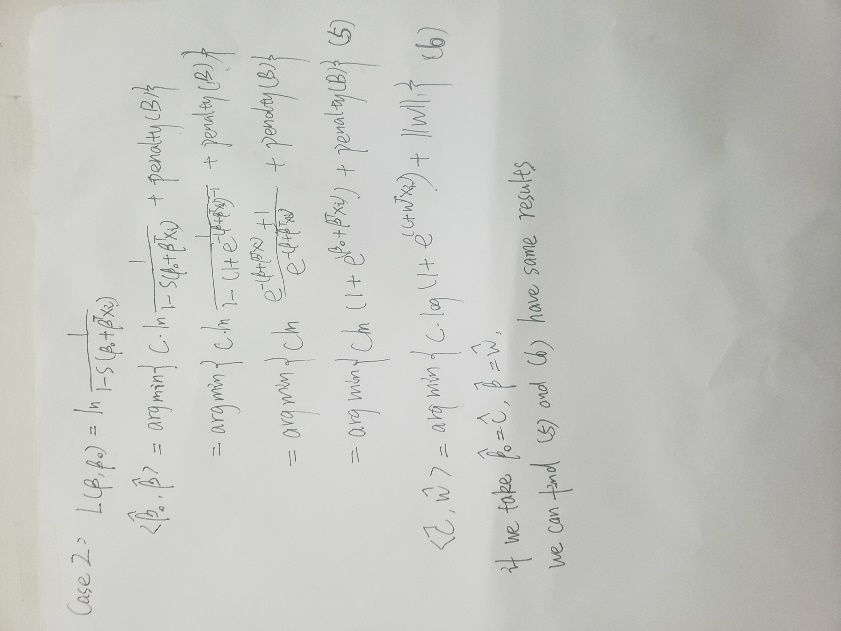
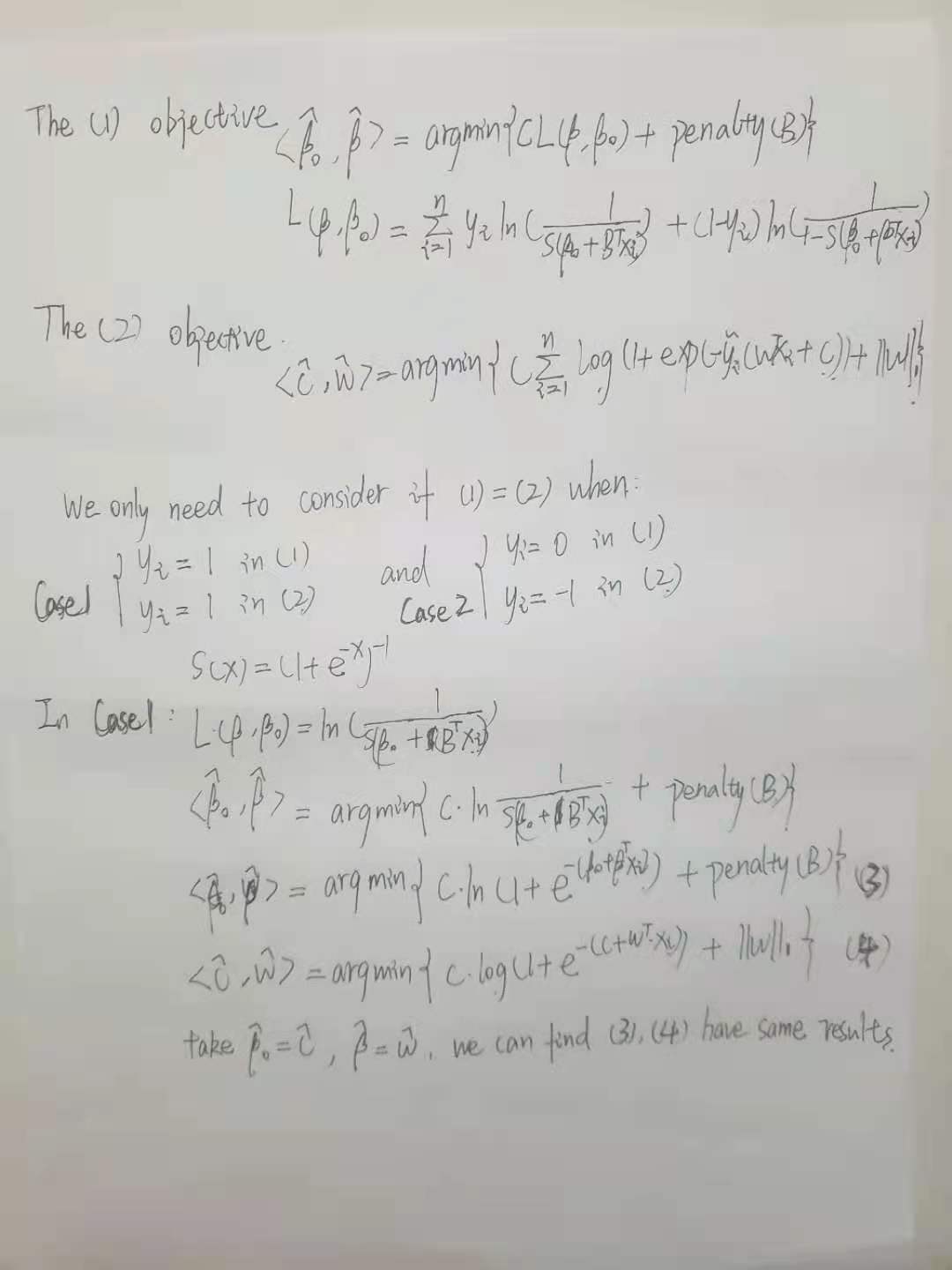
COMP9417 Homework2

