AI5213/EC4213/CT5303/AI4021: Machine learning and deep learning **Spring 2023**

Report for Coding Assignment #1

(due on Wed May. 10, 2023. 11 :59PM)

Instructor: Jeany Son
Student name (GIST ID# / GitHub ID#): Doohyun Kim (20205018 / Luna0413)

REPORT1

There are three ham mail and three spam mail in training set. Therefore, prior is $P(ham) = P(spam) = \frac{3}{3+3} = \frac{1}{2}$.

Likelihood

1. Create a dictionary for all the words in the ham mail and spam mail in the test set.
Dictionary = ['000' '1999' '20' '31' 'ab' 'able' 'about' 'advertisement' · · · 'winner' 'with' 'within' 'would' 'writing' 'you' 'your' 'yrs']

2. Vectorize by counting the number of all the words that appear in each ham mail and spam mail.
3. To apply Laplace smoothing add 1 for all word in word-count-vector to ensure that P (wordi—ham) = 0 or P (wordj —spam) = 0 for a particular word.

count($x_i|ham$)
[1 1 1 1 1 2 4 1 2 2 1 4 1 1 4 1 2 1 1 1 3 2 1 1 2 2 1 5 1 1 2 5 1 1 2 3 1 1 4 1 1 2 1 1 2 2 1 1 1 4 1 2 2 2 2 1 4 1 1 1 4 2 2 1 1 2 3 3 1 1 5 1 2 1 2 2 1 1 2 2 2 2 2 2 2 2 4 1 1 1 2 1 1 1 1 1 1 2 2 1 2 1 1 4 1 1 2 1 5 1 4 1 2 2 2 2 2 1 1 1 4 1 1 3 1 3 1 1 2 1 2 4 2 2 1 2 1 1 3 1 1 2 2 2 1 2 3 2 3 1 2 1 1 2 2 1 2 14 1 2 1 1 4 1 1 1 2 9 3 3 1 1 1 1 1 2 1 2 2 2 4 1 1 1 2 1 1 4 1 2 2 4 4 1]

count($x_i|spam$)
[3 2 2 2 2 1 2 2 2 1 2 1 3 4 4 2 1 2 2 3 3 1 2 2 1 4 2 1 2 2 5 1 2 2 2 1 2 2 1 2 2 1 2 2 1 2 2 1 2 2 2 2 1 2 1 1 1 1 5 1 2 3 3 1 1 1 3 2 1 3 1 4 2 3 2 1 3 4 5 2 2 1 1 1 1 1 1 1 1 1 2 2 2 1 2 2 2 2 4 2 1 1 2 1 3 2 1 2 3 1 2 5 2 2 2 3 1 2 2 1 2 2 2 1 2 2 1 2 1 2 2 1 2 1 1 1 2 1 2 2 1 2 2 1 1 1 2 1 1 1 1 2 1 2 2 1 1 2 1 6 2 1 2 2 3 2 2 2 4 9 1 1 2 2 2 2 2 1 2 1 1 4 1 2 2 2 1 6 2 2 2 1 1 13 5 2]

4. Calculate $P(x_i|ham) = \frac{count(x_i|ham)}{\sum_X count(x|ham)}, P(xi|spam) = \frac{count(x_i|spam)}{\sum_X count(x|spam)}$.
These are likelihood with Laplace smoothing. If we don't apply Laplace smoothing, some $P(x_i|ham)$ can be 0 that means x_i doesn't appear in train ham mail. Then, the posterior about ham can be 0 if unused word in ham mail appear in test mail. The same problem also occurs in spam mail. Therefore, I didn't implement non Laplace smoothing version in code. However, it can be applied change value of variance 'laplace' from 1 to 0.

