MACHINE LEARNING &
COMPUTATIONAL INTELLIGENCE

Single Layer Perceptron

BY :

NADHIFA SOFIA

(19/448721/PPA/05804)

MASTER OF COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE AND ELECTRONICS

FACULTY OF MATHEMATICS AND NATURAL SCIENCES

UNIVERSITAS GADJAH MADA

2020

In Task 1 of Machine Learning I use a dataset from https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data. In the dataset, the first column is x1, the second column is x2, the third column is x3, the fourth column is x4, and the last column is the labeling of the class. To classify 'Iris-versicolor, I made class '0', while for 'Iris-virginica' I made it a class '1'. I took the first 100 data out of 150 data contained in iris.data

A single layer perceptron (SLP) is a feed-forward network based on a threshold transfer function. SLP is the simplest type of artificial neural networks and can only classify linearly separable cases with a binary target (1, 0).

	A	B	C	D	E
1	x1	x2	x3	x4	type
2	7.0	3.2	4.7	1.4	Iris-versicolor
3	6.4	3.2	4.5	1.5	Iris-versicolor
4	6.9	3.1	4.9	1.5	Iris-versicolor
5	5.5	2.3	4.0	1.3	Iris-versicolor
6	6.5	2.8	4.6	1.5	Iris-versicolor
7	5.7	2.8	4.5	1.3	Iris-versicolor
8	6.3	3.3	4.7	1.6	Iris-versicolor
9	4.9	2.4	3.3	1.0	Iris-versicolor
10	6.6	2.9	4.6	1.3	Iris-versicolor
11	5.2	2.7	3.9	1.4	Iris-versicolor
12	5.0	2.0	3.5	1.0	Iris-versicolor
13	5.9	3.0	4.2	1.5	Iris-versicolor
14	6.0	2.2	4.0	1.0	Iris-versicolor
15	6.1	2.9	4.7	1.4	Iris-versicolor
16	5.6	2.9	3.6	1.3	Iris-versicolor
17	6.7	3.1	4.4	1.4	Iris-versicolor
18	5.6	3.0	4.5	1.5	Iris-versicolor
19	5.8	2.7	4.1	1.0	Iris-versicolor
20	6.2	2.2	4.5	1.5	Iris-versicolor

Figure 1. Samples of iris.data

K-Fold Cross-Validation is a procedure in Machine Learning that is used to evaluate Machine Learning Models on a limited amount of data. This procedure uses the K parameter which represents the number of groups formed from the data. The steps in this procedure are as follows:

1. Divide the dataset into K number of groups
2. In each group, determine the majority of data as training data and part of it
others as data validation
3. Evaluate the training data model and validation for each group

Sigmoid Function

Epoch = 100

Alpha or learning rate = 0.01

H(x, theta, bias) = theta1*x1 + theta2*x2 + theta3*x3 + theta4*x4 + bias

slp_iris

File Edit View Insert Format Data Tools Add-ons Help Last edit was made yesterday at 10:03 PM by anonymous

=1/(1+EXP(-M2))

