LAPORAN TUGAS BESAR

PEMBELAJARAN MESIN

Disusun oleh:

1. Chacha Alisha Dewintasari (1305220036)
2. Fasya Agneta Meliatari S.Meliala (1305223042)

Program Studi S1 Data Sains

Fakultas Informatika

Universitas Telkom

2024

DAFTAR ISI

Kata Pengantar

Assalamualaikum Wr. Wb.

Dengan penuh rasa syukur dan hormat, kami, mahasiswa Program Studi Data Sains, mengucapkan salam tulus kepada semua pihak yang telah membantu dan memberikan dukungan selama penyusunan laporan ini. Laporan tugas besar ini kami sajikan untuk memenuhi penilaian tugas besar dalam mata kuliah Pembelajaran Mesin.

Tugas ini dibagi menjadi dua bagian, yaitu supervised learning dengan metode regresi dan unsupervised learning dengan metode clustering. Bagian pertama dari tugas ini bertujuan untuk memprediksi nilai GDP dengan menggunakan atribut-atribut yang relevan seperti Export, Import, Industrial production growth rate, Investment, dan Unemployment rate. Bagian kedua dari tugas ini bertujuan untuk melakukan klustering pada data yang sama guna mengidentifikasi pola dan struktur yang tersembunyi dalam data tersebut.

Kami ingin menyampaikan terima kasih kepada dosen pengampu mata kuliah Pembelajaran Mesin yang telah memberikan bimbingan dan dorongan selama proses pembelajaran. Akhir kata, semoga laporan ini dapat memberikan kontribusi positif dalam pengembangan ilmu Pembelajaran Mesin di masa yang akan datang.

Wassalamualaikum Wr. Wb.

BAB I

Pendahuluan

1.1. Latar Belakang

Pertumbuhan ekonomi suatu negara merupakan hal yang penting untuk meningkatkan kesejahteraan masyarakatnya. Salah satu indikator yang sering digunakan untuk mengukur pertumbuhan ekonomi adalah Gross Domestic Product (GDP), yang mencerminkan nilai total dari semua barang dan jasa yang dihasilkan dalam suatu negara dalam satu periode waktu tertentu. Untuk membantu pemerintah atau lembaga terkait dalam merencanakan kebijakan ekonomi yang tepat, penting untuk dapat memprediksi GDP dengan akurat.

Dalam konteks ini, penggunaan teknik machine learning dapat memberikan kontribusi yang signifikan. Dengan menggunakan data historis mengenai Export, Import, Industrial production growth rate, Investment, dan Unemployment rate, kita dapat membangun model prediksi GDP menggunakan pendekatan supervised learning. Selain itu, dengan menggunakan pendekatan unsupervised learning, kita dapat melakukan analisis clustering untuk mengidentifikasi pola-pola menarik dalam data yang mungkin tidak terlihat secara langsung.

Melalui tugas ini, diharapkan dapat memberikan pemahaman yang lebih baik mengenai penggunaan teknik machine learning dalam menganalisa dan memprediksi pertumbuhan ekonomi suatu negara.

1.2. Rumusan Masalah

Berikut adalah rumusan masalah yang akan dibahas:

1. Pembagian data latih dan data test pada metode Supervised Learning (Regresi).
2. Mengobservasi hyperparameter dari metode yang digunakan dalam membangun model prediksi GDP.
3. Membangun model clustering.
4. Pemilihan jumlah kluster dan kualitas klustering.

1.3. Tujuan

1. Membangun model prediksi GDP yang akurat menggunakan pendekatan supervised learning.
2. Mengobservasi hyperparameter dari metode yang digunakan dalam membangun model prediksi GDP untuk memperoleh performa yang optimal.
3. Membangun model clustering untuk mengidentifikasi pola dalam data GDP menggunakan pendekatan unsupervised learning.

1.4. Manfaat

Mampu melakukan menerapkan metode supervised learning (regresi) dan metode unsupervised learning (clustering) pada data Factbook.

1.5. Deskripsi Tugas

1. Supervised learning (regresi)
Output: GDP
Atribut: Export, Import, Industrial production growth rate, Investment, Unemployment rate.
Ketentuan:
• Boleh menggunakan library, tetapi akan ada nilai tambahan jika model dibangun dari awal.
• Data dapat dinormalisasi terlebih dahulu.
• Skenario:
    - Pembagian data latih dan data test.
    - Observasi hyperparameter dari metode yang digunakan.

2. Unsupervised learning (clustering)
Atribut: Export, Import, Industrial production growth rate, Investment, Unemployment rate.
Ketentuan:
• Model harus dibangun dari awal (tidak boleh menggunakan library).
• Skenario:
    - Pemilihan jumlah kluster dan kualitas klustering.

BAB II

Metode Supervised Learning (Regresi)

2.1. Atribut dan Output

Dalam metode supervised learning (MLP), atribut-atribut yang digunakan sebagai input untuk memprediksi GDP adalah:

1. Export
2. Import
3. Industrial production growth rate
4. Investment
5. Unemployment rate

Output yang ingin kami prediksi adalah GDP (Gross Domestic Product). Dengan menggunakan atribut-atribut tersebut sebagai input, dilatih model untuk mempelajari pola dan hubungan antara atribut-atribut tersebut dengan GDP, sehingga dapat dilakukan prediksi GDP yang akurat berdasarkan nilai atribut yang diberikan.

