

**The Experiment Report of**

***Machine Learning***

**College Software College**

**Subject Software Engineering**

**Members Ana Madeleyn Oporto Guzmán**

**Student ID 201722800070**

**E-mail madeleyn.opg@gmail.com**

**Tutor Prof. Mingkui Tan \_\_\_**

**Date submitted 2017.12 .07**  \_\_\_\_\_\_

**1. Topic: Linear Regression and Linear Classification**

**2. Time: 2017-12-07**

**3. Reporter: Ana Madeleyn Oporto Guzmán**

**4. Purposes:**

The main purpose is to realize the process of optimization and adjusting parameters and further understand of liner regression, linear classification and gradient descent.

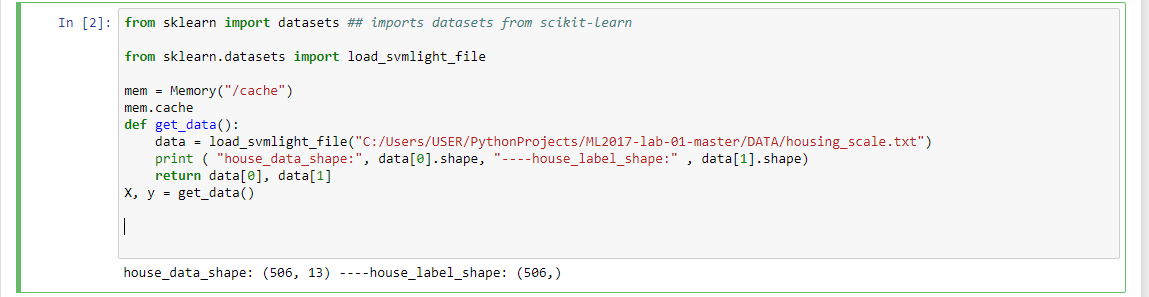
**5. Data sets and data analysis:**

For the first part of the experiment the dataset is **Housing\_Scale** from LIBSVM Data, including 506 samples and each sample has 13 features. The data analysis is **Linear Regression**.

For the second part the dataset is **Australian** from LIBSVM Data, including 690 samples and each sample has 14 features. The data analysis is **Linear Classification**

**6. Experimental steps:**

1. **Linear Regression – Part A**
2. Load the experiment data. You can use [load\_svmlight\_file](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_svmlight_file.html" \t "_blank) function in sklearn library.



from sklearn import datasets ## imports datasets from scikit-learn

from sklearn.datasets import load\_svmlight\_file

mem = Memory("/cache1")

mem.cache

def get\_data():

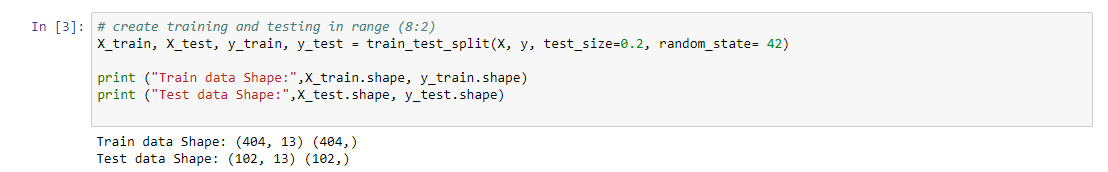
data = load\_svmlight\_file("C:/Users/USER/PythonProjects/ML2017-lab-01-master/DATA/housing\_scale.txt")

print ( "house\_data\_shape:", data[0].shape, "----house\_label\_shape:" , data[1].shape)

return data[0], data[1]

X, y = get\_data()

1. Devide dataset. You should divide dataset into training set and validation set using [train\_test\_split](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html" \t "_blank) function. Test set is not required in this experiment.



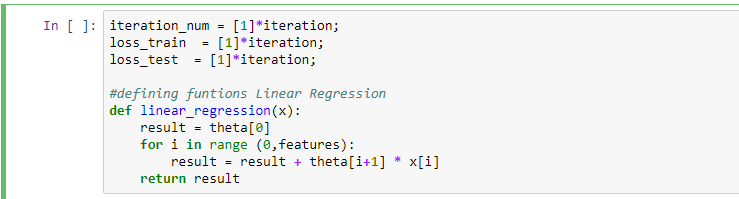
# create training and testing in range (8:2)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state= 42)

print ("Train data Shape:",X\_train.shape, y\_train.shape)

print ("Test data Shape:",X\_test.shape, y\_test.shape)

1. Initialize linear model parameters. You can choose to set all parameter into zero, initialize it randomly or with normal distribution.



