Final Exam Report

2017315051 TANAKA IMANO MUNESATO

2019/06/19

Data Introductions and Goal of Machine Learning

```
# load breaset cancer data
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
print(dir(cancer))
print(cancer.feature_names)
print(cancer.target_names)
```

```
['DESCR', 'data', 'feature_names', 'filename', 'target', 'target_names']
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
['malignant' 'benign']
```

This dataset is breast cancer which provided by Scikit-learn one of default dataset.

feature name ↓

- radius (mean of distances from center to points on the perimeter)
- texture (standard deviation of gray-scale values)
- perimeter
- area
- smoothness (local variation in radius lengths)
- compactness (perimeter^2 / area - 1.0)
- concavity (severity of concave portions of the contour)
- concave points (number of concave portions of the contour)
- symmetry
- fractal dimension ("coastline approximation" - 1)

target name ↓

- malignant
- benign

Model building and experimental process

```
# cancer to DataFrame
import pandas as pd
import numpy as np

data = pd.DataFrame(np.c_[cancer["data"], cancer["target"]], columns = np.append(cancer['feature_names'], ["target"]))
data.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	comp
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.1622	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.1238	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.1444	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.2098	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.1374	

5 rows × 31 columns

We need Cancer data and Target data in one data table.

```
# X: data
X = data.drop(["target"], axis=1)
X.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	wors smoothnes
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	2019.0	0.162
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	1956.0	0.123
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	1709.0	0.144
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	567.7	0.209
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	1575.0	0.137

5 rows × 30 columns

Drop “target” column from data table and named X(variable).

```
#Y: class
y = data["target"].astype(int)
y.head()
```

```
0    0
1    0
2    0
3    0
4    0
Name: target, dtype: int64
```

Get "target" column and named Y(variable).

This table is only for "Target" data. there has 2 type of data. 0 means breast cancer, 1 means non breast cancer.

```
#view columns informations
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
mean radius                569 non-null float64
mean texture               569 non-null float64
mean perimeter             569 non-null float64
mean area                  569 non-null float64
mean smoothness            569 non-null float64
mean compactness           569 non-null float64
mean concavity             569 non-null float64
mean concave points        569 non-null float64
mean symmetry              569 non-null float64
mean fractal dimension     569 non-null float64
radius error               569 non-null float64
texture error              569 non-null float64
perimeter error            569 non-null float64
area error                 569 non-null float64
smoothness error           569 non-null float64
compactness error          569 non-null float64
concavity error            569 non-null float64
concave points error       569 non-null float64
symmetry error             569 non-null float64
fractal dimension error    569 non-null float64
worst radius               569 non-null float64
worst texture              569 non-null float64
worst perimeter            569 non-null float64
worst area                 569 non-null float64
worst smoothness           569 non-null float64
worst compactness          569 non-null float64
worst concavity            569 non-null float64
worst concave points       569 non-null float64
worst symmetry             569 non-null float64
worst fractal dimension    569 non-null float64
dtypes: float64(30)
memory usage: 133.4 KB
```

View X(variable) table information. We have 569 data each column and all value is float.

Split TrainData & TestData

```
from sklearn.model_selection import train_test_split
```

```
#Split by 70:30 (trainData : testData)
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=0)
```

Now we have to separate data. One is for training data and other is for testing data.

train_test_split (Scikit-learn.model_selection) can provide these things.

This time I'm trying to split 7:3 ratio. 70% is for training data and 30% is for testing data.

MLP Performance

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix,f1_score,accuracy_score
```