2.2. Library

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.impute import KNNImputer
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.ensemble import IsolationForest
from sklearn.impute import KNNImputer
import matplotlib.pyplot as plt
```

Library yang kami pakai untuk metode Supervised Learning ini sebagai berikut:

1. pandas: Digunakan untuk membaca dataset dari file Excel dan melakukan manipulasi data tabular.
2. sklearn.model_selection.train_test_split: Digunakan untuk membagi dataset menjadi data latih dan data uji.
3. sklearn.preprocessing.StandardScaler: Digunakan untuk melakukan normalisasi data.
4. sklearn.neural_network.MLPRegressor`: Digunakan untuk membangun model regresi menggunakan Multi-Layer Perceptron (MLP).
5. sklearn.metrics.mean_squared_error,sklearn.metrics.mean_absolute_error, sklearn.metrics.r2_score: Digunakan untuk menghitung evaluasi performa model seperti mean squared error, mean absolute error, dan coefficient of determination (R^2).

6. sklearn.ensemble.IsolationForest: Digunakan untuk mendeteksi dan menghapus outlier dari dataset.
7. sklearn.impute.KNNImputer: Digunakan untuk mengisi nilai yang hilang dalam dataset menggunakan metode K-Nearest Neighbors.
8. Matplotlib: Untuk visualisasi data.

2.3. Pre Processing Data

2.3.1. pembersihan dan meratakan nama kolom

```
def bersihkan_nama_kolom(columns):
    return columns.str.strip().str.lower().str.replace(' ', '').str.replace(r'\W', '', regex=True)

df.columns = bersihkan_nama_kolom(df.columns)
```

pembersihan dan meratakan nama kolom, hal ini dilakukan untuk mempermudah akses ke kolom dan mengurangi kesalahan yang disebabkan oleh perbedaan kecil dalam penamaan.

2.3.2. Konversi nilai moneter dari format string ke format numerik

```
# Fungsi untuk mengonversi nilai moneter dari string ke numerik
def convert_to_numeric(value):
    return float(value.replace("$", "").replace(",", ""))

# Konversi kolom ke numerik
data["gdp"] = data["gdp"].apply(convert_to_numeric)
data["exports"] = data["exports"].apply(convert_to_numeric)
data["imports"] = data["imports"].apply(convert_to_numeric)
print(data.isnull().sum())
```

Fungsi convert to numeric dilakukan dengan menghapus simbol mata uang, sehingga nilai-nilai tersebut dapat digunakan dalam analisis numerik.

2.3.3. Transformasi dan Imputasi Data Numerik

```
# Mengubah kolom menjadi tipe data numerik, nilai non-numerik akan menjadi NaN
data['industrialproductiongrowthrate'] = pd.to_numeric(data['industrialproductiongrowthrate'], errors='coerce')
data['investment'] = pd.to_numeric(data['investment'], errors='coerce')
data['unemploymentrate'] = pd.to_numeric(data['unemploymentrate'], errors='coerce')

# Menggunakan SimpleImputer untuk menggantikan nilai NaN dengan nilai rata-rata kolom
imputer = KNNImputer()

# Mengimputasi kolom-kolom yang memiliki nilai NaN
data[['industrialproductiongrowthrate', 'investment', 'unemploymentrate']] = imputer.fit_transform(data[['industrialproductiongrowthrate', 'investment', 'unemploymentrate']])
```

Data pada kolom 'industrialproductiongrowthrate', 'investment', dan 'unemploymentrate' diubah menjadi tipe data numerik. Nilai non-numerik diubah menjadi NaN. Selanjutnya, nilai NaN digantikan dengan nilai rata-rata kolom

menggunakan SimpleImputer. Hal ini dilakukan untuk menjaga integritas data dan memastikan data yang digunakan dalam analisis lengkap dan dapat diproses secara numerik.

2.3.4. Normalisasi Data

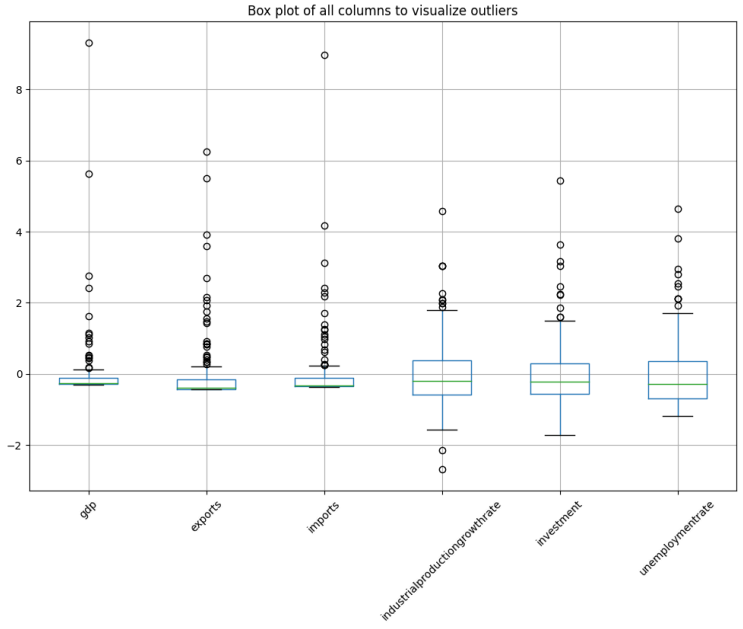
```
[988] # Normalisasi data
      scaler = StandardScaler()
      normalized_data = scaler.fit_transform(data)
      normalized_data = pd.DataFrame(normalized_data, columns=data.columns)
```

Normalisasi data menggunakan StandardScaler. Normalisasi dilakukan untuk mengubah skala data sehingga memiliki rata-rata 0 dan varians 1. Ini membantu dalam memastikan bahwa setiap fitur memiliki pengaruh yang seimbang dalam model yang dibangun. Hasil normalisasi disimpan kembali sebagai DataFrame dengan nama kolom yang sama seperti data asli.

2.3.5. Outliers

Melakukan pengecekan persebaran data yang telah dilakukan normalisasinya