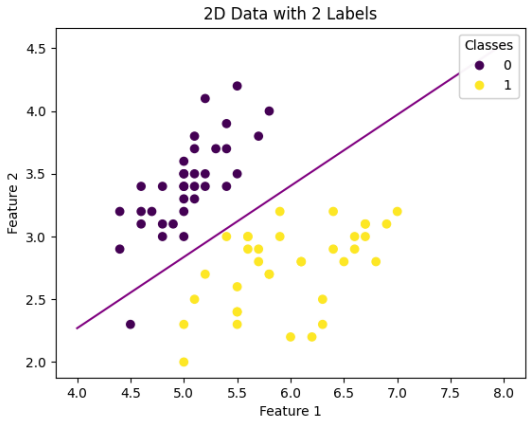
REPORT2

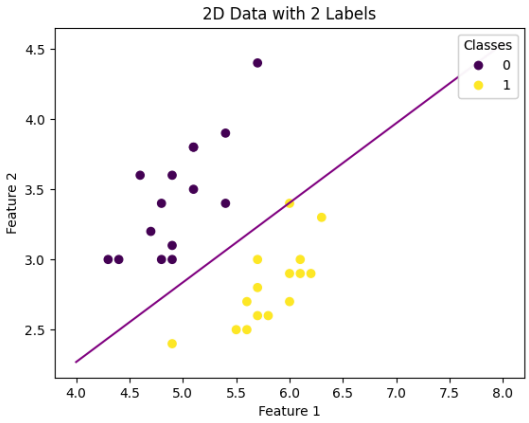
First of all, in gradient ascents, eta was multiplied by 1/n. This is for average of all train sets. Average was used to prevent overflow that occurs in the exp operation by the value of eta. Therefore, the eta value to which this is not applied may appear differently from the appropriate eta value in my code. Eta decreases by 0.9 times per epoch in my code.

Binary classification

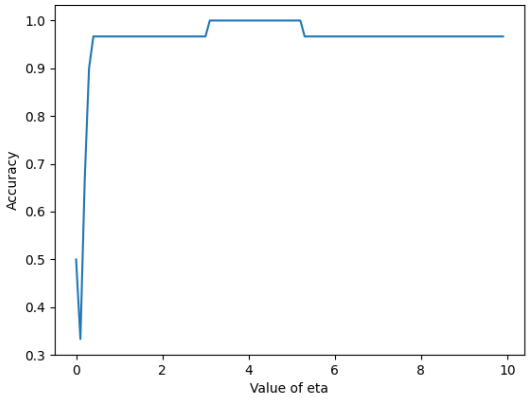
The iris data was preprocessed to have only 0,1 labels and the first two features. By using 'train-test-split (x, y, test-size=0.3, random-state=43)', the iris dataset is classified into 0.7 trainsets and 0.3 testsets. Based on maxIter = 100 and eta = 4.5, the train accuracy is 0.9857142857142858 and the test accuracy was 1. It can be checked through the test-binary-train, test-binary function in the logistic.py.

First graph is train set result and second graph is test(validate) set result. Graphs can be generated through graph-train, graph-test functions in the utils.py.





When sweaping eta based on maxIter= 100, the following accuracy graph is shown. Graphs can be generated through find-max-eta-ga functions in the utils.py. The parameters of the function are boolean values for selecting binary classification and 3 label classification.



When eta is approximately between 3.1 and 5, the acc of the test becomes maximum = 1.

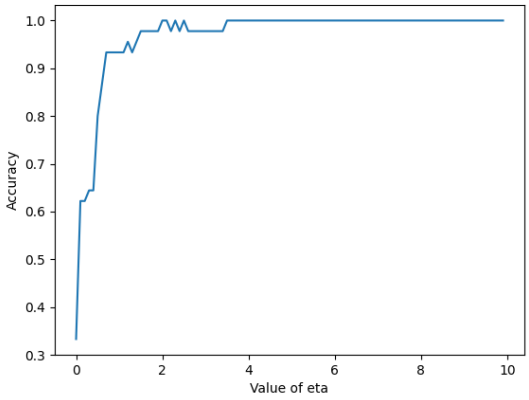
3 label classification

3 label classification use all of feature and lable in iris dataset. By using 'train-test-split (x, y, test-size=0.3, random-state=43)', the iris dataset is classified into 0.7 trainsets and 0.3 testsets. Since there are three decision boundaries, the weight also appears as three vectors. Based on maxIter = 100 and eta = 4.5, the train accuracy is 0.9619047619047619 and the test accuracy was 1. It can be checked through the test-3-label-train(w), test-3-label function in the logistic.py.

