Week 5 Notebook - Multilayered Perceptrons

Multilayered Perceptrons for Regression

```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.collections import LineCollection
from mpl_toolkits.mplot3d.art3d import Line3DCollection
from sklearn.model_selection import train_test_split
```

```
np.random.seed = 47

# If you are NOT using google colab, you need to take this part out starting from here
from google.colab import files

uploaded=files.upload()
# till here

advertising = pd.read_csv('Advertising.csv',usecols=(1,2,3,4))

advertising.head()
```

Choose Files Advertising.csv

- **Advertising.csv**(text/csv) - 5166 bytes, last modified: 10/1/2023 - 100% done

```
Saving Advertising.csv to Advertising.csv
```

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9

```
X = np.array(advertising['TV']).reshape(-1,1)
y = np.array(advertising['Sales'])

X_train, X_test, y_train, y_test =train_test_split(
    X, y, test_size=0.2, random_state=9)

stdscaler = preprocessing.StandardScaler().fit(X_train)
X_train_scaled = stdscaler.transform(X_train)
X_test_scaled  = stdscaler.transform(X_test)
```

```
# Implement your code here
from keras.models import Sequential
from keras.layers import Dense
from keras.regularizers import l2, l1
from keras.optimizers import SGD

# Stochastic Logistic Regression
model = Sequential()

# Model
model.add(Dense(units=2, input_shape=[X_train_scaled.shape[1]],
                activation='sigmoid'))
model.add(Dense(units=1, activation='linear'))

# Slightly better model.
# model.add(Dense(units=4, input_shape=[X_train_scaled.shape[1]],
#                   activation='relu'))
# model.add(Dense(units=2, activation='sigmoid'))
# model.add(Dense(units=1, activation='linear'))
```

```
# Compile model
sgd = SGD(learning_rate=0.001)
model.compile(loss='mean_squared_error', optimizer=sgd)
```

