Scikit-learn exploration

Made by:

- Annisa Rahim / 13518089
- Stefanus Gusega Gunawan / 13518149

Prepare a dict to record the evaluation of each model

In [2]:
```
import pandas as pd

# initialise data of lists.
data = {'algo':['DecisionTreeClassifier', 'Id3Estimator', 'KMeans','Logreg','Logreg_Std
               'NN_StdScaler','NN_MinMaxScaler','SVM'],
        'acc_train':[],
        'acc_valid':[],
        'f1_train':[],
        'f1_valid':[]
       }
```

Breast-Cancer Dataset

Load the datasets first.

In [3]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

load = load_breast_cancer()
cancer = pd.DataFrame(load.data, columns=load.feature_names)
target = pd.DataFrame(load.target, columns =['target'])

# Split 80 : 20
X_train, X_valid, y_train, y_valid = train_test_split(cancer, target, train_size = 0.8,
X_valid
```

Out[3]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	dim
421	14.690	13.98	98.22	656.1	0.10310	0.18360	0.14500	0.063000	0.2086	(
47	13.170	18.66	85.98	534.6	0.11580	0.12310	0.12260	0.073400	0.2128	(
292	12.950	16.02	83.14	513.7	0.10050	0.07943	0.06155	0.033700	0.1730	(
186	18.310	18.58	118.60	1041.0	0.08588	0.08468	0.08169	0.058140	0.1621	(
414	15.130	29.81	96.71	719.5	0.08320	0.04605	0.04686	0.027390	0.1852	(
132	16.160	21.54	106.20	809.8	0.10080	0.12840	0.10430	0.056130	0.2160	(
161	19.190	15.94	126.30	1157.0	0.08694	0.11850	0.11930	0.096670	0.1741	(

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	dim
197	18.080	21.84	117.40	1024.0	0.07371	0.08642	0.11030	0.057780	0.1770	(
245	10.480	19.86	66.72	337.7	0.10700	0.05971	0.04831	0.030700	0.1737	(
453	14.530	13.98	93.86	644.2	0.10990	0.09242	0.06895	0.064950	0.1650	(
411	11.040	16.83	70.92	373.2	0.10770	0.07804	0.03046	0.024800	0.1714	(
214	14.190	23.81	92.87	610.7	0.09463	0.13060	0.11150	0.064620	0.2235	(
283	16.240	18.77	108.80	805.1	0.10660	0.18020	0.19480	0.090520	0.1876	(
107	12.360	18.54	79.01	466.7	0.08477	0.06815	0.02643	0.019210	0.1602	(
542	14.740	25.42	94.70	668.6	0.08275	0.07214	0.04105	0.030270	0.1840	(
518	12.880	18.22	84.45	493.1	0.12180	0.16610	0.04825	0.053030	0.1709	(
324	12.200	15.21	78.01	457.9	0.08673	0.06545	0.01994	0.016920	0.1638	(
488	11.680	16.17	75.49	420.5	0.11280	0.09263	0.04279	0.031320	0.1853	(
376	10.570	20.22	70.15	338.3	0.09073	0.16600	0.22800	0.059410	0.2188	(
237	20.480	21.46	132.50	1306.0	0.08355	0.08348	0.09042	0.060220	0.1467	(
362	12.760	18.84	81.87	496.6	0.09676	0.07952	0.02688	0.017810	0.1759	(
420	11.570	19.04	74.20	409.7	0.08546	0.07722	0.05485	0.014280	0.2031	(
451	19.590	25.00	127.70	1191.0	0.10320	0.09871	0.16550	0.090630	0.1663	(
519	12.750	16.70	82.51	493.8	0.11250	0.11170	0.03880	0.029950	0.2120	(
65	14.780	23.94	97.40	668.3	0.11720	0.14790	0.12670	0.090290	0.1953	(
242	11.300	18.19	73.93	389.4	0.09592	0.13250	0.15480	0.028540	0.2054	(
558	14.590	22.68	96.39	657.1	0.08473	0.13300	0.10290	0.037360	0.1454	(
85	18.460	18.52	121.10	1075.0	0.09874	0.10530	0.13350	0.087950	0.2132	(
180	27.220	21.87	182.10	2250.0	0.10940	0.19140	0.28710	0.187800	0.1800	(
207	17.010	20.26	109.70	904.3	0.08772	0.07304	0.06950	0.053900	0.2026	(
...	...	...	...	...	...	...	...	...	...	
403	12.940	16.17	83.18	507.6	0.09879	0.08836	0.03296	0.023900	0.1735	(
120	11.410	10.82	73.34	403.3	0.09373	0.06685	0.03512	0.026230	0.1667	(
501	13.820	24.49	92.33	595.9	0.11620	0.16810	0.13570	0.067590	0.2275	(
545	13.620	23.23	87.19	573.2	0.09246	0.06747	0.02974	0.024430	0.1664	(
62	14.250	22.15	96.42	645.7	0.10490	0.20080	0.21350	0.086530	0.1949	(
344	11.710	15.45	75.03	420.3	0.11500	0.07281	0.04006	0.032500	0.2009	(
457	13.210	25.25	84.10	537.9	0.08791	0.05205	0.02772	0.020680	0.1619	(
31	11.840	18.70	77.93	440.6	0.11090	0.15160	0.12180	0.051820	0.2301	(

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	dim
555	10.290	27.61	65.67	321.4	0.09030	0.07658	0.05999	0.027380	0.1593	(
443	10.570	18.32	66.82	340.9	0.08142	0.04462	0.01993	0.011110	0.2372	(
400	17.910	21.02	124.40	994.0	0.12300	0.25760	0.31890	0.119800	0.2113	(
5	12.450	15.70	82.57	477.1	0.12780	0.17000	0.15780	0.080890	0.2087	(
59	8.618	11.79	54.34	224.5	0.09752	0.05272	0.02061	0.007799	0.1683	(
496	12.650	18.17	82.69	485.6	0.10760	0.13340	0.08017	0.050740	0.1641	(
289	11.370	18.89	72.17	396.0	0.08713	0.05008	0.02399	0.021730	0.2013	(
346	12.060	18.90	76.66	445.3	0.08386	0.05794	0.00751	0.008488	0.1555	(
531	11.670	20.02	75.21	416.2	0.10160	0.09453	0.04200	0.021570	0.1859	(
305	11.600	24.49	74.23	417.2	0.07474	0.05688	0.01974	0.013130	0.1935	(
425	10.030	21.28	63.19	307.3	0.08117	0.03912	0.00247	0.005159	0.1630	(
347	14.760	14.74	94.87	668.7	0.08875	0.07780	0.04608	0.035280	0.1521	(
462	14.400	26.99	92.25	646.1	0.06995	0.05223	0.03476	0.017370	0.1707	(
165	14.970	19.76	95.50	690.2	0.08421	0.05352	0.01947	0.019390	0.1515	(
550	10.860	21.48	68.51	360.5	0.07431	0.04227	0.00000	0.000000	0.1661	(
295	13.770	13.27	88.06	582.7	0.09198	0.06221	0.01063	0.019170	0.1592	(
119	17.950	20.01	114.20	982.0	0.08402	0.06722	0.07293	0.055960	0.2129	(
172	15.460	11.89	102.50	736.9	0.12570	0.15550	0.20320	0.109700	0.1966	(
3	11.420	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.105200	0.2597	(
68	9.029	17.33	58.79	250.5	0.10660	0.14130	0.31300	0.043750	0.2111	(
448	14.530	19.34	94.25	659.7	0.08388	0.07800	0.08817	0.029250	0.1473	(
442	13.780	15.79	88.37	585.9	0.08817	0.06718	0.01055	0.009937	0.1405	(

114 rows × 30 columns

Check whether there is missing values.

In [4]:

```
cols_missing = [col for col in cancer.columns if (cancer[col].isnull().any())]
cols_missing
```

Out[4]: []

Train datasets

DecisionTreeClassifier

In [5]:
```
from sklearn.tree import DecisionTreeClassifier

classifier = DecisionTreeClassifier(random_state=1)
classifier.fit(X_train, y_train)
```

Out[5]:
```
DecisionTreeClassifier(random_state=1)
```

Print the tree.

In [6]:
```
from sklearn.tree import export_text

tree = export_text(classifier, feature_names = list(cancer.columns))
print(tree)
```

```
|--- worst perimeter <= 106.05
|   |--- worst concave points <= 0.16
|   |   |--- worst concave points <= 0.14
|   |   |   |--- area error <= 48.98
|   |   |   |   |--- class: 1
|   |   |   |--- area error >  48.98
|   |   |   |   |--- worst radius <= 13.55
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- worst radius >  13.55
|   |   |   |   |   |--- class: 1
|   |   |--- worst concave points >  0.14
|   |   |   |--- worst texture <= 29.45
|   |   |   |   |--- class: 1
|   |   |   |--- worst texture >  29.45
|   |   |   |   |--- class: 0
|   |--- worst concave points >  0.16
|   |   |--- worst texture <= 24.78
|   |   |   |--- class: 1
|   |   |--- worst texture >  24.78
|   |   |   |--- class: 0
|--- worst perimeter >  106.05
|   |--- worst texture <= 20.65
|   |   |--- worst perimeter <= 116.80
|   |   |   |--- class: 1
|   |   |--- worst perimeter >  116.80
|   |   |   |--- mean smoothness <= 0.08
|   |   |   |   |--- class: 1
|   |   |   |--- mean smoothness >  0.08
|   |   |   |   |--- class: 0
|   |--- worst texture >  20.65
|   |   |--- mean concave points <= 0.05
|   |   |   |--- concave points error <= 0.01
|   |   |   |   |--- class: 0
|   |   |   |--- concave points error >  0.01
|   |   |   |   |--- class: 1
|   |   |--- mean concave points >  0.05
|   |   |   |--- mean smoothness <= 0.08
|   |   |   |   |--- class: 1
|   |   |   |--- mean smoothness >  0.08
|   |   |   |   |--- class: 0
```

Give prediction to training data so we can see how good the model is. Compared to true value of training data.

In [7]:
```
predict_train = classifier.predict(X_train)
```

Evaluate the model with Accuracy Metrics.

$$Accuracy = \frac{NumberofCorrectpredictions}{Totalnumberofpredictionsmade}$$

In [8]:
```
from sklearn.metrics import accuracy_score

acc_train_score = accuracy_score(y_train, predict_train)

# insert to dict
data['acc_train'].append(acc_train_score)

acc_train_score
```

Out[8]: 1.0

Evaluate the model with F1 score Metrics. The metrics used based on this confusion matrix. F1 Score is the ***Harmonic Mean*** between precision and recall.

		Predicted	
		Negative	**Positive**
Actual	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$

$$F1 = 2 * (\frac{1}{\frac{1}{Precision} + \frac{1}{Recall}}) = 2 * (\frac{TruePositives}{2TruePositives + FalsePositives + FalseNegatives})$$

In [9]:
```
from sklearn.metrics import f1_score

f1_train_score = f1_score(y_train, predict_train)

# insert to dict
data['f1_train'].append(f1_train_score)

f1_train_score
```

Out[9]: 1.0

Now predict the test data.

In [10]:
```
predict_valid = classifier.predict(X_valid)
```

Evaluate the model with Accuracy Metrics.

In [11]:
```
acc_valid_score = accuracy_score(y_valid, predict_valid)

# insert to dict
data['acc_valid'].append(acc_valid_score)

acc_valid_score
```

Out[11]:
```
0.9473684210526315
```

Evaluate the model with F1 Metrics

In [12]:
```
f1_valid_score = f1_score(y_valid, predict_valid)

# insert to dict
data['f1_valid'].append(f1_valid_score)

f1_valid_score
```

Out[12]:
```
0.9594594594594595
```

Id3Estimator

Train the model and print the tree.

In [13]:
```
import six
import sys
sys.modules['sklearn.externals.six'] = six

from id3 import Id3Estimator, export

id3_model1 = Id3Estimator()

id3_model1.fit(X_train, y_train.values.ravel()) # make it numpy 1D array, not a df

# export text
tree_text1 = export.export_text(id3_model1.tree_, feature_names=cancer.columns)

print(tree_text1)
```

```
worst perimeter <=105.15
|   worst concave points <=0.14
|   |   radius error <=0.64: 1 (249)
|   |   radius error >0.64
|   |   |   mean radius <=12.27: 0 (1)
|   |   |   mean radius >12.27: 1 (2)
|   worst concave points >0.14
|   |   worst texture <=25.94: 1 (6)
|   |   worst texture >25.94
|   |   |   mean compactness <=0.12
|   |   |   |   mean radius <=14.06: 1 (2)
|   |   |   |   mean radius >14.06: 0 (1)
|   |   |   mean compactness >0.12: 0 (5)
worst perimeter >105.15
```

```
|   worst concave points <=0.15
|   |   worst texture <=19.91: 1 (13)
|   |   worst texture >19.91
|   |   |   worst radius <=16.80
|   |   |   |   mean smoothness <=0.09: 1 (8)
|   |   |   |   mean smoothness >0.09
|   |   |   |   |   smoothness error <=0.00: 1 (2)
|   |   |   |   |   smoothness error >0.00: 0 (5)
|   |   |   worst radius >16.80
|   |   |   |   worst concavity <=0.21
|   |   |   |   |   mean texture <=21.26: 1 (2)
|   |   |   |   |   mean texture >21.26: 0 (2)
|   |   |   |   worst concavity >0.21: 0 (21)
|   worst concave points >0.15
|   |   mean texture <=15.35
|   |   |   mean radius <=14.89: 1 (1)
|   |   |   mean radius >14.89: 0 (3)
|   |   mean texture >15.35: 0 (132)
```

Evaluate the model with Accuracy and F1 score metrics.

In [14]:

```
# Model Evaluation
predict_train1 = id3_model1.predict(X_train)

print("Accuracy: ", accuracy_score(predict_train1, y_train))
print("F1 score: ", f1_score(predict_train1, y_train))

# insert to dict
data['acc_train'].append(accuracy_score(predict_train1, y_train))
# insert to dict
data['f1_train'].append(f1_score(predict_train1, y_train))
```

```
Accuracy:  1.0
F1 score:  1.0
```

Evaluate the model to test data with Accuracy and F1 score metrics.

In [15]:

```
# Model Prediction
predict_test1 = id3_model1.predict(X_valid)

print("Accuracy: ", accuracy_score(predict_test1,y_valid))
print("F1 score: ", f1_score(predict_test1,y_valid))

# insert to dict
data['acc_valid'].append(accuracy_score(predict_test1,y_valid))
# insert to dict
data['f1_valid'].append(f1_score(predict_test1,y_valid))
```

```
Accuracy:  0.9385964912280702
F1 score:  0.953020134228188
```

KMeans

Train the datasets.

In [16]:

```
from sklearn.cluster import KMeans

# Determine how many clusters
n_clust = y_train['target'].nunique()
```

```
kmeans_model_1 = KMeans(n_clusters=n_clust,random_state=1)
kmeans_model_1.fit(X_train)
```

Out[16]:
```
KMeans(n_clusters=2, random_state=1)
```

Predict the data train and data test.

In [17]:
```
kmeans_predict_train = kmeans_model_1.predict(X_train)
kmeans_predict_test = kmeans_model_1.predict(X_valid)
print(kmeans_predict_train)
print(y_train)
```

```
[1 1 0 0 0 0 0 1 1 0 0 1 0 0 1 1 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0
 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1
 0 0 1 1 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 1 0 0 0 0 0 0 0 0
 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0 1 1 0
 0 0 1 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 1 0 0 0 1 1 0 0 1
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 0
 0 0 0 1 0 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1
 1 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1
 1 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 1 0
 1 0 0 0 0 0 1 0 1 0 0]
     target
408       0
4         0
307       1
386       1
404       1
434       1
19        1
517       0
535       0
445       1
554       1
236       0
117       0
157       1
162       0
78        0
409       1
484       1
334       1
42        0
173       1
223       0
201       0
133       1
232       1
413       1
514       0
244       0
415       1
562       0
..      ...
264       0
209       1
316       1
513       1
```

```
313        1
534        1
319        1
7          0
393        0
141        0
86         0
478        1
503        0
215        0
398        1
490        1
252        0
468        0
357        1
254        0
276        1
178        1
281        1
390        1
508        1
129        0
144        1
72         0
235        1
37         1

[455 rows x 1 columns]
```

Define a function to switch the value from 0 to 1

In [18]:

```python
import numpy as np
def switch(x: int) -> int:
    if (x==0):
        return x+1
    elif (x==1):
        return x-1
```

For evaluation with training data.

In [19]:

```python
# Find out each cluster belong with which class

# First clusterization using first prediction (k_means_train)
# Calculate the accuracy and F1
acc_train = accuracy_score(y_train, kmeans_predict_train)
f1_train = f1_score(y_train, kmeans_predict_train)
print(f"First clusterization accuracy: {acc_train}")
print(f"First clusterization F1: {f1_train}")

# Second clusterization: switch 0 to 1
switched_train = pd.Series(kmeans_predict_train)
fix = switched_train.apply(switch)
acc_train1= accuracy_score(y_train, fix)
f1_train2= f1_score(y_train, fix)
print(f"Second clusterization accuracy: {acc_train1}")
print(f"Second clusterization F1: {f1_train2}")
```

```
First clusterization accuracy: 0.15164835164835164
First clusterization F1: 0.005154639175257732
Second clusterization accuracy: 0.8483516483516483
Second clusterization F1: 0.891679748822606
```

In [20]:
```
# insert to dict
data['acc_train'].append(acc_train1)
# insert to dict
data['f1_train'].append(f1_train2)
```

The performance can be seen by the greatest metrics score. So the cluster 0 belongs to class of 1, cluster 1 belongs to class of 0. Now, for evaluation with validation data (datatest).

In [21]:
```
# First clusterization
acc_valid = accuracy_score(y_valid, kmeans_predict_test)
f1_valid = f1_score(y_valid, kmeans_predict_test)
print(f"First clusterization accuracy: {acc_valid}")
print(f"First clusterization F1: {f1_train}")

# Second clusterization: switch 0 to 1
switched_valid = pd.Series(kmeans_predict_test).apply(switch)
acc_valid_switch = accuracy_score(y_valid, switched_valid)
f1_valid_switch = f1_score(y_valid, switched_valid)
print(f"Second clusterization accuracy: {acc_valid_switch}")
print(f"Second clusterization F1: {f1_valid_switch}")
```

```
First clusterization accuracy: 0.19298245614035087
First clusterization F1: 0.005154639175257732
Second clusterization accuracy: 0.8070175438596491
Second clusterization F1: 0.8674698795180723
```

In [22]:
```
# insert to dict
data['acc_valid'].append(acc_valid_switch)
# insert to dict
data['f1_valid'].append(f1_valid_switch)
```

Same as when we evaluate with training data, the second clusterization is better.

LogisticRegression

In [23]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score

model_1 = LogisticRegression(max_iter=2500, random_state=1)

model_1.fit(X_train,y_train.values.ravel())
```

Out[23]:
```
LogisticRegression(max_iter=2500, random_state=1)
```

In [24]:
```
predict_train = model_1.predict(X_train)

acc = accuracy_score(y_train, predict_train)
f1 = f1_score(y_train, predict_train)

print("Training Data with only 2500 maximum iteration")
print("Accuracy: ", acc)
print("F1 score: ", f1)

# insert to dict
data['acc_train'].append(acc)
```

```
# insert to dict
data['f1_train'].append(f1)
```

```
Training Data with only 2500 maximum iteration
Accuracy:  0.9604395604395605
F1 score:  0.9685314685314685
```

In [25]:

```
predict_test = model_1.predict(X_valid)

acc = accuracy_score(y_valid, predict_test)
f1 = f1_score(y_valid, predict_test)

print("Test Data with only 2500 maximum iteration")
print("Accuracy: ", acc)
print("F1 score: ", f1)

# insert to dict
data['acc_valid'].append(acc)
# insert to dict
data['f1_valid'].append(f1)
```

```
Test Data with only 2500 maximum iteration
Accuracy:  0.9473684210526315
F1 score:  0.9594594594594595
```

Scaling the dataset (using Standard Scaler)

In [26]:

```
# Scaling
from sklearn.preprocessing import StandardScaler, MinMaxScaler
scaler = StandardScaler()

scaled_X_train = scaler.fit_transform(X_train)
scaled_X_valid = scaler.fit_transform(X_valid)
```

Modelling the scaled dataset.