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	x1	x2	x3	x4	type	target	prediction	theta1	theta2	theta3	theta4	bias	sigma	sigmoid	error	dt1	dt2	dt3	dt4	db
2	7.0	3.2	4.7	1.4	Iris-versicolor	0	1	0.5	0.5	0.5	0.5	0.5	8.65	0.9998249	0.9996498	0.0024504	0.0011202	0.0016453	0.0004900	0.0003500
3	6.4	3.2	4.5	1.5	Iris-versicolor	0	1	0.4997549	0.4998879	0.4998354	0.4999509	0.4999649	8.297224297	0.9997508	0.9995017	0.0031874	0.0015937	0.0022411	0.0007470	0.0004980
4	6.9	3.1	4.9	1.5	Iris-versicolor	0	1	0.4994362	0.4997286	0.4996113	0.4998762	0.4999151	8.693093698	0.9998322	0.9996646	0.0023136	0.0010394	0.0016430	0.0005029	0.0003353
5	5.5	2.3	4.0	1.3	Iris-versicolor	0	1	0.4992048	0.4996246	0.4994470	0.4998259	0.4998816	7.042206946	0.9991265	0.9982538	0.0095909	0.0040107	0.0069752	0.0022669	0.0017438
6	6.5	2.8	4.6	1.5	Iris-versicolor	0	1	0.4982457	0.4992235	0.4987495	0.4995992	0.4997072	8.17977735	0.9997198	0.9994397	0.0036403	0.0015681	0.0025762	0.0008400	0.0005600
7	5.7	2.8	4.5	1.3	Iris-versicolor	0	1	0.4978817	0.4990667	0.4984918	0.4995152	0.4996512	7.627547312	0.9995133	0.9990270	0.0055420	0.0027224	0.0043752	0.0012639	0.0009722
8	6.3	3.3	4.7	1.6	Iris-versicolor	0	1	0.4973275	0.4987945	0.4980543	0.4993888	0.4995540	8.418616958	0.9997793	0.9995587	0.0027792	0.0014557	0.0020733	0.0007058	0.0004411
9	4.9	2.4	3.3	1.0	Iris-versicolor	0	1	0.4970495	0.4986489	0.4978470	0.4993183	0.4995099	6.274023815	0.9981189	0.9962413	0.0183653	0.0089952	0.0123684	0.0037480	0.0037480
10	6.6	2.9	4.6	1.3	Iris-versicolor	0	1	0.4952130	0.4977494	0.4966101	0.4989434	0.4991351	8.144047887	0.9997096	0.9994193	0.0038307	0.0016831	0.0026698	0.0007545	0.0005804
11	5.2	2.7	3.9	1.4	Iris-versicolor	0	1	0.4948299	0.4975810	0.4963431	0.4988680	0.4990770	7.049815595	0.9991331	0.9982671	0.0089992	0.0046726	0.0067494	0.0024228	0.0017306
12	5.0	2.0	3.5	1.0	Iris-versicolor	0	1	0.4939300	0.4971138	0.4956682	0.4986257	0.4989040	6.196246501	0.9979670	0.9959382	0.0202466	0.0080986	0.0141726	0.0040493	0.0040493
13	5.9	3.0	4.2	1.5	Iris-versicolor	0	1	0.4919053	0.4963039	0.4942509	0.4982208	0.4984990	7.712838058	0.9995531	0.9991064	0.0052681	0.0026787	0.0037501	0.0013393	0.0008929
14	6.0	2.2	4.0	1.0	Iris-versicolor	0	1	0.4913785	0.4960360	0.4938759	0.4980868	0.4984097	7.011551317	0.9990994	0.9981996	0.0107877	0.0039554	0.0071918	0.0017979	0.0017979
15	6.1	2.9	4.7	1.4	Iris-versicolor	0	1	0.4902997	0.4956405	0.4931567	0.4979070	0.4982300	7.941323109	0.9996443	0.9992889	0.0043353	0.0020610	0.0033403	0.0009949	0.0007107
16	5.6	2.9	3.6	1.3	Iris-versicolor	0	1	0.4898662	0.4954344	0.4928227	0.4978075	0.4981589	7.0994816	0.9991751	0.9983509	0.0092231	0.0047762	0.0059291	0.0021410	0.0016469
17	6.7	3.1	4.4	1.4	Iris-versicolor	0	1	0.4889439	0.4949568	0.4922298	0.4975934	0.4979942	8.170726938	0.9997172	0.9994346	0.0037864	0.0017519	0.0024866	0.0007912	0.0005651
18	5.6	3.0	4.5	1.5	Iris-versicolor	0	1	0.4885653	0.4947816	0.4919811	0.4975143	0.4979377	7.67843505	0.9995375	0.9990752	0.0051750	0.0027723	0.0041585	0.0013861	0.0009241
19	5.8	2.7	4.1	1.0	Iris-versicolor	0	1	0.4880478	0.4945043	0.4915653	0.4973757	0.4978453	7.176477908	0.9992362	0.9984730	0.0088462	0.0041180	0.0062533	0.0015252	0.0015252
20	6.2	2.2	4.5	1.5	Iris-versicolor	0	1	0.4871631	0.4940925	0.4909399	0.4972232	0.4976927	7.560172888	0.9994794	0.9989592	0.0064476	0.0022878	0.0046797	0.0015599	0.0010399
21	5.6	2.5	3.9	1.1	Iris-versicolor	0	1	0.4865184	0.4938637	0.4904720	0.4970672	0.4975887	6.916366131	0.9990095	0.9980200	0.0110710	0.0049424	0.0077101	0.0021746	0.0019769
22	5.9	3.2	4.8	1.8	Iris-versicolor	0	1	0.4854113	0.4933695	0.4897009	0.4968497	0.4973910	8.184994685	0.9997212	0.9994426	0.0032871	0.0017828	0.0026742	0.0010028	0.0005571
23	6.1	2.8	4.0	1.3	Iris-versicolor	0	1	0.4850825	0.4931912	0.4894335	0.4967494	0.4973353	7.440783281	0.9994135	0.9988273	0.0071466	0.0032804	0.0046863	0.0015230	0.0011715
24	6.3	2.5	4.9	1.5	Iris-versicolor	0	1	0.4843679	0.4928632	0.4889649	0.4965971	0.4972182	7.921718069	0.9996373	0.9992748	0.0045660	0.0018119	0.0035513	0.0010871	0.0007247
25	6.1	2.8	4.7	1.2	Iris-versicolor	0	1	0.4839113	0.4926820	0.4886097	0.4964884	0.4971457	7.720766615	0.9995566	0.9991135	0.0054037	0.0024804	0.0041635	0.0010630	0.0008858
26	6.4	2.9	4.3	1.3	Iris-versicolor	0	1	0.4833709	0.4924339	0.4881934	0.4963821	0.4970571	7.763218316	0.9995750	0.9991503	0.0054341	0.0024623	0.0036510	0.0011038	0.0008490
27	6.6	3.0	4.4	1.4	Iris-versicolor	0	1	0.4828275	0.4921877	0.4878283	0.4962717	0.4969722	8.001422281	0.9996651	0.9993303	0.0044173	0.0020078	0.0029449	0.0009370	0.0006692

Figure 2. The manual calculations of linear classification with Google Sheet

(source : https://bit.ly/slp_csv)

With trials

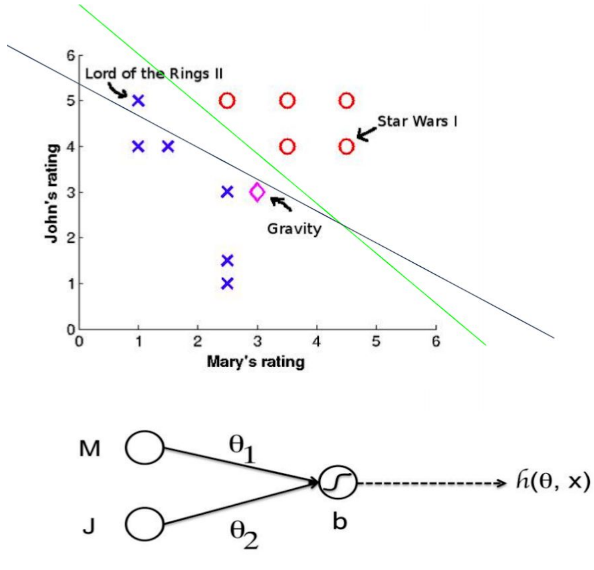
Theta1 = 0.5

Theta2 = 0.5

Theta3 = 0.5

Theta4 = 0.5

Bias = 0.5



$$h(x;\theta,b) = \theta_1 x_1 + \theta_2 x_2 + b$$

$$h(x;\theta,b) = \theta^T x + b$$

$$h(x;\theta,b) = g(\theta^T x + b)$$

$$g(z) = \frac{1}{1+\exp(-z)}$$

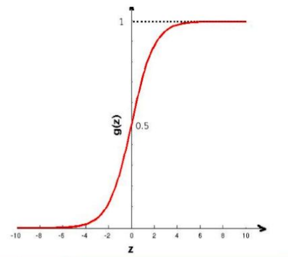


Figure 3. Mathematical formula for sigma and sigmoid

Sigmoid(h) = 1/(1+e^(-(h))) — fx =1/(1+(2.71828182845904)^(-(G3)))