```
import time;
```

```
mlp = MLPClassifier(hidden_layer_sizes=(1),
                    activation='logistic',
                    solver='lbfgs',
                    tol=1e-4,
                    learning_rate_init=.1,
                    verbose=True,
                    random_state=0)
```

Multiple layer perceptron (neural network) is one of powerful machine learning algorithm.

Because we can choose number of layer and number of nodes by yourself. furthermore, we d on't have to care about units. That's why people call MLP is black box. I felt the possibility of neural networks that's why this time I used this model.

My model

Import MLPClassifier which provided by Scikit-learn.

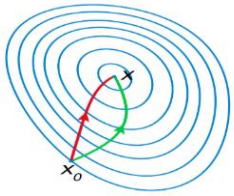
And import confusion_matrix, f1_score, accuracy_score to do performance testing.

- hidden_layer_sizes = 1: default is 1. I'm going to make this value as random number (1to30) because we have 30 input so I thought that 30 was the best number of node s.

- activation = 'logistic': {'identity', 'logistic', 'tanh', 'relu'}, default 'relu'. I choice 'logistic' function because it exits between (0-1) and we can find the slope of the sigmoid curve at any two points.

  Range of 'tanh' is (-1 to 1), 'relu' is (0-infinity), 'identity' is no-op activation.

- solver = 'lbfgs': red line is lbfgs. Green line is gradient descent. Red line (lbfgs) reaching from the initial value to the optimal solution in a short distance.



- tol = 1e-4: when the loss function or score is no improving stop learning.

- learning_rate_init = 0.1: Learning ratio.

- Verbose = True: print progress message to my command line.

```
Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
 2446      3      5      1     0     0   4.794D-06   6.612D-01
  F =  0.66124571639176388

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL

 Cauchy                time 0.000E+00 seconds.
 Subspace minimization time 0.000E+00 seconds.
 Line search           time 0.000E+00 seconds.

 Total User time 0.000E+00 seconds.
```

- random_state = 0: learn same values.

MLP performance hidden1

```
index = 0
mini = 200

start = time.time()

#random nodes (1 ~ 30)
for i in range(1,30):

    mlp.hidden_layer_sizes= (i)
    mlp.fit(X_train,y_train)

    predictions = mlp.predict(X_test)

    #Add FN , FP (a + b)values and conpare to minimum value
    #If FN + FP value is smaller than minimum value , then it can be best classified  of our models

    confusionM = confusion_matrix(y_test, predictions)
    a = confusionM[:1,1:] .reshape(1)
    b = confusionM[1:2,:1] .reshape(1)

    if a+b <= mini:
        mini = a+b
        index = i
        bestMLP1 = mlp
        prediction1 = predictions
        matrix = confusionM

end = time.time()
```

Hidden layer = 1

hidden_layer_size = (i)

Number of nodes (i) is 1 to 30 and find best model. Definition of the best model is if FN (F alse Negative) + FP (Fales Positive) goes minimum number. In this code FN = a, FP = b. And if accuracy close to 1 and f1 score close to 1, our model is best.

We are counting learning time using python library as time.time().

```
print("Hidden layer1 Node : ",index)
print("Learning time : ",end-start)


hidden1Report = pd.DataFrame(matrix)
hidden1Report.index.name = 'True'
hidden1Report.columns.name = "Predicted"
hidden1Report
```

```
Hidden layer1 Node :  29
Learning time :  3.169603109359741
```

Predicted	0	1
True		
0	62	1
1	7	101

```
print("Accuracy score : ",accuracy_score(prediction1,y_test))
```

```
Accuracy score :  0.9532163742690059
```

```
print("F1 score : " ,f1_score(y_test, prediction1, average='macro'))
```

```
F1 score :  0.9506493506493507
```

This case the best number of nodes is 29

Learning time is 3sec.

TP = 62, FN = 1, FP = 7, TN = 101

Accuracy = 0.953...

F1 score = 0.950...

Ok. Let's see another model.

Hidden layer 2

MLP performance hidden2

```
index = 0
index2 = 0
mini = 200

start = time.time()

#random nodes (1 ~ 30)
for i in range(1,30):
    for j in range(1,30):
        mlp.hidden_layer_sizes = (i,j)
        mlp.fit(X_train,y_train)

        predictions = mlp.predict(X_test)

        #Add FN , FP (a + b) values and compare to minimum value
        #If FN + FP value is smaller than minimum value , then it can be best classified  of our models

        confusionM = confusion_matrix(y_test, predictions)
        a = confusionM[:1,1:] .reshape(1)
        b = confusionM[1:2,:1] .reshape(1)

        if a+b <= mini:
            mini = a+b
            index = i
            index2 = j
            bestMLP2 = mlp
            prediction2 = predictions
            matrix = confusionM

end = time.time()
```

Hidden layer = 2

hidden_layer_size = (i, j)

Number of nodes (i, j) are 1 to 30 and find best model. Definition of the best model is if FN (False Negative) + FP (Fales Positive) goes minimum number. In this code FN = a, FP = b. And if accuracy close to 1 and f1 score close to 1, our model is best.

We are counting learning time using python library as time.time().

