COMP 4211 Machine Learning
Assignment 1 Report
CHEN Yifei (20328874)

Environment Setting

In this assignment, I use several packages including numpy, matplotlib, scikit-learn, model_selection, time, and metrics in Python 3.7.

```
from __future__ import print_function

import numpy as np
import matplotlib
import matplotlib.pyplot as plt
#import pandas as pd

import time

from sklearn import datasets, linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import mean_squared_error, r2_score

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
```

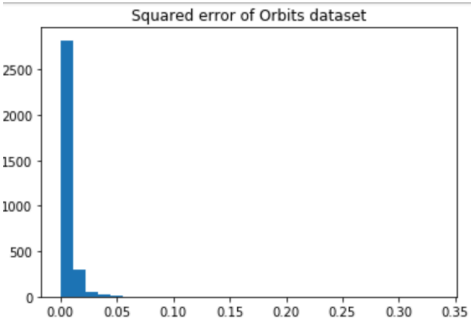
Empirical Study on Linear Regression

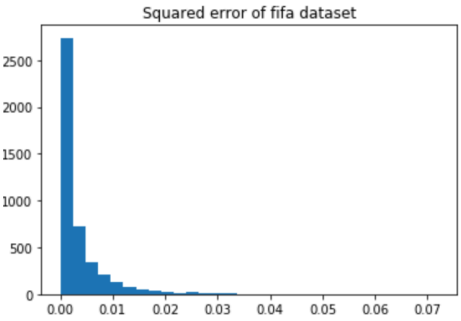
In this part, I implement linear regression on all three datasets. Below are the results:

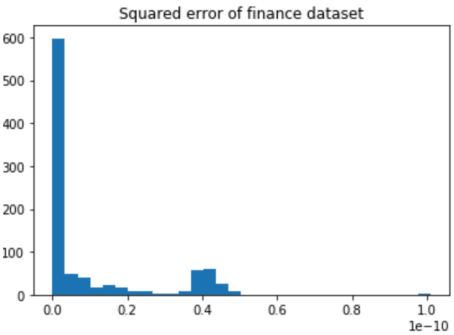
(Training time, R2 score on training & Testing Dataset)

```
----------
Linear Regression for dataset orbits:


time needed to complete the task: 0.00282 s
Variance score on training dataset: 0.69
Variance score on test dataset: 0.70
----------
----------
Linear Regression for dataset fifa:


time needed to complete the task: 0.00634 s
Variance score on training dataset: 0.84
Variance score on test dataset: 0.84
----------
----------
Linear Regression for dataset finance:


time needed to complete the task: 0.0025 s
Variance score on training dataset: 1.00
Variance score on test dataset: 1.00
----------
```

Both R2 scores of training and test parts in three datasets are very close. Even though it is the simplest model, the scores are good, the squared error histograms are shown below:







The main squared error concentrate in the low-value interval, which means the error is not huge and the model performs well.

Empirical Study on Logistic Regression

In this part, I set the function: (take orbits dataset as an example)

```
orbits_logistic = linear_model.SGDClassifier(loss = 'log', max_iter=50, tol=1e-3, verbose=0)
```