result weight

$$[1.68980407, 0.60136855, 4.88851657, -8.42776678, -3.21437916]$$
$$[0.83291043, -0.74922135, -2.76297635, 2.24299301, 0.24017361]$$
$$[-1.09103329, -4.67459307, -5.07792489, 7.2790465, 5.4597195]$$

When sweaping eta based on maxIter= 100, the following accuracy graph is shown. Graphs can be generated through find-max-eta-ga functions in the utils.py with parameter false. When eta is 2.0, acc first becomes 1, and after 3.5, acc converges to 1.

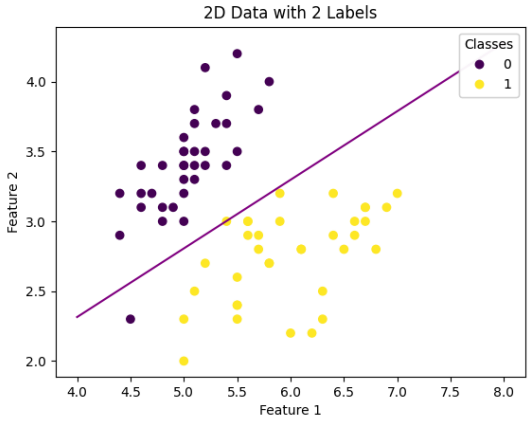


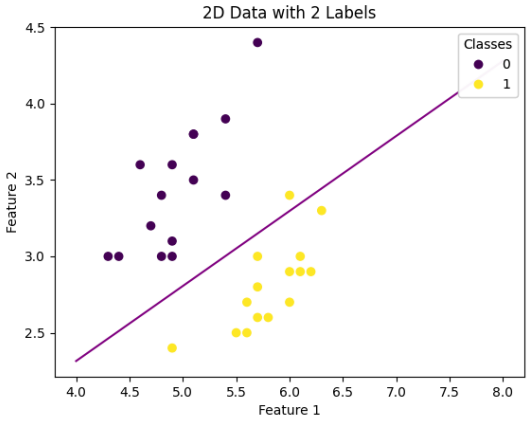
REPORT3

In Stochastic gradient ascent, I use random.randrange(0,len(trainset)) for choose random value in train set. Eta decreases by 0.9 times per one learning in my code.

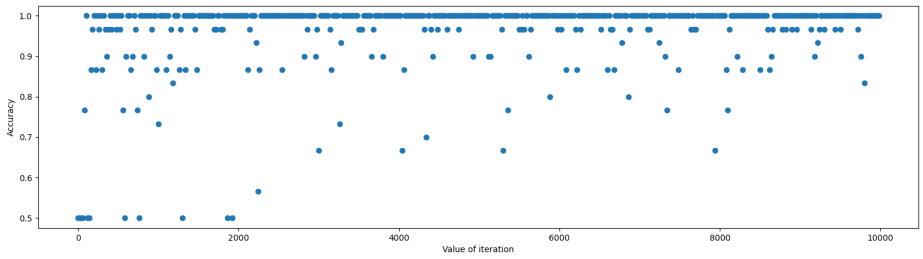
Binary classification

The iris data was preprocessed to have only 0,1 labels and the first two features. By using 'train-test-split (x, y, test-size=0.3, random-state=43)', the iris dataset is classified into 0.7 trainsets and 0.3 testsets. Based on maxIter = 10000 and eta = 0.7, the train accuracy is 0.9857142857142858 and the test accuracy was 0.9666666666666667. eta used the same eta as report4. The value of the weights learned by the random function may vary because the method of selecting the data to be learned from the code I implemented selected random values throughout the train set.