In [27]:

```
model_2 = LogisticRegression(max_iter=2500, random_state=1)
model_2.fit(scaled_X_train,y_train.values.ravel())
```

Out[27]:

```
LogisticRegression(max_iter=2500, random_state=1)
```

Evaluate the model to train data with Accuracy and F1 score metrics.

In [28]:

```
predict_train = model_2.predict(scaled_X_train)

acc = accuracy_score(y_train, predict_train)
f1 = f1_score(y_train, predict_train)

print("Training Data with using max 2500 iteration and StandardScaler")
print("Accuracy: ", acc)
print("F1 score: ", f1)

# insert to dict
data['acc_train'].append(acc)
# insert to dict
data['f1_train'].append(f1)
```

```
Training Data with using max 2500 iteration and StandardScaler
Accuracy:  0.9912087912087912
F1 score:  0.993006993006993
```

Evaluate the model to test data with Accuracy and F1 score metrics.

In [29]:

```
predict_test = model_2.predict(scaled_X_valid)

acc = accuracy_score(y_valid, predict_test)
f1 = f1_score(y_valid, predict_test)

print("Test Data with using max 2500 iteration and StandardScaler")
print("Accuracy: ", acc)
print("F1 score: ", f1)

# insert to dict
data['acc_valid'].append(acc)
# insert to dict
data['f1_valid'].append(f1)
```

```
Test Data with using max 2500 iteration and StandardScaler
Accuracy:  0.9736842105263158
F1 score:  0.9793103448275863
```

Scaling the data (Using MinMax Scaler)

In [30]:

```
scaler = MinMaxScaler()

scaled_X_train = scaler.fit_transform(X_train)
scaled_X_valid = scaler.fit_transform(X_valid)

model_3 = LogisticRegression(max_iter=2500, random_state=1)
model_3.fit(scaled_X_train,y_train.values.ravel())
```

Out[30]:

```
LogisticRegression(max_iter=2500, random_state=1)
```

In [31]:

```
predict_train = model_3.predict(scaled_X_train)

acc = accuracy_score(y_train, predict_train)
f1 = f1_score(y_train, predict_train)

print("Training Data with using max 2500 iteration and MinMaxScaler")
print("Accuracy: ", acc)
print("F1 score: ", f1)

# insert to dict
data['acc_train'].append(acc)
# insert to dict
data['f1_train'].append(f1)
```

```
Training Data with using max 2500 iteration and MinMaxScaler
Accuracy:  0.9626373626373627
F1 score:  0.9709401709401708
```

In [32]:

```
predict_test = model_3.predict(scaled_X_valid)

acc = accuracy_score(y_valid, predict_test)
f1 = f1_score(y_valid, predict_test)
```

```
print("Test Data with using max 2500 iteration and MinMaxScaler")
print("Accuracy: ", acc)
print("F1 score: ", f1)

# insert to dict
data['acc_valid'].append(acc)
# insert to dict
data['f1_valid'].append(f1)
```

```
Test Data with using max 2500 iteration and MinMaxScaler
Accuracy:  0.9736842105263158
F1 score:  0.9795918367346939
```

Conclusion

This data is better with MinMaxScaler because the metrics of the validation dataset bigger than the metrics of the training dataset, even this is a unusual condition. Possible answer: the data was divided into two imbalance dataset, so the training data harder to solve, and the validation data easier to solve. So, we still concluded that with MinMaxScaler is better, because it proves that model is not overfit.

Neural_network

Define the algorithm

In [33]:
```
from sklearn.neural_network import MLPClassifier

model_1 = MLPClassifier(max_iter = 700, random_state=1)

model_1.fit(X_train,y_train.values.ravel())
```

Out[33]:
```
MLPClassifier(max_iter=700, random_state=1)
```

Predict with training data

In [34]:
```
predict_train = model_1.predict(X_train)

acc = accuracy_score(y_train, predict_train)
f1 = f1_score(y_train, predict_train)

print("Accuracy: ", acc)
print("F1 score: ", f1)

# insert to dict
data['acc_train'].append(acc)
# insert to dict
data['f1_train'].append(f1)
```

```
Accuracy:  0.9428571428571428
F1 score:  0.9559322033898305
```

Predict with validation data

In [35]:
```
predict_test = model_1.predict(X_valid)
```

```
acc = accuracy_score(y_valid, predict_test)
f1 = f1_score(y_valid, predict_test)

print("Accuracy: ", acc)
print("F1 score: ", f1)

# insert to dict
data['acc_valid'].append(acc)
# insert to dict
data['f1_valid'].append(f1)
```

```
Accuracy:  0.9473684210526315
F1 score:  0.9594594594594595
```

Standard Scaler

Train the training data

In [36]:

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
scaler = StandardScaler()

scaled_X_train = scaler.fit_transform(X_train)
scaled_X_valid = scaler.fit_transform(X_valid)

model_2 = MLPClassifier(max_iter = 700, random_state=1)

model_2.fit(scaled_X_train,y_train.values.ravel())
```

Out[36]:

```
MLPClassifier(max_iter=700, random_state=1)
```

Predict with training data

In [37]:

```
predict_train = model_2.predict(scaled_X_train)

acc = accuracy_score(y_train, predict_train)
f1 = f1_score(y_train, predict_train)

print("Accuracy: ", acc)
print("F1 score: ", f1)

# insert to dict
data['acc_train'].append(acc)
# insert to dict
data['f1_train'].append(f1)
```

```
Accuracy:  1.0
F1 score:  1.0
```

Predict with validation data

In [38]:

```
predict_test = model_2.predict(scaled_X_valid)

acc = accuracy_score(y_valid, predict_test)
f1 = f1_score(y_valid, predict_test)

print("Accuracy: ", acc)
print("F1 score: ", f1)

# insert to dict
```

```
data['acc_valid'].append(acc)
# insert to dict
data['f1_valid'].append(f1)
```

```
Accuracy:  0.9649122807017544
F1 score:  0.9722222222222222
```

MinMax Scaler

Train the traning data

In [39]:

```
scaler = MinMaxScaler()

scaled_X_train = scaler.fit_transform(X_train)
scaled_X_valid = scaler.fit_transform(X_valid)

model_3 = MLPClassifier(max_iter = 700, random_state=1)
model_3.fit(scaled_X_train,y_train.values.ravel())
```

Out[39]:

```
MLPClassifier(max_iter=700, random_state=1)
```

Predict with training data

In [40]:

```
predict_train = model_3.predict(scaled_X_train)

acc = accuracy_score(y_train, predict_train)
f1 = f1_score(y_train, predict_train)

print("Accuracy: ", acc)
print("F1 score: ", f1)
# insert to dict
data['acc_train'].append(acc)
# insert to dict
data['f1_train'].append(f1)
```

```
Accuracy:  0.9934065934065934
F1 score:  0.9947643979057591
```

Predict with validation data

In [41]:

```
predict_test = model_3.predict(scaled_X_valid)

acc = accuracy_score(y_valid, predict_test)
f1 = f1_score(y_valid, predict_test)

print("Accuracy: ", acc)
print("F1 score: ", f1)

# insert to dict
data['acc_valid'].append(acc)
# insert to dict
data['f1_valid'].append(f1)
```

```
Accuracy:  0.9035087719298246
F1 score:  0.9172932330827067
```

SVM

Define the SVM (Support Vector Machines) constructor

In [42]:
```
from sklearn.svm import SVC

svm = SVC(random_state=1)
```

Train the traning dataset

In [43]:
```
svm.fit(X_train, y_train.values.ravel())
# .values will give the values in an array (shape: (n,1)
# .ravel will convert that array shape to (n, )
```

Out[43]:
```
SVC(random_state=1)
```

Predict the dataset.

In [44]:
```
pred_train_svm_breast = svm.predict(X_train)
pred_val_svm_breast = svm.predict(X_valid)
pred_val_svm_breast
```

Out[44]:
```
array([1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1,
       0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1])
```

Evaluate data train prediction with using Accuracy and F1 metrics

In [45]:
```
acc_score = accuracy_score(y_train, pred_train_svm_breast)
f1 = f1_score(y_train, pred_train_svm_breast)
print("Accuracy : ", acc_score)
print("F1 : ",f1)

# insert to dict
data['acc_train'].append(acc_score)
# insert to dict
data['f1_train'].append(f1)
```

```
Accuracy :  0.9230769230769231
F1 :  0.9405772495755519
```

Evaluate data test prediction with using Accuracy and F1 metrics

In [46]:
```
acc_score = accuracy_score(y_valid, pred_val_svm_breast)
f1 = f1_score(y_valid, pred_val_svm_breast)
print("Accuracy : ", acc_score)
print("F1 : ",f1)

# insert to dict
data['acc_valid'].append(acc_score)
# insert to dict
data['f1_valid'].append(f1)
```

```
Accuracy :  0.9035087719298246
F1 :  0.9290322580645161
```

In [47]:
```
breast_cancer = pd.DataFrame(data)
```

Play-Tennis Dataset

Load the dataset first.

In [48]:
```
# initialise data of lists.
data_tennis = {'algo':['DecisionTreeClassifier', 'Id3Estimator', 'KMeans','Logreg','Log
               'NN_StdScaler','NN_MinMaxScaler','SVM'],
        'acc_train':[],
        'acc_valid':[],
        'f1_train':[],
        'f1_valid':[]
       }
```

In [49]:
```
tennis = pd.read_csv('datasets/PlayTennis.csv')
X_tennis = tennis.drop(['play'],axis=1)
y_tennis = pd.DataFrame(tennis['play'])

# Split 80 : 20
X_train_tennis, X_valid_tennis, y_train_tennis, y_valid_tennis = train_test_split(X_ten
y_valid_tennis
```

Out[49]:

	play
3	yes
7	no
6	yes

Check whether there are missing values

In [50]:
```
cols_with_missing_tennis = [col for col in tennis.columns if tennis[col].isnull().any()
cols_with_missing_tennis
```

Out[50]: []

Train datasets

DecisionTreeClassifier

Make the categorical variable to numeric variable

In [51]:
```
from sklearn.preprocessing import LabelEncoder

cond = (X_train_tennis.dtypes == 'object')
object_cols = list(cond[cond].index)

label_X_train = X_train_tennis.copy()
label_X_valid = X_valid_tennis.copy()
```

```
label_encoder = LabelEncoder()
for col in object_cols:
    label_X_train[col] = label_encoder.fit_transform(X_train_tennis[col])
    label_X_valid[col] = label_encoder.transform(X_valid_tennis[col])

# In case needed
label_y_train = label_encoder.fit_transform(y_train_tennis.values.ravel())
label_y_valid = label_encoder.transform(y_valid_tennis.values.ravel())

label_X_train
```

Out[51]:

	outlook	temp	humidity	windy
2	0	1	0	False
10	2	2	1	True
4	1	0	1	False
1	2	1	0	True
12	0	1	1	False
0	2	1	0	False
13	1	2	0	True
9	1	2	1	False
8	2	0	1	False
11	0	2	0	True
5	1	0	1	True

Train the model.

In [52]:

```
tennis_classifier = DecisionTreeClassifier(random_state=1)
tennis_classifier.fit(label_X_train, y_train_tennis)
```

Out[52]:

```
DecisionTreeClassifier(random_state=1)
```

Print the tree.

In [53]:

```
tree_tennis = export_text(tennis_classifier, feature_names = list(X_tennis.columns))
print(tree_tennis)
```

```
|--- outlook <= 0.50
|   |--- class: yes
|--- outlook >  0.50
|   |--- humidity <= 0.50
|   |   |--- class: no
|   |--- humidity >  0.50
|   |   |--- windy <= 0.50
|   |   |   |--- class: yes
|   |   |--- windy >  0.50
|   |   |   |--- outlook <= 1.50
|   |   |   |   |--- class: no
|   |   |   |--- outlook >  1.50
|   |   |   |   |--- class: yes
```

Predict training data with the model.

In [54]:
```
predict_train_tennis = tennis_classifier.predict(label_X_train)
predict_train_tennis
```

Out[54]:
```
array(['yes', 'yes', 'yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'yes',
       'no'], dtype=object)
```

Evaluate the model to training data with Accuracy Metrics

In [55]:
```
acc_train_tennis = accuracy_score(y_train_tennis,predict_train_tennis)
acc_train_tennis
# insert to dict
data_tennis['acc_train'].append(acc_train_tennis)
```

Evaluate the model to training data with F1 Metrics

In [56]:
```
f1_train_tennis = f1_score(y_train_tennis,predict_train_tennis,pos_label='yes')
# insert to dict
data_tennis['f1_train'].append(f1_train_tennis)
f1_train_tennis
```

Out[56]:
```
1.0
```

Predict the test data using model.

In [57]:
```
predict_valid_tennis = tennis_classifier.predict(label_X_valid)
predict_valid_tennis
```

Out[57]:
```
array(['no', 'no', 'yes'], dtype=object)
```

Evaluate the model with datatest using Accuracy Metrics

In [58]:
```
acc_valid_tennis = accuracy_score(y_valid_tennis,predict_valid_tennis)
# insert to dict
data_tennis['acc_valid'].append(acc_valid_tennis)
acc_valid_tennis
```

Out[58]:
```
0.6666666666666666
```

Evaluate the model with datatest using F1 Metrics

In [59]:
```
f1_valid_tennis = f1_score(y_valid_tennis,predict_valid_tennis,pos_label='yes')
# insert to dict
data_tennis['f1_valid'].append(f1_valid_tennis)
f1_valid_tennis
```

Out[59]:
```
0.6666666666666666
```

Id3Estimator

Train the model and print the tree produced.

In [60]:
```
id3_model2 = Id3Estimator()
id3_model2.fit(label_X_train, label_y_train)

tree_text2 = export.export_text(id3_model2.tree_, feature_names=list(tennis.columns))
print(tree_text2)
```