Zid: z5239803

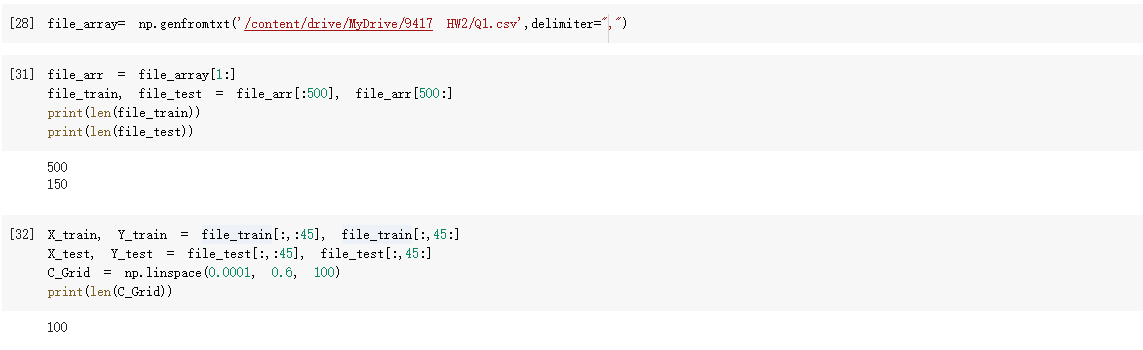
Name: Zhengyang Jia

**Q1**:

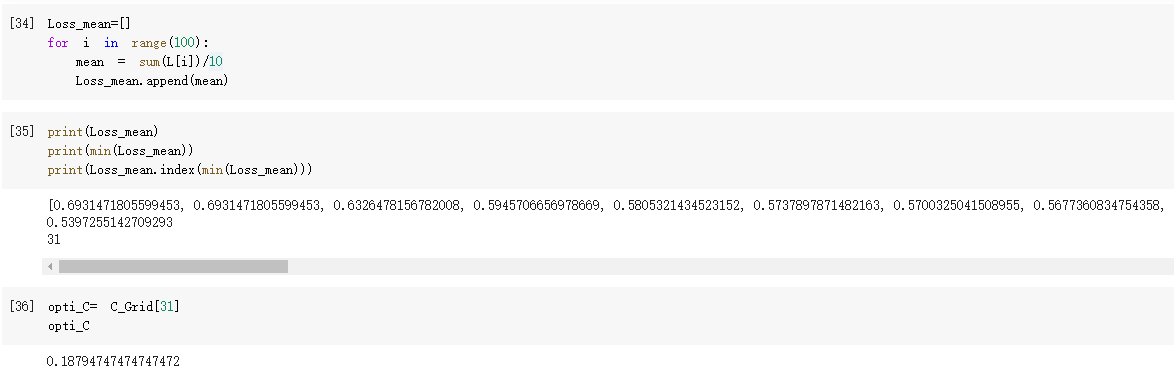
(a)