iteration = 100

m = 506

m\_train = 404

m\_test = 102

features=13

theta=[0,0,0,0,0, 0,0,0,0,0, 0,0,0,0]

alpha = 0.001

iteration\_num = [1]\*iteration;

loss\_train = [1]\*iteration;

loss\_test = [1]\*iteration;

**#defining functions Linear Regression**

def linear\_regression(x):

result = theta[0]

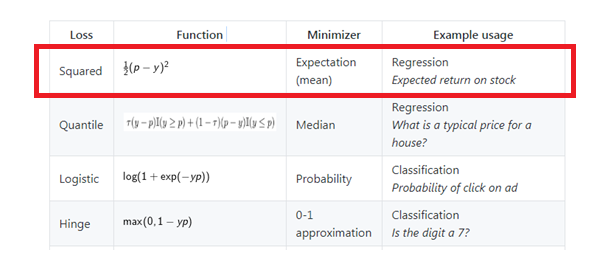
for i in range (0,features):

result = result + theta[i+1] \* x[i]

return result

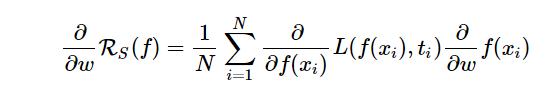
1. Choose loss function and derivation: Find more detail in PPT.

For this experiment least Squared I used as Loss Function.



Derivatives:

When both the loss function and the model are differentiable, it is possible to calculate the derivative of the empirical risk with respect to the model parameters ω:



1. Calculate gradient G toward loss function from all samples.

**#defining functions Loss**

def loss(m,X,y):

sum=0

for i in range(0,m):

sum = sum + ( linear\_regression(X[i]) - y[i] ) \*\*2

sum = sum / (2\*m)

return sum

**#defining functions Gradient**

def derivative(j,m,X,y):

sum=0

if(j==0):

for i in range(0,m):

sum = sum + ( linear\_regression(X[i]) - y[i] )

else:

for i in range(0,m):

sum = sum + (linear\_regression(X[i]) - y[i] ) \* X[i][j-1]

sum = sum / m

return sum

**def train():**

for i in range(0,iteration):

for j in range(0,features+1):

theta[j] = theta[j] - alpha \*derivative(j,m\_train,X\_train,y\_train)

iteration\_num[i] = i;

loss\_train[i] = loss(m\_train,X\_train,y\_train);

loss\_test[i] = loss(m\_test,X\_test,y\_test);

1. Denote the opposite direction of gradient  G as D .
2. Update model:  .  Is learning rate, a hyper-parameter that we can adjust.
3. Get the loss *Ltrain* under the training set and *Lvalidation* by validating under validation set.
4. Repeat step 5 to 8 for several times, and **drawing graph** *Ltrain* **of  as well as***Lvalidation***with the number of iterations**.

%matplotlib inline

import matplotlib.pyplot as plt

plt.figure(figsize=(16,8))

plt.title("Train and Validation Loss with Linear Model ")

plt.plot(iteration\_num, loss\_train,color = 'Red', label='Train Lost')

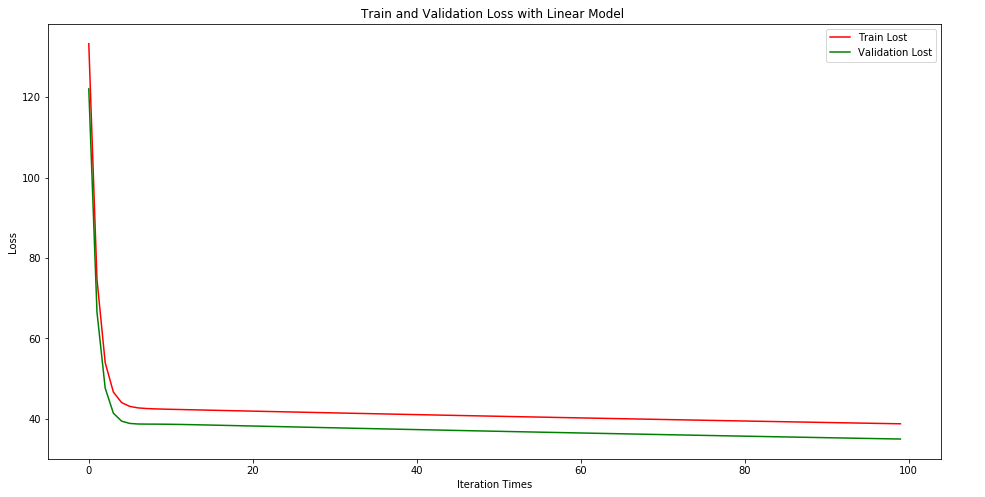
plt.plot(iteration\_num, loss\_test, color = 'Green', label='Validation Lost')

plt.legend(loc='upper right')