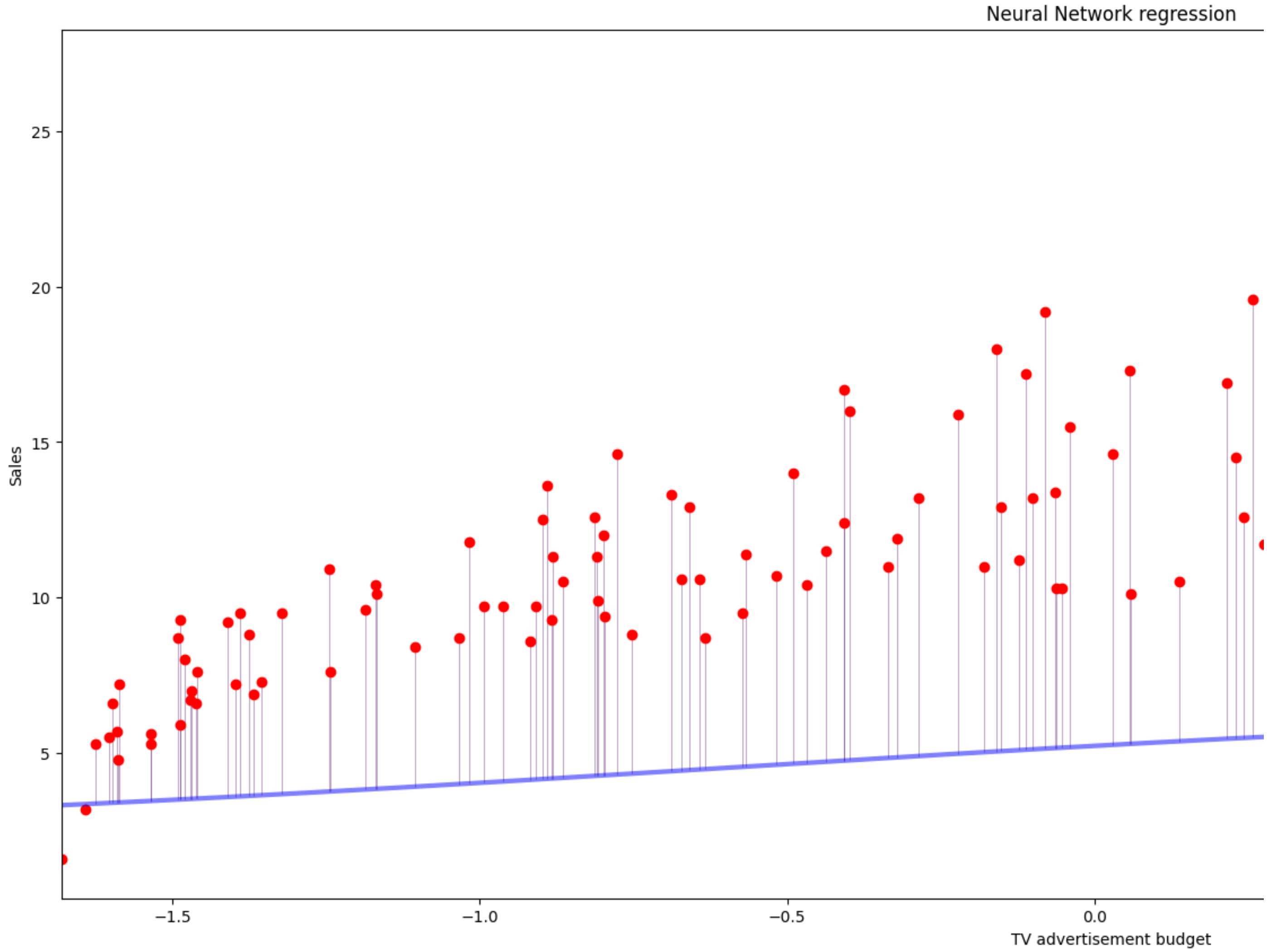
```
# Fit the model
history = model.fit(X_train_scaled, y_train.reshape(-1,1), batch_size = 256,
          epochs = 100, verbose=0, validation_data=(X_test_scaled,y_test.reshape(-1,1)))
```

```
%matplotlib inline
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(6,4))
# # summarize history for loss
plt.plot(np.sqrt(history.history['loss']))
plt.plot(np.sqrt(history.history['val_loss']), 'g--')
plt.title('MLP Regression Model Loss')
plt.ylabel('Root Mean Squared Error (RMSE)')
plt.xlabel('Epoch')
plt.legend(['Training Loss', 'Testing Loss'], loc='upper right')
print("RMSE Loss after final iteration: ", np.sqrt(history.history['val_loss'][-1]))
plt.show()
```

```
RMSE Loss after final iteration:  10.646112267907869
```


```
from matplotlib.collections import LineCollection

y_predicted = model.predict(X_train_scaled)
N = len(y_train)

segments = [[[X_train_scaled[i], y_train[i]], [X_train_scaled[i], y_predicted[i]]] for i in range(N)]
lc = LineCollection(segments, zorder=0)
lc.set_array(np.ones(len(y_train)))
lc.set_alpha(0.5)
lc.set_linewidths(0.5 * np.ones(len(y_train)))

fig = plt.figure(figsize=[24,10])
# plot the training data
plt.plot(X_train_scaled, y_train, 'r.', markersize=12)
# plot the prediction line
x_lin = np.linspace(X_train_scaled.min(),X_train_scaled.max(),1000).reshape(-1,1)
plt.plot(x_lin, model.predict(x_lin), color='blue',linewidth=3,alpha=0.5)
# plot the redisuals
plt.gca().add_collection(lc)

plt.xlim([X_train_scaled.min(),X_train_scaled.max()])
plt.xlabel('TV advertisement budget')
plt.ylabel('Sales')
plt.legend(('Data', 'Regression Fit'), loc='lower right')
plt.title('Neural Network regression')

# plot the regression line
plt.show()
```

```
5/5 [==============================] - 0s 2ms/step
32/32 [==============================] - 0s 1ms/step
```