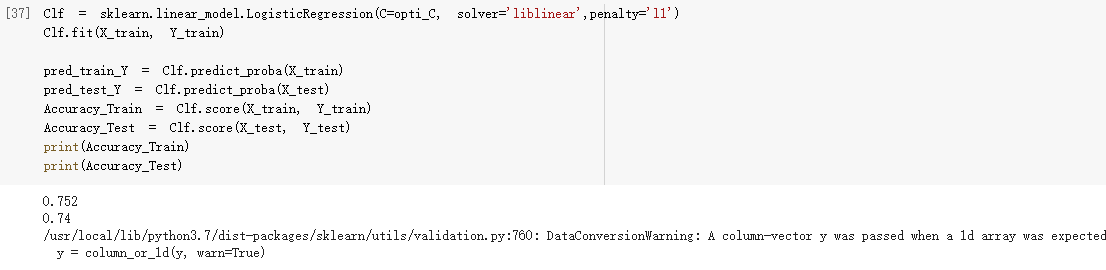
**(b)**





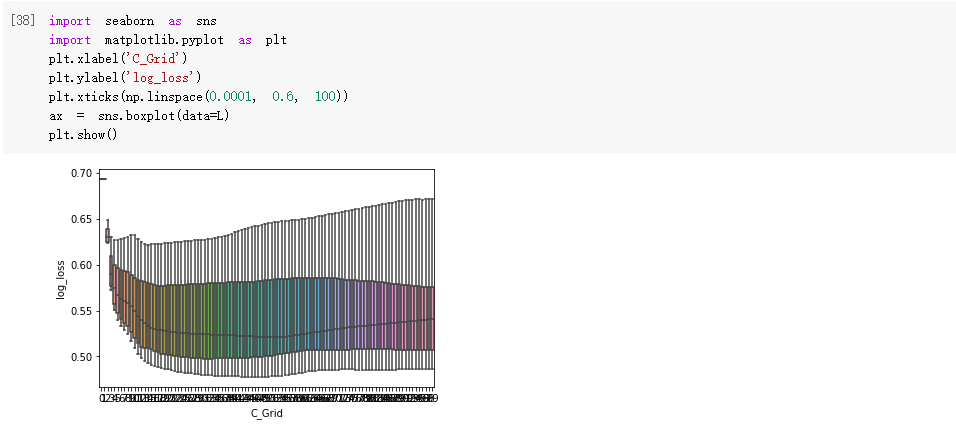


We found that when the index of C\_grid is 31 (32th element), the mean of log\_loss is minimum, which is around 0.539725. The corresponding C value is 0.18794747474747472.



The Train\_Accuracy is 0.752, the Test\_Accuracy is 0.74

The boxplot looks like this:



**(c)**

Two reasons for different results:

**Reason-1:** In 1(b) we used log\_loss function to evaluate, while here in 1(c), we use different scoring method, the default value of the GridSearchCV parameter scoring = None, which makes the result different to 1(b).

**Reason-2:** The data segmentation methods. In 1(b) we segmented the 500-row train\_set into 10 parts sequentially, while the method GridSearchCV here have different rules, it segments train\_set due to the label-y. It tries to make the data with label y=0 and y=1 have same proportion in each validation. Thus, it can’t have same segmentation method with1(b), which leads to the difference.



After changing some parameters:

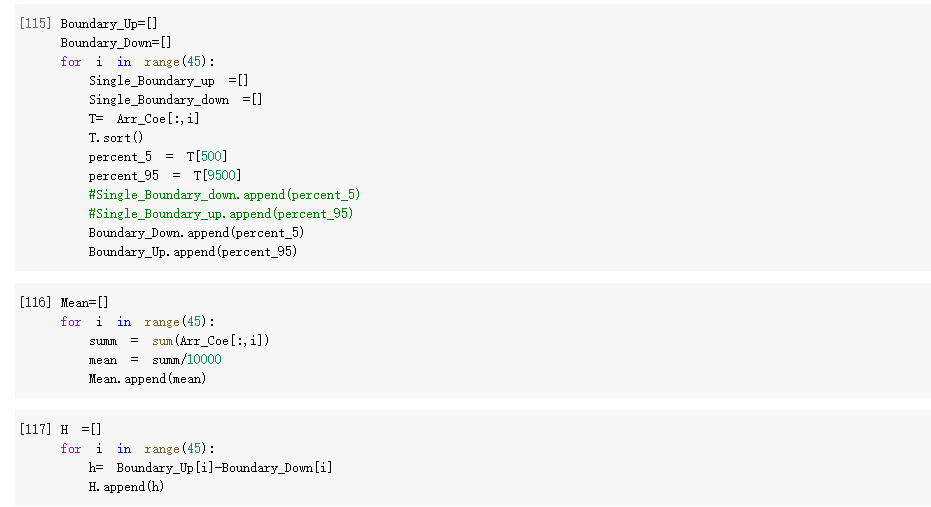
cv= KFold(n\_splits=10), param\_grid= {“C”: C\_grid}, scoring=”neg\_log\_loss. We can get the same C value with 1(b)