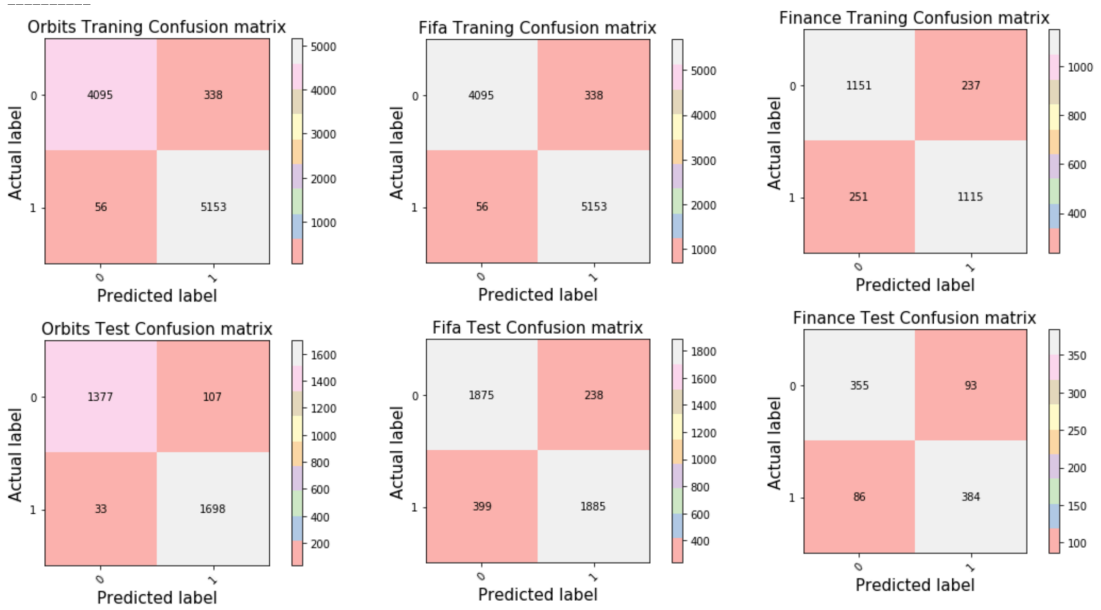
Here I do not set learning rate (default), and I set early stop for iteration 50. Also, I choose the SGD classifier for its advantage on smaller size of batches, which reduce the computation cost as well as proving good accuracy in these datasets.

Relevant results:

(For each dataset: the first number is the time needed for construct the model (fit); the second is the accuracy of using the model to predict the training set; while the last number is the accuracy of using the model to predict the test set.)

```
----------
Logistic regression for Orbits dataset:


time needed to complete the task: 0.0144 s
Accuracy on traning dataset:  0.959137108483717
Accuracy on test dataset:  0.9564541213063764
----------
----------
Logistic regression for fifa dataset:


time needed to complete the task: 0.08595 s
Accuracy on traning dataset:  0.8596012432719279
Accuracy on test dataset:  0.8551284967022971
----------
----------
Logistic regression for finance dataset:


time needed to complete the task: 0.02791 s
Accuracy on traning dataset:  0.8079157588961511
Accuracy on test dataset:  0.7810457516339869
----------
```

And the confusion matrix:



From the result, I find that the method of logistic regression is also suitable for the three provided datasets (especially for the orbits data, where the accuracy is around 95%).

Empirical Study on Single-hidden-layer Neural Network

In this part, I mainly use 'GridSearchCV' (to find the optimal parameter set) and 'MLPClassifier' (to implement the neural network model).

1. For the orbits dataset, for convenience, I set 10 learning rates to be 0.1 (which are manually adjusted, I find this learning rate performs good).

```
tuned_parameters_1 = [{'hidden_layer_sizes': [(1,)], 'learning_rate':['constant'], 'learning_rate_init':[0.1]},
                      {'hidden_layer_sizes': [(2,)], 'learning_rate':['constant'], 'learning_rate_init':[0.1]},
                      {'hidden_layer_sizes': [(3,)], 'learning_rate':['constant'], 'learning_rate_init':[0.1]},
                      {'hidden_layer_sizes': [(4,)], 'learning_rate':['constant'], 'learning_rate_init':[0.1]},
                      {'hidden_layer_sizes': [(5,)], 'learning_rate':['constant'], 'learning_rate_init':[0.1]},
                      {'hidden_layer_sizes': [(6,)], 'learning_rate':['constant'], 'learning_rate_init':[0.1]},
                      {'hidden_layer_sizes': [(7,)], 'learning_rate':['constant'], 'learning_rate_init':[0.1]},
                      {'hidden_layer_sizes': [(8,)], 'learning_rate':['constant'], 'learning_rate_init':[0.1]},
                      {'hidden_layer_sizes': [(9,)], 'learning_rate':['constant'], 'learning_rate_init':[0.1]},
                      {'hidden_layer_sizes': [(10,)], 'learning_rate':['constant'], 'learning_rate_init':[0.1]}]
```

Then I run GridSearchCV to find the best of 10.