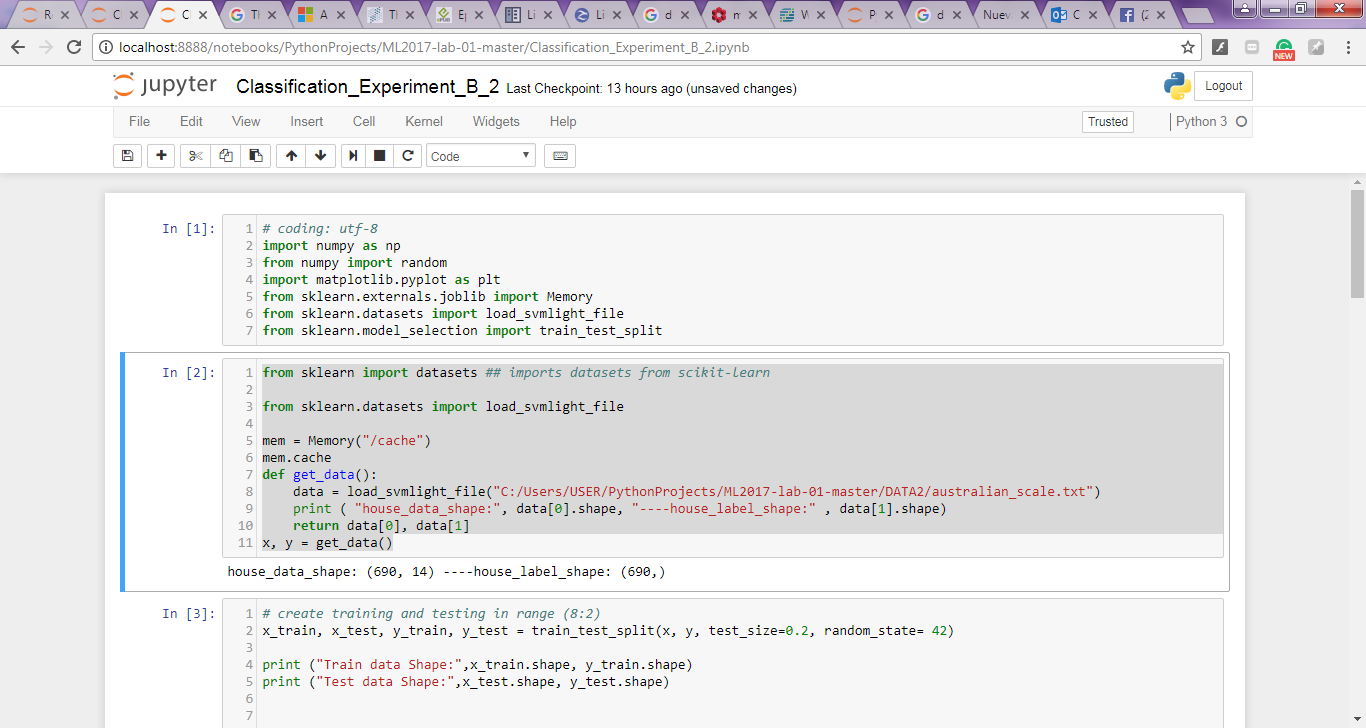
plt.xlabel('Iteration Times')

plt.ylabel('Loss')

plt.show()

****

1. **Linear Classification – Part B**
2. Load the experiment data.



from sklearn import datasets ## imports datasets from scikit-learn

from sklearn.datasets import load\_svmlight\_file

mem = Memory("/cache")

mem.cache

def get\_data():

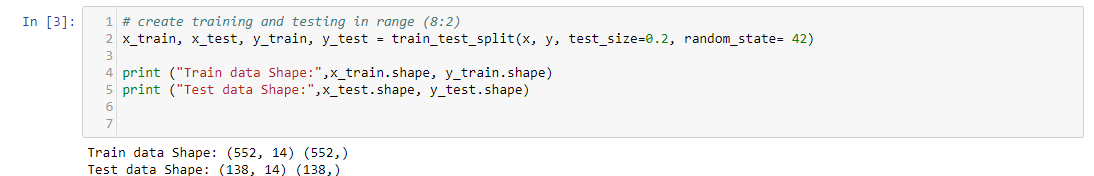
data = load\_svmlight\_file("C:/Users/USER/PythonProjects/ML2017-lab-01-master/DATA2/australian\_scale.txt")

print ( "house\_data\_shape:", data[0].shape, "----house\_label\_shape:" , data[1].shape)

return data[0], data[1]

x, y = get\_data()

1. Divide dataset into training set and validation set.



1. Initialize SVM model parameters. You can choose to set all parameter into zero, initialize it randomly or with normal distribution.
2. Choose loss function and derivation: Find more detail in PPT.
3. Calculate gradient  toward loss function from all samples.
4. Denote the opposite direction of gradient  as .
5. Update model: .  is learning rate, a hyper-parameter that we can adjust.
6. **Select the appropriate threshold, mark the sample whose predict scores greater than the threshold as positive, on the contrary as negative.** Get the loss  under the trainin set and  by validating under validation set.
7. Repeate step 5 to 8 for several times, and **drawing graph of  as well as  with the number of iterations**.

