Laporan Jobsheet 11

Teknologi Data (Linear Regression in Python)

Disusun oleh :

Nama :Dhuta Pamungkas Ibnusiqin

Nim : 1941723014

Class:TI-3D

Jurusan Teknologi Informasi

Politeknik Negeri Malang
2021

Linear Regression in Python

- Regresi

  Analisis regresi adalah salah satu bidang terpenting dalam statistik dan pembelajaran mesin. Ada banyak metode regresi yang tersedia. Regresi linier adalah salah satunya.

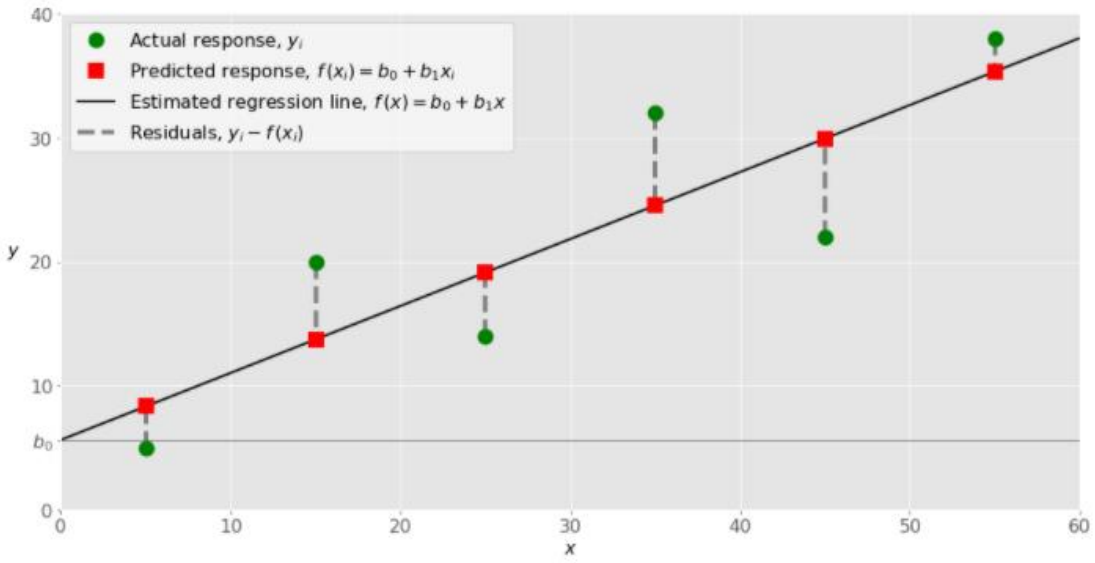
- Regresi linier

  Regresi linier mungkin salah satu teknik regresi yang paling penting dan banyak digunakan. Ini adalah salah satu metode regresi yang paling sederhana. Salah satu keuntungan utamanya adalah kemudahan menafsirkan hasil.

1. **Regresi Linier Sederhana**

   Regresi linier sederhana atau satu-variate adalah kasus regresi linier yang paling sederhana dengan variabel independen tunggal, **x** = x.

   Gambar berikut menggambarkan regresi linier sederhana:



   Saat menerapkan regresi linier sederhana, Anda biasanya mulai dengan kumpulan pasangan input-output (x-y) tertentu (lingkaran hijau). Pasangan ini adalah pengamatan Anda. Misalnya, pengamatan paling kiri (lingkaran hijau) memiliki masukan x = 5 dan keluaran aktual (tanggapan) y = 5. Yang berikutnya memiliki x = 15 dan y = 20, dan seterusnya.

   Fungsi estimasi regresi (garis hitam) memiliki persamaan $f(x) = b_0 + b_1 x$. Tujuan Anda adalah menghitung nilai optimal dari bobot prediksi b_0 dan b_1 yang meminimalkan SSR dan menentukan fungsi regresi yang diperkirakan. Nilai b_0, juga disebut intersep, menunjukkan titik di mana garis estimasi regresi melintasi sumbu y. Ini adalah nilai perkiraan respon $f(x)$ untuk x = 0. Nilai b_1 menentukan kemiringan garis regresi yang diperkirakan.

Respons yang diprediksi (kotak merah) adalah titik-titik pada garis regresi yang sesuai dengan nilai input. Misalnya, untuk input $x = 5$, respons yang diprediksi adalah $f(5) = 8{,}33$ (diwakili dengan kotak merah paling kiri).