When sweaping iteration based on eta = 0.7, the following accuracy graph is shown. Graphs can be generated through find-max-ir-sga functions in the utils.py. The parameters of the function are boolean values for selecting binary classification and 3 label classification.



As the number of repetitions increases, the accuracy increases from 0 to 4000 in the initial section, but after that, the accuracy within the error range is shown. This is thought to be related to the value of eta and the decay of eta.

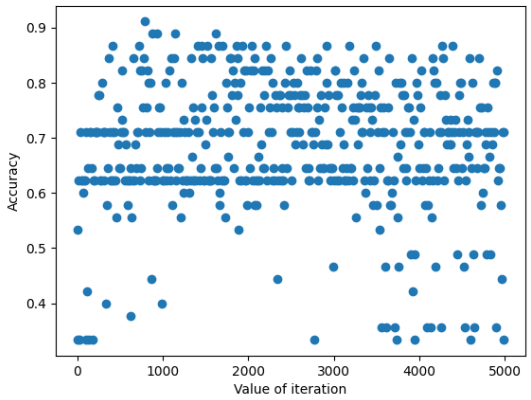
3 label classification

3 label classification use all of feature and lable in iris dataset. By using 'train-test-split (x, y, test-size=0.3, random-state=43)', the iris dataset is classified into 0.7 trainsets and 0.3 testsets. Since there are three decision boundaries, the weight also appears as three vectors. Based on iteration = 3000 and eta = 4.5, the train accuracy is 0.8761904761904762 and the test accuracy was 0.8222222222222222. Random selection issues exist the same. eta used the same eta as report4.

result weight

$$[6.15214595, 6.13894871, 23.07600868, -35.15012858, -15.57178104]$$
$$[106.48159559, 54.59344925, -165.24656925, 66.93531499, -215.33034718]$$
$$[-129.91465248, -185.49134615, -172.63546812, 273.07310964, 270.16851582]$$

When sweaping iteration based on eta = 4.5, the following accuracy graph is shown. Graphs can be generated through find-max-ir-sga functions in the utils.py with parameter false. It generally shows an accuracy of 0.6 to 0.85. If the accuracy is less than 0.5 after 3,500 iteration, overflow occurs in the exp calculation, resulting in an error due to the weight becoming nan(Not a number).



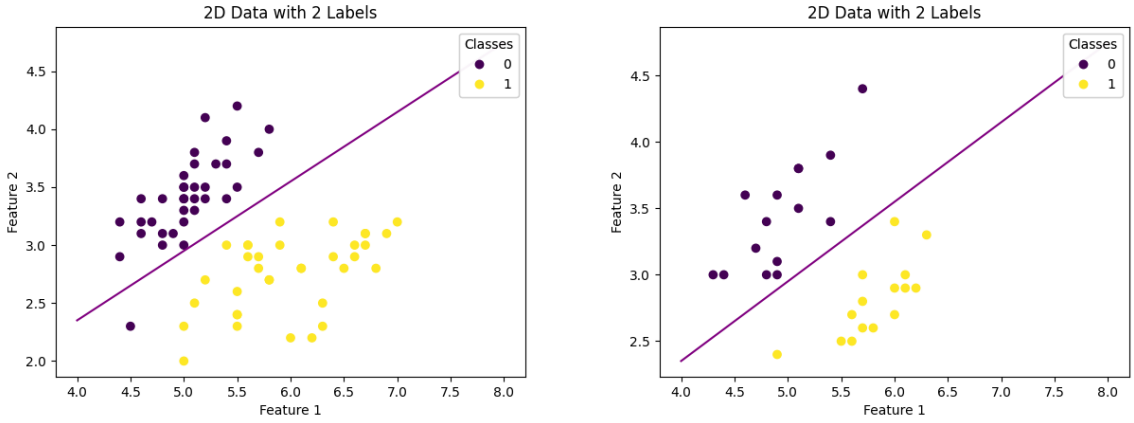
REPORT4

In the same way as gradient ascents, 1/n was multiplied by eta to use the average. Eta decreases by 0.9 times per epoch in my code.