Prediction Class = IF (sigmoid > 0.5, 1, 0) — fx =IF(H3>0.5,1,0)

Error = (Prediction – Sigmoid)^2 — fx =(I3-H3)^2

Delta theta = (2*(sigmoid – fakta))*(1-fakta)*x*sigmoid — fx =(2*(H3-F3))*(1-H3)*A3*H3

Delta bias = (2*(sigmoid – fakta))*(1-fakta)*sigmoid — fx =(2*(H3-F3))*(1-H3)*H3

Theta (baru) = teta – (alpha * delta theta) — fx =P2-(0.8*K3)

Bias (baru) = bias – (alpha * delta bias) — fx =T2-(0.8*O3)

Figure 4. Formula on Google Sheet (source https://bit.ly/slp_csv)

Error Function

Error = (0 - sigmoid)^2 = (0 - 0.9998249038) ^ 2 = 0.9996498383

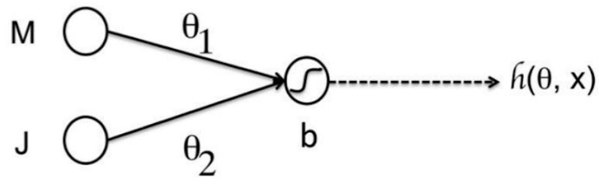
Gradient Descent Algorithm

Error calculation is really crucial to update the weight. I minimize error value by finding the best weight for every input or in a mathematical way is known as "Stochastic Gradient Descent (SGD)".

- new_weight = weight - learning_rate * (dError/dWeight)
- dError/dWeight = (dError/dSigmoid) * (dSigmoid/dSigma) * (dSigma/dWeight)
- dError/dWeight = -2 * (Target-Sigmoid) * Sigmoid * (1-Sigmoid) * Input

- dError/dWeight1 = (-2*(0.99) * 0.99 * (1-0.99) * 7.0 = 0.0024
- dError/dWeight2 = (-2*(0.99) * 0.99 * (1-0.99) * 3.2 = 0.0011
- dError/dWeight3 = (-2*(0.99) * 0.99 * (1-0.99) * 4.7 = 0.0016
- dError/dWeight4 = (-2*(0.99) * 0.99 * (1-0.99) * 1.4 = 0.0004
- dError/dBias = (-2*(0.99) * 0.99 * (1-0.99) = 0.00035

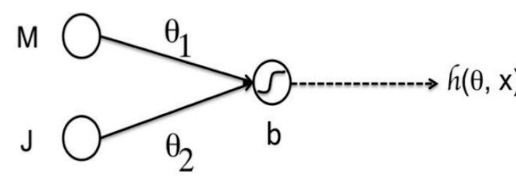
By using SGD, I get new_weight for every input;
- new_weight1 = 0.5 - 0.1 * 0.0024 = 0.4997549512
- new_weight2 = 0.5 - 0.1 * 0.0011 = 0.4998879777
- new_weight3 = 0.5 - 0.1 * 0.0016 = 0.4998354672
- new_weight4 = 0.5 - 0.1 * 0.0004 = 0.4999509902
- bias = 0.5 - 0.1 * 0.00035 = 0.499964993



$$\theta_1 = \theta_1 - \alpha\Delta\theta_1$$
$$\theta_2 = \theta_2 - \alpha\Delta\theta_2$$
$$b = b - \alpha\Delta b$$

$$\begin{aligned}\Delta\theta_1 &= \frac{\partial}{\partial\theta_1}\left(h(x^{(i)};\theta,b) - y^{(i)}\right)^2\\ &= 2\left(h(x^{(i)};\theta,b) - y^{(i)}\right)\frac{\partial}{\partial\theta_1}h(x^{(i)};\theta,b)\\ &= 2\left(g(\theta^T x^{(i)}+b) - y^{(i)}\right)\frac{\partial}{\partial\theta_1}g(\theta^T x^{(i)}+b)\end{aligned}$$

$$\begin{aligned}\Delta\theta_1 &= \frac{\partial}{\partial\theta_1}\left(h(x^{(i)};\theta,b) - y^{(i)}\right)^2\\ &= 2\left(h(x^{(i)};\theta,b) - y^{(i)}\right)\frac{\partial}{\partial\theta_1}h(x^{(i)};\theta,b)\\ &= 2\left(g(\theta^T x^{(i)}+b) - y^{(i)}\right)\frac{\partial}{\partial\theta_1}g(\theta^T x^{(i)}+b)\end{aligned}$$



$$\frac{\partial g}{\partial z} = [1 - g(z)]g(z)$$

$$\begin{aligned}\frac{\partial}{\partial\theta_1}g(\theta^T x^{(i)}+b) &= \frac{\partial g(\theta^T x^{(i)}+b)}{\partial(\theta^T x^{(i)}+b)}\frac{\partial(\theta^T x^{(i)}+b)}{\partial\theta_1}\\ &= \left[1 - g(\theta^T x^{(i)}+b)\right]g(\theta^T x^{(i)}+b)\frac{\partial(\theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + b)}{\partial\theta_1}\\ &= \left[1 - g(\theta^T x^{(i)}+b)\right]g(\theta^T x^{(i)}+b)x_1^{(i)}\end{aligned}$$