```
#Box Plot Visulisasi Outliers
plt.figure(figsize=(12, 8))
normalized_data.boxplot()
plt.title('Box plot of all columns to visualize outliers')
plt.xticks(rotation=45)
plt.show()
```



```
] # Menghapus outliers
  outlier_detector = IsolationForest(contamination=0.1)
  outliers = outlier_detector.fit_predict(normalized_data)
  cleaned_data = normalized_data[outliers == 1]
```

Metode menghilangkan outlier digunakan untuk membersihkan atau menghapus nilai-nilai ekstrem yang jauh dari sebagian besar data dalam sebuah dataset. Outlier adalah nilai yang secara signifikan berbeda dari nilai-nilai lain dalam dataset, baik itu berbeda secara positif maupun negatif.

Outlier dapat mempengaruhi analisis statistik dan model yang dibangun dari dataset. Oleh karena itu, menghilangkan outlier dapat membantu meningkatkan keandalan dan interpretasi hasil analisis data.

Dengan dilakukannya penghapusan outlier didapatkan data yang lebih seimbang sehingga prediksi yang dilakukan oleh model akan lebih optimal.

2.4. Pembagian Data latih dan Data Test

```
]#Membagi Data Set
 X = cleaned_data.drop('gdp', axis=1)
 y = cleaned_data['gdp']
```

```
#Membagi data train dan data test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Pembagian data menjadi data latih (training set) dan data uji (test set) dilakukan menggunakan fungsi train_test_split dari library sklearn.model_selection.

X adalah atribut yang telah dinormalisasi dan y adalah target. test_size=0.2 menentukan bahwa 20% data akan digunakan sebagai data uji, sedangkan 80% data akan digunakan sebagai data latih. random_state=42 digunakan untuk menjaga konsistensi pembagian data jika kode dijalankan ulang.

Setelah proses pembagian, data latih akan disimpan dalam variabel X_train (atribut data latih) dan y_train (target data latih), sedangkan data uji akan disimpan dalam variabel X_test (atribut data uji) dan y_test (target data uji).

2.5. Membangun model MLP

Metode supervised learning(Multi Layer Perceptron) merupakan pendekatan dalam pembelajaran mesin di mana model atau algoritma belajar dari contoh-contoh yang telah dilabeli. Dalam tugas besar ini, kami menggunakan metode supervised learning yang disebut Multi Layer Perceptron (MLP) untuk memprediksi GDP suatu negara berdasarkan atribut-atribut tertentu.

2.5.1 Inisiasi dan Melatih Model

```
# Inisialisasi dan melatih MLPRegressor
mlp_regressor = MLPRegressor(hidden_layer_sizes=(100, 50), activation='relu', solver='adam', max_iter=1000)
mlp_regressor.fit(X_train, y_train)
```

Membuat dan melatih model Regresi menggunakan Multi-layer Perceptron (MLP) dengan struktur jaringan yang telah ditentukan. Model ini akan digunakan untuk mempelajari pola dalam data latih (`X_train`, `y_train`) dan menghasilkan prediksi yang akurat pada data uji (`X_test`, `y_test`). Dengan melakukan pelatihan ini, kita dapat menghasilkan model yang dapat digunakan untuk memprediksi nilai target berdasarkan fitur-fitur yang ada dalam data.

2.5.2 Prediksi Data

```
# Membuat prediksi pada data latih
y_pred_train = mlp_regressor.predict(X_train)
y_pred_test = mlp_regressor.predict(X_test)
```

Membuat prediksi pada data Latih dan data uji, menggunakan model yang telah dilatih untuk membuat prediksi pada data latih (X_train) dan data uji (X_test) dengan memanggil method predict.