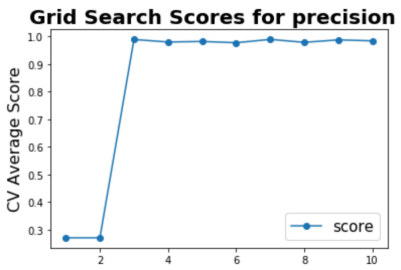
```
print("# Tuning hyper-parameters for Orbits" )
scores = ['precision', 'recall']
for score in scores:
    print("# Tuning hyper-parameters for %s" % score)
    print()
    mlp =MLPClassifier(max_iter=300, alpha=1e-4, solver='sgd', verbose=0, tol=1e-4, random_state=1)
    clf1 = GridSearchCV(mlp, tuned_parameters_1, cv=5, scoring='%s_macro' % score)
    clf1.fit(orbits_train_X , orbits_classification_train_Y )
```

Here I still use SGD method, and the best method will be selected based on precision score and recall score (The one with highest average score will be chosed as the best).

Here is the result:

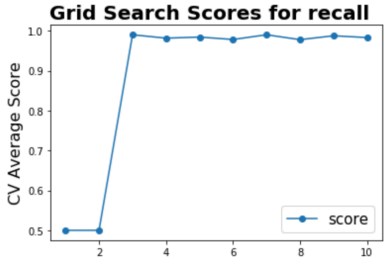
(Precision)

```
Grid scores on development set:

0.270 (+/-0.000) for {'hidden_layer_sizes': (1,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.270 (+/-0.000) for {'hidden_layer_sizes': (2,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.990 (+/-0.011) for {'hidden_layer_sizes': (3,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.980 (+/-0.016) for {'hidden_layer_sizes': (4,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.983 (+/-0.023) for {'hidden_layer_sizes': (5,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.978 (+/-0.027) for {'hidden_layer_sizes': (6,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.990 (+/-0.005) for {'hidden_layer_sizes': (7,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.979 (+/-0.031) for {'hidden_layer_sizes': (8,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.988 (+/-0.007) for {'hidden_layer_sizes': (9,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.985 (+/-0.025) for {'hidden_layer_sizes': (10,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}


Best parameters set found on development set based on precision score :

{'hidden_layer_sizes': (7,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
```



(Recall)

```
Grid scores on development set:

0.500 (+/-0.000) for {'hidden_layer_sizes': (1,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.500 (+/-0.000) for {'hidden_layer_sizes': (2,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.990 (+/-0.011) for {'hidden_layer_sizes': (3,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.981 (+/-0.015) for {'hidden_layer_sizes': (4,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.984 (+/-0.021) for {'hidden_layer_sizes': (5,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.978 (+/-0.026) for {'hidden_layer_sizes': (6,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.990 (+/-0.004) for {'hidden_layer_sizes': (7,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.978 (+/-0.031) for {'hidden_layer_sizes': (8,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.987 (+/-0.011) for {'hidden_layer_sizes': (9,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.983 (+/-0.034) for {'hidden_layer_sizes': (10,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}


Best parameters set found on development set based on recall score :

{'hidden_layer_sizes': (3,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
```



(The final choice based on the two scores)

```
index = np.argmax(average)
print()
print ('Over all the optimal parameter setting is:')
print ( tuned_parameters_1[index])
print()
```

```
Over all the optimal parameter setting is:
{'hidden_layer_sizes': [(7,)], 'learning_rate': ['constant'], 'learning_rate_init': [0.1]}
```