$$\begin{aligned}\Delta\theta_1 &= \frac{\partial}{\partial\theta_1}\left(h(x^{(i)};\theta,b) - y^{(i)}\right)^2\\ &= 2\left(h(x^{(i)};\theta,b) - y^{(i)}\right)\frac{\partial}{\partial\theta_1}h(x^{(i)};\theta,b)\\ &= 2\left(g(\theta^T x^{(i)}+b) - y^{(i)}\right)\frac{\partial}{\partial\theta_1}g(\theta^T x^{(i)}+b)\end{aligned}$$

$$\begin{aligned}\Delta\theta_1 &= 2\left[g(\theta^T x^{(i)}+b) - y^{(i)}\right]\left[1 - g(\theta^T x^{(i)}+b)\right]g(\theta^T x^{(i)}+b)x_1^{(i)}\\ \Delta\theta_2 &= 2\left[g(\theta^T x^{(i)}+b) - y^{(i)}\right]\left[1 - g(\theta^T x^{(i)}+b)\right]g(\theta^T x^{(i)}+b)x_2^{(i)}\\ \Delta b &= 2\left[g(\theta^T x^{(i)}+b) - y^{(i)}\right]\left[1 - g(\theta^T x^{(i)}+b)\right]g(\theta^T x^{(i)}+b)\end{aligned}$$

$$\begin{aligned}\frac{\partial}{\partial\theta_1}g(\theta^T x^{(i)}+b) &= \frac{\partial g(\theta^T x^{(i)}+b)}{\partial(\theta^T x^{(i)}+b)}\frac{\partial(\theta^T x^{(i)}+b)}{\partial\theta_1}\\ &= \left[1 - g(\theta^T x^{(i)}+b)\right]g(\theta^T x^{(i)}+b)\frac{\partial(\theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + b)}{\partial\theta_1}\\ &= \left[1 - g(\theta^T x^{(i)}+b)\right]g(\theta^T x^{(i)}+b)x_1^{(i)}\end{aligned}$$

$$\Delta\theta_1 = 2\left[\overbrace{g(\theta^T x^{(i)} + b)}^{\text{prediksi}} - \overbrace{y^{(i)}}^{\text{target}}\right]\left[1 - \overbrace{g(\theta^T x^{(i)} + b)}^{\text{prediksi}}\right]\overbrace{g(\theta^T x^{(i)} + b)}^{\text{prediksi}}x_1^{(i)}$$

$$\Delta\theta_2 = 2\left[g(\theta^T x^{(i)} + b) - y^{(i)}\right]\left[1 - g(\theta^T x^{(i)} + b)\right]g(\theta^T x^{(i)} + b)x_2^{(i)}$$

$$\Delta b = 2\left[g(\theta^T x^{(i)} + b) - y^{(i)}\right]\left[1 - g(\theta^T x^{(i)} + b)\right]g(\theta^T x^{(i)} + b)$$

Figure 5. Mathematical explanation for SGD

Train the Neural Network - SLP

In machine learning, epoch = iteration for training process. I just combine the training data with the same training dataset + use the recent weight and bias that I got before from the previous epoch. The full journey is at https://bit.ly/slp_csv

- Train the Neural Network - SLP algorithm on the training data with learning rate : 0.001 and maximum iteration : 100
- Find theta that minimize the cost function based on Figure 4.
- Test the linear classifier on the test data, and calculate the accuracy
- Plot the cost/error function plot

Implementation

Full version :

https://github.com/dhifaaans/slp_iris/blob/master/linearclassifier_nadhifasofia_KKPMDD.ipynb

Binary-Classification using Linear Classification

Using The part of Iris data (2 classes and two features, choosen by yourself => 100 data).
http://archive.ics.uci.edu/ml/datasets/iris

1. Load the Iris data from scikit learn and divide the data into two parts: training and test data with ration 80:20. Make sure that the class within each parts of the data is balance. (score: 0.5)

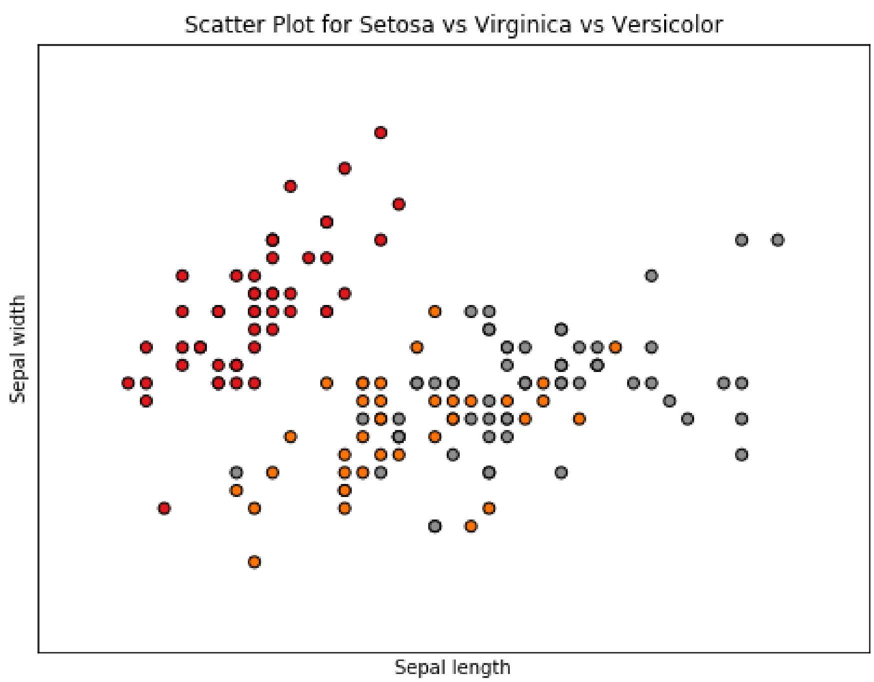
In [2]:
```
#initialize the data
cols = ['x1','x2','x3','x4','class','binary','t1','t2','t3','t4','bia
df_training = pd.read_csv('iris.data', header=None, names=cols)
#to choose 100 first data
df_training.head(100)
df_training = df_training.head(100)

#to generate 2 classes, i chose these
for i in range(100):
    if df_training.at[i,'class']=='Iris-versicolor':
        df_training.at[i,'binary'] = 0
    elif df_training.at[i,'class']=='Iris-virginica':
        df_training.at[i,'binary'] = 1

#divide data into k-fold, 100 epoch
df_k1 = df_training.iloc[0:80]
df_epoch_1 = df_k1
for i in range(0,99):
    df_k1 = df_k1.append(df_epoch_1,ignore_index=True, sort=False)
df_k1_validation = df_training.iloc[80:100]
df_k1_validation = df_k1_validation.reset_index(drop=True)
```