KDD Cup 1999 Network Security Dataset

In this next example, we will look at the KDD Cup 1999 dataset. (10% subset)

```
%matplotlib inline
import matplotlib.pyplot as plt

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

from sklearn import preprocessing

from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.regularizers import l2,l1
from keras.optimizers import SGD
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

np.random.seed = 47

# If you are NOT using google colab, you need to take this part out starting from here
from google.colab import files

uploaded=files.upload()
# till here

data = pd.read_csv('kddcup.data_10_percent.csv',header=None)
dataCols = ['duration','protocol_type','service','flag','src_bytes','dst_bytes',
    'land','wrong_fragment','urgent','hot','num_failed_logins','logged_in','num_compromised',
```

```
    'root_shell','su_attempted','num_root','num_file_creations','num_shells',
    'num_access_files','num_outbound_cmds','is_host_login','is_guest_login',
    'count','srv_count','serror_rate','srv_serror_rate','rerror_rate','srv_rerror_rate',
    'same_srv_rate','diff_srv_rate','srv_diff_host_rate','dst_host_count','dst_host_srv_count',
    'dst_host_same_srv_rate','dst_host_diff_srv_rate','dst_host_same_src_port_rate',
    'dst_host_srv_diff_host_rate','dst_host_serror_rate','dst_host_srv_serror_rate',
    'dst_host_rerror_rate','dst_host_srv_rerror_rate','target']
data.columns = dataCols

print("Shape: ", data.shape)
print("Targets: ", data['target'].unique())
data = data.reindex(np.random.permutation(data.index)).reset_index(drop=True)

target = data['target'].copy()
data = data.drop('target', axis=1)

data.head()
```

Choose Files kddcup.dat…_percent.csv

- **kddcup.data_10_percent.csv**(text/csv) - 74889749 bytes, last modified: 10/1/2023 - 100% done

```
Saving kddcup.data_10_percent.csv to kddcup.data_10_percent.csv
Shape:  (494021, 42)
Targets:  ['normal.' 'buffer_overflow.' 'loadmodule.' 'perl.' 'neptune.' 'smurf.'
 'guess_passwd.' 'pod.' 'teardrop.' 'portsweep.' 'ipsweep.' 'land.'
 'ftp_write.' 'back.' 'imap.' 'satan.' 'phf.' 'nmap.' 'multihop.'
 'warezmaster.' 'warezclient.' 'spy.' 'rootkit.']
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_count	dst_host_srv_cour
0	0	udp	private	SF	105	0	0	0	0	0	...	255	2
1	0	icmp	ecr_i	SF	1032	0	0	0	0	0	...	255	2
2	0	tcp	private	REJ	0	0	0	0	0	0	...	255	
3	0	tcp	http	SF	224	495	0	0	0	0	...	255	2
4	0	tcp	http	SF	270	9590	0	0	0	0	...	9	2

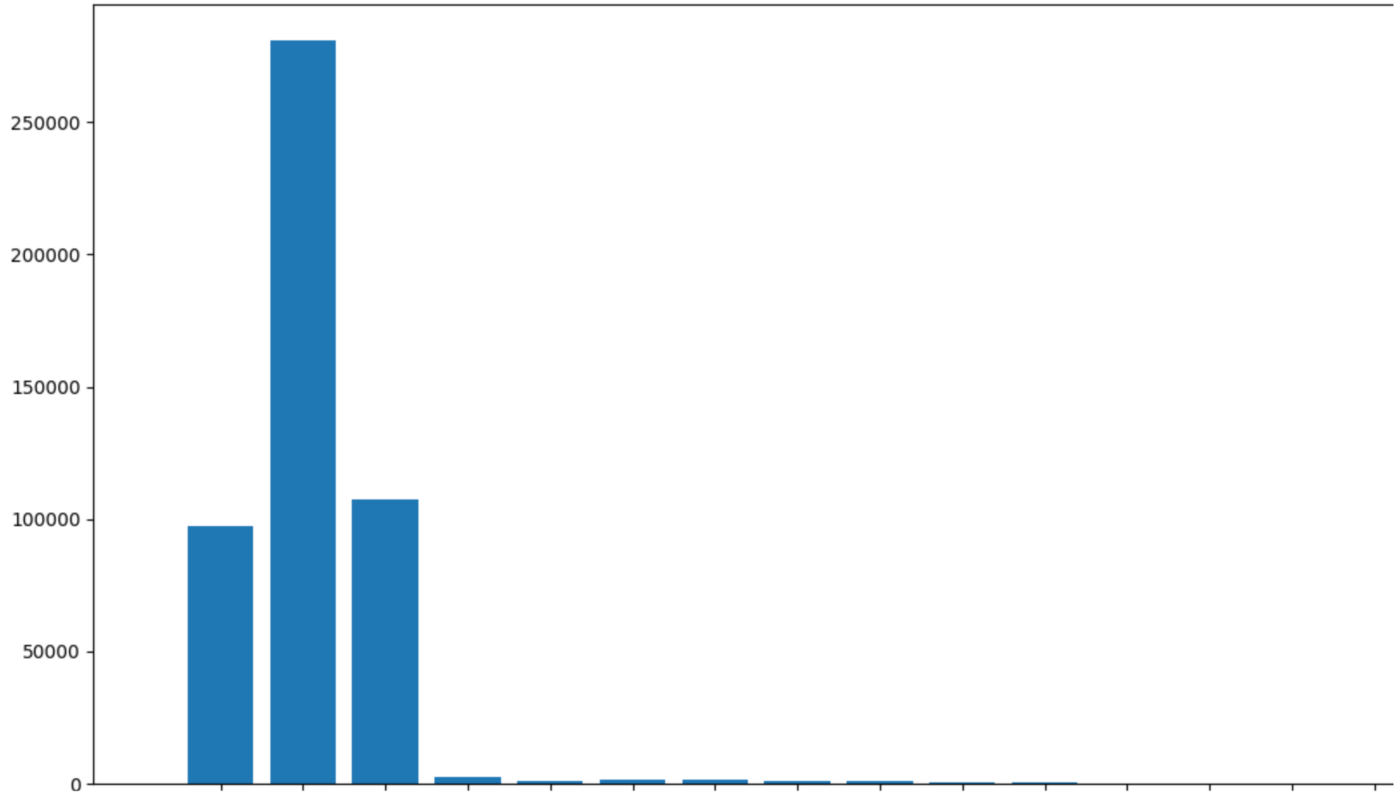
5 rows × 41 columns