The Next I do is to use the parameters I chose to construct the model

```
mlp1 = MLPClassifier(hidden_layer_sizes=tuned_parameters_1[index]['hidden_layer_sizes'][0], activation='relu', solver='adam', alpha=0.0001,
                     batch_size='auto', learning_rate='constant',learning_rate_init=tuned_parameters_1[index]['learning_rate_init'][0], power_t=0.5,
                     max_iter=50, shuffle=True, random_state=None, tol=0.0001, verbose=1,
                     warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=True,
                     validation_fraction=0.2, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10)
```

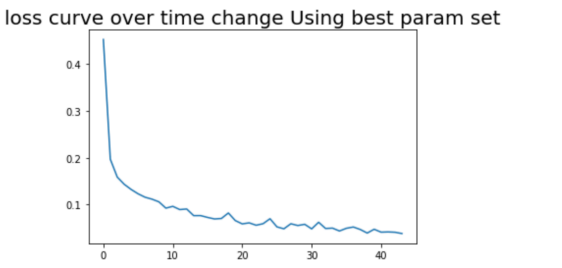
Here I do not use the SGD, the activation function is ReLU, early stopping is set to be 50 iterations.

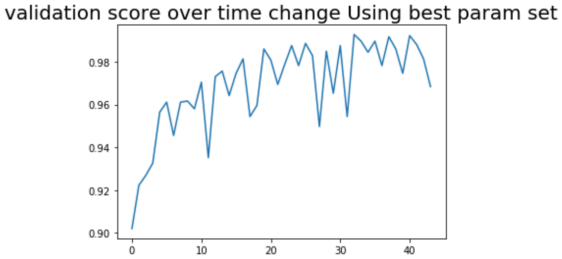
And here are relevant results of constructing and evaluating the model:

(Relevant scores & Time for training the model (fit) and time for predicting the test set)

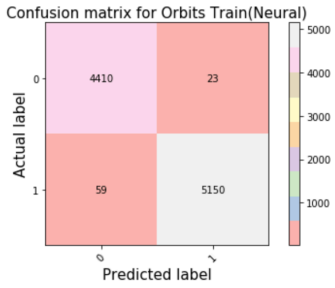
```
Iteration 44, loss = 0.03841481
Validation score: 0.968377
Validation score did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
time for training: 0.51419 s
time for predicting: 0.00074 s
```

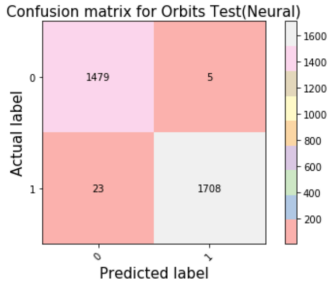
(Loss curve and validation score during training (x axis: iteration))





(Confusion Matrix on both training and test set)





From the confusion matrix, we find that the accuracy is really close to 99%, which is really high compared with the former 2.

