Nama : Akmal Zuhdy Prasetya
NIM : H071191035
Kelas : Machine Learning

Resume Pertemuan 6

A. Support Vector Machine

Algoritma SVM adalah algoritma pembelajaran terawasi yang dikategorikan dalam teknik Klasifikasi. Ini adalah teknik klasifikasi biner yang menggunakan dataset pelatihan untuk memprediksi hyperplane yang optimal dalam ruang n-dimensi. Hyperplane ini digunakan untuk mengklasifikasikan set data baru. Menjadi pengklasifikasi biner, kumpulan data pelatihan hyperplane membagi kumpulan data pelatihan menjadi dua kelas.

Algoritma SVM digunakan untuk mengklasifikasikan data dalam bidang 2 dimensi serta hyperplane multidimensi. Hyperplane multidimensi menggunakan "Kernel" untuk mengkategorikan data multidimensi. Hal itu selalu diinginkan untuk memiliki perbedaan maksimum antara titik data yang diklasifikasikan. Ini berarti bahwa mereka harus memiliki jarak maksimum, atau hyperplane harus memiliki margin maksimum antara titik data.

B. Hpyerlane

Hyperplane adalah batas yang membagi bidang. Ini adalah batas keputusan yang mengklasifikasikan titik data menjadi 2 kelas yang berbeda. Karena SVM digunakan untuk mengklasifikasikan data dalam multi-dimensi, hyperplane dapat berupa garis lurus jika ada 2 input atau bidang 2 D jika ada lebih dari 2 input.

C. Support Vectors

Support Vectors adalah titik data yang membantu kita untuk mengoptimalkan hyperplane. Vektor-vektor ini terletak paling dekat dengan hyperplane dan paling sulit untuk diklasifikasikan. Posisi hyperplane keputusan tergantung pada support vector. Jika support vector ini dihilangkan, maka posisi hyperplane juga akan berubah.

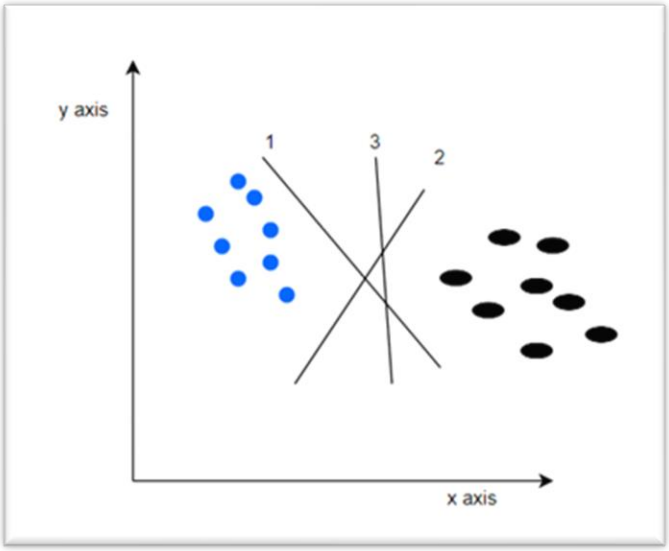
Support Vector Machine (SVM) menggunakan titik data input atau fitur yang disebut vektor dukungan untuk memaksimalkan batas keputusan yaitu ruang di sekitar hyperplane.

D. Mekanisme SVM

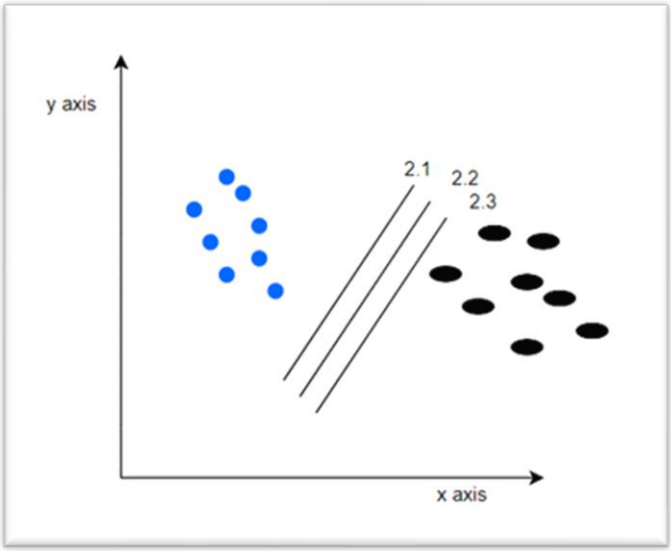
Seperti yang kita ketahui, tujuan dari SVM adalah untuk memaksimalkan margin antara titik data yang diklasifikasikan. Ini akan membawa hasil yang lebih optimal untuk mengklasifikasikan kumpulan data baru yang tidak terlatih. Dengan demikian, dapat dicapai dengan memiliki hyperplane pada posisi di mana margin maksimum.