Iris Dataset Visualization

2. Using scatter plot (matplotlib), visualise the Iris data (training part only) (score: 0.5)

In [3]:
```
# import some data to play with
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data[:, :2]  # we only take the first two features.
y = iris.target

x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5

plt.figure(2, figsize=(8, 6))
plt.clf()

# Plot the training points
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Set1,
            edgecolor='k')
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('Scatter Plot for Setosa vs Virginica vs Versicolor')
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.grid(True)
plt.xticks(())
plt.yticks(())
```

Out[3]: ([], <a list of 0 Text yticklabel objects>)



Sigmoid Function

3. Define a python function for sigmoid function (score:0.5)

By way of example, we have data that is saved into our model; you can see it thru https://bit.ly/slp_csv

- input1 * weight1 = 7.0 * 0.5 = 3.5
- input2 * weight2 = 3.2 * 0.5 = 1.6
- input3 * weight3 = 4.7 * 0.5 = 2.35
- input4 * weight4 = 1.4 * 0.5 = 0.7
- bias = 0.5
- sigma(input * weight) + bias = (3.5+1.6+2.35+0.7) + 0.5 = 8.65
- **sigmoid = 1/(1 + (exp(-8.65))) = 0.9998249038**

In [6]:
```
def sigmoid(ha):
    return (1/(1+math.exp(-1*ha)))
```

Error Function

4. Define a python function of the cost function (score:0.5)

Error = (0 - sigmoid)^2 = (0 - 0.9998249038) ^ 2 = 0.9996498383

In [8]:
```
def local_error(a,b):
    return math.fabs((a-b))
```

Gradient Descent Algorithm

5. Define a python function for the Gradient Descent algorithm (score: 0.5)

Error calculation is really crucial to update the weight. I minimize error value by finding the best weight for every input or in mathematical way is known as "Stochastic Gradient Descent (SGD)".

- **new_weight = weight - learning_rate * (dError/dWeight)**
- **dError/dWeight = (dError/dSigmoid) * (dSigmoid/dSigma) * (dSigma/dWeight)**
- **dError/dWeight = -2 * (Target-Sigmoid) * Sigmoid * (1-Sigmoid) * Input**

- dError/dWeight1 = (-2*(0.99) * 0.99 * (1-0.99) * 7.0 = 0.0024
- dError/dWeight2 = (-2*(0.99) * 0.99 * (1-0.99) * 3.2 = 0.0011
- dError/dWeight3 = (-2*(0.99) * 0.99 * (1-0.99) * 4.7 = 0.0016
- dError/dWeight4 = (-2*(0.99) * 0.99 * (1-0.99) * 1.4 = 0.0004
- dError/dBias = (-2*(0.99) * 0.99 * (1-0.99) = 0.00035

By using SGD, I get new_weight for every input;

- new_weight1 = 0.5 - 0.1 * 0.0024 = 0.4997549512
- new_weight2 = 0.5 - 0.1 * 0.0011 = 0.4998879777
- new_weight3 = 0.5 - 0.1 * 0.0016 = 0.4998354672
- new_weight4 = 0.5 - 0.1 * 0.0004 = 0.4999509902
- bias = 0.5 - 0.1 * 0.00035 = 0.499964993

In [10]:
```
def sgd(g, y, x):
    return (2*(g-y)*(1-g)*g*x)
```

Train the Neural Network - SLP

6. Train your NN-SLP algorithm on the training data, set learning rate: 0.001 and maximum iteration: 100 (score: 0.5)

In machine learning, epoch = iteration for training process. I just combine the training data with the same training dataset + use the recent weight and bias that I got before from previous epoch. The full journey is at https://bit.ly/slp_csv

7. Find thetha(s) that minimize the cost function, and plot the decision boundary using matplotlib. (score: 1.0)

In [14]:
```
learningrate = input("input learning rate: ")
learningrate = float(learningrate)


fig, plotaccuracy = plt.subplots()
fig, ploterror = plt.subplots()

for i in range(1,df.shape[0]):
    counter = counter + 1
    df.at[i,'t1'] = df.at[i-1,'t1']+learningrate*df.at[i-1,'dt1']
    df.at[i,'t2'] = df.at[i-1,'t2']+learningrate*df.at[i-1,'dt2']
    df.at[i,'t3'] = df.at[i-1,'t3']+learningrate*df.at[i-1,'dt3']
    df.at[i,'t4'] = df.at[i-1,'t4']+learningrate*df.at[i-1,'dt4']
    df.at[i,'bias'] = df.at[i-1,'bias']+learningrate*df.at[i-1,'dbias

    df.at[i,'target']= df.at[i,'x1']*df.at[i,'t1']+df.at[i,'x2']*df.a
    df.at[i,'sigmoid']= 1/(1+math.exp(-df.at[i,'target']))

    if df.at[i,'sigmoid']>0.5:
        df.at[i,'prediction']= 1
    else:
        df.at[i,'prediction']= 0

    df.at[i,'error']= math.pow(df.at[i,'binary']-df.at[i,'sigmoid'],2
    df.at[i,'dt1']= 2*df.at[i,'x1']*(df.at[i,'binary']-df.at[i,'sigmo
    df.at[i,'dt2']= 2*df.at[i,'x2']*(df.at[i,'binary']-df.at[i,'sigmo
    df.at[i,'dt3']= 2*df.at[i,'x3']*(df.at[i,'binary']-df.at[i,'sigmo
    df.at[i,'dt4']= 2*df.at[i,'x4']*(df.at[i,'binary']-df.at[i,'sigmo
```