```
discreteCols = ['protocol_type','service','flag']
dataDummies = pd.get_dummies(data[discreteCols])
data = data.drop(discreteCols, axis=1)

data = dataDummies.join(data)
data.head()
```

	protocol_type_icmp	protocol_type_tcp	protocol_type_udp	service_IRC	service_X11	service_Z39_50	service_auth	service_bgp	service
0	0	0	1	0	0	0	0	0	
1	1	0	0	0	0	0	0	0	
2	0	1	0	0	0	0	0	0	
3	0	1	0	0	0	0	0	0	
4	0	1	0	0	0	0	0	0	

5 rows × 118 columns

```
posteriorCount = {i:(target==i).sum() for i in target.unique()}
fig = plt.figure(figsize=(20,8))
plt.bar(range(len(posteriorCount)), posteriorCount.values(), align='center')
plt.xticks(range(len(posteriorCount)), zip(posteriorCount.keys(),posteriorCount.values()), rotation='vertical')
plt.subplots_adjust(bottom=0.15)
plt.show()
```




```
targetDummies = pd.get_dummies(target)
targetFull = targetDummies
targetFull.head()
```

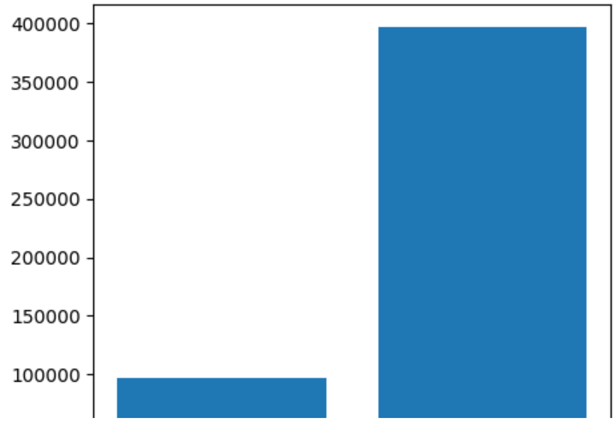
	back.	buffer_overflow.	ftp_write.	guess_passwd.	imap.	ipsweep.	land.	loadmodule.	multihop.	neptune.	...	phf.	pod.	portswee
0	0	0	0	0	0	0	0	0	0	0	...	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	
2	0	0	0	0	0	0	0	0	0	1	...	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	

5 rows × 23 columns

