is',R_squared)
```

```
The R-squared value of our linear model is [[0.97225157]]
```

VI. Challenges

The challenges of this homework are the abstraction of models' theory, the syntax of programming, and the selection of variables.

Unlike the simple regression model I've learned before, the four supervised learning models in this homework are more generic and abstract. For example, I've not until this homework studied logistic regression in the linear algebra approach. When we run the gradient descent to update the weight values of logistic regression, matrix and vectors indeed shorten the calculations step; however, it also makes the functions more obscure and abstract.

To overcome this abstraction, I resort to visualized data. For instance, I try to plot the data points in a 3-d space before constructing the PLA model and this helps me a lot in pinning down what plains in a 3-d space divide the data points from label 1 to -1.

I hit the second challenge—the syntax of programming—when training the Pocket algorithm. The construction of the Pocket algorithm relies heavily on nested iteration and condition expression. Writing nested iteration for updating weight values in Pocket algo, specifically, is a demanding task. It costs me great effort to arrange multiple conditions, one layer after another, for when to stop iteration and when to jump out of the current loop.

The third challenge is the selection of variables—mainly the selection of models' initial weights and learning rates. Too often the different random weight values or learning rate yield completely different results. For example, I once incorrectly choose an overly minute learning rate alpha that leaves the Pocket algo's weight values unchanged even after several iterations. The other case is four positive initial weight values that can barely derive negative values from threshold functions, and the outcome is my model incorrectly classifies all data points to class '1'.