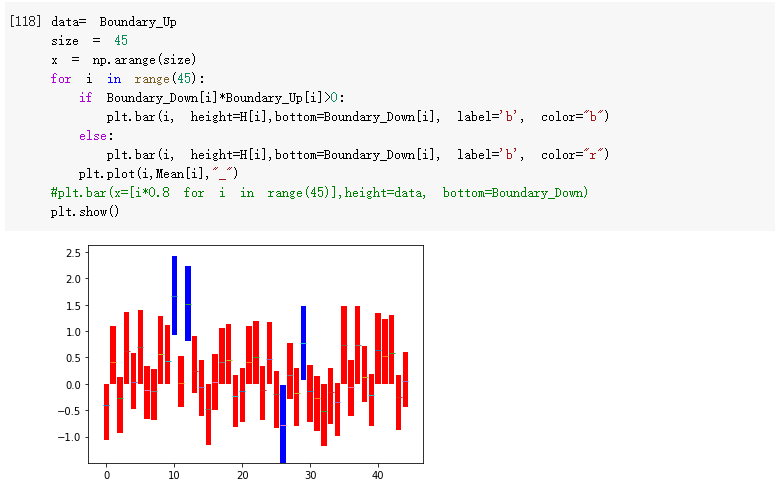
**(d)**

There are 10000 iteration, and in each iteration, we emerge 500 random row to train the model. Then, we get the vector coeffient, there are 10000 these vector in total, we need to calculate coef values of the position 5% and 95% of each sorted feature, along with the the mean of each coef.





From the below figure, we can find that the , , , features are painted blue, which means these four features are more significant to predict y than other ‘painted red’ features.



**(e)**

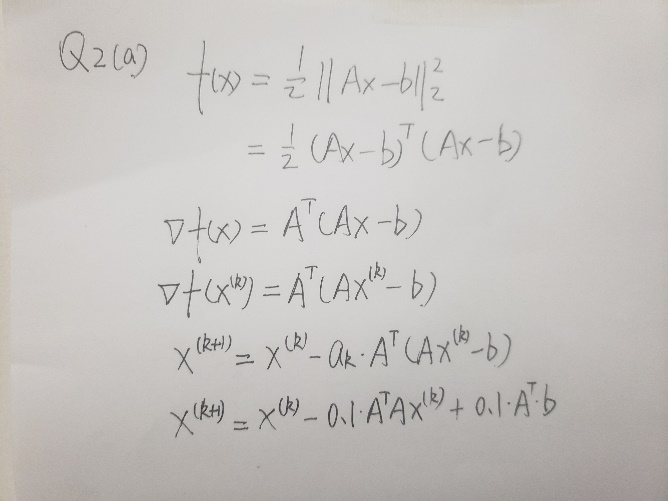
The red bar features have limited effect on prediction, they are less important. The blue bar features have considerate effect on prediction, so they are more important. If we have too many red bars, we can gradually decrease value C to decrease the punishment. If we have too many blue bars, we can gradually increase value C to increase punishment.

If regularization is necessary depends on whether we have too many red bars. If we have too many red bars(0 in the range), regularization is necessary, else regularization is not necessary.