```
target[target != 'normal.'] = 1.0
target[target == 'normal.'] = 0.0
target.head()
```

```
0    0.0
1    1.0
2    1.0
3    0.0
4    0.0
Name: target, dtype: object
```

```
posteriorCount = {i:(target==i).sum() for i in target.unique()}
fig = plt.figure(figsize=(5,5))
plt.bar(range(len(posteriorCount)), posteriorCount.values(), align='center')
plt.xticks(range(len(posteriorCount)), zip(posteriorCount.keys(),posteriorCount.values()), rotation='vertical')
plt.subplots_adjust(bottom=0.15)
plt.show()
```




```
# split the data into training and testing sets
X_train, X_test, y_train, y_test =train_test_split(
    data, target, test_size=0.5, random_state=50)

X_train = X_train.reset_index(drop=True)
X_test  = X_test.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)
y_test  = y_test.reset_index(drop=True)

print(X_train.shape)
print(y_train.shape)

print(X_test.shape)
print(y_test.shape)
```

```
(247010, 118)
(247010,)
(247011, 118)
(247011,)
```

```
# standardize the data
# turn off the error message, we're not setting indivdual values.
pd.options.mode.chained_assignment = None

toStandardize = ['src_bytes', 'dst_bytes', 'count', 'srv_count',
                 'dst_host_count', 'dst_host_srv_count']
stdscaler = preprocessing.MinMaxScaler().fit(X_train[toStandardize])
X_train[toStandardize] = stdscaler.transform(X_train[toStandardize])
X_test[toStandardize]  = stdscaler.transform(X_test[toStandardize])

X_train.head()
```

	protocol_type_icmp	protocol_type_tcp	protocol_type_udp	service_IRC	service_X11	service_Z39_50	service_auth	service_bgp	service
0	1	0	0	0	0	0	0	0	
1	1	0	0	0	0	0	0	0	
2	1	0	0	0	0	0	0	0	
3	1	0	0	0	0	0	0	0	
4	0	1	0	0	0	0	0	0	

5 rows × 118 columns

Logistic Regression Model on the KDD Cup 1999 dataset

```
from keras.models import Sequential
from keras.layers import Dense
from keras.regularizers import l2, l1
from keras.optimizers import SGD

# Stochastic Logistic Regression
model = Sequential()

# Model
```

```
model.add(Dense(units=1, input_shape=[X_train.shape[1]],
                activation='sigmoid', kernel_regularizer=l2(0.001)))

# Compile model
sgd = SGD(learning_rate=0.1)
model.compile(loss='binary_crossentropy', optimizer=sgd)
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 1)                 119

=================================================================
Total params: 119 (476.00 Byte)
Trainable params: 119 (476.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
#!pip show theano
import tensorflow as tf
os.environ["KERAS_BACKEND"] = "theano"
import keras.backend
keras.backend.set_image_data_format('channels_last')
```