Berikut merupakan contoh mekanisme dari SVM:

1. Temukan hyperplane yang benar dari berbagai kemungkinan. Untuk memutuskan hyperplane terbaik, temukan semua kemungkinan bidang yang membagi data, dan kemudian pilih salah satu yang paling baik mengklasifikasikan set data input. Pada grafik di bawah ini terdapat tiga kemungkinan hyperplane. Hyperplane 3 membagi titik data dengan lebih baik.

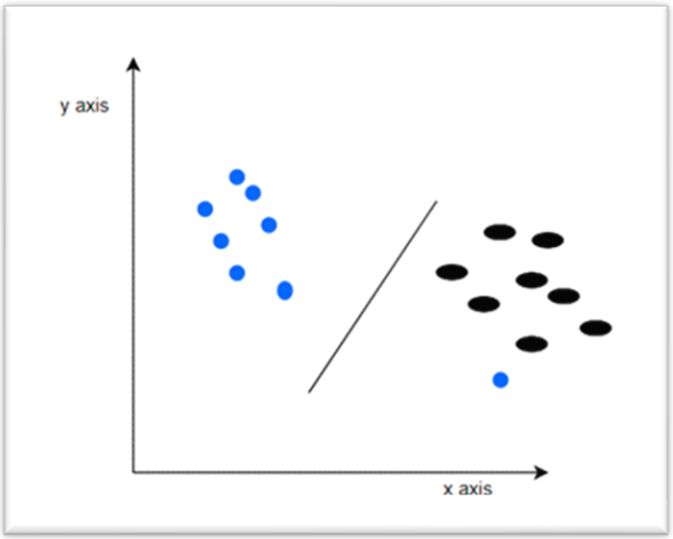


2. Pilih hyperplane yang memiliki margin maksimum antara titik data terdekat. Dimana margin itu didefinisikan sebagai jarak antara hyperplane dan titik data terdekat. Jadi, optimal untuk memiliki margin maksimum. Ketika 2 atau lebih dari 2 hyperplane mengklasifikasikan data secara merata, maka carilah marginnya.

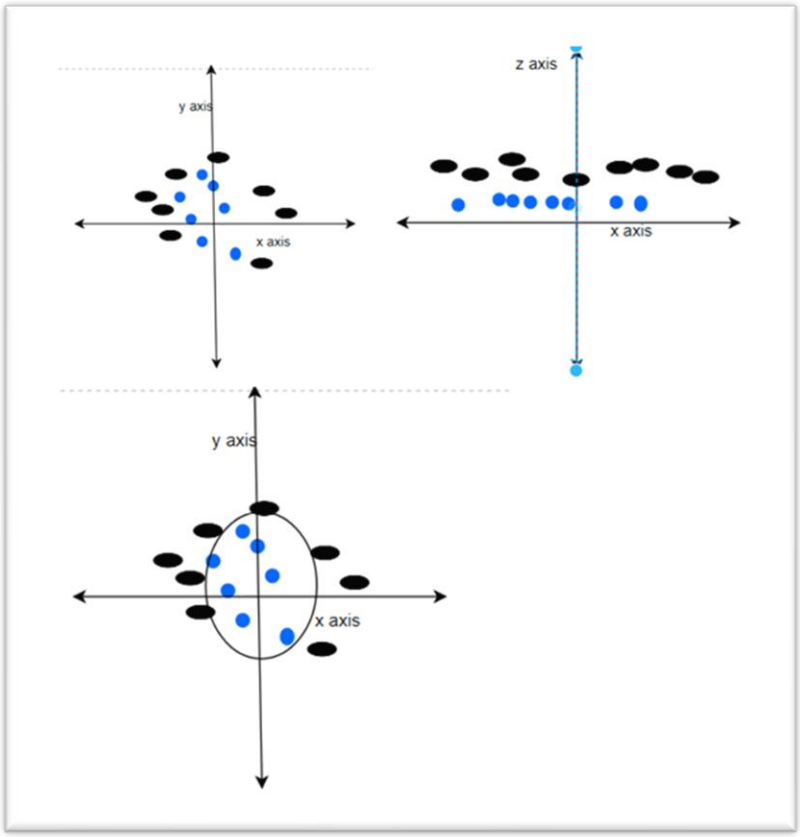


Pada kasus ini, Hyperplane dengan margin maksimum dipilih. Pada gambar diatas, hyperplane 2.1,2.2 dan 2.3 membagi titik data tetapi hyperplane 2.2 memiliki margin yang maksimum.

