Tugas perbaikan

Bab 6

Nama : muhammad makhlufi makbullah

Kelas : TK 45 01

NIM : 1103210171

"Introduction to Machine Learning with Python"

Bab ini membahas cara menggabungkan beberapa langkah pemrosesan data dan pelatihan model menjadi sebuah proses yang terstruktur menggunakan **pipelines**. Pipelines membantu membuat kode lebih modular, terorganisir, dan meminimalkan kemungkinan kesalahan.

Sub-Bab Penjelasan

1. What are Pipelines?

- **Pipeline** adalah alat untuk menggabungkan beberapa langkah pemrosesan data (seperti preprocessing, scaling, atau encoding) dan pelatihan model menjadi satu objek.
- Pipelines memastikan bahwa semua langkah diatur secara berurutan dan dapat diproses bersamaan.
- Building Pipelines adalah cara untuk menyusun beberapa langkah preprocessing dan pelatihan model menjadi satu objek.

```
from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC


pipeline = Pipeline([

   ('scaler', StandardScaler()),

   ('svc', SVC())

])
```

- Contoh sederhana:
    - Langkah 1: Normalisasi data.
    - Langkah 2: Pelatihan model dengan data yang dinormalisasi.

2. Benefits of Pipelines

- **Reproducibility**: Mengurangi risiko kesalahan, seperti menerapkan preprocessing hanya pada data pelatihan tetapi tidak pada data uji.
- **Code Modularity**: Langkah-langkah preprocessing dan model dapat didefinisikan secara terpisah dan digabungkan dalam pipeline.
- **Ease of Use**: Mempermudah penggunaan cross-validation dan grid search tanpa harus menulis ulang proses preprocessing.

3. Building Pipelines in scikit-learn

- Scikit-learn menyediakan kelas Pipeline untuk membuat pipeline.
- Contoh:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
pipeline = Pipeline([
   ('scaler', StandardScaler()),  # Step 1: Scaling
   ('svc', SVC())                 # Step 2: Model
])
pipeline.fit(X_train, y_train)
```

4. Pipelines with Grid Search

- Pipelines dapat dikombinasikan dengan **GridSearchCV** untuk melakukan pencarian parameter secara otomatis.
- Parameter dalam setiap langkah pipeline dapat dioptimalkan secara bersamaan.
- Contoh:

```
from sklearn.model_selection import GridSearchCV
param_grid = {'svc__C': [0.1, 1, 10], 'svc__kernel': ['linear', 'rbf']}
grid = GridSearchCV(pipeline, param_grid, cv=5)
```

```
grid.fit(X_train, y_train)
print(grid.best_params_)
```

5. More Complex Pipelines

- Pipelines dapat memiliki langkah preprocessing tambahan, seperti:
    - Encoding untuk data kategori.
    - Imputasi nilai yang hilang.
- Contoh pipeline yang lebih kompleks:

```
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
# Define preprocessing for numerical and categorical data
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Combine preprocessors
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
```

```
    ]
)


# Add model to the pipeline
clf = Pipeline(steps=[('preprocessor', preprocessor),
                      ('classifier', SVC())])
```

6. Caching Transformers

- Caching memungkinkan langkah preprocessing yang berat untuk dijalankan sekali saja, sehingga meningkatkan efisiensi.
- Di scikit-learn, gunakan parameter memory pada pipeline untuk menyimpan hasil sementara.

7. Summary

- Pipelines sangat membantu untuk:
    - Mempermudah preprocessing data.
    - Mengintegrasikan proses machine learning menjadi satu alur.
    - Mengurangi risiko kesalahan.
- Dengan pipelines, proses seperti grid search, cross-validation, dan deployment model menjadi lebih efisien dan terstruktur.

penjelasan masing-masing sub-bab dari Bab 6: Algorithm Chains and Pipelines:

1. Parameter Selection with Preprocessing

- Parameter tuning sering kali memerlukan preprocessing data terlebih dahulu, seperti scaling atau encoding.
- Contohnya:
    - Model SVM membutuhkan data yang di-scale agar performanya optimal.
    - Langkah preprocessing harus diterapkan pada data pelatihan dan data uji secara konsisten.
- **Tantangan**: Jika preprocessing dilakukan secara terpisah dari tuning, ada risiko kebocoran data (data leakage).
- **Solusi**: Gunakan pipeline untuk menyatukan preprocessing dan model dalam satu alur, sehingga parameter tuning mempertimbangkan langkah preprocessing.

---

2. Building Pipelines

- **Pipeline** adalah cara untuk menyusun beberapa langkah preprocessing dan pelatihan model menjadi satu objek.
- Langkah-langkah yang disusun dalam pipeline:
    1. Preprocessing (misalnya, imputasi nilai hilang, normalisasi).
    2. Model machine learning.
- Contoh:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
pipeline = Pipeline([
   ('scaler', StandardScaler()),
   ('svc', SVC())
])
```