```
outlook <=0.50: 1 (3)
outlook >0.50
|   humidity <=0.50: 0 (3)
|   humidity >0.50
|   |   windy <=0.50: 1 (3)
|   |   windy >0.50
|   |   |   temp <=1.00: 0 (1)
|   |   |   temp >1.00: 1 (1)
```

Evaluate the model to training data with Accuracy and F1 Metrics

In [61]:
```
predict_train2 = id3_model2.predict(label_X_train)

print("Accuracy: ", accuracy_score(predict_train2,label_y_train))
print("F1 score: ", f1_score(predict_train2,label_y_train))
# insert to dict
data_tennis['acc_train'].append(accuracy_score(predict_train2,label_y_train))
# insert to dict
data_tennis['f1_train'].append(f1_score(predict_train2,label_y_train))
```

```
Accuracy:  1.0
F1 score:  1.0
```

Evaluate the model to test data with using Accuracy and F1 Metrics

In [62]:
```
predict_test2 = id3_model2.predict(label_X_valid)

print("Accuracy: ", accuracy_score(predict_test2,label_y_valid))
print("F1 score: ", f1_score(predict_test2,label_y_valid))
# insert to dict
data_tennis['acc_valid'].append(accuracy_score(predict_test2,label_y_valid))
# insert to dict
data_tennis['f1_valid'].append(f1_score(predict_test2,label_y_valid))
```

```
Accuracy:  0.6666666666666666
F1 score:  0.6666666666666666
```

KMeans

Define KMeans and then train the dataset.

In [63]:
```
from sklearn.cluster import KMeans

# Determine how many clusters
n_clst = 2
# Define the KMeans
kmeans_model_2 = KMeans(n_clusters = n_clst, random_state=1)
kmeans_model_2.fit(label_X_train)
```

Out[63]:
```
KMeans(n_clusters=2, random_state=1)
```

Predict and produce clusters.

In [64]:
```
predict_train = kmeans_model_2.predict(label_X_train)
predict_test = kmeans_model_2.predict(label_X_valid)
```

Evaluate the training data now

In [65]:
```
# Find out each cluster belong with which class

# First clusterization using first prediction (k_means_train)
# Calculate the accuracy and F1
acc_train = accuracy_score(label_y_train, predict_train)
f1_train = f1_score(label_y_train, predict_train)
print(f"First clusterization accuracy: {acc_train}")
print(f"First clusterization F1: {f1_train}")

# Second clusterization: switch 0 to 1
switched_train = pd.Series(predict_train)
fix = switched_train.apply(switch)
acc_train1= accuracy_score(label_y_train, fix)
f1_train2= f1_score(label_y_train, fix)
print(f"Second clusterization accuracy: {acc_train1}")
print(f"Second clusterization F1: {f1_train2}")

# insert to dict
data_tennis['acc_train'].append(acc_train1)
# insert to dict
data_tennis['f1_train'].append(f1_train2)
```

```
First clusterization accuracy: 0.36363636363636365
First clusterization F1: 0.4615384615384615
Second clusterization accuracy: 0.6363636363636364
Second clusterization F1: 0.6666666666666666
```

Second clusterization is better, so we take the metrics as the final metrics. Evaluate the validation data now

In [66]:
```
# First clusterization
acc_valid = accuracy_score(label_y_valid, predict_test)
f1_valid = f1_score(label_y_valid, predict_test)
print(f"First clusterization accuracy: {acc_valid}")
print(f"First clusterization F1: {f1_train}")

# Second clusterization: switch 0 to 1
switched_valid = pd.Series(predict_test).apply(switch)
acc_valid_switch = accuracy_score(label_y_valid, switched_valid)
f1_valid_switch = f1_score(label_y_valid, switched_valid)
print(f"Second clusterization accuracy: {acc_valid_switch}")
print(f"Second clusterization F1: {f1_valid_switch}")

# insert to dict
data_tennis['acc_valid'].append(acc_valid_switch)
# insert to dict
data_tennis['f1_valid'].append(f1_valid_switch)
```

```
First clusterization accuracy: 0.0
First clusterization F1: 0.4615384615384615
Second clusterization accuracy: 1.0
Second clusterization F1: 1.0
```

Same as when we evaluate with training data, the second clusterization is better and then will be our final metrics score.

LogisticRegression

Modelling

In [67]:
```
model_1 = LogisticRegression(random_state=1)

model_1.fit(label_X_train,label_y_train)
```

Out[67]:
```
LogisticRegression(random_state=1)
```

Predict the train data and evaluate the prediction with Accuracy and F1 score.

In [68]:
```
predict_train = model_1.predict(label_X_train)

acc = accuracy_score(label_y_train, predict_train)
f1 = f1_score(label_y_train, predict_train)

print("local data")
print("Accuracy: ", acc)
print("F1 score: ", f1)
# insert to dict
data_tennis['acc_train'].append(acc)
# insert to dict
data_tennis['f1_train'].append(f1)
```

```
local data
Accuracy:  0.8181818181818182
F1 score:  0.8750000000000001
```

In [69]:
```
predict_test = model_1.predict(label_X_valid)

acc = accuracy_score(label_y_valid, predict_test)
f1 = f1_score(label_y_valid, predict_test)

print("test data")
print("Accuracy: ", acc)
print("F1 score: ", f1)
# insert to dict
data_tennis['acc_valid'].append(acc)
# insert to dict
data_tennis['f1_valid'].append(f1)
```

```
test data
Accuracy:  0.6666666666666666
F1 score:  0.8
```

Scaling : Standard

In [70]:
```
# Scaling if needed
from sklearn.preprocessing import StandardScaler, MinMaxScaler
scaler = StandardScaler()

scaled_label_X_train = scaler.fit_transform(label_X_train)
scaled_label_X_valid = scaler.fit_transform(label_X_valid)
```

```
model_2 = LogisticRegression(random_state=1)
model_2.fit(scaled_label_X_train,label_y_train)
```

Out[70]: LogisticRegression(random_state=1)

In [71]:

```
predict_train = model_2.predict(scaled_label_X_train)

acc = accuracy_score(label_y_train, predict_train)
f1 = f1_score(label_y_train, predict_train)

print("local data")
print("Accuracy: ", acc)
print("F1 score: ", f1)
# insert to dict
data_tennis['acc_train'].append(acc)
# insert to dict
data_tennis['f1_train'].append(f1)
```

```
local data
Accuracy:  0.9090909090909091
F1 score:  0.9333333333333333
```

In [72]:

```
predict_test = model_2.predict(scaled_label_X_valid)

acc = accuracy_score(label_y_valid, predict_test)
f1 = f1_score(label_y_valid, predict_test)

print("test data")
print("Accuracy: ", acc)
print("F1 score: ", f1)
# insert to dict
data_tennis['acc_valid'].append(acc)
# insert to dict
data_tennis['f1_valid'].append(f1)
```

