Tucil_13520101_13520119

February 26, 2023

1 Tugas Kecil 1 IF3270

1.1 Eksplorasi Library Algoritma Pembelajaran pada Jupyter Notebook

1.1.1 Anggota

1. 13520101 - Aira Thalca Avila Putra
2. 13520119 - Marchotridyo

2 Bagian 1: Mengenal dataset breast-cancer

[1]:
```
# Load data breast_cancer
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
```

[2]:
```
# Mengenal isi dari data breast_cancer
import pandas as pd
import numpy as np

def sklearn_to_df(sklearn_dataset):
    df = pd.DataFrame(sklearn_dataset.data, columns=sklearn_dataset.
 ↪feature_names)
    df['target'] = pd.Series(sklearn_dataset.target)
    return df

dataset = sklearn_to_df(data)

dataset.head()
```

[2]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	\
0	17.99	10.38	122.80	1001.0	0.11840	
1	20.57	17.77	132.90	1326.0	0.08474	
2	19.69	21.25	130.00	1203.0	0.10960	
3	11.42	20.38	77.58	386.1	0.14250	
4	20.29	14.34	135.10	1297.0	0.10030	

	mean compactness	mean concavity	mean concave points	mean symmetry	\
0	0.27760	0.3001	0.14710	0.2419	

```
1           0.07864          0.0869            0.07017         0.1812
2           0.15990          0.1974            0.12790         0.2069
3           0.28390          0.2414            0.10520         0.2597
4           0.13280          0.1980            0.10430         0.1809

   mean fractal dimension  …  worst texture  worst perimeter  worst area  \
0                 0.07871  …          17.33           184.60      2019.0
1                 0.05667  …          23.41           158.80      1956.0
2                 0.05999  …          25.53           152.50      1709.0
3                 0.09744  …          26.50            98.87       567.7
4                 0.05883  …          16.67           152.20      1575.0

   worst smoothness  worst compactness  worst concavity  worst concave points  \
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625

   worst symmetry  worst fractal dimension  target
0          0.4601                  0.11890       0
1          0.2750                  0.08902       0
2          0.3613                  0.08758       0
3          0.6638                  0.17300       0
4          0.2364                  0.07678       0

[5 rows x 31 columns]
```

[3]:
```
# Lihat data fitur apa saja yang dimiliki oleh dataset dan nama target variabelnya
print(data.feature_names)
print(data.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']
```

Jadi, apa yang ingin dilakukan setelah melakukan pembelajaran? Yang nantinya ingin dilakukan: Dari suatu data (yang berisikan nilai dari feature 'mean radius' s.d. 'worst fractal dimension') yang diberikan, kita ingin mengklasifikasikan data tersebut apakah bersifat 'malignant'

(0) atau 'benign' (1).

Sebelum dimulai learning, cek beberapa informasi tambahan dahulu...

[4]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  target                   569 non-null    int32
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
```

Beberapa hal yang bisa diperhatikan dari pemanggilan df.info():

1. Tidak ada missing values, dan
2. Semua tipe datanya numerik

3 Bagian 2: Membagi dataset menjadi 80% data training dan 20% data testing

[5]:
```
from sklearn.model_selection import train_test_split

X = data.data
Y = data.target

# Bagi menjadi 80% data training dan 20% data testing
X_train, X_test, Y_train, Y_test =  train_test_split(X, Y, test_size = 0.20)
```

4 Bagian 3: Melakukan pembelajaran untuk membentuk classifier dengan berbagai algoritma

4.1 Decision Tree

[6]:
```
from sklearn import tree
decisionTreeClassifier = tree.DecisionTreeClassifier()
decisionTreeClassifier = decisionTreeClassifier.fit(X_train, Y_train)

# Model pohon decision tree
print(tree.export_text(decisionTreeClassifier, feature_names=list(data.
  ↪feature_names)))
```

```
|--- mean concave points <= 0.05
|   |--- worst area <= 957.45
|   |   |--- worst symmetry <= 0.16
|   |   |   |--- class: 0
|   |   |--- worst symmetry >  0.16
|   |   |   |--- worst perimeter <= 108.25
|   |   |   |   |--- smoothness error <= 0.00
|   |   |   |   |   |--- worst concave points <= 0.10
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- worst concave points >  0.10
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |--- smoothness error >  0.00
|   |   |   |   |   |--- worst texture <= 33.35
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- worst texture >  33.35
|   |   |   |   |   |   |--- mean texture <= 23.20
|   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |--- mean texture >  23.20
|   |   |   |   |   |   |   |--- class: 1
|   |   |   |--- worst perimeter >  108.25
|   |   |   |   |--- worst smoothness <= 0.14
|   |   |   |   |   |--- texture error <= 1.48
```