---

3. Using Pipelines in Grid Searches

- **Grid Search** mencari kombinasi parameter terbaik, termasuk parameter dari preprocessing.
- Contoh:
    - Parameter untuk langkah preprocessing (misalnya, jenis scaler atau jumlah fitur).
    - Parameter model (misalnya, nilai C dan kernel pada SVM).
- Dalam pipeline, nama langkah dan parameter dapat digabungkan menggunakan notasi step__parameter. Contoh:

param_grid = {'svc__C': [0.1, 1, 10], 'svc__kernel': ['linear', 'rbf']}

---

4. The General Pipeline Interface

- Pipeline di scikit-learn memiliki antarmuka yang sama dengan estimator lainnya (fit, predict, dll.).
- Setiap langkah dalam pipeline harus memiliki:
    - fit untuk menyesuaikan data.
    - transform untuk memodifikasi data (kecuali langkah terakhir, yaitu model).

---

5. Convenient Pipeline Creation with make_pipeline

- Fungsi make_pipeline adalah cara lebih singkat untuk membuat pipeline tanpa memberikan nama langkah.
- Nama langkah akan dibuat secara otomatis berdasarkan nama kelas.
- Contoh:

from sklearn.pipeline import make_pipeline

pipeline = make_pipeline(StandardScaler(), SVC())

---

6. Accessing Step Attributes

- Atribut dari setiap langkah pipeline dapat diakses menggunakan nama langkah.
- Contoh:

pipeline.named_steps['scaler']

- Berguna untuk memeriksa parameter atau hasil pada langkah tertentu.

---

7. Accessing Attributes in a Grid-Searched Pipeline

- Ketika menggunakan grid search, pipeline hasil grid search tetap memungkinkan akses ke langkah individu.
- Contoh:

```
grid.best_estimator_.named_steps['svc']
```

- Anda dapat melihat parameter model yang dipilih atau nilai transformasi dari langkah preprocessing.

---

8. Grid-Searching Preprocessing Steps and Model Parameters

- Grid search dapat mencakup pencarian parameter preprocessing dan parameter model secara bersamaan.
- Contoh:
  - Parameter preprocessing (misalnya, jumlah fitur yang dipilih).
  - Parameter model (misalnya, nilai C untuk SVM).
- Contoh grid search:

```
param_grid = {
    'scaler': [StandardScaler(), MinMaxScaler()],
    'svc__C': [0.1, 1, 10]
}
```

---

9. Grid-Searching Which Model To Use

- Grid search juga memungkinkan pemilihan model terbaik di antara beberapa model.
- Gunakan parameter pipeline untuk menentukan model yang digunakan.
- Contoh:

```
from sklearn.ensemble import RandomForestClassifier
param_grid = {
    'model': [SVC(), RandomForestClassifier()],
    'model__C': [0.1, 1, 10],  # untuk SVC
```

'model__n_estimators': [50, 100] # untuk RandomForest

}

---

10. Summary and Outlook

- **Pipelines** membantu menyatukan preprocessing, tuning, dan pelatihan model menjadi satu alur kerja.
- **Keuntungan**:
    - Meminimalkan risiko kesalahan (seperti data leakage).
    - Meningkatkan efisiensi pengembangan model.
    - Mempermudah integrasi preprocessing dan pelatihan.