```
X_train = tf.convert_to_tensor(X_train, dtype=tf.float32)
y_train = tf.convert_to_tensor(y_train, dtype=tf.float32)
X_test = tf.convert_to_tensor(X_test, dtype=tf.float32)
y_test = tf.convert_to_tensor(y_test, dtype=tf.float32)
```

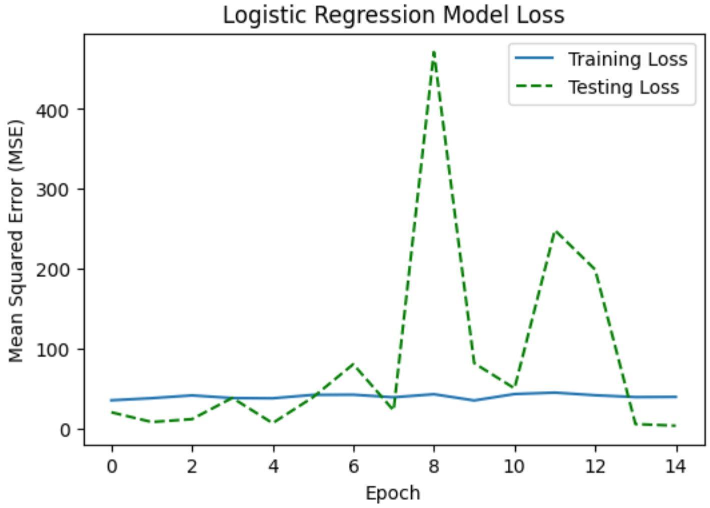
```
history = model.fit(X_train, y_train, batch_size=256,
                    epochs=15, verbose=2, validation_data=(X_test,y_test))
```

```
Epoch 1/15
965/965 - 2s - loss: 0.0268 - val_loss: 0.0270 - 2s/epoch - 2ms/step
Epoch 2/15
965/965 - 2s - loss: 0.0264 - val_loss: 0.0267 - 2s/epoch - 2ms/step
Epoch 3/15
965/965 - 2s - loss: 0.0261 - val_loss: 0.0265 - 2s/epoch - 2ms/step
Epoch 4/15
965/965 - 2s - loss: 0.0258 - val_loss: 0.0262 - 2s/epoch - 2ms/step
Epoch 5/15
965/965 - 2s - loss: 0.0257 - val_loss: 0.0260 - 2s/epoch - 2ms/step
Epoch 6/15
965/965 - 2s - loss: 0.0255 - val_loss: 0.0259 - 2s/epoch - 2ms/step
Epoch 7/15
965/965 - 2s - loss: 0.0254 - val_loss: 0.0258 - 2s/epoch - 2ms/step
Epoch 8/15
965/965 - 2s - loss: 0.0254 - val_loss: 0.0258 - 2s/epoch - 2ms/step
Epoch 9/15
965/965 - 2s - loss: 0.0253 - val_loss: 0.0257 - 2s/epoch - 2ms/step
Epoch 10/15
965/965 - 2s - loss: 0.0253 - val_loss: 0.0257 - 2s/epoch - 2ms/step
Epoch 11/15
965/965 - 2s - loss: 0.0252 - val_loss: 0.0257 - 2s/epoch - 2ms/step
Epoch 12/15
965/965 - 2s - loss: 0.0252 - val_loss: 0.0257 - 2s/epoch - 2ms/step
Epoch 13/15
965/965 - 2s - loss: 0.0251 - val_loss: 0.0256 - 2s/epoch - 2ms/step
Epoch 14/15
965/965 - 2s - loss: 0.0251 - val_loss: 0.0256 - 2s/epoch - 2ms/step
Epoch 15/15
965/965 - 2s - loss: 0.0251 - val_loss: 0.0256 - 2s/epoch - 2ms/step
```

```
%matplotlib inline
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(6,4))
# # summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'], 'g--')
plt.title('Logistic Regression Model Loss')
plt.ylabel('Mean Squared Error (MSE)')
plt.xlabel('Epoch')
plt.legend(['Training Loss', 'Testing Loss'], loc='upper right')
```

```
print("Loss after final iteration: ", history.history['val_loss'][-1])
plt.show()
```

```
Loss after final iteration:  2.991057872772217
```




```
predictions = (pd.DataFrame(model.predict(X_test.to_numpy())))

M11= predictions.to_numpy()

predictions[M11 > (0.5)] = 'normal'
predictions[M11 <= (0.5)] = 'anamoly'