3. Saat terdapat outlier. Outlier atau data pencilan adalah titik data yang berbeda dari sekumpulan titik data. Dalam kasus 2 set titik data, outlier mungkin ada. SVM mengabaikan outlier tersebut dalam data dan kemudian menemukan hyperplane margin maksimum.



4. Dalam hal titik data yang terpisah secara non-linear, SVM menggunakan trik kernel. Ini akan mengubah bidang yang tidak dapat dipisahkan secara linier menjadi panel yang dapat dipisahkan dengan memperkenalkan dimensi baru. Trik kernel adalah persamaan matematika kompleks yang melakukan transformasi data kompleks untuk mengoptimalkan hyperplane. Gambar di bawah ini menunjukkan titik-titik data yang tidak dapat dipisahkan secara linier yang kemudian ditransformasikan menjadi berdimensi tinggi dengan bidang z. Hyperplane yang membagi dua set data adalah lingkaran.



E. Kernel

Algoritma SVM diimplementasikan dalam praktik menggunakan kernel. Kernel mengubah ruang data input ke dalam bentuk yang diperlukan. SVM menggunakan teknik yang disebut trik kernel. Di sini, kernel mengambil ruang input berdimensi rendah dan mengubahnya menjadi ruang berdimensi lebih tinggi. Dengan kata lain, kita dapat mengatakan bahwa itu mengubah masalah yang tidak dapat dipisahkan menjadi masalah yang dapat dipisahkan dengan menambahkan lebih banyak dimensi padanya. Hal ini paling berguna dalam masalah pemisahan non-linear. Trik kernel membantu kita membangun pengklasifikasi yang lebih akurat.

Berikut ini adalah beberapa kernel di dalam SVM:

1. Linear Kernel

Kernel linier dapat digunakan sebagai produk titik normal setiap dua pengamatan yang diberikan. Dimana diketahui bawah produk antara dua vektor adalah jumlah perkalian dari setiap pasangan nilai input.

$$K(\bar{x}) = 1\,, if\left||\bar{x}|\right| \leq 1$$

$$K(\bar{x}) = 0, Otherwise$$

2. Polynomial Kernel

Kernel polinomial adalah bentuk yang lebih umum dari kernel linier. Kernel polinomial dapat membedakan ruang input melengkung atau nonlinier.

$$K(x, y) = \tanh(\gamma \cdot x^T y + r)^d, \gamma > 0$$

3. Gaussian Kernel Radial Basis Function

Kernel fungsi basis Radial adalah fungsi kernel populer yang biasa digunakan dalam mendukung klasifikasi mesin vektor. RBF dapat memetakan ruang input dalam ruang dimensi tak terhingga.

$$K(x, y) = e^{-}\left(\gamma\left||x - y|\right|^2\right)$$

$$K(x, x1) + K(x, x2)(Simplified - Formula)$$

$$K(x, x1) + K(x, x2) > 0\ (Green)$$

$$K(x, x1) + K(x, x2) = 0\ (Red)$$

4. Sigmoid Kernel

ungsi ini setara dengan model jaringan saraf perceptron dua lapis, yang digunakan sebagai fungsi aktivasi untuk neuron buatan.

$$K(x, y) = \tanh(\gamma \cdot x^T y + r)$$

F. Kekurangan & Kelebihan

Kekurangan dari SVM adalah:

- Tidak berfungsi dengan baik dengan kumpulan data yang ukurannya lebih besar
- Terkadang, waktu pelatihan dengan SVM bisa sangat tinggi

Kelebihan dari SVM adalah:

- Akurat
- Bekerja sangat baik dengan kumpulan data terbatas
- Kernel SVM berisi fungsi transformasi non-linier untuk mengubah data rumit yang tidak dapat dipisahkan secara linier menjadi data yang dapat dipisahkan secara linier.

G. Implementasi

Berikut merupakan contoh implementasi sederhana dari SVM dengan menggunakan dataset tentang klasifikasi tipe bintang.