2.5.3. Evaluasi Kinerja Model

```
# Evaluasi Kinerja Model
mse_train = mean_squared_error(y_train, y_pred_train)
mse_test = mean_squared_error(y_test, y_pred_test)
mae_test = mean_absolute_error(y_test, y_pred_test)
r2_test = r2_score(y_test, y_pred_test)
```

- Mean Squared Error (MSE): Mengukur rata-rata dari kuadrat selisih antara nilai aktual (y_train, y_test) dan nilai prediksi (y_pred_train, y_pred_test). Semakin rendah nilai MSE, semakin baik kinerja model.
- Mean Absolute Error (MAE): Mengukur rata-rata dari nilai absolut dari selisih antara nilai aktual dan nilai prediksi. Semakin rendah nilai MAE, semakin baik kinerja model.
- R-squared (R^2): Mengukur seberapa baik variabilitas dari nilai target yang dapat dijelaskan oleh model. Nilai R^2 berkisar antara 0 dan 1, dan semakin mendekati 1, semakin baik modelnya.

2.6. Hasil dan Analisis

```
print("Training Score (R^2):", mlp_regressor.score(X_train, y_train))
print("Test Score (R^2):", mlp_regressor.score(X_test, y_test))
print("R^2 Score:", r2_test)
print("Mean Squared Error:", mse_test)
print("Mean Absolute Error:", mae_test)
```

```
Training Score (R^2): 0.6315334179824482
Test Score (R^2): 0.5146872522993096
R^2 Score: 0.5146872522993096
Mean Squared Error: 0.018423464133069826
Mean Absolute Error: 0.09258073034013033
```

- Training Score (R^2): Nilai R^2 untuk data latih menunjukkan seberapa baik model dapat menjelaskan variabilitas dalam data. Nilai tersebut adalah sekitar 0.632, yang berarti sekitar 63,2% dari variabilitas dalam data latih dapat dijelaskan oleh model.
- Test Score (R^2): Nilai R^2 untuk data uji menunjukkan seberapa baik model dapat menjelaskan variabilitas dalam data uji yang tidak pernah dilihat sebelumnya. Nilai tersebut adalah sekitar 0.515, yang berarti sekitar 51,5% dari variabilitas dalam data uji dapat dijelaskan oleh model.
- R^2 Score: Ini adalah nilai R^2 untuk data uji.
  Mean Squared Error (MSE): Ini adalah rata-rata dari kuadrat selisih antara nilai aktual dan nilai prediksi. Semakin rendah nilai MSE, semakin baik kinerja model. Nilai MSE model kami adalah sekitar 0.0184.
- Mean Absolute Error (MAE): Ini adalah rata-rata dari nilai absolut dari selisih antara nilai aktual dan nilai prediksi. Semakin rendah nilai MAE, semakin baik kinerja model. Nilai MAE model kami adalah sekitar 0.0926.

2.7. Observasi hyperparameter

Arsitektur MLP (Multilayer Perceptron) yang digunakan dalam model kami adalah sebagai berikut:

- Jumlah layer: MLP memiliki 2 hidden layer.
- Jumlah neuron: Hidden layer pertama memiliki 100 neuron, dan hidden layer kedua memiliki 50 neuron.
- Fungsi aktivasi: Fungsi aktivasi yang digunakan pada setiap neuron adalah ReLU (Rectified Linear Unit).
- Solver: Solver yang digunakan adalah Adam. Adam merupakan optimizer yang menggabungkan algoritma Adam dan RMSprop untuk efisiensi komputasi dan kecepatan konvergensi yang lebih baik.
- Jumlah iterasi: MLP akan melatih model dengan melakukan maksimal 1000 iterasi.

Pemilihan hyperparameter pada MLP dapat mempengaruhi kinerja dan performa model. Dalam kode tersebut, hyperparameter yang dipilih secara subjektif dan dapat dieksplorasi lebih lanjut.

Beberapa pertimbangan dalam pemilihan hyperparameter adalah:

- Jumlah layer dan jumlah neuron: Jumlah layer dan neuron dapat mempengaruhi kapasitas model. Jika terlalu sedikit, model mungkin tidak dapat menangkap pola yang kompleks dalam data. Jika terlalu banyak, model mungkin menjadi terlalu kompleks dan rentan terhadap overfitting. Oleh karena itu, pemilihan jumlah layer dan neuron harus dilakukan berdasarkan karakteristik dataset dan kompleksitas masalah yang ingin dipecahkan.
- Fungsi aktivasi: Fungsi aktivasi mengenalkan non-linearitas ke dalam model, yang diperlukan untuk memodelkan hubungan yang kompleks antara atribut dan target. Fungsi ReLU merupakan pilihan umum karena efisiensi komputasi dan kemampuan untuk mengatasi masalah gradien yang menghilang (vanishing gradient).
- Solver: Solver digunakan untuk mengoptimalkan parameter model selama proses pelatihan. Adam adalah pilihan yang umum digunakan karena efisiensi dan kemampuan adaptasinya terhadap laju pembelajaran pada setiap parameter.
- Jumlah iterasi: Jumlah iterasi menentukan berapa kali model akan melihat dataset selama pelatihan. Jumlah iterasi yang cukup memungkinkan model untuk menemukan titik optimal dalam ruang parameter, tetapi terlalu banyak iterasi dapat menyebabkan overfitting atau waktu pelatihan yang lama.