```
test data
Accuracy:  0.6666666666666666
F1 score:  0.8
```

Scaling : MinMax

In [73]:

```
model_3 = LogisticRegression(random_state=1)
model_3.fit(scaled_label_X_train,label_y_train)

predict_train = model_3.predict(scaled_label_X_train)

acc = accuracy_score(label_y_train, predict_train)
f1 = f1_score(label_y_train, predict_train)

print("local data")
print("Accuracy: ", acc)
print("F1 score: ", f1)

# insert to dict
data_tennis['acc_train'].append(acc)
# insert to dict
data_tennis['f1_train'].append(f1)
```

```
local data
Accuracy:  0.9090909090909091
F1 score:  0.9333333333333333
```

In [74]:

```
predict_test = model_3.predict(scaled_label_X_valid)

acc = accuracy_score(label_y_valid, predict_test)
f1 = f1_score(label_y_valid, predict_test)

print("test data")
print("Accuracy: ", acc)
print("F1 score: ", f1)

# insert to dict
data_tennis['acc_valid'].append(acc)
# insert to dict
data_tennis['f1_valid'].append(f1)
```

```
test data
Accuracy:  0.6666666666666666
F1 score:  0.8
```

Neural_network

Train the dataset

In [75]:

```
from sklearn.neural_network import MLPClassifier

model_1 = MLPClassifier(max_iter=700, random_state=1)

model_1.fit(label_X_train,label_y_train)
```

Out[75]:

```
MLPClassifier(max_iter=700, random_state=1)
```

Predict the training data

In [76]:

```
predict_train = model_1.predict(label_X_train)

acc = accuracy_score(label_y_train, predict_train)
f1 = f1_score(label_y_train, predict_train)

print("Accuracy: ", acc)
print("F1 score: ", f1)
# insert to dict
data_tennis['acc_train'].append(acc)
# insert to dict
data_tennis['f1_train'].append(f1)
```

```
Accuracy:  1.0
F1 score:  1.0
```

Predict the validation data

In [77]:

```
predict_test = model_1.predict(label_X_valid)

acc = accuracy_score(label_y_valid, predict_test)
f1 = f1_score(label_y_valid, predict_test)
```

```
print("Accuracy: ", acc)
print("F1 score: ", f1)
# insert to dict
data_tennis['acc_valid'].append(acc)
# insert to dict
data_tennis['f1_valid'].append(f1)
```

```
Accuracy:  1.0
F1 score:  1.0
```

StandardScaler

Train the dataset

In [78]:

```
scaler = StandardScaler()



scaled_label_X_train = scaler.fit_transform(label_X_train)
scaled_label_X_valid = scaler.fit_transform(label_X_valid)

model_2 = MLPClassifier(max_iter=700, random_state=1)
model_2.fit(scaled_label_X_train,label_y_train)
```

Out[78]:

```
MLPClassifier(max_iter=700, random_state=1)
```

Predict the training data

In [79]:

```
predict_train = model_2.predict(scaled_label_X_train)

acc = accuracy_score(label_y_train, predict_train)
f1 = f1_score(label_y_train, predict_train)

print("Accuracy: ", acc)
print("F1 score: ", f1)
# insert to dict
data_tennis['acc_train'].append(acc)
# insert to dict
data_tennis['f1_train'].append(f1)
```

```
Accuracy:  1.0
F1 score:  1.0
```

Predict with validation data

In [80]:

```
predict_test = model_2.predict(scaled_label_X_valid)

acc = accuracy_score(label_y_valid, predict_test)
f1 = f1_score(label_y_valid, predict_test)

print("Accuracy: ", acc)
print("F1 score: ", f1)
# insert to dict
data_tennis['acc_valid'].append(acc)
# insert to dict
data_tennis['f1_valid'].append(f1)
```

```
Accuracy:  0.6666666666666666
F1 score:  0.6666666666666666
```

MinMax Scaler

Train the dataset

In [81]:
```
scaler = MinMaxScaler()

scaled_label_X_train = scaler.fit_transform(label_X_train)
scaled_label_X_valid = scaler.fit_transform(label_X_valid)

model_3 = MLPClassifier(max_iter=700, random_state=1)
model_3.fit(scaled_label_X_train,label_y_train)
```

Out[81]:
```
MLPClassifier(max_iter=700, random_state=1)
```

Predict the training data

In [82]:
```
predict_train = model_3.predict(scaled_label_X_train)

acc = accuracy_score(label_y_train, predict_train)
f1 = f1_score(label_y_train, predict_train)

print("Accuracy: ", acc)
print("F1 score: ", f1)
# insert to dict
data_tennis['acc_train'].append(acc)
# insert to dict
data_tennis['f1_train'].append(f1)
```

```
Accuracy:  1.0
F1 score:  1.0
```

Predict the validation data

In [83]:
```
predict_test = model_3.predict(scaled_label_X_valid)

acc = accuracy_score(label_y_valid, predict_test)
f1 = f1_score(label_y_valid, predict_test)

print("Accuracy: ", acc)
print("F1 score: ", f1)
# insert to dict
data_tennis['acc_valid'].append(acc)
# insert to dict
data_tennis['f1_valid'].append(f1)
```

```
Accuracy:  0.6666666666666666
F1 score:  0.8
```

SVM

Define the SVM (Support Vector Machines) constructor

In [84]:
```
svm = SVC(random_state=1)
```

Train the dataset

In [85]:
```
svm.fit(label_X_train, y_train_tennis.values.ravel())
# .values will give the values in an array (shape: (n,1)
# .ravel will convert that array shape to (n, )
# y_train_tennis.values.ravel()
```

Out[85]:
```
SVC(random_state=1)
```

Predict the data train and the data test

In [86]:
```
pred_svm_train_tennis = svm.predict(label_X_train)
pred_svm_val_tennis = svm.predict(label_X_valid)
pred_svm_train_tennis
# pred_svm_val_tennis
```

Out[86]:
```
array(['yes', 'yes', 'yes', 'no', 'yes', 'no', 'yes', 'yes', 'yes', 'yes',
       'yes'], dtype=object)
```

Evaluate the prediction of data train with using Accuracy and F1 metrics

In [87]:
```
acc_score_svm = accuracy_score(y_train_tennis, pred_svm_train_tennis)
# must define the positive label = yes
f1_score_svm = f1_score(y_train_tennis, pred_svm_train_tennis, pos_label="yes")
print("Accuracy : ",acc_score_svm)
print("F1 : ",f1_score_svm)
# insert to dict
data_tennis['acc_train'].append(acc_score_svm)
# insert to dict
data_tennis['f1_train'].append(f1_score_svm)
```