My11=y_test.to_numpy().reshape(-1,1).copy()

y_test_labels = y_test.to_numpy().reshape(-1,1).copy()
y_test_labels[My11 > 0.5] = 'normal'
y_test_labels[My11 <= 0.5] = 'anamoly'

print('accuracy', accuracy_score(predictions,y_test_labels))
print('confusion matrix\n', confusion_matrix(predictions,y_test_labels))

print(classification_report(predictions,y_test_labels))
```

```
7720/7720 [==============================] - 7s 895us/step
accuracy 0.9950042710648513
confusion matrix
 [[ 48217    614]
 [   620 197560]]
              precision    recall  f1-score   support

     anamoly       0.99      0.99      0.99     48831
      normal       1.00      1.00      1.00    198180

    accuracy                           1.00    247011
   macro avg       0.99      0.99      0.99    247011
weighted avg       1.00      1.00      1.00    247011
```

MLP Model on the KDD Cup 1999 dataset

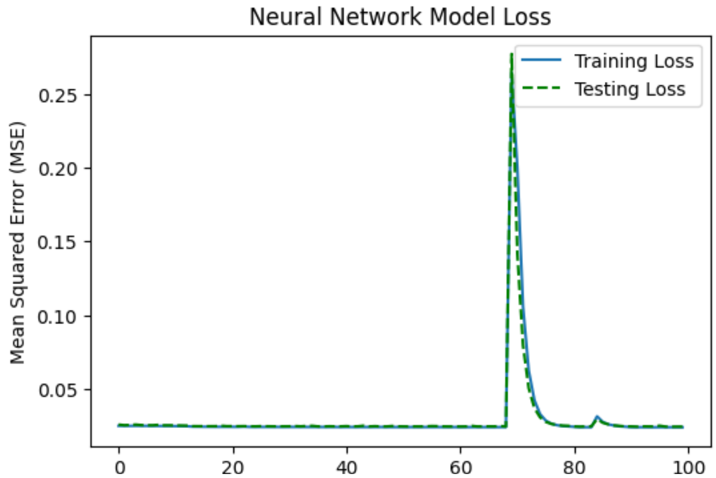
```
from keras.models import Sequential
from keras.layers import Dense
from keras.regularizers import l2, l1
from keras.optimizers import SGD

# Stochastic Logistic Regression
model = Sequential()

# Model
model.add(Dense(units=4, input_shape=[X_train.shape[1]],
                activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dense(units=2,
                activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dense(units=1,
                activation='sigmoid', kernel_regularizer=l2(0.001)))
```

```
# Compile model
sgd = SGD(learning_rate=0.1)
model.compile(loss='binary_crossentropy', optimizer=sgd)


history = model.fit(X_train, y_train, batch_size=128,
                    epochs=100, verbose=2, validation_data=(X_test,y_test))
```