Pemilihan hyperparameter yang tepat melibatkan eksperimen dan penyesuaian berdasarkan performa model pada dataset yang spesifik.

BAB III

Metode Unsupervised Learning (Clustering)

3.1 Atribut dan Output

Atribut yang digunakan dalam model ini adalah sebagai berikut:

- Export: Nilai ekspor suatu negara.
- Import: Nilai impor suatu negara.
- Industrial production growth rate: Tingkat pertumbuhan produksi industri.
- Investment: Tingkat investasi.
- Unemployment rate: Tingkat pengangguran.

Output yang dihasilkan dari model ini adalah pengelompokan (clustering) ke dalam beberapa kelompok berdasarkan kemiripan atribut-atribut tersebut. Dengan menggunakan metode ini, kita dapat mengidentifikasi pola dan hubungan antara variabel-variabel ekonomi yang berbeda.

3.2 Metodologi

3.2.1 Principal Component Analysis (PCA)

PCA adalah teknik reduksi dimensi yang digunakan untuk mengurangi jumlah variabel dalam dataset sambil mempertahankan sebanyak mungkin informasi yang ada. Dalam laporan ini, PCA digunakan untuk mengurangi kompleksitas data dari 5 variabel menjadi 2 variabel dan membantu visualisasi hasil clustering.

3.2.2 K-means Clustering

K-means adalah algoritma clustering yang membagi data ke dalam k kelompok berdasarkan jarak terdekat antara titik data dengan pusat cluster (centroid). Langkah-langkah dalam K-means adalah:

1. Menentukan jumlah cluster k.
2. Menentukan titik pusat awal (centroid) secara acak.
3. Mengelompokkan setiap titik data ke centroid terdekat.
4. Menghitung ulang centroid berdasarkan rata-rata dari titik-titik data dalam setiap cluster.
5. Mengulangi proses hingga konvergensi tercapai.

3.3 Implementasi

3.3.1 Library yang digunakan

Untuk mengimplementasikan model ini, kami menggunakan bahasa pemrograman Python dengan beberapa library yang mendukung analisis data dan machine learning.

```
# import library
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

- Pandas: Untuk manipulasi dan analisis data
- NumPy: Untuk operasi numerik
- Matplotlib: Untuk visualisasi data

3.3.2 Pre Processing

1. **Membersihkan Nama Kolom dan Membuat Data Frame**

```
#Membersihkan nama kolom dari spasi, karakter non-alfanumerik dan untuk mengkonversi karakter menjadi huruf kecil
def bersihkan_nama_kolom(columns):
    return columns.str.strip().str.lower().str.replace(' ', '').str.replace(r'\W', '', regex=True)

df.columns = bersihkan_nama_kolom(df.columns)
```

```
# Membuat data frame baru dengan fitur yang akan digunakan
data = df[['exports', 'imports', 'industrialproductiongrowthrate', 'investment', 'unemploymentrate']]

data.info()
```

Membersihkan nama kolom bertujuan untuk memastikan bahwa nama-nama kolom dalam data frame bersih dan konsisten, hal ini digunakan agar lebih mudah analisis dan mencegah kesalahan karena inkonsisten penamaan.

2. **Mengkonversi Tipe Data**

```
# Mengkonversi tipe data dari object ke float
def convert_to_numeric(value):
    try:
        return float(value.replace("$", "").replace(",", ""))
    except ValueError:
        return np.nan

data["exports"] = data["exports"].apply(convert_to_numeric)
data["imports"] = data["imports"].apply(convert_to_numeric)
data['industrialproductiongrowthrate'] = pd.to_numeric(data['industrialproductiongrowthrate'], errors='coerce')
data['investment'] = pd.to_numeric(data['investment'], errors='coerce')
data['unemploymentrate'] = pd.to_numeric(data['unemploymentrate'], errors='coerce')

data.info()
```

Fungsi convert_to_numeric dirancang untuk mengubah nilai dalam bentuk string (object) yang mungkin mengandung simbol dolar ($) dan tanda koma (,) menjadi nilai float.

Kode selanjutnya digunakan untuk mengkonversi tipe data dari object ke float dan menangani nilai tidak valid dengan mengubahnya menjadi NaN.

3. Mengatasi Null/NaN Values

```
# Mengecek null value
data.isnull().sum()
```

```
exports                           0
imports                           0
industrialproductiongrowthrate    15
investment                        5
unemploymentrate                  22
dtype: int64
```

```
# Mengisi data kosong dengan rata rata
avg_ipgr = data['industrialproductiongrowthrate'].astype('float').mean(axis=0)
data.loc[:, 'industrialproductiongrowthrate'] = data['industrialproductiongrowthrate'].replace(np.nan, avg_ipgr)

avg_investment = data['investment'].astype('float').mean(axis=0)
data.loc[:, 'investment'] = data['investment'].replace(np.nan, avg_investment)

avg_unemploymentrate = data['unemploymentrate'].astype('float').mean(axis=0)
data.loc[:, 'unemploymentrate'] = data['unemploymentrate'].replace(np.nan, avg_unemploymentrate)