**7. Code:**

(Fill in the contents of 8-12 respectively for linear regression and linear classification)

**8. Selection of validation (hold-out, cross-validation, k-folds cross-validation, etc.):**

1. **Linear Regression:**

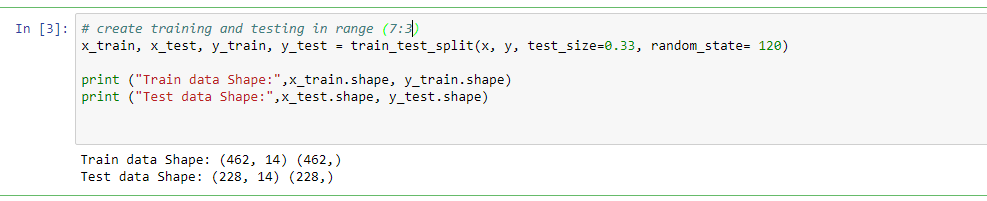
**# create training and testing in range (8:2)**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state= 42)

print ("Train data Shape:",X\_train.shape, y\_train.shape)

print ("Test data Shape:",X\_test.shape, y\_test.shape)

1. **Linear Classification:**

****

# create training and testing in range (7:3)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.33, random\_state= 120)

print ("Train data Shape:",x\_train.shape, y\_train.shape)

print ("Test data Shape:",x\_test.shape, y\_test.shape)

**9. The initialization method of model parameters:**

1. **Linear Regression:**

iteration = 100

m = 506

m\_train = 404

m\_test = 102

features=13

theta=[0,0,0,0,0, 0,0,0,0,0, 0,0,0,0]

alpha = 0.001

iteration\_num = [1]\*iteration;

loss\_train = [1]\*iteration;

loss\_test = [1]\*iteration;

**#defining functions Linear Regression**

def linear\_regression(x):

result = theta[0]

for i in range (0,features):

result = result + theta[i+1] \* x[i]

return result

1. **Linear Classification:**

W = random.random(size=(D, C))

Iterations=100

th = 0

eta = 0.001

L\_train=[];

L\_test=[];

for t in range(Iterations):

y\_train\_pred = np.dot(x\_train,W)

y\_train\_pred[y\_train\_pred> th] = 1

y\_train\_pred[y\_train\_pred<=th] = 0

y\_test\_pred = np.dot(x\_test,W)

y\_test\_pred[y\_test\_pred> th] = 1

y\_test\_pred[y\_test\_pred<=th] = 0

**10. The selected loss function and its derivatives:**

1. **Linear Regression:**

**#defining functions Loss**

def loss(m,X,y):

sum=0

for i in range(0,m):

sum = sum + ( linear\_regression(X[i]) - y[i] ) \*\*2

sum = sum / (2\*m)

return sum

**#defining functions Gradient**

def derivative(j,m,X,y):

sum=0

if(j==0):

for i in range(0,m):

sum = sum + ( linear\_regression(X[i]) - y[i] )

else:

for i in range(0,m):

sum = sum + (linear\_regression(X[i]) - y[i] ) \* X[i][j-1]

sum = sum / m

return sum

1. **Linear Classification:**

#definition function

def svm(W, xtrain, ytrain, xtest, ytest, reg):

gW = np.zeros(W.shape)

num\_classes = W.shape[1]

train\_loss = 0

scores\_train = xtrain.dot(W)

num\_train = xtrain.shape[0]

scores\_train\_correct = scores\_train[np.arange(num\_train), ytrain]

scores\_train\_correct = np.reshape(scores\_train\_correct, (num\_train, 1))

margins\_train = scores\_train - scores\_train\_correct + 1.0

margins\_train[np.arange(num\_train), ytrain] = 0.0

margins\_train[margins\_train <= 0] = 0.0

train\_loss += np.sum(margins\_train) / num\_train

train\_loss += 0.5 \* reg \* np.sum(W \* W)

margins\_train[margins\_train > 0] = 1.0

row\_sum = np.sum(margins\_train, axis=1)