```
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- texture error >  1.48
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |--- worst smoothness >  0.14
|   |   |   |   |   |--- class: 0
|   |--- worst area >  957.45
|   |   |--- worst fractal dimension <= 0.06
|   |   |   |--- class: 1
|   |   |--- worst fractal dimension >  0.06
|   |   |   |--- class: 0
|--- mean concave points >  0.05
|   |--- worst concavity <= 0.23
|   |   |--- mean radius <= 16.59
|   |   |   |--- class: 1
|   |   |--- mean radius >  16.59
|   |   |   |--- class: 0
|   |--- worst concavity >  0.23
|   |   |--- mean radius <= 11.18
|   |   |   |--- class: 1
|   |   |--- mean radius >  11.18
|   |   |   |--- mean texture <= 15.38
|   |   |   |   |--- worst perimeter <= 119.55
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- worst perimeter >  119.55
|   |   |   |   |   |--- class: 0
|   |   |   |--- mean texture >  15.38
|   |   |   |   |--- fractal dimension error <= 0.01
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- fractal dimension error >  0.01
|   |   |   |   |   |--- class: 1
```

4.2 ID3

[7]:
```
import six
import sys
sys.modules['sklearn.externals.six'] = six

from id3 import Id3Estimator

id3Classifier = Id3Estimator()
id3Classifier = id3Classifier.fit(X_train, Y_train)
```

4.3 K-Means

[8]:
```
from sklearn.cluster import KMeans
kMeansClassifier = KMeans(n_clusters = 2, n_init = 10)
kMeansClassifier = kMeansClassifier.fit(X_train)
```

4.4 Logistic Regression

[9]:
```
from sklearn.linear_model import LogisticRegression
logisticRegressionClassifier = LogisticRegression(C=100, max_iter=10000)
logisticRegressionClassifier = logisticRegressionClassifier.fit(X_train,␣
 ↪Y_train)
```

```
C:\Python310\lib\site-packages\sklearn\linear_model\_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

4.5 Neural Network

[10]:
```
from sklearn.neural_network import MLPClassifier
mlpClassifier = MLPClassifier(hidden_layer_sizes=(100, 100, 100), max_iter=1000)
mlpClassifier.fit(X_train, Y_train)
```

[10]:
```
MLPClassifier(hidden_layer_sizes=(100, 100, 100), max_iter=1000)
```

4.6 SVM

[11]:
```
from sklearn.svm import SVC

svmClassifier = SVC(C=100, kernel='linear')
svmClassifier.fit(X_train, Y_train)
```

[11]:
```
SVC(C=100, kernel='linear')
```

5 Bagian 4: Menyimpan model hasil pembelajaran

[12]:
```
import pickle

pickle.dump(decisionTreeClassifier, open("decisionTreeClassifier.sav", "wb"))
```

```
pickle.dump(id3Classifier, open("id3Classifier.sav", "wb"))
pickle.dump(kMeansClassifier, open("kMeansClassifier.sav", "wb"))
pickle.dump(logisticRegressionClassifier, open("logisticRegressionClassifier.
  ↪sav", "wb"))
pickle.dump(mlpClassifier, open("mlpClassifier.sav", "wb"))
pickle.dump(svmClassifier, open("svmClassifier.sav", "wb"))
```

6 Bagian 5: Melakukan prediksi dengan loaded model

6.1 Set up

[13]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
  ↪f1_score, confusion_matrix

def predictFromModel(modelFileName):
  loadedModel = pickle.load(open(modelFileName, "rb"))
  modelName = modelFileName.rstrip(".sav")
  # Lakukan prediksi
  Y_pred = loadedModel.predict(X_test)
  accuracy = accuracy_score(Y_test, Y_pred)
  precision = precision_score(Y_test, Y_pred)
  recall = recall_score(Y_test, Y_pred)
  f1 = f1_score(Y_test, Y_pred)
  cmat = confusion_matrix(Y_test, Y_pred)
  # Output!
  print(f'''Hasil prediksi dari model {modelName}:

  Accuracy: {accuracy}
  Precision: {precision}
  Recall: {recall}
  F1: {f1}

  Confusion matrix:
  {cmat}
  ''')
```

6.2 Decision Tree

[14]:
```
predictFromModel("decisionTreeClassifier.sav")
```

```
Hasil prediksi dari model decisionTreeClassifier:

  Accuracy: 0.8947368421052632
  Precision: 0.92
  Recall: 0.92
  F1: 0.92
```

```
  Confusion matrix:
  [[33  6]
 [ 6 69]]
```

6.3 ID3

[15]:
```
predictFromModel("id3Classifier.sav")
```

```
Hasil prediksi dari model id3Classifier:

  Accuracy: 0.9035087719298246
  Precision: 0.9324324324324325
  Recall: 0.92
  F1: 0.9261744966442953

  Confusion matrix:
  [[34  5]
 [ 6 69]]
```

6.4 K-Means

[16]:
```
predictFromModel("kMeansClassifier.sav")
```

```
Hasil prediksi dari model kMeansClassifier:

  Accuracy: 0.15789473684210525
  Precision: 0.0
  Recall: 0.0
  F1: 0.0

  Confusion matrix:
  [[18 21]
 [75  0]]
```

6.5 Logistic Regression

[17]:
```
predictFromModel("logisticRegressionClassifier.sav")
```

