Project 2 Breast Cancer Wisconsin Data Set

by Yuheng Lin for B455 Principles of Machine Learning

Import the Packages

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.model_selection import ShuffleSplit
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_score, learning_curve
```

Read Breast Cancer Wisconsin Data From UCI Learning

```
patient_data = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-
                          names = ["ID", "diagnosis",
                                  "radius_mean", "texture_mean", "perimeter_mean", "area-mean", "sm
                                  "radius_se", "texture_se", "perimeter_se", "area-se", "smoothness
                                  "radius_worst", "texture_worst", "perimeter_worst", "area-worst",
```

Check the Data

By using head()

```
patient_data.head()
```

	ID	diagnosis	radius_mean	texture_mean	perimeter_mean	area-mean	smoothness_
0	842302	M	17.99	10.38	122.80	1001.0	0.1
1	842517	M	20.57	17.77	132.90	1326.0	0.0
2	84300903	M	19.69	21.25	130.00	1203.0	0.1
3	84348301	M	11.42	20.38	77.58	386.1	0.1
4	84358402	M	20.29	14.34	135.10	1297.0	0.1

5 rows × 32 columns

Check the Info of Our Data

```
print(patient_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
ID                         569 non-null int64
diagnosis                  569 non-null object
radius_mean                569 non-null float64
texture_mean               569 non-null float64
perimeter_mean             569 non-null float64
area-mean                  569 non-null float64
smoothness_mean            569 non-null float64
compactness_mean           569 non-null float64
concavity_mean             569 non-null float64
concave_points_mean        569 non-null float64
symmetry_mean              569 non-null float64
fractal_dimension_mean     569 non-null float64
radius_se                  569 non-null float64
texture_se                 569 non-null float64
perimeter_se               569 non-null float64
area-se                    569 non-null float64
smoothness_se              569 non-null float64
compactness_se             569 non-null float64
concavity_se               569 non-null float64
concave_points_se          569 non-null float64
symmetry_se                569 non-null float64
fractal_dimension_se       569 non-null float64
radius_worst               569 non-null float64
texture_worst              569 non-null float64
perimeter_worst            569 non-null float64
area-worst                 569 non-null float64
smoothness_worst           569 non-null float64
compactness_worst          569 non-null float64
concavity_worst            569 non-null float64
concave_points_worst       569 non-null float64
symmetry_worst             569 non-null float64
fractal_dimension_worst    569 non-null float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.3+ KB
```

Set Our Inputs and Outputs

We drop the ID and diagnosis in our data as our inputs

Diagnosis will be our outputs

```
inputs = patient_data.drop('ID', axis=1).drop('diagnosis', axis=1)
outputs = patient_data['diagnosis']
```

Randomly Select Train and Test Inputs and Outputs

```
inputs_train, inputs_test, outputs_train, outputs_test = train_test_split(inputs, outputs)
```

Train the Model With 5 Hidden Layer With 2000 Max Iterations

```
BC_mlp = MLPClassifier(hidden_layer_sizes=(30, 30, 30, 30, 30), max_iter=2000)
BC_mlp.fit(inputs_train, outputs_train)
```

Check the Accuracy of Our MLP by Random selections of Inputs and Outputs

```
mlp_prediction = BC_mlp.predict(inputs_test)
print(classification_report(outputs_test,mlp_prediction))
```

```
              precision    recall  f1-score   support

           B       0.94      0.95      0.94        95
           M       0.89      0.88      0.88        48

   micro avg       0.92      0.92      0.92       143
   macro avg       0.92      0.91      0.91       143
weighted avg       0.92      0.92      0.92       143
```

Check the Accuracy of Our MLP by Using 5-Fold Cross Validation

```
five_fold_mlp_accuracy = cross_val_score(BC_mlp, inputs, outputs, cv=5)
print(five_fold_mlp_accuracy)
```