```
print("Hidden layer1 Nodes : ",index)
print("Hidden layer2 Nodes : ",index2)
print("Learning time : ",end-start)

hidden2Report = pd.DataFrame(matrix)
hidden2Report.index.name = 'True'
hidden2Report.columns.name = "Predicted"
hidden2Report
```

```
Hidden layer1 Nodes :  24
Hidden layer2 Nodes :  27
Learning time :  137.05825781822205
```

Predicted	0	1
True		
0	62	1
1	5	103

```
print("Accuracy score : ",accuracy_score(prediction2,y_test))
```

```
Accuracy score :  0.9649122807017544
```

```
print("F1 score : " ,f1_score(y_test, prediction2, average='macro'))
```

```
F1 score :  0.9627721335268504
```

This case the best number of nodes is hidden layer 1 = 24, hidden layer 2 = 27

Learning time is 137sec

TP = 62, FN = 1, FP = 5, TN = 103

Accuracy = 0.964...

F1 score = 0.962...

Oh, better than hidden layer1. ok let's look another pattern.

Hidden layer 3

MLP performance hidden3

```
mini = 200
index = 0
index2 = 0
index3 = 0

start = time.time()

#random nodes (1 ~ 30)
for i in range(1,30):
    for j in range(1,30):
        for z in range(1,30):

            mlp.hidden_layer_sizes= (i,j,z)

            mlp.fit(X_train,y_train)

            predictions = mlp.predict(X_test)

            #Add FN , FP (a + b)values and compare to minimum value
            #If FN + FP value is smaller than minimum value , then it can be best classified  of our models

            confusionM = confusion_matrix(y_test, predictions)
            a = confusionM[:1,1:] .reshape(1)
            b = confusionM[1:2,:1] .reshape(1)

            if a+b <=mini:
                index = i
                index2 = j
                index3 = z
                mini = a+b
                bestMLP3 = mlp
                prediction3 = predictions
                matrix = confusionM

end = time.time()
```

Hidden layer = 3

hidden_layer_size = (i, j, z)

Number of nodes (i, j, z) are 1 to 30 and find best model. Definition of the best model is if FN (False Negative) + FP (Fales Positive) goes minimum number. In this code FN = a, FP = b. And if accuracy close to 1 and f1 score close to 1, our model is best.

We are counting learning time using python library as time.time().

```
print("Hidden layer1 Nodes : ",index)
print("Hidden layer2 Nodes : ",index2)
print("Hidden layer3 Nodes : ",index3)
print("Learning time : ",end-start)

hidden3Report = pd.DataFrame(matrix)
hidden3Report.index.name = 'True'
hidden3Report.columns.name = "Predicted"
hidden3Report
```

```
Hidden layer1 Nodes :  2
Hidden layer2 Nodes :  22
Hidden layer3 Nodes :  27
Learning time :  5178.96866774559
```

Predicted	0	1
True		
0	62	1
1	3	105

```
print("Accuracy score : ",accuracy_score(prediction3,y_test))
```

```
Accuracy score :  0.9649122807017544
```

```
print("F1 score : " ,f1_score(y_test, prediction3, average='macro'))
```

```
F1 score :  0.9750292056074766
```

This case the best number of nodes is hidden layer1 = 2, hidden layer2 = 22, hidden layer3 = 27

Learning time is 5178sec

TP = 62, FN = 1, FP = 3, TN = 105

Accuracy = 0.976…

F1 score = 0.975…

Wow! Better than hidden layer2 one. Let's look another pattern…

Hold on sec, I notice one thing that Actually, this hidden layer3 model took time. If I increas e hidden layer as node 1 to 30 randomly, it takes 5178 sec * 30 … There is no time to chec k out next model. But I found an alternative idea. At this points our best hidden layer is 3 a nd each layer has nodes. first hidden layer has 2 nodes, second hidden layer has 22 nodes, t hird hidden layer has 27 nodes. we are going to use these hidden layers nodes and add one another hidden layer number 4. Let's upgrade these values for next learning instead.

hidden layer 4