Testing Process

8. Test your Linear Classifier on the test data, and calculate the accuracy (score: 0.5)

```
In [15]:        for i in range(1,df_validating.shape[0]):

                    df_validating.at[i,'t1'] = df_validating.at[i-1,'t1']+lea
                    df_validating.at[i,'t2'] = df_validating.at[i-1,'t2']+lea
                    df_validating.at[i,'t3'] = df_validating.at[i-1,'t3']+lea
                    df_validating.at[i,'t4'] = df_validating.at[i-1,'t4']+lea
                    df_validating.at[i,'bias'] = df_validating.at[i-1,'bias']

                    df_validating.at[i,'target']= df_validating.at[i,'x1']*d
                    df_validating.at[i,'sigmoid']= 1/(1+math.exp(-df_validat

                    if df_validating.at[i,'sigmoid']>0.5:
                        df_validating.at[i,'prediction']= 1
                    else:
                        df_validating.at[i,'prediction']= 0

                    df_validating.at[i,'error']= math.pow(df_validating.at[i,
                    df_validating.at[i,'dt1']= 2*df_validating.at[i,'x1']*(d
                    df_validating.at[i,'dt2']= 2*df_validating.at[i,'x2']*(d
                    df_validating.at[i,'dt3']= 2*df_validating.at[i,'x3']*(d
                    df_validating.at[i,'dt4']= 2*df_validating.at[i,'x4']*(d
                    df_validating.at[i,'dbias']= 2*(df_validating.at[i,'bina

                    if df_validating.at[i,'prediction'] == df_validating.at[
                        correctvalidate = correctvalidate+1

                ploterrorvalidate.append(df_validating.at[19,'error'] )
                #print("correct training: ",correcttraining, "correct valida
                plotcorrecttrain.append(100*correcttraining/80)
                plotcorrectvalidate.append(100*correctvalidate/19)
                plotcounter.append(counter/80)

                correcttraining = correctvalidate = 0

        print("K-1 Learning Rate: ",learningrate," Epochs:100")
        plotaccuracy.plot(plotcounter, plotcorrecttrain, color="red",label="
        plotaccuracy.plot(plotcounter, plotcorrectvalidate, color = "green",
        plotaccuracy.legend(loc = "lower right")
        plotaccuracy.set_title("Accuracy Diagram")
```

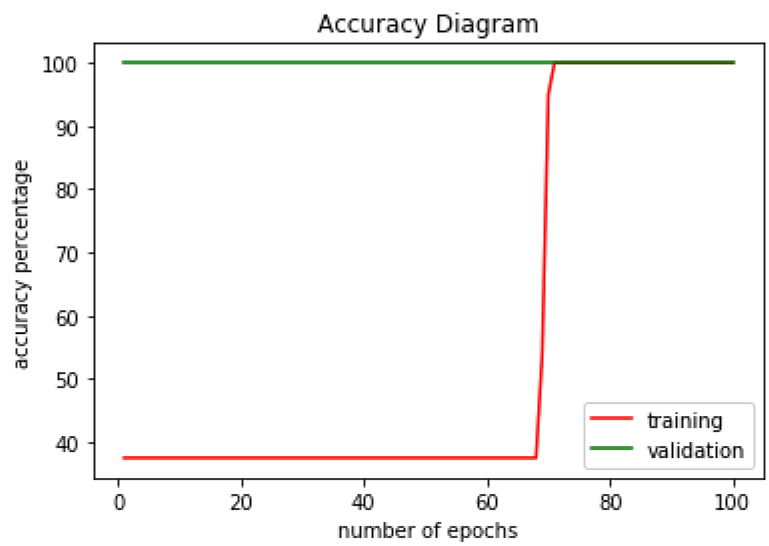
Error Function Plot

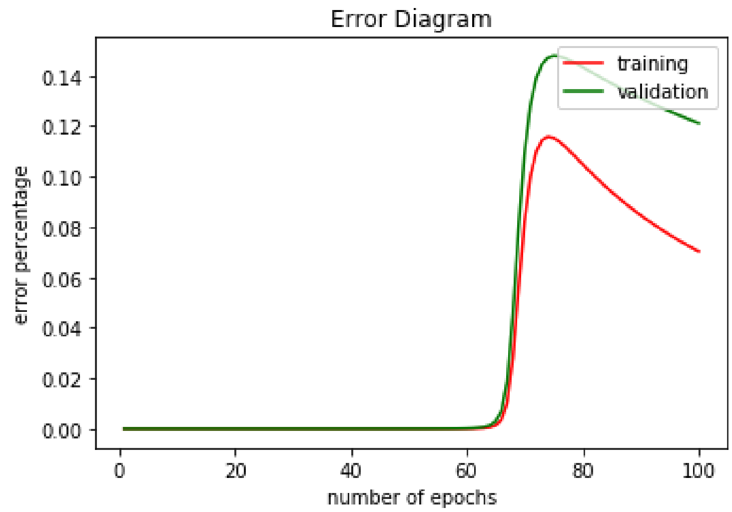
9. Plot your cost function using matplotlib (cost function vs iteration) (score: 0.5)

In [17]:
```
ploterror.plot(plotcounter, ploterrortrain, color = "red",label = "tı
ploterror.plot(plotcounter, ploterrorvalidate, color = "green", labe
ploterror.legend(loc = "upper right")
ploterror.set_title("Error Diagram")
ploterror.set_xlabel("number of epochs")
ploterror.set_ylabel("error percentage")
```

```
input k-fold index: 2
input learning rate: 0.001
K-1 Learning Rate:  0.001  Epochs:100
```