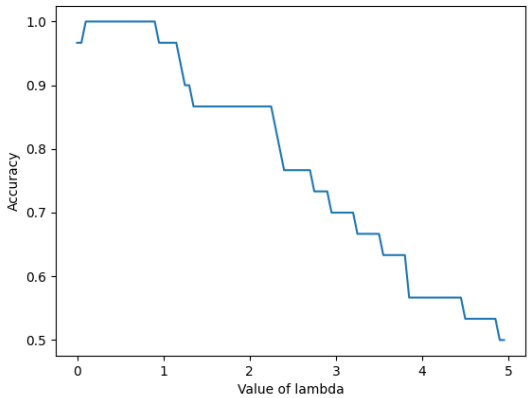
Binary classification

The iris data was preprocessed to have only 0,1 labels and the first two features. By using 'train-test-split (x, y, test-size=0.3, random-state=43)', the iris dataset is classified into 0.7 trainsets and 0.3 testsets. Based on maxIter =

100 and eta = 0.7 and lambda = 0.5, the train accuracy is 0.9857142857142858 and the test accuracy was 1.



When sweaping lambda based on maxIter= 100 and eta = 0.7, the following accuracy graph is shown. Graphs can be generated through find-max-lambda-MACP functions in the utils.py. The parameters of the function are boolean values for selecting binary classification and 3 label classification.



When eta is approximately between 0.1 and 0.9, the acc of the test becomes maximum = 1 .
3 label classification

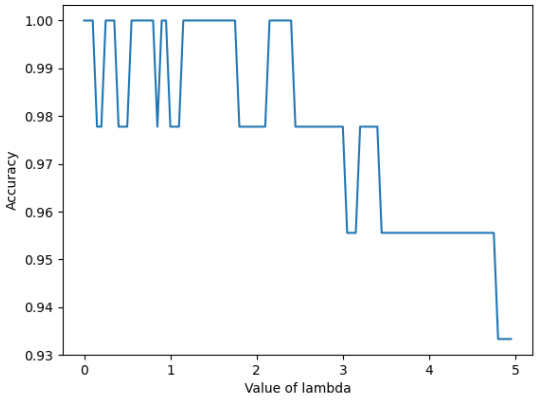
3 label classification use all of feature and lable in iris dataset. By using 'train-test-split (x, y, test-size=0.3, random-state=43)', the iris dataset is classified into 0.7 trainsets and 0.3 testsets. Since there are three decision boundaries, the weight also appears as three vectors. Based on maxIter = 100 and eta = 4.5 and lambda = 1.5, the train accuracy is 0.9619047619047619 and the test accuracy was 1.

result weight

$$[-3.16215553, 3.39978771, 4.65914114, -12.35386852, -9.47169378]$$
$$[-4.3156749, 2.04228457, -4.47458773, 2.23891599, -4.0938886]$$
$$[-6.26785732, -1.95337617, -6.95615842, 7.44855616, 1.22055368]$$

When sweaping eta based on maxIter= 100 and eta = 4.5, the following accuracy graph is shown. Graphs can be generated through find-max-lambda-MACP functions in the utils.py with parameter false. The value of Lambda tends

to decrease in acc after 2.4.