```
mini = 200
index = 0

start = time.time()

#random nodes (1 ~ 30)
for i in range(1,30):
    bestMLP3.hidden_layer_sizes = (2,22,27,i)
    bestMLP3.fit(X_train,y_train)
    predictions = mlp.predict(X_test)

    #Add FN , FP (a + b)values and compare to minimum value
    #If FN + FP value is smaller than minimum value , then it can be best classified  of our models

    confusionM = confusion_matrix(y_test, predictions)
    a = confusionM[:1,1:] .reshape(1)
    b = confusionM[1:2,:1] .reshape(1)

    if a+b <=mini:
        index = i
        mini = a+b
        bestMLP4 = mlp
        prediction4 = predictions
        matrix = confusionM

end = time.time()
```

Hidden layer = 4

hidden_layer_size = (2, 22, 27, i)

This case I used our best models hidden layer3 nodes number.

Number of nodes (i) is 1-30 and find best model. Definition of the best model is if FN (False Negative) + FP (Fales Positive) goes minimum number. In this code FN = a, FP = b. And if accuracy close to 1 and f1 score close to 1, our model is best.

We are counting learning time using python library as time.time().

```
print("Hidden layer4 Nodes : ",index)
print("Learning time : ",end-start)

hidden4Report = pd.DataFrame(matrix)
hidden4Report.index.name = 'True'
hidden4Report.columns.name = "Predicted"
hidden4Report
```

```
Hidden layer4 Nodes :  14
Learning time :  6.064229488372803
```

Predicted	0	1
True		
0	57	6
1	2	106

```
print("Accuracy score : ",accuracy_score(prediction4,y_test))
```

```
Accuracy score :  0.9532163742690059
```

```
print("F1 score : " ,f1_score(y_test, prediction4, average='macro'))
```

```
F1 score :  0.9490312965722802
```

This case the best number of nodes is of course hidden layer1 = 2, hidden layer2 = 22, hidd en layer3 = 27 and hidden layer4 = 14

Learning time is 6sec.

TP = 57, FN =6, FP = 2, TN = 106

Accuracy = 0.952...

F1 score = 0.949...

Umm. Accuracy and F1 score are decreased. How about five hidden layers?

hidden layer 5

```
mini = 200
index = 0

start = time.time()

#random nodes (1 ~ 30)
for i in range(1,30):
    bestMLP4.hidden_layer_sizes = (2,22,27,29,14,i)
    bestMLP4.fit(X_train,y_train)
    predictions = mlp.predict(X_test)

    #Add FN , FP (a + b)values and compare to minimum value
    #If FN + FP value is smaller than minimum value , then it can be best classified  of our models

    confusionM = confusion_matrix(y_test, predictions)
    a = confusionM[:1,1:] .reshape(1)
    b = confusionM[1:2,:1] .reshape(1)

    if a+b <=mini:
        index = i
        mini = a+b
        bestMLP5 = mlp
        prediction5 = predictions
        matrix = confusionM

end = time.time()
```

Hidden layer = 5

hidden_layer_size = (2, 22, 27, 14, i)

This case I used our best models hidden layer4 nodes number.

Number of nodes (i) is 1-30 randomly and find best model. Definition of the best model is if FN (False Negative) + FP (Fales Positive) goes minimum number. In this code FN = a, FP = b. And if accuracy close to 1 and f1 score close to 1, our model is best.

We are counting learning time using python library as time.time().