Out[17]: `Text(0, 0.5, 'error percentage')`



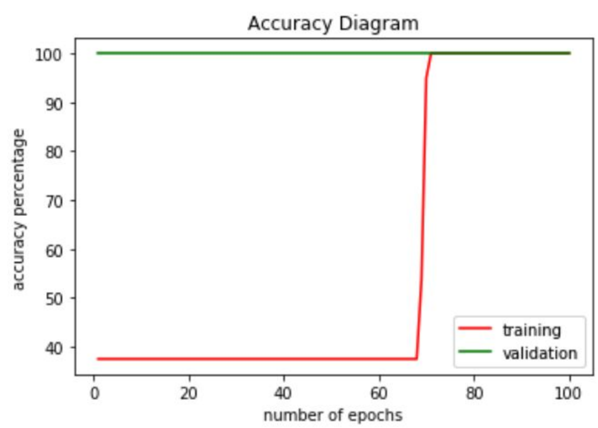


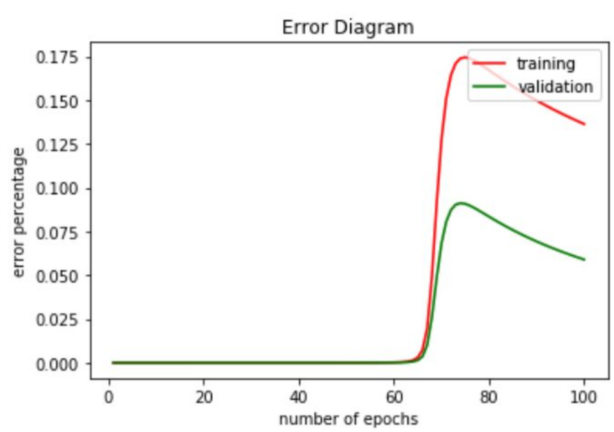
ACCURACY FOR EVERY K-FOLD (K= 1:5)

K=1, Learning Rate = 0.001, Epoch = 100

```
input k-fold index: 1
input learning rate: 0.001
K-1 Learning Rate:  0.001  Epochs:100
```

```
Out[4]: Text(0, 0.5, 'error percentage')
```

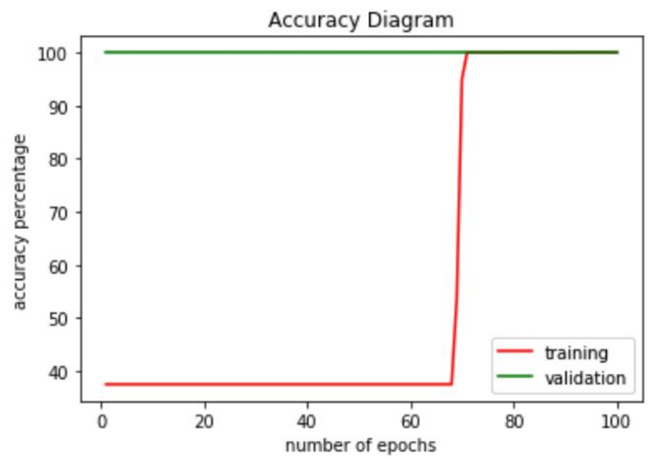


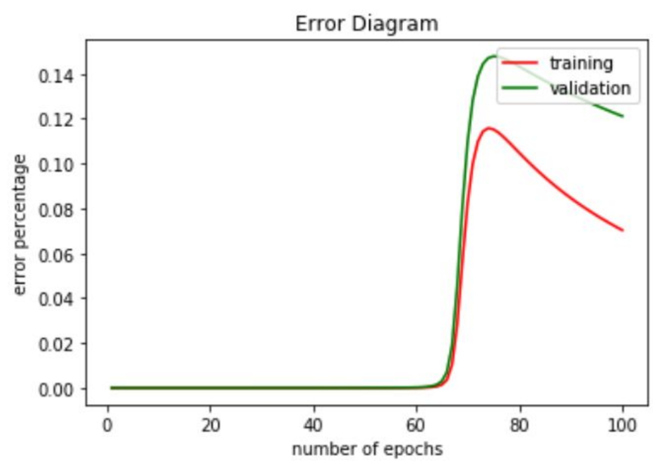


K=2, Learning Rate = 0.001, Epoch = 100

```
input k-fold index: 2
input learning rate: 0.001
K-1 Learning Rate:  0.001  Epochs:100
```

```
Out[5]: Text(0, 0.5, 'error percentage')
```