# Mengecek Null Values
data.isnull().sum()
```

```
exports                           0
imports                           0
industrialproductiongrowthrate    0
investment                        0
unemploymentrate                  0
dtype: int64
```

Langkah pertama adalah mengecek apakah terdapat null value dalam kolom yang berada dalam data frame. Dalam pengecekan didapatkan bahwa masih ada kolom yang memiliki Null Values. Kemudian Null Values diisi menggunakan rata rata dalam kolom tersebut.

4. Normalisasi Data

```
# Melakukan Normalisasi terhadap data menggunakan min_max scaling
def min_max_scaling(data):
    min_val = data.min()
    max_val = data.max()
    scaled_data = (data - min_val) / (max_val - min_val)
    return scaled_data

scaled_data = min_max_scaling(data)
```

Data dilakukan normalisasi dengan menggunakan Min-Max Scaling. Normalisasi diperlukan karena algoritma K-means sensitif terhadap skala data. Normalisasi membantu memastikan bahwa setiap fitur berkontribusi secara proporsional terhadap model.

5. **Mereduksi Dimensi**

```
# Mereduksi dimensi data menggunakan teknik PCA secara manual

def pca_manual(data, n_components):
    # Standarisasi data
    data_meaned = data - np.mean(data, axis=0)
    # Menghitung matriks kovariansi
    cov_matrix = np.cov(data_meaned, rowvar=False)
    # Menghitung eigenvalues dan eigenvectors
    eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)
    # Mengurutkan eigenvalues dan eigenvectors
    sorted_index = np.argsort(eigenvalues)[::-1]
    sorted_eigenvalues = eigenvalues[sorted_index]
    sorted_eigenvectors = eigenvectors[:, sorted_index]
    # Memilih komponen utama
    eigenvector_subset = sorted_eigenvectors[:, 0:n_components]
    # Proyeksi data ke komponen utama
    data_reduced = np.dot(eigenvector_subset.transpose(), data_meaned.transpose()).transpose()
    return data_reduced

# Menggunakan PCA manual untuk mereduksi data ke 2 dimensi
data_pca = pca_manual(scaled_data, 2)
```

Mereduksi data menggunakan teknik Principal Component Analysis (PCA). Kami melakukan pengurangan dimensi dari 5 menjadi 2. Dengan mengurangi dimensi data, kita dapat memperoleh representasi yang lebih sederhana dan kurang rumit dari data asli.

3.3.3 K-Means

1. **Fungsi-fungsi yang diperlukan**

```
# Fungsi untuk menghitung jarak
def euclidean_distance(a, b):
    return np.sqrt(np.sum((a - b)**2))
```

Fungsi ini menghitung jarak euclidean antara dua titik a dan b.

```
# Menginisialisasi posisi awal centroid
def initialize_centroids(data, k):
    indices = np.random.choice(data.shape[0], k, replace=False)
    return data[indices]
```

Fungsi ini menginisialisasi posisi awal dari k centroid secara acak dari data.

```python
# Menempatkan titik ke kluster terdekat
def assign_clusters(data, centroids):
    clusters = []
    for point in data:
        distances = [euclidean_distance(point, centroid) for centroid in centroids]
        cluster = np.argmin(distances)
        clusters.append(cluster)
    return np.array(clusters)
```

Fungsi ini menetapkan setiap titik data ke cluster terdekat berdasarkan jarak ke centroid.

```python
#Update Centroid
def update_centroids(data, clusters, k):
    new_centroids = []
    for i in range(k):
        cluster_points = data[clusters == i]
        if len(cluster_points) > 0:
            new_centroid = cluster_points.mean(axis=0)
            new_centroids.append(new_centroid)
        else:
            new_centroids.append(data[np.random.choice(data.shape[0])])
    return np.array(new_centroids)
```

Fungsi ini memperbarui posisi centroid berdasarkan rata-rata dari titik-titik dalam kluster.

```python
# Algoritma K-means
def kmeans(data, k, max_iters=100):
    centroids = initialize_centroids(data, k)
    for _ in range(max_iters):
        clusters = assign_clusters(data, centroids)
        new_centroids = update_centroids(data, clusters, k)
        if np.all(centroids == new_centroids):
            break
        centroids = new_centroids
    return clusters, centroids
```

Fungsi ini merupakan implementasi algoritma K-Means clustering. Iteratif memperbarui posisi centroid dan menetapkan titik-titik data ke kluster yang sesuai hingga konvergensi atau maksimum iterasi tercapai.