Sumber dataset: https://www.kaggle.com/brsdincer/star-type-classification

1. Import Library & Dataset

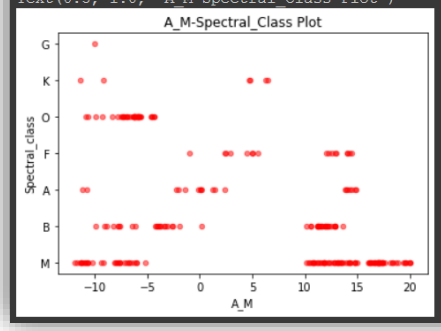
```
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

os.environ['KAGGLE_USERNAME'] = "akmalzuhdyprasetya"
os.environ['KAGGLE_KEY'] = "3f1fd1ffa5d36294293dfb098b3b7392"

!kaggle datasets download -d brsdincer/star-type-classification

!unzip -q /content/star-type-classification.zip
```

```
data = pd.read_csv("/content/Stars.csv")
data.head(11)
```

	Temperature	L	R	A_M	Color	Spectral_Class	Type
0	3068	0.002400	0.1700	16.12	Red	M	0
1	3042	0.000500	0.1542	16.60	Red	M	0
2	2600	0.000300	0.1020	18.70	Red	M	0
3	2800	0.000200	0.1600	16.65	Red	M	0
4	1939	0.000138	0.1030	20.06	Red	M	0
5	2840	0.000650	0.1100	16.98	Red	M	0
6	2637	0.000730	0.1270	17.22	Red	M	0
7	2600	0.000400	0.0960	17.40	Red	M	0
8	2650	0.000690	0.1100	17.45	Red	M	0
9	2700	0.000180	0.1300	16.05	Red	M	0
10	3600	0.002900	0.5100	10.69	Red	M	1

```
data.plot(kind='scatter', x ='A_M', y='Spectral_Class', alpha=0.5, color='red')
plt.xlabel('A_M')
plt.ylabel('Spectral_class')
plt.title('A_M-Spectral_Class Plot')
```

```
Text(0.5, 1.0, 'A_M-Spectral_Class Plot')
```



2. Pre-Processing Data

```
# label encoding
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
data['Color'] = le.fit_transform(data['Color'])
data['Spectral_Class'] = le.fit_transform(data['Spectral_Class'])
```

```
data.describe()
```

	Temperature	L	R	A_M	Color	Spectral_Class	Type
count	240.000000	240.000000	240.000000	240.000000	240.000000	240.000000	240.000000
mean	10497.462500	107188.361635	237.157781	4.382396	5.766667	3.758333	2.500000
std	9552.425037	179432.244940	517.155763	10.532512	4.208446	2.090007	1.711394
min	1939.000000	0.000080	0.008400	-11.920000	0.000000	0.000000	0.000000
25%	3344.250000	0.000865	0.102750	-6.232500	1.000000	1.000000	1.000000
50%	5776.000000	0.070500	0.762500	8.313000	8.000000	5.000000	2.500000
75%	15055.500000	198050.000000	42.750000	13.697500	8.000000	5.000000	4.000000
max	40000.000000	849420.000000	1948.500000	20.060000	16.000000	6.000000	5.000000

```
# Separate Feature and Target Matrix
x = data.drop('Type', axis=1)
y = data['Type']
```

```
# Split dataset into training set and test set
# 70% training and 30% test
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=100)
```

3. Modelling

```
# import Support Vector Classifier
from sklearn.svm import SVC
```

Linear Kernel

```
svc_linear = SVC(kernel='linear')
svc_linear.fit(x_train, y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
y_pred_linear = svc_linear.predict(x_test)
y_pred_linear
```

```
array([5, 2, 2, 1, 2, 4, 2, 4, 2, 0, 3, 0, 5, 5, 1, 3, 0, 4, 5, 5, 2, 2,
       0, 2, 3, 1, 4, 0, 0, 0, 1, 3, 2, 3, 4, 2, 2, 1, 1, 0, 0, 4, 3, 3,
       1, 3, 5, 2, 4, 0, 4, 2, 4, 3, 2, 5, 2, 5, 3, 1, 0, 1, 1, 5, 1, 2,
       4, 5, 1, 3, 5, 4])
```

Polynomial Kernel

```
svc_poly = SVC(kernel='poly', degree=8)
svc_poly.fit(x_train, y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=8, gamma='scale', kernel='poly',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
y_pred_poly = svc_poly.predict(x_test)
y_pred_poly
```

```
array([0, 0, 0, 0, 0, 4, 0, 4, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 5, 0, 0, 0,
       0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 4, 0, 0,
       0, 0, 3, 0, 0, 0, 4, 0, 0, 0, 0, 4, 0, 4, 0, 0, 0, 0, 0, 4, 0, 0,
       0, 3, 0, 0, 5, 4])
```