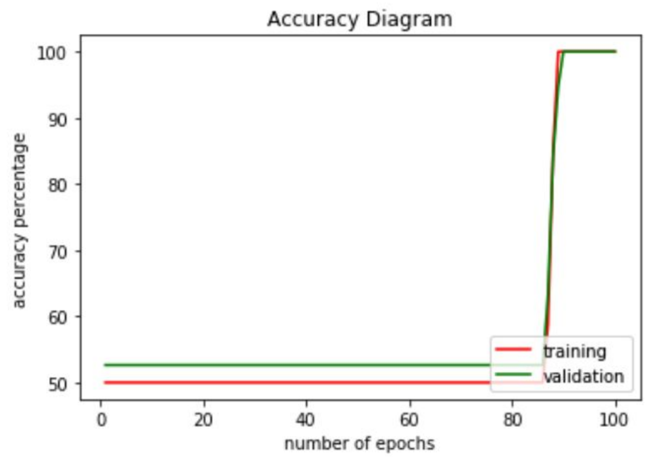


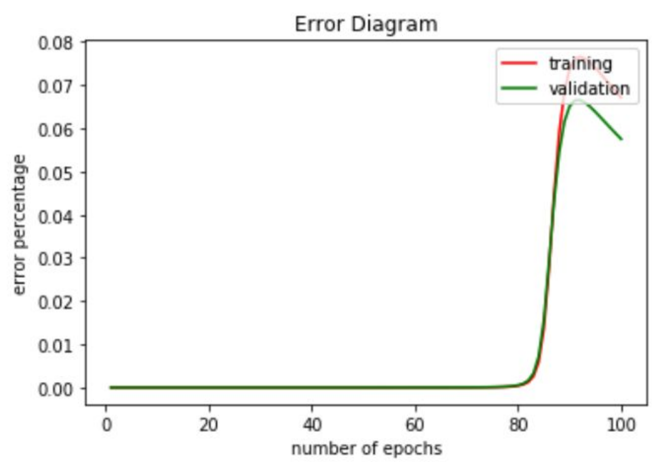


K=3, Learning Rate = 0.001, Epoch = 100

```
input k-fold index: 3
input learning rate: 0.001
K-1 Learning Rate:  0.001  Epochs:100
```

```
Out[6]: Text(0, 0.5, 'error percentage')
```

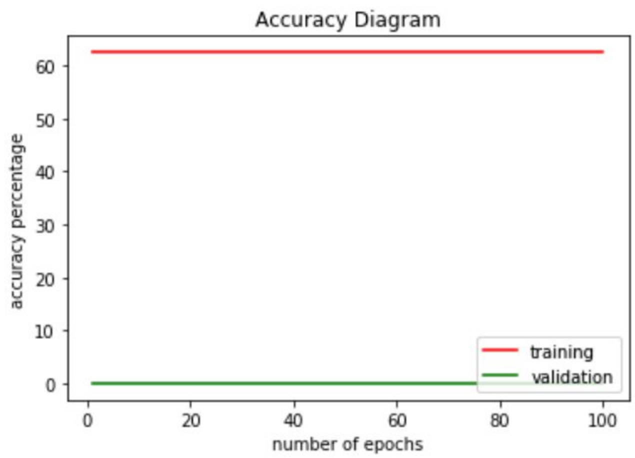


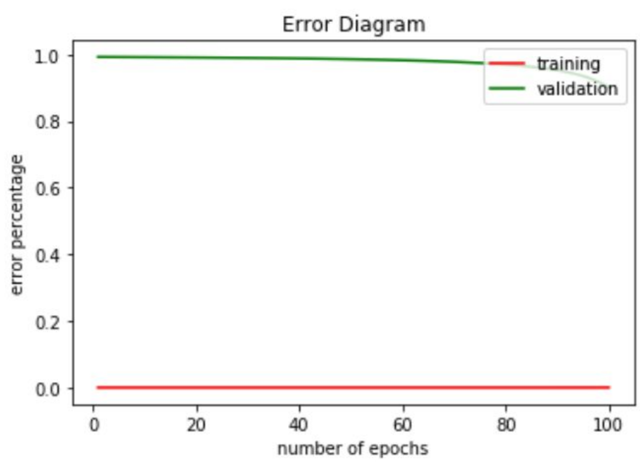


K=4, Learning Rate = 0.001, Epoch = 100

```
input k-fold index: 4
input learning rate: 0.001
K-1 Learning Rate:  0.001  Epochs:100
```

```
Out[7]: Text(0, 0.5, 'error percentage')
```

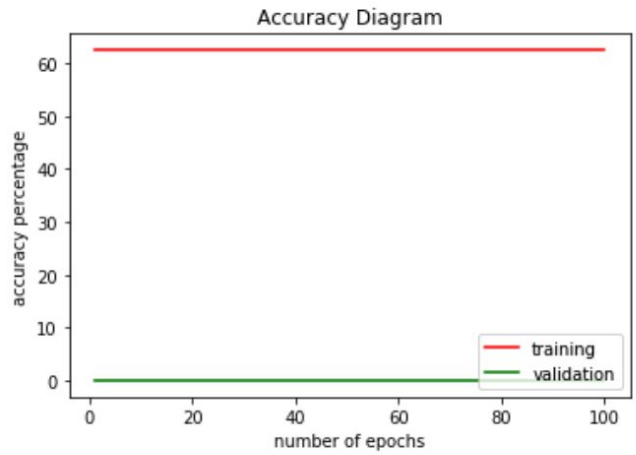


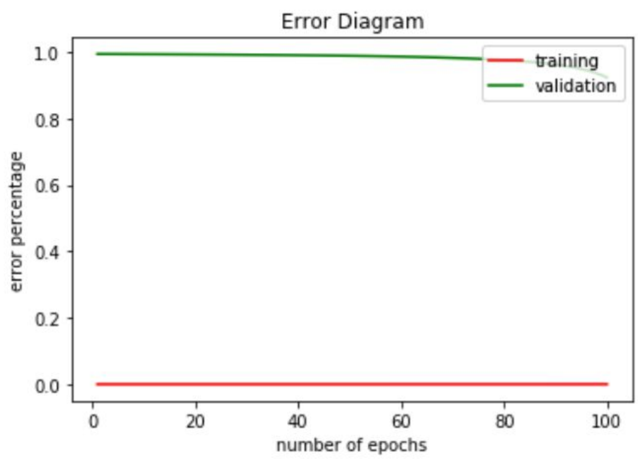


K=5, Learning Rate = 0.001, Epoch = 100

```
input k-fold index: 5
input learning rate: 0.001
K-1 Learning Rate:  0.001  Epochs:100
```

Out[8]: Text(0, 0.5, 'error percentage')





SUMMARY

From the algorithm, we get the best performance on K=1 and K=2 with well accuracy and error improvements. On K=1, we get steady 100% accuracy on validation, 100% accuracy after 70+ epochs for the training process. After that, the error lowers from 0.075% -> 0.050% after 70+ epochs on validation. In addition, on K=2, we still get steady 100% accuracy like the K1 gets and error reduction from 0.14% -> 0.12% after 70+ epochs.