2. For the Fifa dataset, the procedure is similar, I still set the 10 learning rates to be 0.1.

```
# Set the parameters by cross-validation
tuned_parameters_2 = [{'hidden_layer_sizes': [(1,)], 'learning_rate':['constant'], 'learning_rate_init':[0.1]},
                      {'hidden_layer_sizes': [(2,)], 'learning_rate':['constant'], 'learning_rate_init':[0.1]},
                      {'hidden_layer_sizes': [(3,)], 'learning_rate':['constant'], 'learning_rate_init':[0.1]},
                      {'hidden_layer_sizes': [(4,)], 'learning_rate':['constant'], 'learning_rate_init':[0.1]},
                      {'hidden_layer_sizes': [(5,)], 'learning_rate':['constant'], 'learning_rate_init':[0.1]},
                      {'hidden_layer_sizes': [(6,)], 'learning_rate':['constant'], 'learning_rate_init':[0.1]},
                      {'hidden_layer_sizes': [(7,)], 'learning_rate':['constant'], 'learning_rate_init':[0.1]},
                      {'hidden_layer_sizes': [(8,)], 'learning_rate':['constant'], 'learning_rate_init':[0.1]},
                      {'hidden_layer_sizes': [(9,)], 'learning_rate':['constant'], 'learning_rate_init':[0.1]},
                      {'hidden_layer_sizes': [(10,)], 'learning_rate':['constant'], 'learning_rate_init':[0.1]}]
```

```
average = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

print("# Tuning hyper-parameters for Fifa" )
scores = ['precision', 'recall']
for score in scores:
    print('%s scores:' % score)
    print()
    mlp =MLPClassifier(max_iter=300, alpha=1e-4, solver='sgd', verbose=0, tol=1e-4, random_state=1)
    clf2 = GridSearchCV(mlp, tuned_parameters_2, cv=5, scoring='%s_macro' % score)
    clf2.fit(fifa_train_X , fifa_classification_train_Y )

    print()
    print("Grid scores on development set:")
    print()
    means = clf2.cv_results_['mean_test_score']
    average = average + clf2.cv_results_['mean_test_score']
    stds = clf2.cv_results_['std_test_score']
    for mean, std, params in zip(means, stds, clf2.cv_results_['params']):
        print("%0.3f (+/-%0.03f) for %r"
              % (mean, std * 2, params))
```

What I get:

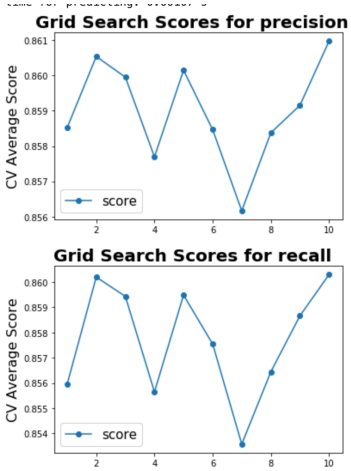
(precision score)

```
# Tuning hyper-parameters for Fifa
precision scores:


Grid scores on development set:

0.859 (+/-0.016) for {'hidden_layer_sizes': (1,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.861 (+/-0.010) for {'hidden_layer_sizes': (2,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.860 (+/-0.013) for {'hidden_layer_sizes': (3,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.858 (+/-0.013) for {'hidden_layer_sizes': (4,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.860 (+/-0.012) for {'hidden_layer_sizes': (5,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.858 (+/-0.014) for {'hidden_layer_sizes': (6,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.856 (+/-0.016) for {'hidden_layer_sizes': (7,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.858 (+/-0.014) for {'hidden_layer_sizes': (8,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.859 (+/-0.012) for {'hidden_layer_sizes': (9,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.861 (+/-0.013) for {'hidden_layer_sizes': (10,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}


Best parameters set found on development set based on score precision:

{'hidden_layer_sizes': (10,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
recall scores:
```

(Recall score & the final choice based on the two scores)

```
Grid scores on development set:

0.856 (+/-0.020) for {'hidden_layer_sizes': (1,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.860 (+/-0.011) for {'hidden_layer_sizes': (2,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.859 (+/-0.015) for {'hidden_layer_sizes': (3,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.856 (+/-0.018) for {'hidden_layer_sizes': (4,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.859 (+/-0.013) for {'hidden_layer_sizes': (5,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.858 (+/-0.014) for {'hidden_layer_sizes': (6,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.854 (+/-0.021) for {'hidden_layer_sizes': (7,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.856 (+/-0.017) for {'hidden_layer_sizes': (8,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.859 (+/-0.013) for {'hidden_layer_sizes': (9,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}
0.860 (+/-0.013) for {'hidden_layer_sizes': (10,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}


Best parameters set found on development set based on score recall:

{'hidden_layer_sizes': (10,), 'learning_rate': 'constant', 'learning_rate_init': 0.1}


Over all the optimal parameter setting is:
{'hidden_layer_sizes': [(10,)], 'learning_rate': ['constant'], 'learning_rate_init': [0.1]}
```