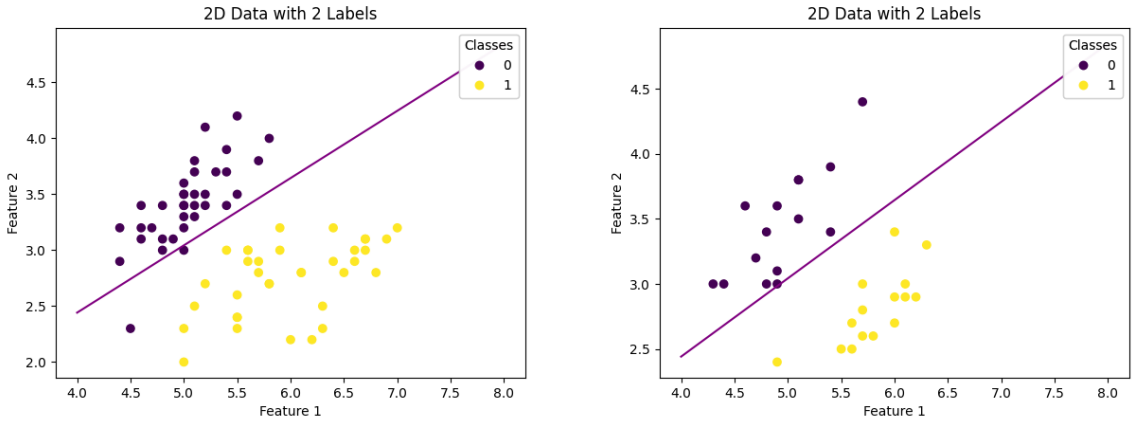
REPORT5

The time point at which the learning of the perceptron ends was set to convergence of weights and 200 repetitive times. The maximum number of iterations was set to prepare for the case where the test set was not linearly separable.

Binary classification

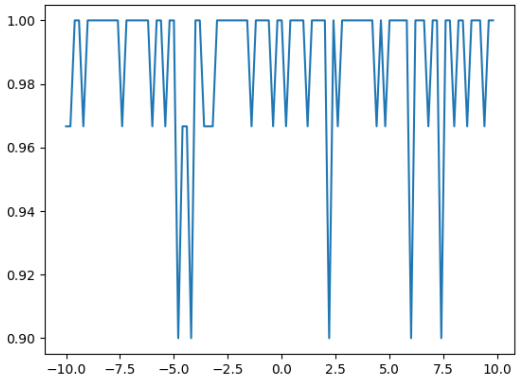
The iris data was preprocessed to have only -1, 1 labels and the first two features. By using 'train-test-split (x, y, test-size=0.3, random-state=43)', the iris dataset is classified into 0.7 trainsets and 0.3 testsets. Based on threshold = 3.0, the train accuracy is 0.9714285714285714 and the test accuracy was 1. It can be checked through the test-perceptron-binary-train, test-perceptron-binary function in the perceptron.py.

First graph is train set result and second graph is test(validate) set result. Graphs can be generated through graph-train, graph-test functions in the utils.py.



When sweaping threshold, the following accuracy graph is shown. Graphs can be generated through find-threshold-

perceptron-binary functions in the utils.py.



In general, it shows an accuracy of nearly 1.

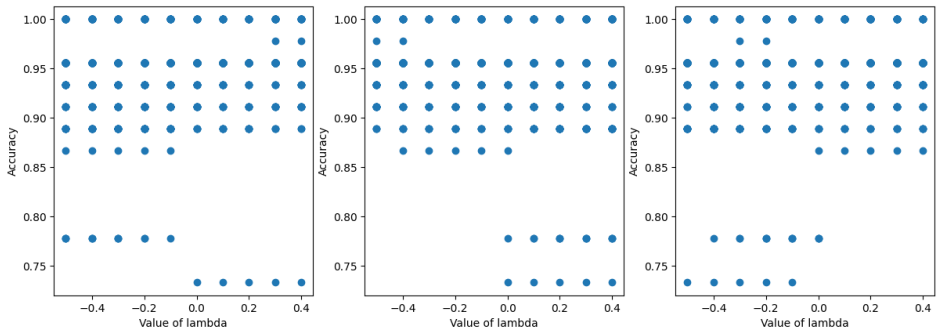
3 label classification

3 label classification use all of feature and lable in iris dataset. By using 'train-test-split (x, y, test-size=0.3, random-state=43)', the iris dataset is classified into 0.7 trainsets and 0.3 testsets. Since there are three decision boundaries, the weight also appears as three vectors. Based threshold=[-0.5, -0.5, -0.4], the train accuracy is 0.9428571428571428 and the test accuracy was 1. It can be checked through the test-perceptron-train, test-perceptron function in the perceptron.py.