```
[0.93043478 0.93043478 0.9380531  0.92035398 0.91150442]
```

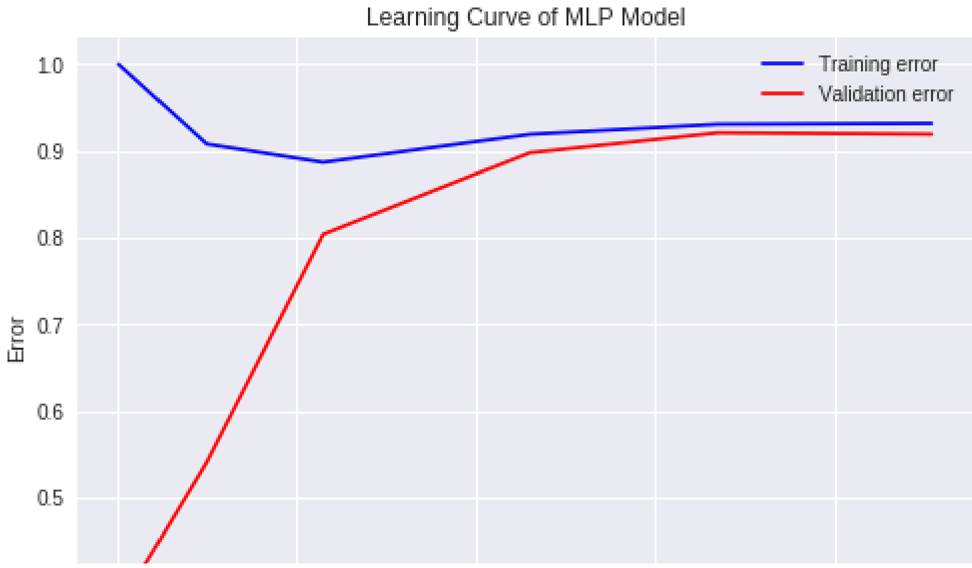
Conclusion for Model Parameters for Our MLP

In the mlp model, I used 5 hidden layers with 2000 iterations. The accuracy of the mlp model is good and around 0.92. And the learning curve of mlp model converfent. As result, using 5 hidden layers for the mlp model is valid and suitable.

Plot the Learning Curve of the Model

```
train_sizes = [1, 50, 115, 230, 335, 454]
def plot_learning_curve(estimator, title, inputs, outputs, train_sizes, cv):
  plt.figure()
  plt.title(title)
  train_sizes, train_error, validation_error = learning_curve(estimator, inputs, outputs, train_siz
  train_error_mean = train_error.mean(axis=1)
  validation_error_mean = validation_error.mean(axis=1)
  plt.xlabel('Training set size')
  plt.ylabel('Error')
  plt.plot(train_sizes, train_error_mean, color = 'b', label = 'Training error')
  plt.plot(train_sizes, validation_error_mean, color = 'r', label = 'Validation error')
  plt.legend(loc="best")
  return plt

plot_learning_curve(BC_mlp, "Learning Curve of MLP Model", inputs, outputs, train_sizes, 5)
plt.show()
```



Learning Curve Conclusion

From the graph above, we can see that the train and validation error both converge and they are around 0.92 to 0.95. And training error is higher than validation error.

Build the SVM for the Same Train and Test Data

Using **SVC** from the **sklearn** package

```
BC_svm = SVC(kernel='linear')
BC_svm.fit(inputs_train, outputs_train)
```

Check the Accuracy of Our SVM Model by Using Random Selections of Inputs

```
svm_prediction = BC_svm.predict(inputs_test)
print(classification_report(outputs_test,svm_prediction))
```

```
              precision    recall  f1-score   support

           B       0.95      0.98      0.96        95
           M       0.96      0.90      0.92        48

   micro avg       0.95      0.95      0.95       143
   macro avg       0.95      0.94      0.94       143
weighted avg       0.95      0.95      0.95       143
```

Check the Accuracy of Our SVM by Using 5-Fold Cross Validation

```
five_fold_svm_accuracy = cross_val_score(BC_svm, inputs, outputs, cv=5)
print(five_fold_svm_accuracy)
```

```
[0.94782609 0.93043478 0.97345133 0.92035398 0.95575221]
```

Conclusion

We can see that the average accuracy of mlp is around 0.92. And the average accuracy of svm is around 0.95. So, the accuracy of svm is higher. By running the code, svm takes a little longer to run than mlp.