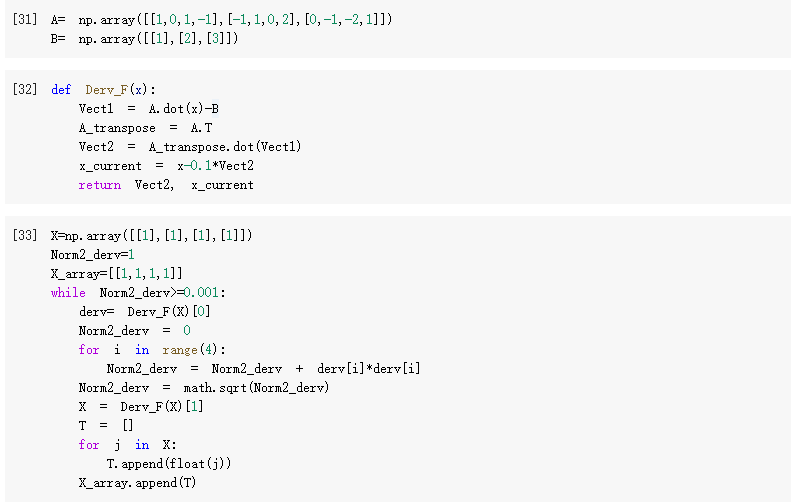
**Q2**:

**(a)**

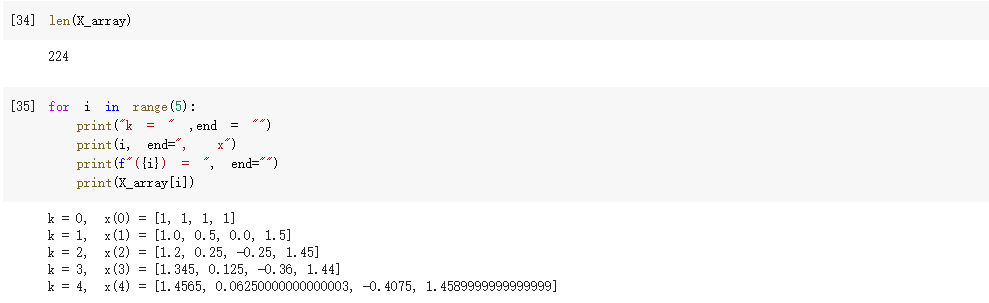
For the gradient step(3), the derivation I made is like this:

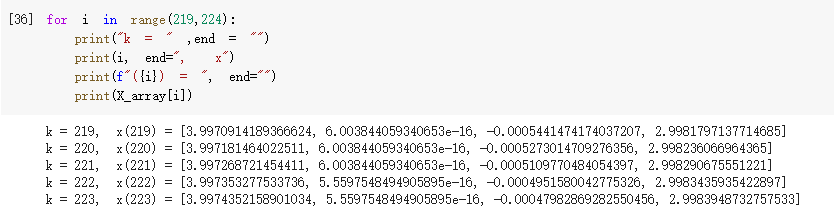


After getting the derivative of f(x), I used vect2 to represent the f(x) derivative. When , we end loop.