(Graph for the above scores, x-axi is the number of processing units)



Using the optimal set H* :

```
mlp2 = MLPClassifier(hidden_layer_sizes=tuned_parameters_2[index]['hidden_layer_sizes'][0], activation='relu', solver='adam', alpha=0.0001,
                     batch_size='auto', learning_rate='constant', learning_rate_init=tuned_parameters_2[index]['learning_rate_init'][0], power_t=0.5,
                     max_iter=50, shuffle=True, random_state=None, tol=0.0001, verbose=1,
                     warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=True,
                     validation_fraction=0.2, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10)
```

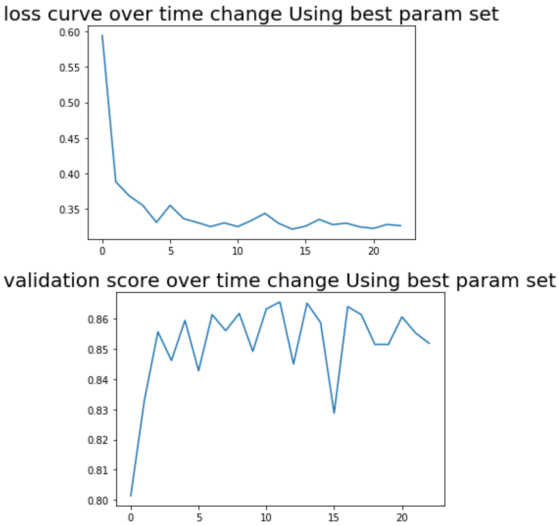
(The same, not SGD, activation function ReLU, early stop 50 iterations)

Relevant results:

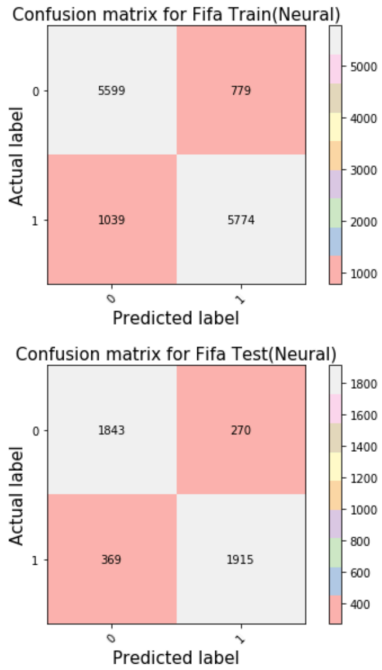
(Relevant scores & Time for training the model (fit) and time for predicting the test set)

```
Iteration 23, loss = 0.32601284
Validation score: 0.851838
Validation score did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
time for training: 0.381 s
time for predicting: 0.00107 s
```

(Loss curve and validation score during training (x axis: iteration))



(Confusion Matrix on both training and test set)



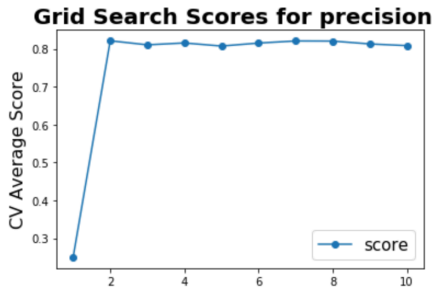
From the confusion matrix, I find that the accuracy does not improve significantly compared with logistic regression.