```
print("Hidden layer5 Nodes : ",index)
print("Learning time : ",end-start)

hidden4Report = pd.DataFrame(matrix)
hidden4Report.index.name = 'True'
hidden4Report.columns.name = "Predicted"
hidden4Report
```

```
Hidden layer5 Nodes :  29
Learning time :  0.887709379196167
```

Predicted	0	1
True		
0	0	63
1	0	108

```
print("Accuracy score : ",accuracy_score(prediction5,y_test))
```

```
Accuracy score :  0.631578947368421
```

```
print("F1 score : " ,f1_score(y_test, prediction5, average='macro'))
```

```
F1 score :  0.3870967741935484
```

This case the best number of nodes is hidden layer1 = 2, hidden layer2 = 22, hidden layer3 = 27, hidden layer4 = 14, hidden layer5 = 29.

Learning time is 0.8sec.

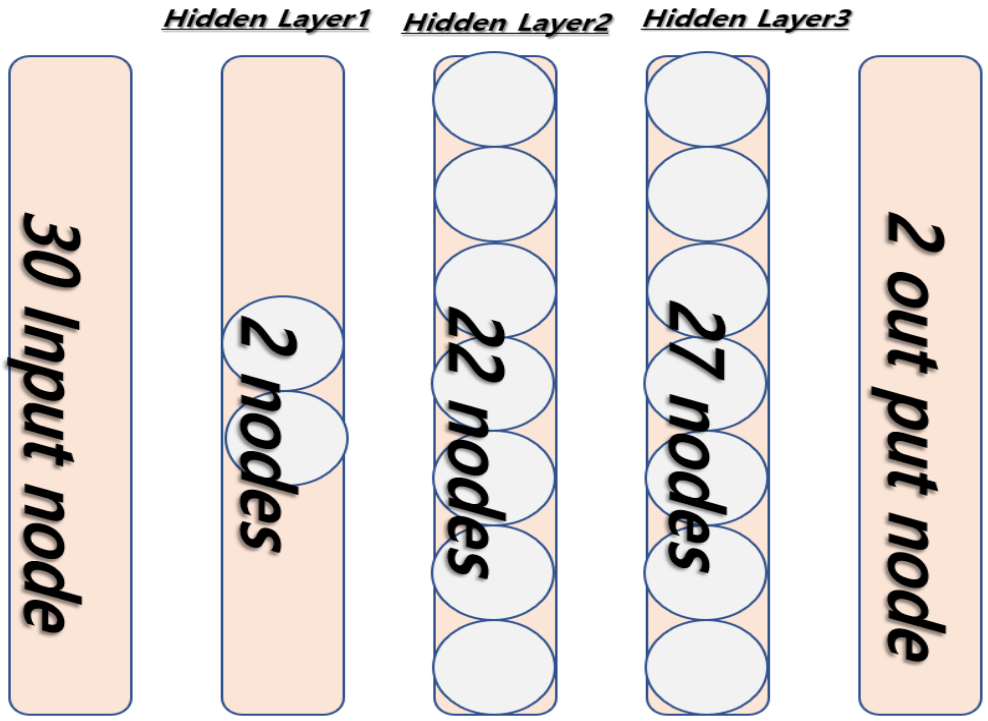
TP = 0, FN = 63, FT = 0, FN = 108

Accuracy = 0.631...

F1 score = 0.387...

Oh no. this model is worst.

Final model and performance



Predicted	0	1
True		
0	62	1
1	3	105

```
print("Accracy score : ",accuracy_score(prediction3,y_test))
```

```
Accracy score :  0.9766081871345029
```

```
print("F1 score : " ,f1_score(y_test, prediction3, average='macro'))
```

```
F1 score :  0.9750292056074766
```

Compare to other models, this three hidden layer model is top of accuracy and f1 score in o ur models. thus, **the best our model is three hidden layers. accuracy is 0.976, f1 scor e is 0.975.**

Conclusion

It turns out that increasing the number of layers does not always lead to good results. On the contrary, I thought that only the god knew what model could be classified correctly. But this time I challenged with a neural network to get close to the model.

A neural network is a combination of many layers and nodes, and it takes more time as the number of layers and nodes increases when learning. I can decide as I want, but how exactly is the model so good, are there any other good patterns? In order to know, I had to do it at random. I actually felt the limit in having machine learning on my own personal computer. Because I have to spend a lot of time. However, it was very interesting because I could try many patterns in machine learning for the first time. furthermore, this is my first time to write report in English. It may not be an appropriate statement but it was a good experience.