margins\_train[np.arange(num\_train), ytrain] = -row\_sum

gW += np.dot(xtrain.T, margins\_train)/num\_train + reg \* W

test\_loss = 0

scores\_test = xtest.dot(W)

num\_test = xtest.shape[0]

scores\_test\_correct = scores\_test[np.arange(num\_test), ytest]

scores\_test\_correct = np.reshape(scores\_test\_correct, (num\_test, 1))

margins\_test = scores\_test - scores\_test\_correct + 1.0

margins\_test[np.arange(num\_test), ytest] = 0.0

margins\_test[margins\_test <= 0] = 0.0

test\_loss += np.sum(margins\_test) / num\_test

test\_loss += 0.5 \* reg \* np.sum(W \* W)

return train\_loss, test\_loss, gW

**11. Experimental results and curve:**

## Hyper-parameter selection (η, epoch, etc.):

1. **Linear Regression**

Epoch = 100 and 500

1. **Linear Classification**

Epoch = 100 and 500

## Assessment Results (based on selected validation):

**Linear regression** the dataset used is this case is Housing\_Scale and the experiment was running with 100 and 500 iterations with the train data in the range 8:2. And the second one is with **Linear Classification** the dataset is Australian the experiment was running with 100 and 500 iterations but in this case I used the train data in the range 8:2. The graphs are totally different when you do it many iterations, the result change, the graphs change.

## Predicted Results (Best Results):

## Loss curve:

**Result for the first Experiment part A – LINEAR REGRESSION:**

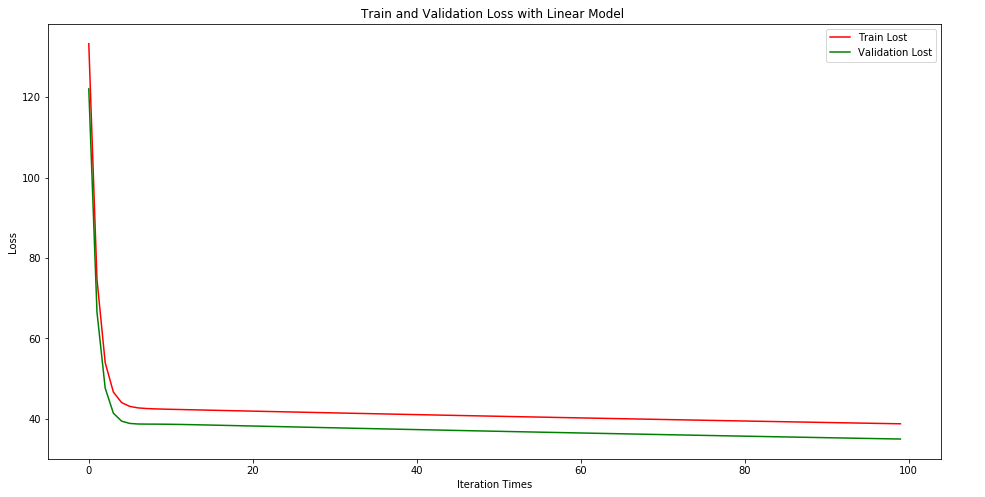
# Training and Testing in range (8:2)

* Dataset Housing\_Scale content 506 examples with 13 features
* Test\_size=0.2

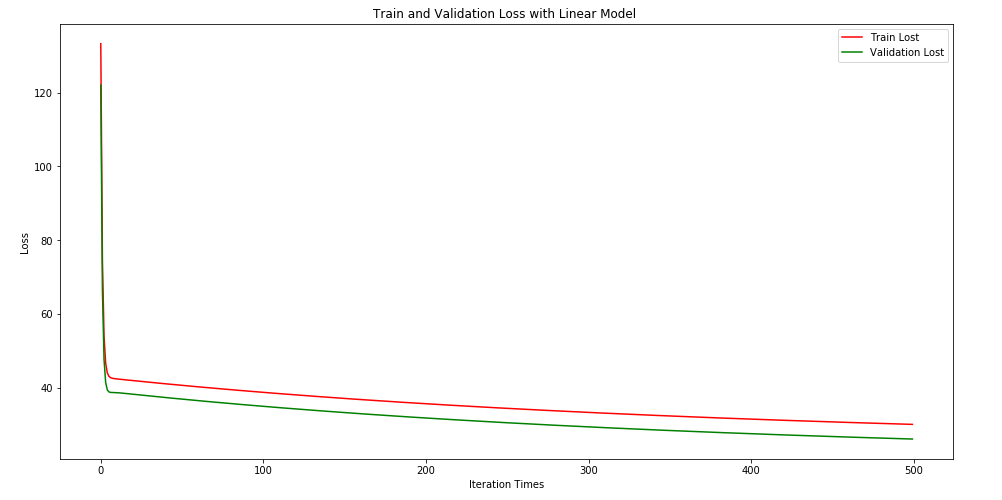
Train data Shape: (404, 13) (404,)