```python
# Menghitung within-cluster sum of squares
def calculate_wcss(data, clusters, centroids):
    wcss = 0
    for i, centroid in enumerate(centroids):
        cluster_points = data[clusters == i]
        wcss += np.sum((cluster_points - centroid) ** 2)
    return wcss
```

Fungsi ini menghitung within-cluster sum of squares (WCSS), yaitu jumlah kuadrat jarak antara setiap titik dalam kluster dengan centroidnya.

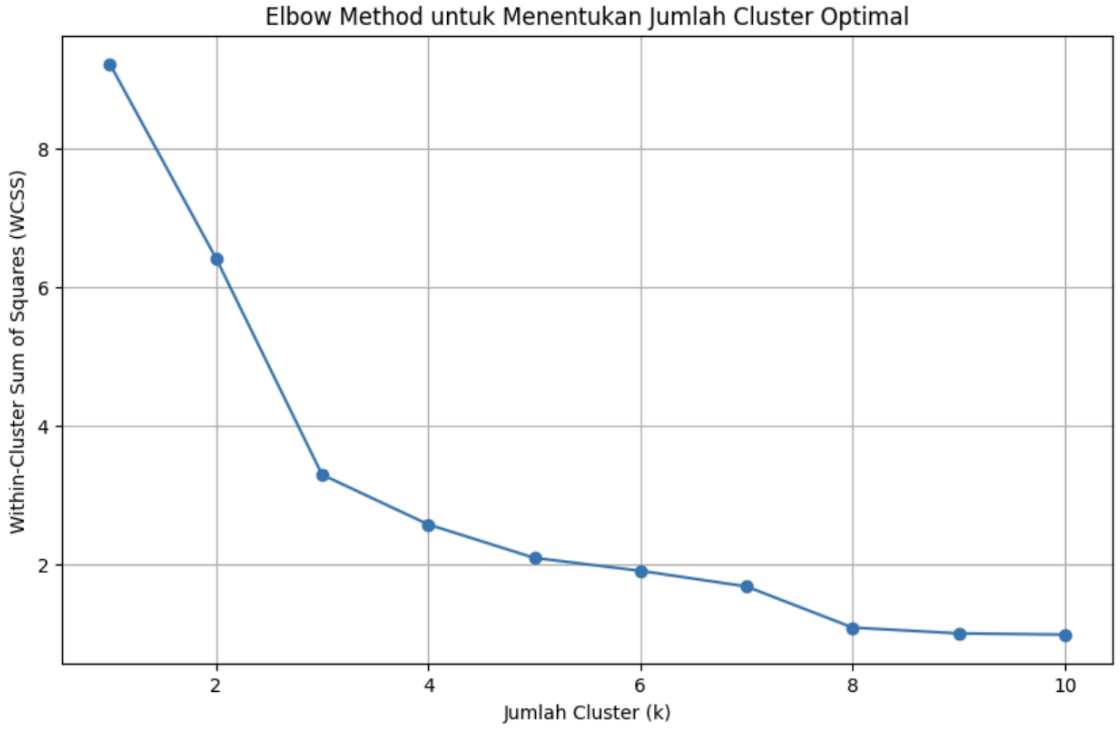
2. Fungsi Menentukan jumlah Cluster (Elbow Method)

```
# Menentukan rentang nilai k yang akan diuji
k_values = range(1, 11)
wcss_values = []
```

```
# menentukan jumlah kluster yang optimal
for k in k_values:
    clusters, centroids = kmeans(data_pca, k)
    wcss = calculate_wcss(data_pca, clusters, centroids)
    wcss_values.append(wcss)
```

Iterasi dilakukan untuk setiap nilai k dari 1 hingga 10. WCSS dihitung dengan menjalankan K-Means clustering untuk setiap nilai k.

```
# Plot Elbow Method
plt.figure(figsize=(10, 6))
plt.plot(k_values, wcss_values, marker='o')
plt.title('Elbow Method untuk Menentukan Jumlah Cluster Optimal')
plt.xlabel('Jumlah Cluster (k)')
plt.ylabel('Within-Cluster Sum of Squares (WCSS)')
plt.grid(True)
plt.show()
```



Plot menunjukkan hubungan antara jumlah cluster (k) dan WCSS. Tujuannya adalah untuk menemukan "siku" dalam plot, dimana penurunan WCSS mulai melambat secara signifikan. Ini menandakan bahwa penambahan lebih banyak cluster tidak memberikan penurunan signifikan dalam WCSS.

3. Menjalankan K-Means

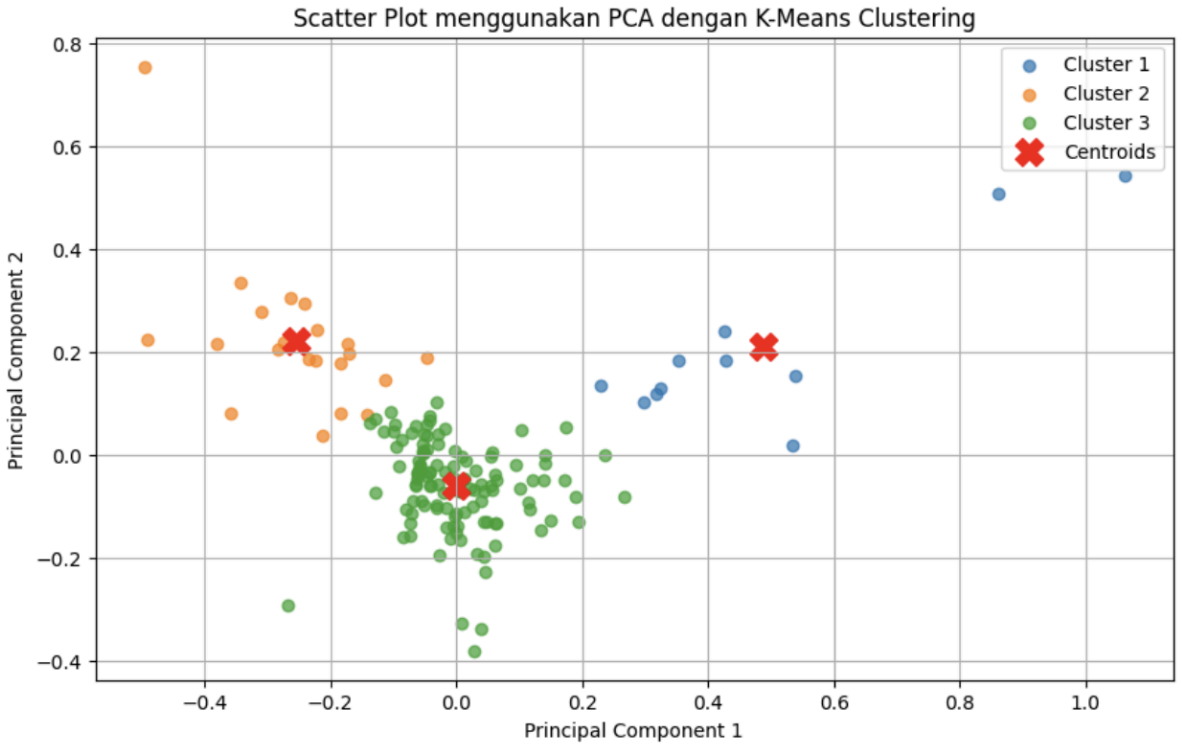
```
# Menentukan jumlah cluster optimal berdasarkan Elbow Method
optimal_k = 3