result weight

$$[-0.5, 38.7, 76.2, -100.2, -51.9]$$
$$[-0.5, 43., 33.3, -30.5, -92.6]$$
$$[-0.4, -81.7, -109.5, 130.7, 144.5]$$

When sweaping threshold, the following accuracy graph is shown. Graphs can be generated through find-threshold-perceptron functions in the utils.py. It was difficult to judge the overall shape of the graph. Since there are three decision boundaries, it is difficult to express them in a two-dimensional picture, which seems to be the reason for choosing such a graph, but it can be seen that a number of points achieving accuracy 1 are found for each threshold.



REPORT6

1. Both my code and scikit-learn use CountVecorizer in the preprocess stage of corpus.
2. In scikit-learn, tf-idf can be used to avoid bias from differences in length of the document, and to give the word more subjective predictability to improve accuracy model performance

3. My code is the same as scikit-learn using multinomial naive bays classifier. In scikit-learn library, default value of smoothing parameter on multinomial naive bayes classifier functions is 1 = Laplace smoothing.

4. Scikit-learn use scaler that learned from test-set with 'fit-transform' for preprocessing test data with 'transform' the result of my NBcode and scikit-learn are same : ham
sklearn-NB-result() in naivebayes.py.

REPORT7

The default solver of LogisticRegression in scikit-learn library is "lbfgs"

lbfgs is an optimization algorithm that approximately calculates the Hessian matrix of a function from the slope vector and previously calculated values, and uses it to find the value of the variable to be updated next. In addition, scikit-learn can use 'sag' or stochastic average gredient ascents as a solver. A solver similar to (stochastic) gradient ascents is 'sag', and there is a slight difference between (stochastic) gradient ascents and the basic solver, lbfgs. In addition, 'newton-cg' and , 'saga', 'newton-cholesky' also can be used in situations where there is no regulation.

Both accuracy of scikit-learn Logistic Regression with 'lbfgs' and 'sag are 1 in binary classifier case. However, accuracy of my code are 1 and 0.9666666666666667 that is for gradient ascent and stochastic gradient ascent. Due to differences in the process of selecting an object to learn, the accuracy of the stochastic gradient is different from that of the scikit-learn. Funcion sklearn-result with binart parameter True in util.py is code for test accuracy of scikit-learn library.

In multinomial case, also both accuracy of scikit-learn Logistic Regression with 'lbfgs' and 'sag are 1. However, result of my code are 1 and 0.6 0.85 in method gradient ascent and stochastic gradient ascent. use can check result in sklearn-result in util.py with binary parameter false.

REPORT8

The default solver of Logistic Regression in scikit-learn library is also "lbfgs" Accuracy of scikit-learn logistic regression with L2 regularizer is 1. And my result is also 1 in binary classifier.

In multinomial case, accuracy of scikit-learn Logistic Regression with L2 regularizer is 0.9777777777777777. and result of my code is 1. In this case my accuaray is better than scikit-learn. However, we have very little amount of data. So, result can be biased. Futhermore, It can be seen that the selection of hyperparameters influenced it.

REPORT9

Accuracy of scikit-learn Perceptron library is 1. And my result is also 1 in binary classifier.

In multinomial case, accuracy of scikit-learn Perceptron library is 0.9777777777777777. But my result is 1. It can be said that my code showed higher accuracy than scikit-learn because I can directly selected the hyperparameter suitable for the given situation.

...