Test data Shape: (102, 13) (102,)

**Graph with 100 iterations:**

****

**Graph with 500 iterations:**

****

**Result for the first Experiment part B – LINEAR CLASSIFICATION:**

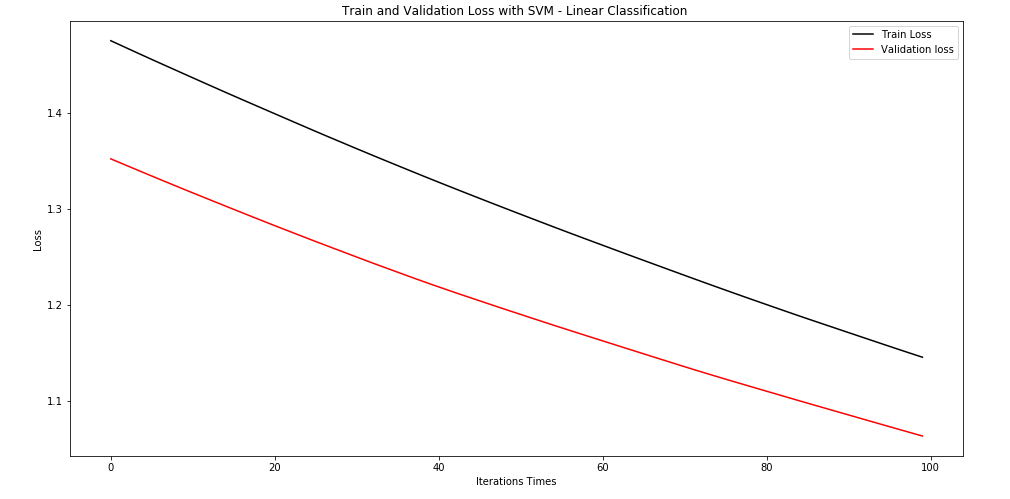
# Training and Testing in range (8:2)

* Dataset **Australian** content 690 examples with 14 features
* Test\_size=0.2

Train data Shape: (552, 14) (552,)

Test data Shape: (138, 14) (138,)