# Menjalankan K-Means clustering dengan jumlah cluster optimal
clusters, centroids = kmeans(data_pca, optimal_k)
```

K-Means clustering dijalankan lagi dengan jumlah cluster optimal yang telah ditentukan. Jika dilihat dari Plot Elbow Method cluster optimal berada di cluster 3.

3.4 Hasil dan Analisis

```
# Plot hasil clustering dengan jumlah cluster optimal
plt.figure(figsize=(10, 6))
for i in range(optimal_k):
    cluster_points = data_pca[clusters == i]
    plt.scatter(cluster_points[:, 0], cluster_points[:, 1], alpha=0.7, label=f'Cluster {i+1}')
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', marker='X', s=200, label='Centroids')
plt.title('Scatter Plot menggunakan PCA dengan K-Means Clustering')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.grid(True)
plt.show()
```



```python
# Fungsi untuk menghitung Silhouette Score
def silhouette_score(data, clusters, centroids):
    silhouette_scores = []
    for i in range(data.shape[0]):
        own_cluster = clusters[i]
        own_cluster_points = data[clusters == own_cluster]
        a = np.mean([euclidean_distance(data[i], point) for point in own_cluster_points])

        b = np.min([np.mean([euclidean_distance(data[i], point) for point in data[clusters == j]])
                    for j in range(len(centroids)) if j != own_cluster])

        silhouette_scores.append((b - a) / max(a, b))

    return np.mean(silhouette_scores)
```

```python
silhouette_avg = silhouette_score(data_pca, clusters, centroids)
print(f'Silhouette Score: {silhouette_avg:.2f}')
```

```
Silhouette Score: 0.58
```

Fungsi tersebut menghitung Silhouette Score untuk hasil clustering yang diberikan, yang memberikan ukuran seberapa baik setiap titik cocok dengan klusternya sendiri dan seberapa terpisah titik tersebut dari kluster lain. Silhouette Score yang lebih tinggi menunjukkan bahwa hasil clustering lebih baik, dengan nilai maksimum 1 menunjukkan clustering yang sempurna. Dalam kasus ini, Silhouette Score sebesar 0.58 menunjukkan bahwa hasil clustering relatif baik.

BAB IV

Analisis dan Kesimpulan

4.1 Metode Supervised Learning (Regresi)

Regresi Multilayer Perceptron (MLP) adalah metode regresi yang menggunakan jaringan saraf tiruan dengan beberapa lapisan neuron. Model ini belajar dengan mengoptimalkan fungsi kerugian seperti Mean Squared Error (MSE) antara nilai prediksi dan nilai sebenarnya.

Mean Squared Error (MSE) sebesar 0.0184 menunjukkan bahwa rata-rata dari kuadrat selisih antara nilai aktual dan nilai prediksi relatif kecil. Hal ini menunjukkan bahwa model memiliki tingkat akurasi yang tinggi dalam melakukan prediksi.

4.2 Metode Unsupervised Learning (Clustering)

K-Means adalah salah satu algoritma clustering yang paling efisien secara komputasi, dan menggunakan PCA untuk mereduksi dimensi data dapat meningkatkan efisiensi algoritma.

Silhouette Score sebesar 0.58 menunjukkan bahwa sebagian besar titik data cocok dengan kluster mereka sendiri lebih baik daripada dengan kluster lain. Ini menunjukkan adanya sejumlah besar titik yang terklaster dengan baik.

BAB V

Lampiran Tugas

Link colab regresi : REGRESI.ipynb

Link colab clustering : Clustering ML Tubes.ipynb

Link PPT: PPT