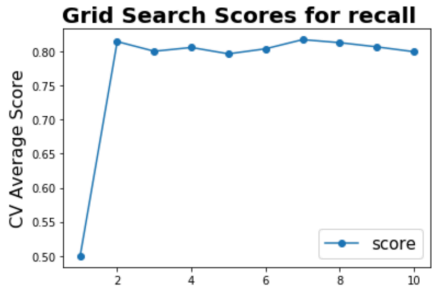
3. For the Finance dataset, the procedure is similar. However, I set the learning rates to be 0.01 after several test.

```
tuned_parameters_3 = [{'hidden_layer_sizes': [(1,)], 'learning_rate':['constant'], 'learning_rate_init':[0.01]},
                      {'hidden_layer_sizes': [(2,)], 'learning_rate':['constant'], 'learning_rate_init':[0.01]},
                      {'hidden_layer_sizes': [(3,)], 'learning_rate':['constant'], 'learning_rate_init':[0.01]},
                      {'hidden_layer_sizes': [(4,)], 'learning_rate':['constant'], 'learning_rate_init':[0.01]},
                      {'hidden_layer_sizes': [(5,)], 'learning_rate':['constant'], 'learning_rate_init':[0.01]},
                      {'hidden_layer_sizes': [(6,)] , 'learning_rate':['constant'], 'learning_rate_init':[0.01]},
                      {'hidden_layer_sizes': [(7,)] , 'learning_rate':['constant'], 'learning_rate_init':[0.01]},
                      {'hidden_layer_sizes': [(8,)], 'learning_rate':['constant'], 'learning_rate_init':[0.01]},
                      {'hidden_layer_sizes': [(9,)], 'learning_rate':['constant'], 'learning_rate_init':[0.01]},
                      {'hidden_layer_sizes': [(10,)], 'learning_rate':['constant'], 'learning_rate_init':[0.01]}]
```

Due to the similar process, here I just post the results for the dataset.

(Selecting the optimal parameter setting)





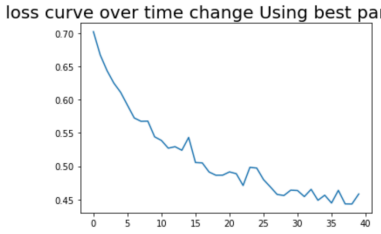

```
0.250 (+/-0.004) for {'hidden_layer_sizes': (1,), 'learning_rate': 'constant', 'learning_rate_init': 0.01}
0.821 (+/-0.013) for {'hidden_layer_sizes': (2,), 'learning_rate': 'constant', 'learning_rate_init': 0.01}
0.810 (+/-0.018) for {'hidden_layer_sizes': (3,), 'learning_rate': 'constant', 'learning_rate_init': 0.01}
0.815 (+/-0.024) for {'hidden_layer_sizes': (4,), 'learning_rate': 'constant', 'learning_rate_init': 0.01}
0.807 (+/-0.029) for {'hidden_layer_sizes': (5,), 'learning_rate': 'constant', 'learning_rate_init': 0.01}
0.815 (+/-0.015) for {'hidden_layer_sizes': (6,), 'learning_rate': 'constant', 'learning_rate_init': 0.01}
0.821 (+/-0.018) for {'hidden_layer_sizes': (7,), 'learning_rate': 'constant', 'learning_rate_init': 0.01}
0.820 (+/-0.015) for {'hidden_layer_sizes': (8,), 'learning_rate': 'constant', 'learning_rate_init': 0.01}
0.813 (+/-0.018) for {'hidden_layer_sizes': (9,), 'learning_rate': 'constant', 'learning_rate_init': 0.01}
0.808 (+/-0.019) for {'hidden_layer_sizes': (10,), 'learning_rate': 'constant', 'learning_rate_init': 0.01}


Best parameters set found on development set based on score precision:

{'hidden_layer_sizes': (2,), 'learning_rate': 'constant', 'learning_rate_init': 0.01}
```