**Graph with 100 iterations:**

****

**Graph with 500 iterations:**

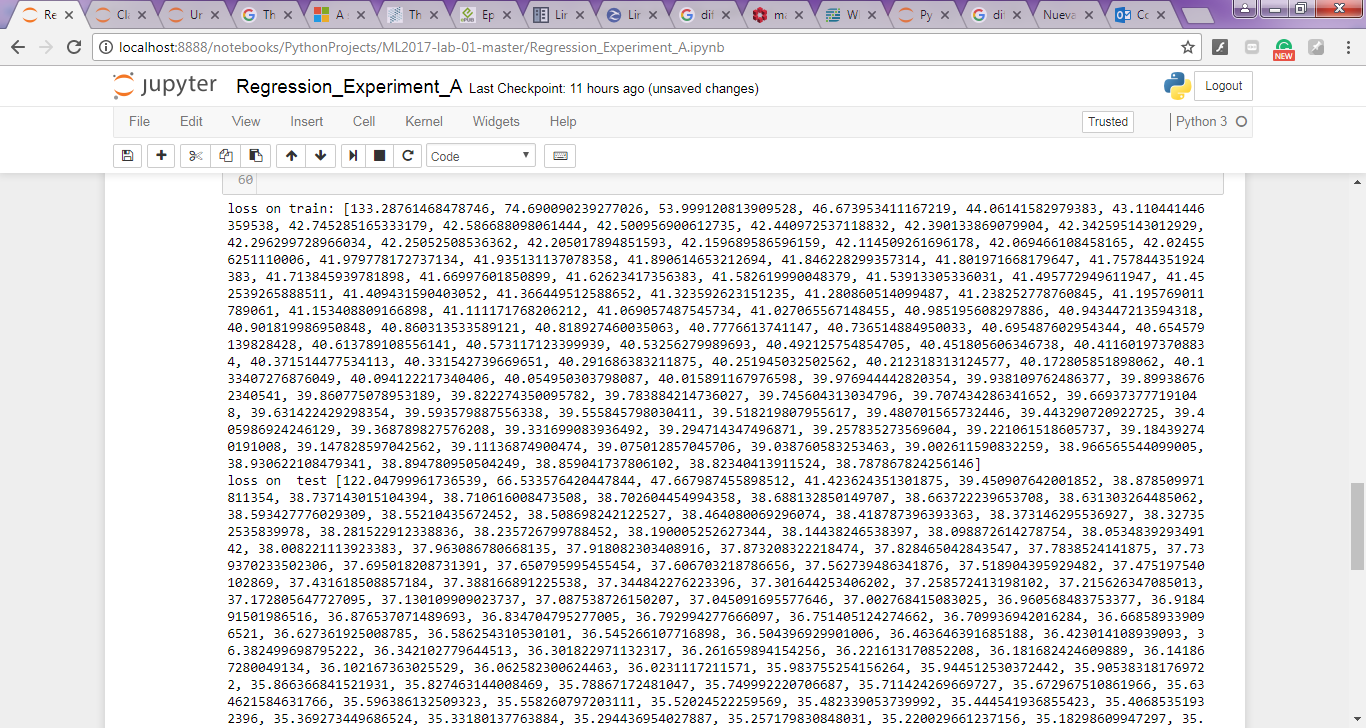
****

**12. Results analysis:**

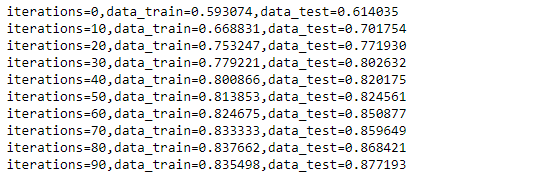
**Result of the Loss Function for Linear Regression**

**loss on train**: [133.28761468478746, 74.690090239277026, 53.999120813909528, 46.673953411167219, 44.06141582979383, 43.110441446359538, 42.745285165333179, 42.586688098061444, 42.500956900612735, 42.440972537118832, 42.390133869079904, 42.342595143012929, 42.296299728966034, 42.25052508536362, 42.205017894851593, 42.159689586596159, 42.114509261696178, 42.069466108458165, 42.024556251110006, 41.979778172737134, 41.935131137078358, 41.890614653212694, 41.846228299357314, 41.801971668179647, 41.757844351924383, 41.713845939781898, 41.66997601850899, 41.62623417356383, 41.582619990048379, 41.53913305336031, 41.495772949611947, 41.452539265888511, 41.409431590403052, 41.366449512588652, 41.323592623151235, 41.280860514099487, 41.238252778760845, 41.195769011789061, 41.153408809166898, 41.111171768206212, 41.069057487545734, 41.027065567148455, 40.985195608297886, 40.943447213594318, 40.901819986950848, 40.860313533589121, 40.818927460035063, 40.7776613741147, 40.736514884950033, 40.695487602954344, 40.654579139828428, 40.613789108556141, 40.573117123399939, 40.53256279989693, 40.492125754854705, 40.451805606346738, 40.411601973708834, 40.371514477534113, 40.331542739669651, 40.291686383211875, 40.251945032502562, 40.212318313124577, 40.172805851898062, 40.133407276876049, 40.094122217340406, 40.054950303798087, 40.015891167976598, 39.976944442820354, 39.938109762486377, 39.899386762340541, 39.860775078953189, 39.822274350095782, 39.783884214736027, 39.745604313034796, 39.707434286341652, 39.669373777191048, 39.631422429298354, 39.593579887556338, 39.555845798030411, 39.518219807955617, 39.480701565732446, 39.443290720922725, 39.405986924246129, 39.368789827576208, 39.331699083936492, 39.294714347496871, 39.257835273569604, 39.221061518605737, 39.184392740191008, 39.147828597042562, 39.11136874900474, 39.075012857045706, 39.038760583253463, 39.002611590832259, 38.966565544099005, 38.930622108479341, 38.894780950504249, 38.859041737806102, 38.82340413911524, 38.787867824256146]