```
Epoch 72/100
1930/1930 - 4s - loss: 0.1060 - val_loss: 0.0795 - 4s/epoch - 2ms/step
Epoch 73/100
1930/1930 - 4s - loss: 0.0626 - val_loss: 0.0505 - 4s/epoch - 2ms/step
Epoch 74/100
1930/1930 - 5s - loss: 0.0424 - val_loss: 0.0369 - 5s/epoch - 2ms/step
Epoch 75/100
1930/1930 - 4s - loss: 0.0329 - val_loss: 0.0307 - 4s/epoch - 2ms/step
Epoch 76/100
1930/1930 - 4s - loss: 0.0285 - val_loss: 0.0276 - 4s/epoch - 2ms/step
Epoch 77/100
1930/1930 - 4s - loss: 0.0264 - val_loss: 0.0263 - 4s/epoch - 2ms/step
Epoch 78/100
1930/1930 - 3s - loss: 0.0254 - val_loss: 0.0255 - 3s/epoch - 2ms/step
Epoch 79/100
1930/1930 - 4s - loss: 0.0248 - val_loss: 0.0251 - 4s/epoch - 2ms/step
Epoch 80/100
1930/1930 - 4s - loss: 0.0246 - val_loss: 0.0250 - 4s/epoch - 2ms/step
Epoch 81/100
1930/1930 - 4s - loss: 0.0243 - val_loss: 0.0245 - 4s/epoch - 2ms/step
Epoch 82/100
1930/1930 - 5s - loss: 0.0241 - val_loss: 0.0244 - 5s/epoch - 2ms/step
Epoch 83/100
1930/1930 - 4s - loss: 0.0241 - val_loss: 0.0246 - 4s/epoch - 2ms/step
Epoch 84/100
1930/1930 - 3s - loss: 0.0240 - val_loss: 0.0246 - 3s/epoch - 2ms/step
Epoch 85/100
1930/1930 - 4s - loss: 0.0314 - val_loss: 0.0296 - 4s/epoch - 2ms/step
Epoch 86/100
1930/1930 - 4s - loss: 0.0276 - val_loss: 0.0271 - 4s/epoch - 2ms/step
Epoch 87/100
1930/1930 - 4s - loss: 0.0260 - val_loss: 0.0259 - 4s/epoch - 2ms/step
Epoch 88/100
1930/1930 - 4s - loss: 0.0252 - val_loss: 0.0252 - 4s/epoch - 2ms/step
Epoch 89/100
1930/1930 - 3s - loss: 0.0246 - val_loss: 0.0249 - 3s/epoch - 2ms/step
Epoch 90/100
1930/1930 - 4s - loss: 0.0244 - val_loss: 0.0246 - 4s/epoch - 2ms/step
Epoch 91/100
1930/1930 - 3s - loss: 0.0241 - val_loss: 0.0246 - 3s/epoch - 2ms/step
Epoch 92/100
1930/1930 - 3s - loss: 0.0241 - val_loss: 0.0244 - 3s/epoch - 2ms/step
Epoch 93/100
1930/1930 - 3s - loss: 0.0241 - val_loss: 0.0246 - 3s/epoch - 2ms/step
Epoch 94/100
1930/1930 - 5s - loss: 0.0241 - val_loss: 0.0246 - 5s/epoch - 2ms/step
Epoch 95/100
1930/1930 - 3s - loss: 0.0240 - val_loss: 0.0246 - 3s/epoch - 2ms/step
Epoch 96/100
1930/1930 - 3s - loss: 0.0240 - val_loss: 0.0251 - 3s/epoch - 2ms/step
Epoch 97/100
1930/1930 - 4s - loss: 0.0240 - val_loss: 0.0244 - 4s/epoch - 2ms/step
Epoch 98/100
1930/1930 - 3s - loss: 0.0240 - val_loss: 0.0244 - 3s/epoch - 2ms/step
Epoch 99/100
1930/1930 - 3s - loss: 0.0240 - val_loss: 0.0244 - 3s/epoch - 2ms/step
Epoch 100/100
1930/1930 - 4s - loss: 0.0240 - val_loss: 0.0244 - 4s/epoch - 2ms/step
```

```
%matplotlib inline
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(6,4))
# # summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'], 'g--')
plt.title('Neural Network Model Loss')
plt.ylabel('Mean Squared Error (MSE)')
plt.xlabel('Epoch')
plt.legend(['Training Loss', 'Testing Loss'], loc='upper right')
print("Loss after final iteration: ", history.history['val_loss'][-1])
plt.show()
```

```
Loss after final iteration:  0.02443923056125641
```




```
predictions = pd.DataFrame(model.predict(X_test))

#print(predictions)

#print(type(predictions))
M1= predictions.to_numpy()



predictions[M1 > 0.5] = 'normal'
predictions[M1 <= 0.5] = 'anamoly'

#print(predictions)

My12=y_test

#y_test_labels = y_test.to_numpy().reshape(-1,1).copy()
y_test_labels[My12 > 0.5] = 'normal'
y_test_labels[My12 <= 0.5] = 'anamoly'


print()
print('accuracy', accuracy_score(predictions,y_test_labels))
print('confusion matrix\n', confusion_matrix(predictions,y_test_labels))

print(classification_report(predictions,y_test_labels))
```

```
7720/7720 [==============================] - 7s 876us/step

accuracy 0.9972592313702628
confusion matrix
 [[ 48746    586]
 [    91 197588]]
              precision    recall  f1-score   support

     anamoly       1.00      0.99      0.99     49332
      normal       1.00      1.00      1.00    197679

    accuracy                           1.00    247011
   macro avg       1.00      0.99      1.00    247011
weighted avg       1.00      1.00      1.00    247011
```