We now know that X\_array has 224 elements, the last X\_array index is 223

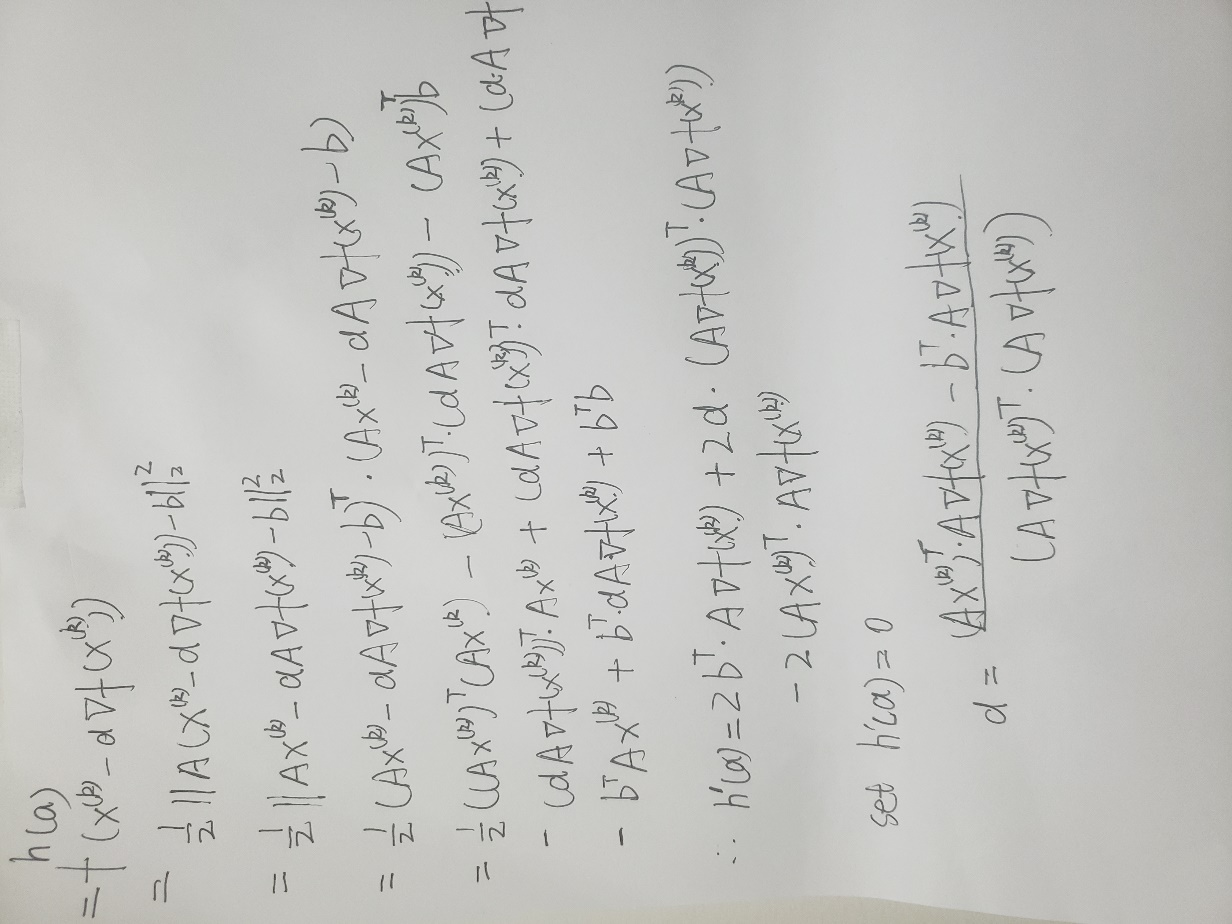


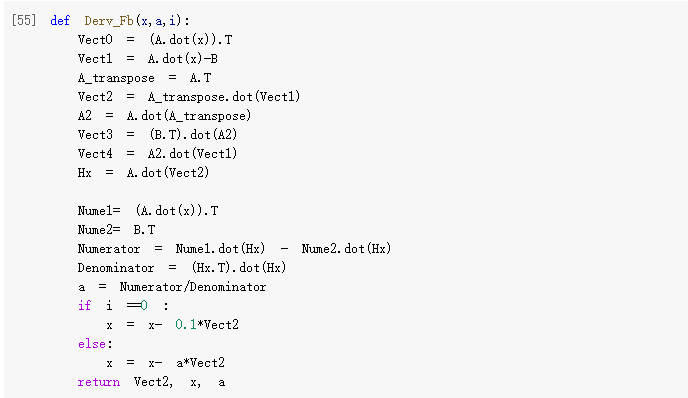


**(b)**

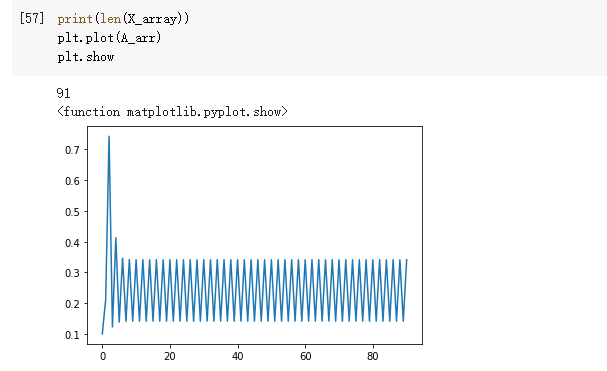
the choice of in 2(b) differs 2(a).