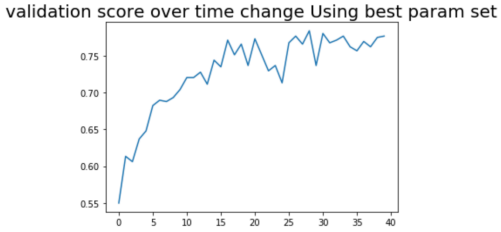
```
0.500 (+/-0.001) for {'hidden_layer_sizes': (1,), 'learning_rate': 'constant', 'learning_rate_init': 0.01}
0.815 (+/-0.030) for {'hidden_layer_sizes': (2,), 'learning_rate': 'constant', 'learning_rate_init': 0.01}
0.801 (+/-0.041) for {'hidden_layer_sizes': (3,), 'learning_rate': 'constant', 'learning_rate_init': 0.01}
0.806 (+/-0.027) for {'hidden_layer_sizes': (4,), 'learning_rate': 'constant', 'learning_rate_init': 0.01}
0.797 (+/-0.061) for {'hidden_layer_sizes': (5,), 'learning_rate': 'constant', 'learning_rate_init': 0.01}
0.804 (+/-0.025) for {'hidden_layer_sizes': (6,), 'learning_rate': 'constant', 'learning_rate_init': 0.01}
0.818 (+/-0.020) for {'hidden_layer_sizes': (7,), 'learning_rate': 'constant', 'learning_rate_init': 0.01}
0.813 (+/-0.019) for {'hidden_layer_sizes': (8,), 'learning_rate': 'constant', 'learning_rate_init': 0.01}
0.807 (+/-0.019) for {'hidden_layer_sizes': (9,), 'learning_rate': 'constant', 'learning_rate_init': 0.01}
0.800 (+/-0.029) for {'hidden_layer_sizes': (10,), 'learning_rate': 'constant', 'learning_rate_init': 0.01}


Best parameters set found on development set based on score recall:

{'hidden_layer_sizes': (7,), 'learning_rate': 'constant', 'learning_rate_init': 0.01}

Over all the optimal parameter setting is:
{'hidden_layer_sizes': [(7,)], 'learning_rate': ['constant'], 'learning_rate_init': [0.01]}
```

(select the single layer with 7 units)

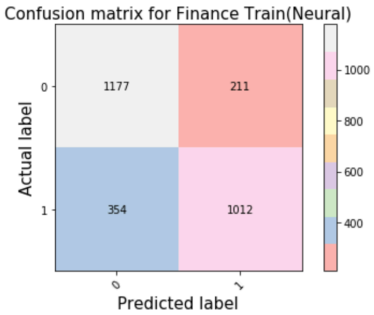
(Training time & Loss Curve & Validation Scores)

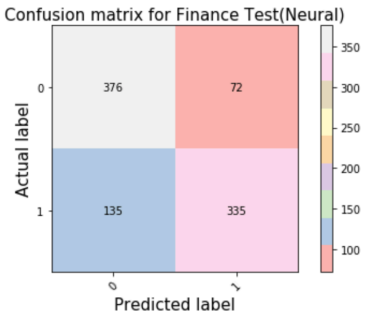
```
Iteration 40, loss = 0.45817180
Validation score: 0.776770
Validation score did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
time for training: 0.26632 s
time for predicting: 0.00122 s
```





(Confusion Matrix)





Training accuracy: 0.7948. Test accuracy: 0.7745

The accuracy does not increase significantly compared with logistic regression method.

Compare & Analysis

I try three methods (linear regression, logistic regression, and neural network) required for each dataset. From the results I get:

1. Linear regression focus on the regression problem. It takes shorter time to train the model, it is suitable for simple dataset due to its linear function (W) can do well and the advantage of saving time. Additionally, overfitting problem occurs less in linear regression.
2. Logistic regression and Neural network are used to solve the classification problems. Neural network has a more complicated structure compared with logistic regression. Even though in this assignment the neural model is the single-layer one, it generally takes longer time than logistic regression to train the model. But it can bring better performance sometimes (like the performance difference dealing with Orbits dataset).
3. The results also show that the performance of neural network sometimes is worse or not better than logistic regression (like what happen is dataset Fifa and Finance). I think this depends on dataset: for some simple dataset, more complicated model like neural network may face overfitting problem. And the simple model can save time and perform well. While, when the dataset is in a complex form, the neural network may show its real power. But obviously it takes more time.