```
Hasil prediksi dari model logisticRegressionClassifier:

  Accuracy: 0.9649122807017544
  Precision: 0.961038961038961
  Recall: 0.9866666666666667
  F1: 0.9736842105263157
```

```
  Confusion matrix:
  [[36  3]
 [ 1 74]]
```

6.6 Neural Network

[18]:
```
predictFromModel("mlpClassifier.sav")
```

```
Hasil prediksi dari model mlpClassifier:

  Accuracy: 0.9035087719298246
  Precision: 0.9102564102564102
  Recall: 0.9466666666666667
  F1: 0.9281045751633987

  Confusion matrix:
  [[32  7]
 [ 4 71]]
```

6.7 SVM

[19]:
```
predictFromModel("svmClassifier.sav")
```

```
Hasil prediksi dari model svmClassifier:

  Accuracy: 0.9649122807017544
  Precision: 0.961038961038961
  Recall: 0.9866666666666667
  F1: 0.9736842105263157

  Confusion matrix:
  [[36  3]
 [ 1 74]]
```

7 Bagian 6: Analisis hasil metrik evaluasi

Dari beberapa model yang sudah dibuat dan diuji, didapatkan hasil bahwa: 1. Skor accuracy tertinggi didapatkan oleh model Logistic Regression 2. Skor precision tertinggi didapatkan oleh model Logistic Regression 3. Skor recall tertinggi didapatkan oleh model K-Means 4. Skor F1 tertinggi didapatkan oleh model Logistic Regression

Secara umum, seluruh model yang dibuat memiliki skor yang cukup baik. Namun, ada beberapa hal yang perlu diperhatikan: - Skor accuracy yang didapatkan oleh model K-Means adalah satu-satunya yang memiliki skor dibawah 0.9. Hal ini bisa disebabkan karena model K-Means memiliki

skor yang rendah jika ia mengklasifikasikan kelasnya salah (misal, yang seharusnya targetnya 0 dia klasifikasikan menjadi 1). - Model paling cocok yang digunakan adalah model Logistic Regression, karena memiliki skor yang paling tinggi di semua metrik evaluasi. Hal ini bisa disebabkan karena model Logistic Regression cocok digunakan untuk klasifikasi biner (mengklasifikasikan suatu data menjadi dua kelas, yaitu 0 dan 1). Tipe data yang digunakan juga cocok digunakan untuk model Logistic Regression, yaitu data numerik. - Model SVM dengan kernel linear dan Decision Tree juga cocok untuk digunakan sebagai model karena keduanya cocok digunakan untuk klasifikasi biner. Selain itu, model SVM cocok digunakan untuk data yang memiliki banyak feature.

8 Bagian 7: K-Fold Cross Validation pada DecisionTreeClassifier

[20]:
```
from sklearn.model_selection import cross_validate

scores = cross_validate(decisionTreeClassifier, X, Y, cv=10,␣
 ↪scoring=["accuracy", "f1"])
print(f"""Data 10-fold cross validation:
  Accuracy: {scores["test_accuracy"]}, with mean {np.
 ↪mean(scores["test_accuracy"])}
  F1: {scores["test_f1"]}, with mean {np.mean(scores["test_f1"])}
""")

print('Average accuracy: ', scores['test_accuracy'].mean())
print('Average f1: ', scores['test_f1'].mean())
```

```
Data 10-fold cross validation:
  Accuracy: [0.9122807  0.84210526 0.9122807  0.85964912 0.96491228 0.9122807
 0.9122807  0.94736842 0.92982456 0.98214286], with mean 0.9175125313283209
  F1: [0.92957746 0.87323944 0.93150685 0.88571429 0.97222222 0.93150685
 0.93506494 0.95774648 0.94117647 0.98591549], with mean 0.9343670485459251

Average accuracy:  0.9175125313283209
Average f1:  0.9343670485459251
```

8.1 Perbandingan dengan skor prediksi DecisionTreeClassifier sebelumnya

Dapat dilihat bahwa menurut akurasi antara K-Fold Cross Validation dan prediksi DecisionTreeClassifier sebelumnya, kedua skor tersebut hampir sama. Hal ini bisa disebabkan karena jumlah data yang digunakan untuk melakukan K-Fold Cross Validation adalah 80% dari data yang ada, sehingga jumlah data yang digunakan untuk melakukan K-Fold Cross Validation hampir sama dengan jumlah data yang digunakan untuk melakukan prediksi DecisionTreeClassifier sebelumnya. Dalam beberapa percobaa, nilai akurasi yang didapatkan dari K-Fold Cross Validation kadang lebih baik dan kadang lebih buruk dari prediksi DecisionTreeClassifier sebelumnya. Secara keseluruhan, K-Fold Cross Validation memberikan gambaran yang baik mengenai performa model, dalam hal ini DecisionTreeClassifier.