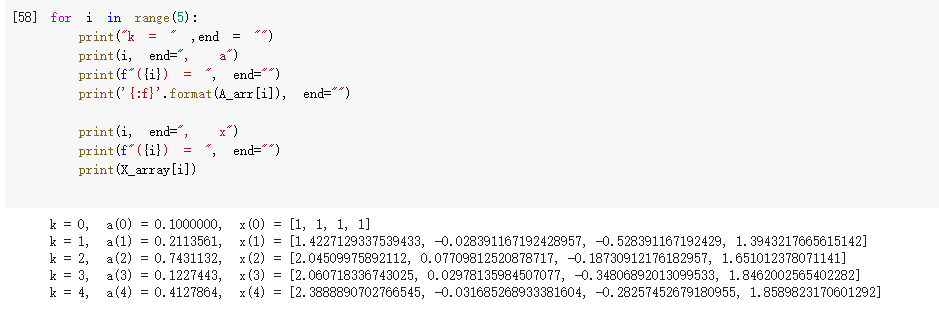
The deduction of is shown here:

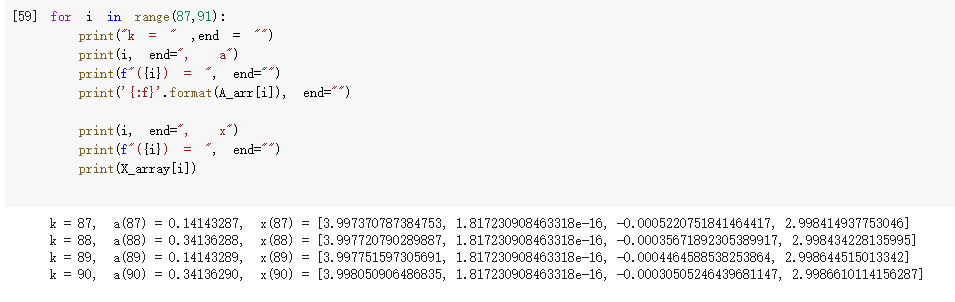












**(c)**

**Gradient Descent**: focuses descent in the negative gradient direction.

**Steepest Gradient Descent**: The directional of steepest descent (or ascent) is the direction amongst all nearby directions that lowers or raises the value of f the most.

The direction of gradient descent method is negative gradient, while the direction of steepest descent method is the direction:



In 2(b), we used less steps to converge the loss, which indicates steepest gradient in such case is better than gradient descent.

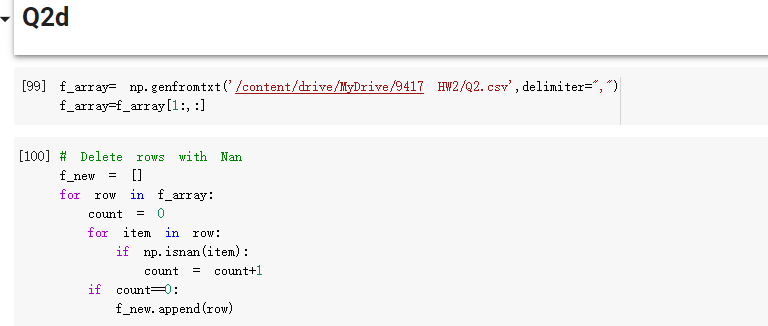
Steepest gradient can converge faster than gradient descent. Besides steepest gradient is easier to find appropriate stepsize.

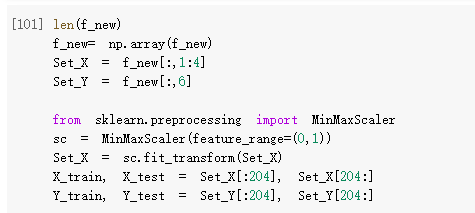
Why it is a reasonable condition to terminate:

When the derivative of f(x) is very close to 0, we assume we got the appropriate alpha, we also don’t want over fitting. If the threshold is too small, we will have overfitting problems.

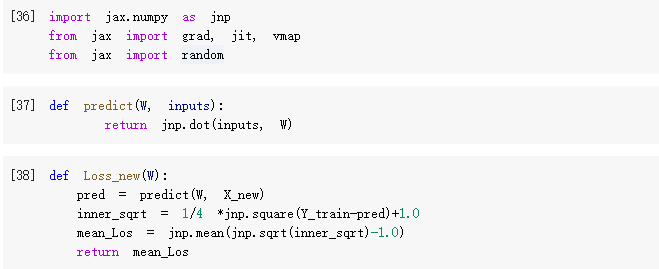
so setting is reasonable.