```
Accuracy :  0.8181818181818182
F1 :  0.8750000000000001
```

Evaluate the prediction of data validation with using Accuracy and F1 metrics

In [88]:
```
acc_score_svm = accuracy_score(y_valid_tennis, pred_svm_val_tennis)
# must define the positive label = yes
f1_score_svm = f1_score(y_valid_tennis, pred_svm_val_tennis, pos_label="yes")
print("Accuracy : ",acc_score_svm)
print("F1 : ",f1_score_svm)
# insert to dict
data_tennis['acc_valid'].append(acc_score_svm)
# insert to dict
data_tennis['f1_valid'].append(f1_score_svm)
```

```
Accuracy :  1.0
F1 :  1.0
```

In [89]:
```
play_tennis = pd.DataFrame(data_tennis)
```

Analysis accuracy and F1 score for each model

The metrics data from Breast Cancer Dataset

In [90]:
```
breast_cancer
```

Out[90]:

	algo	acc_train	acc_valid	f1_train	f1_valid
0	DecisionTreeClassifier	1.000000	0.947368	1.000000	0.959459
1	Id3Estimator	1.000000	0.938596	1.000000	0.953020
2	KMeans	0.848352	0.807018	0.891680	0.867470
3	Logreg	0.960440	0.947368	0.968531	0.959459
4	Logreg_StdScaler	0.991209	0.973684	0.993007	0.979310
5	Logreg_MinMaxScaler	0.962637	0.973684	0.970940	0.979592
6	NN	0.942857	0.947368	0.955932	0.959459
7	NN_StdScaler	1.000000	0.964912	1.000000	0.972222
8	NN_MinMaxScaler	0.993407	0.903509	0.994764	0.917293
9	SVM	0.923077	0.903509	0.940577	0.929032

Analisis Tiap Model

Decision Tree dan id3 Estimator:

- akurasi dan f1 pada train set sempurna, namun cenderung turun pada validation set hal ini diakibatkan oleh kecenderungan dari terjadinya overfitting saat membentuk model Decision Tree, karena decision tree membentuk model dengan menyesuaikan rule-rule ke data yang sedang ditinjau.
- pada data breast cancer, akurasi dan f1 validation dengan DTL biasa lebih baik daripada menggunakan id3 estimator

KMeans:

- akurasi dan f1 cenderung tidak terlalu tinggi, karena persebaran data tidak terbagi dua dengan jelas, dan kemungkinan terdapat outlier. Ada kemungkinan model mengambil centroid yang kurang tepat. Selain itu, KMeans merupakan algoritma unsupervised, sedangkan data yang kami gunakan memiliki label yang sebaiknya dapat dimanfaatkan untuk mempelajari data dengan lebih tepat

Logistic Regression

- akurasi dan f1 cenderung tinggi, agak mengalami overfit saat menggunakan standard scaler. Hal ini mungkin terjadi karena standard scaler mengubah nilai-nilai kolom pada range yang sama, sehingga hasil klasifikasi model pada data training cukup besar.

Neural network

- akurasi dan f1 tidak terlalu tinggi saat tidak menggunakan scaler
- saat mengguakan scaler, akurasi bertambah namun mengalami overfit saat menggunakan minmax scaler

SVM

- akurasi dan f1 cenderung tidak terlalu tinggi. Kemungkinan hasil pengelompokan tidak memiliki dua bagian yang jelas sehingga sulit untuk membagi kedalam dua bagian data

Kesimpulan

Kasus 1:
Jika tidak memperhatikan model-model yang menggunakan *scaled dataset*, maka model-model yang akan dibandingkan adalah model 0, 1, 2, 3, 6, dan 9. Pertama-tama, kita melihat menggunakan *accuracy metrics* pada bagian *validation dataset*. Ketika dilihat, yang memiliki nilai *accuracy metrics* terbesar adalah 3 algoritma, yaitu DecisionTreeClassifier, LogisticRegression, dan NeuralNetwork dengan nilai sebesar 0.947368. Lalu, untuk melihat apakah algoritma *overfit* atau tidak, maka bisa dilihat pada bagian *training dataset*. Yang memiliki akurasi terkecil adalah NN. Lalu, kita cek pada bagian *F1 score metrics*, berlaku hal yang sama, NN memiliki F1 score terbesar pada validation dataset dan memiliki F1 score yang cukup kecil pada training datasetnya. Lalu, dalam hal ini KMeans memiliki performansi yang paling rendah, jika dilihat dari metrics dari tiap metode metrics.

Kasus 2:
Jika memperhitungkan model-model yang menggunakan scaled dataset, maka Logistic Regression Model dengan Standard Scaler memiliki performansi terbaik, jika kita mempertimbangkan nilai accuracy metrics. Jika kita mempertimbangkan nilai F1 score, Logistic Regression Model dengan MinMaxScaler memiliki performansi terbaik. Yang jelas, jika Logistic Regression Model diberi Scaler, akan menjadi model yang paling baik. Untuk model yang memiliki performansi terburuk tetap jatuh pada KMeans, baik dari segi accuracy metrics maupun F1 score metrics.

The metrics data from Play Tennis Dataset

In [91]:
```
play_tennis
```

Out[91]:

	algo	acc_train	acc_valid	f1_train	f1_valid
0	DecisionTreeClassifier	1.000000	0.666667	1.000000	0.666667
1	Id3Estimator	1.000000	0.666667	1.000000	0.666667
2	KMeans	0.636364	1.000000	0.666667	1.000000
3	Logreg	0.818182	0.666667	0.875000	0.800000
4	Logreg_StdScaler	0.909091	0.666667	0.933333	0.800000
5	Logreg_MinMaxScaler	0.909091	0.666667	0.933333	0.800000
6	NN	1.000000	1.000000	1.000000	1.000000
7	NN_StdScaler	1.000000	0.666667	1.000000	0.666667
8	NN_MinMaxScaler	1.000000	0.666667	1.000000	0.800000
9	SVM	0.818182	1.000000	0.875000	1.000000

Lalu, kita lihat pada dataset yang ukurannya lebih kecil -- bahkan jauh lebih kecil -- dari dataset sebelumnya. Jika kita menggunakan accuracy metrics sebagai acuan, maka ada tiga kandidat terbaik (jika melihat juga dari validation dataset), yaitu KMeans, NN, dan SVM. Kita lihat bagaimana kinerja ketika diterapkan pada training dataset. Hal yang cukup mengejutkan adalah hasil evaluasi pada validation dataset yang terjadi pada model KMeans dan SVM adalah lebih besar dari hasil evaluasi pada training dataset. Hal ini tidak biasa. Namun, yang jelas pada KMeans, selisihnya cukup jauh, sehingga tidak cukup baik untuk dijadikan pertimbangan, karena bisa saja data yang terjadi pada training dataset sangat random, sehingga banyak outlier daripada validation datasetnya. Lalu, ketika dilihat, NN sangat konsisten baik dalam accuracy metrics maupun F1 score metrics. Maka bisa disimpulkan NN memiliki performansi terbaik untuk dataset yang ukurannya kecil.