**loss on test** [122.04799961736539, 66.533576420447844, 47.667987455898512, 41.423624351301875, 39.450907642001852, 38.878509971811354, 38.737143015104394, 38.710616008473508, 38.702604454994358, 38.688132850149707, 38.663722239653708, 38.631303264485062, 38.593427776029309, 38.55210435672452, 38.508698242122527, 38.464080069296074, 38.418787396393363, 38.373146295536927, 38.327352535839978, 38.281522912338836, 38.235726799788452, 38.190005252627344, 38.14438246538397, 38.098872614278754, 38.053483929349142, 38.008221113923383, 37.963086780668135, 37.918082303408916, 37.873208322218474, 37.828465042843547, 37.7838524141875, 37.739370233502306, 37.695018208731391, 37.650795995455454, 37.606703218786656, 37.562739486341876, 37.518904395929482, 37.475197540102869, 37.431618508857184, 37.388166891225538, 37.344842276223396, 37.301644253406202, 37.258572413198102, 37.215626347085013, 37.172805647727095, 37.130109909023737, 37.087538726150207, 37.045091695577646, 37.002768415083025, 36.960568483753377, 36.918491501986516, 36.876537071489693, 36.834704795277005, 36.792994277666097, 36.751405124274662, 36.709936942016284, 36.668589339096521, 36.627361925008785, 36.586254310530101, 36.545266107716898, 36.504396929901006, 36.463646391685188, 36.423014108939093, 36.382499698795222, 36.342102779644513, 36.301822971132317, 36.261659894154256, 36.221613170852208, 36.181682424609889, 36.141867280049134, 36.102167363025529, 36.062582300624463, 36.0231117211571, 35.983755254156264, 35.944512530372442, 35.905383181769722, 35.866366841521931, 35.827463144008469, 35.78867172481047, 35.749992220706687, 35.711424269669727, 35.672967510861966, 35.634621584631766, 35.596386132509323, 35.558260797203111, 35.52024522259569, 35.482339053739992, 35.444541936855423, 35.40685351932396, 35.369273449686524, 35.33180137763884, 35.294436954027887, 35.257179830848031, 35.220029661237156, 35.18298609947297, 35.146048800969339, 35.109217422272259, 35.072491621056429, 35.035871056121351, 34.999355387387709]



**Result of the Loss Function for Linear Classification**

****

**13. Similarities and differences between linear regression and linear classification:**

1. Regression: the output variable takes continuous values.

Classification: the output variable takes class labels.

1. Regression involves estimating or predicting a response.

Classification is identifying group membership.

1. Given the following

f : x→y

If Y is discrete/categorical variable, then this is classification problem.

If Y is real number/continuous, then this is a regression problem**.**

1. Regression and classification are both related to prediction, where regression predicts a value from a continuous set, whereas classification predicts the 'belonging' to the class.

For example:

The price of a house depending on the 'size' (in some unit) and say 'location' of the house can be some 'numerical value' (which can be continuous): this relates to regression.

Similarly, the prediction of price can be in words, viz., 'very costly', 'costly', 'affordable', 'cheap', and 'very cheap': this relates to classification.

1. Regression: given a set of data, find the best relationship that represents the set of data.

Classification: given a known relationship, identify the class that the data belongs to.

We can see that regression and classification start from opposing ends: to find a pattern or to find the pattern that it belongs to.

1. Regression and classification can work on some common problems where the response variable is respectively continuous and ordinal.

But the result is what would make us choose between the two.

For example, simple/hard classifiers (e.g. SVM) simply try to put the example in specific class (e.g. whether the project is "profitable" or "not profitable", and doesn't account for how much), where regression can give an exact profit value as some continuous value.

However, in the case of classification, we can consider probabilistic models (e.g. logistic regression), where each class (or label) has some probability, which can be weighted by the cost associated with each label (or class), and thus give us with a final value on basis of which we can decide to put it some label or not. For instance, label AA has a probability of 0.30.3, but the payoff is huge (e.g. 1000). However, label BB has probability 0.70.7, but the payoff is very low (e.g. 1010). So, for maximizing the profit, we might label the example as label AA instead of BB.

1. Regression means to predict the output value using training data.

Classification means to group the output into a class.

For example, we use regression to predict the house price (a real value) from training data and we can use classification to predict the type of tumor (e.g. "benign" or "malign") using training data.

1. It is important to be clear when using terms like regression, classification and prediction to discriminate between the task you are performing and the method used to perform it. A classification task involves taking an input and labelling it as belonging to a given class, so the output is categorical. On the other hand, a prediction task involves predicting a continuous valued output.

Methods for achieving these tasks include regression, in which a continuous valued output is estimated (or, rather, the expected value of a distribution on a continuous variable is estimated, conditional on a given set of input values). This can be used to carry out a prediction task, as you would expect. It can also be used to carry out a classification task, for example using logistic regression to estimate the log odds of the input pattern belonging to a given class. In this case, the task is classification, the method is regression.

Classification methods simply generate a class label rather than estimating a distribution parameter. K nearest neighbour is a good example where the task and the method are both called classification.

|  |  |
| --- | --- |
|  |  |

**14. Summary:**

In this report, you discovered 2 different experiments, 1 of them is with linear regression the dataset used is this case is Housing\_Scale and the experiment was running with 100 and 500 iterations with the train data 8:2. And the second one is with Linear Classification the dataset is Australian the experiment was running with 100 and 500 iterations but in this case I used the train data in 7:3 you can running and put your own range for make some text. The graphs are totally different when you use different iterations like in this case 100 and 500 times, the result changes the graphs changes. I discovered the simple linear regression model and how to train it using gradient descent. I work through the experiment of the update rule for gradient descent. I also learned how to make predictions with a learned linear regression model.