Gaussian Kernel

```
svc_gauss = SVC(kernel='rbf')
svc_gauss.fit(x_train, y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
y_pred_gauss = svc_gauss.predict(x_test)
y_pred_gauss
```

```
array([5, 0, 0, 0, 0, 4, 0, 4, 0, 0, 0, 0, 4, 4, 0, 0, 0, 5, 4, 5, 0, 0,
       0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 4, 0, 0,
       0, 0, 4, 0, 4, 0, 5, 0, 4, 0, 0, 4, 0, 4, 0, 0, 0, 0, 0, 5, 0, 0,
       5, 4, 0, 0, 5, 4])
```

Sigmoid Kernel

```
svc_sigmoid = SVC(kernel='sigmoid')
svc_sigmoid.fit(x_train, y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='sigmoid',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
y_pred_sigmoid = svc_sigmoid.predict(x_test)
y_pred_sigmoid
```

```
array([4, 0, 0, 0, 0, 5, 0, 4, 0, 0, 0, 0, 4, 4, 0, 0, 0, 4, 5, 4, 0, 0,
       0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 4, 0, 0,
       0, 0, 4, 0, 4, 0, 5, 0, 4, 0, 0, 4, 0, 4, 0, 0, 0, 0, 0, 5, 0, 0,
       4, 4, 0, 0, 5, 5])
```

4. Model Evaluation

```
# import evaluating helper package
from sklearn.metrics import classification_report as cr, confusion_matrix as cm
```

Linear Kernel

```
print(cm(y_test, y_pred_linear))
print(cr(y_test, y_pred_linear))
```

```
[[11  0  0  0  0  0]
 [ 0 12  0  0  0  0]
 [ 0  0 16  0  0  0]
 [ 0  0  0 11  0  0]
 [ 0  0  0  0 11  0]
 [ 0  0  0  0  0 11]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        11
           1       1.00      1.00      1.00        12
           2       1.00      1.00      1.00        16
           3       1.00      1.00      1.00        11
           4       1.00      1.00      1.00        11
           5       1.00      1.00      1.00        11

    accuracy                           1.00        72
   macro avg       1.00      1.00      1.00        72
weighted avg       1.00      1.00      1.00        72
```

Polynomial Kernel

```
print(cm(y_test, y_pred_poly))
print(cr(y_test, y_pred_poly))
```

```
[[11  0  0  0  0  0]
 [12  0  0  0  0  0]
 [16  0  0  0  0  0]
 [11  0  0  0  0  0]
 [ 4  0  0  0  7  0]
 [ 3  0  0  2  4  2]]
              precision    recall  f1-score   support

           0       0.19      1.00      0.32        11
           1       0.00      0.00      0.00        12
           2       0.00      0.00      0.00        16
           3       0.00      0.00      0.00        11
           4       0.64      0.64      0.64        11
           5       1.00      0.18      0.31        11

    accuracy                           0.28        72
   macro avg       0.30      0.30      0.21        72
weighted avg       0.28      0.28      0.19        72
```

Gaussian Kernel

```
print(cm(y_test, y_pred_gauss))
print(cr(y_test, y_pred_gauss))
```

```
[[11  0  0  0  0  0]
 [12  0  0  0  0  0]
 [16  0  0  0  0  0]
 [11  0  0  0  0  0]
 [ 0  0  0  0  7  4]
 [ 0  0  0  0  7  4]]
              precision    recall  f1-score   support

           0       0.22      1.00      0.36        11
           1       0.00      0.00      0.00        12
           2       0.00      0.00      0.00        16
           3       0.00      0.00      0.00        11
           4       0.50      0.64      0.56        11
           5       0.50      0.36      0.42        11

    accuracy                           0.31        72
   macro avg       0.20      0.33      0.22        72
weighted avg       0.19      0.31      0.20        72
```

Sigmoid Kernel

```
print(cm(y_test, y_pred_sigmoid))
print(cr(y_test, y_pred_sigmoid))
```

```
[[11  0  0  0  0  0]
 [12  0  0  0  0  0]
 [16  0  0  0  0  0]
 [11  0  0  0  0  0]
 [ 0  0  0  0  7  4]
 [ 0  0  0  0  8  3]]
              precision    recall  f1-score   support

           0       0.22      1.00      0.36        11
           1       0.00      0.00      0.00        12
           2       0.00      0.00      0.00        16
           3       0.00      0.00      0.00        11
           4       0.47      0.64      0.54        11
           5       0.43      0.27      0.33        11

    accuracy                           0.29        72
   macro avg       0.19      0.32      0.21        72
weighted avg       0.17      0.29      0.19        72
```

Source Code:

https://github.com/trueazp/Machine-Learning/blob/main/assignments/assignment05/Tugas05_ML_H071191035.ipynb