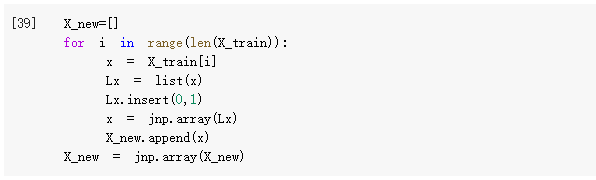
**(d)**

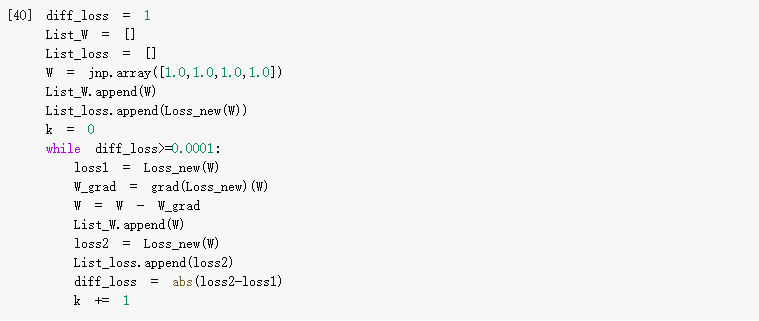


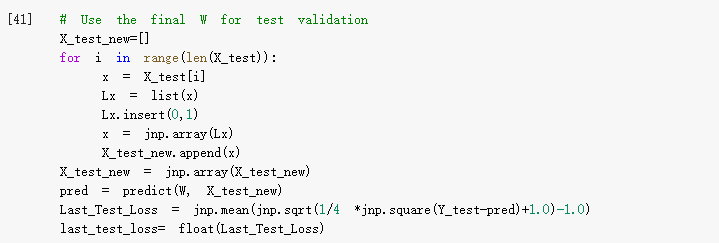


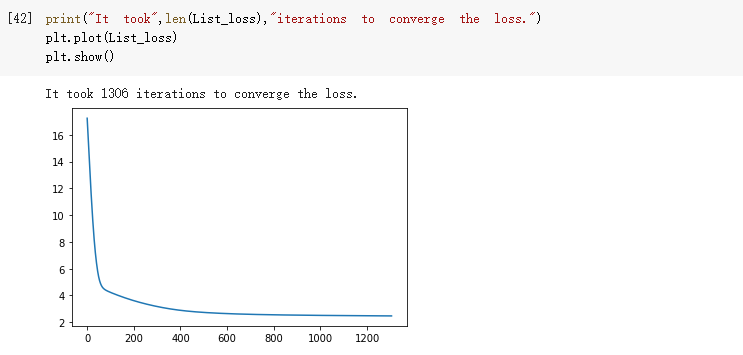
**(e)**

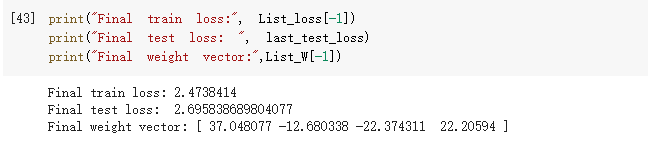






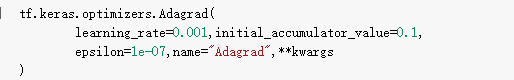


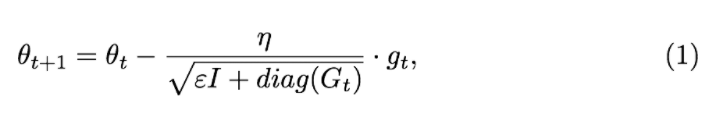


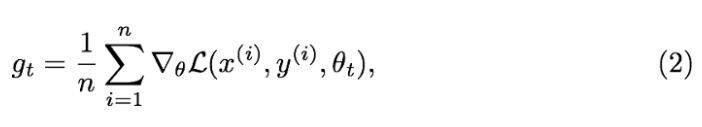


**(g)**

Adagrad is an optimizer with parameter-specific learning rates, which are adapted relative to how frequently a parameter gets updated during training. The more updates a parameter receives, the smaller the updates.





is the parameter to be updated, is the initial learning rate, is some small quantity that used to avoid the division of zero, is the identity matrix, is the gradient estimate in time-step that we can get with the equation



**Feature:**

While the is small, the regularizer is large, so it could amplify the gradient. While the is larger, the regularizer is smaller so it could constrain the gradient.

This Adagrad method suits for dealing with sparse gradients

**Drawbacks:**

In the later period of training, the speed of decreasing gradient gets faster, which could leads to an early end of training.

learning rate can not be set large. If it’s large, regularizer is very sensitive and affects the precise regulation of gradient.