Sisa (garis abu-abu putus-putus vertikal) dapat dihitung sebagai $y_i - f(\mathbf{x}_i) = y_i - b_0 - b_1 x_i$ untuk $i = 1,\ldots, n$. Itu adalah jarak antara lingkaran hijau dan kotak merah. Saat Anda menerapkan regresi linier, Anda sebenarnya mencoba meminimalkan jarak ini dan membuat kotak merah sedekat mungkin dengan lingkaran hijau yang telah ditentukan sebelumnya.

2. Underfitting dan Overfitting

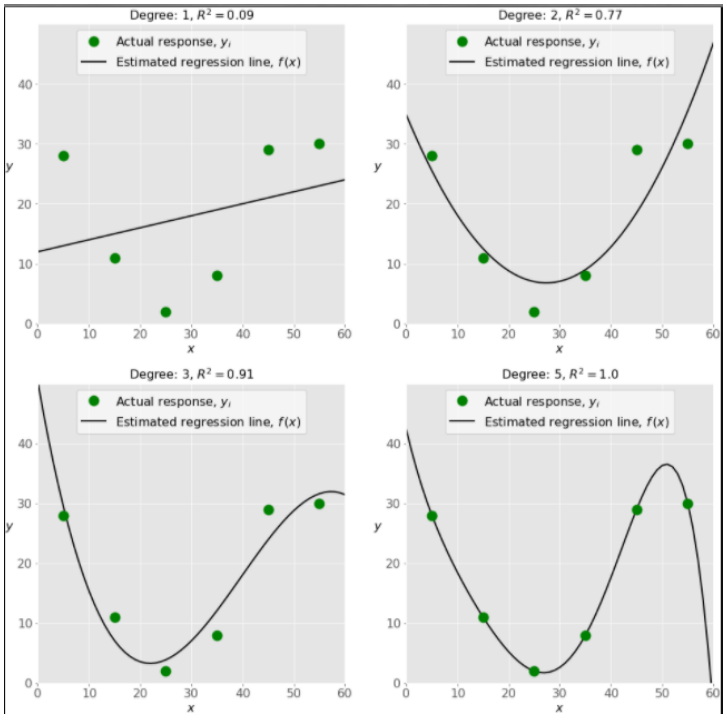
Satu pertanyaan sangat penting yang mungkin muncul saat Anda menerapkan regresi polinomial terkait dengan pilihan derajat optimal dari fungsi regresi polinomial.

Tidak ada aturan langsung untuk melakukan ini. Tergantung kasusnya. Namun, Anda harus menyadari dua masalah yang mungkin mengikuti pilihan derajat: underfitting dan overfitting.

Underfitting terjadi jika model tidak dapat menangkap dependensi di antara data secara akurat, biasanya sebagai konsekuensi dari kesederhanaannya sendiri. Seringkali menghasilkan R^2 rendah dengan data yang diketahui dan kemampuan generalisasi yang buruk saat diterapkan dengan data baru.

Overfitting terjadi saat model mempelajari dependensi di antara data dan fluktuasi acak. Dengan kata lain, model mempelajari data yang ada dengan terlalu baik. Model kompleks, yang memiliki banyak fitur atau istilah, sering kali cenderung overfitting. Ketika diterapkan pada data yang diketahui, model seperti itu biasanya menghasilkan R^2 yang tinggi. Namun, mereka sering tidak menggeneralisasi dengan baik dan memiliki lebih rendah secara signifikan R^2 saat digunakan dengan data baru.

Gambar berikut mengilustrasikan model underfitted, well-fit, dan overfitted:



Plot kiri atas menunjukkan garis regresi linier yang memiliki R^2 rendah. Mungkin juga penting bahwa garis lurus tidak dapat memperhitungkan fakta bahwa respons aktual meningkat saat x menjauh dari 25 menuju nol. Ini mungkin contoh underfitting.

Plot kanan atas menggambarkan regresi polinomial dengan derajat sama dengan 2. Dalam contoh ini, ini mungkin derajat optimal untuk memodelkan data ini. Model tersebut memiliki nilai R^2 yang memuaskan dalam banyak kasus dan menunjukkan tren dengan baik.

Plot kiri bawah menampilkan regresi polinomial dengan derajat sama dengan 3. Nilai R^2 lebih tinggi dari pada kasus sebelumnya. Model ini berperilaku lebih baik dengan data yang diketahui daripada yang sebelumnya. Namun, ini menunjukkan beberapa tanda overfitting, terutama untuk nilai input yang mendekati 60 di mana garis mulai menurun, meskipun data sebenarnya tidak menunjukkannya.

Akhirnya, pada plot kanan bawah, Anda dapat melihat kesesuaian yang sempurna: enam titik dan garis polinomial dengan derajat 5 (atau lebih tinggi) menghasilkan $R^2 = 1$. Setiap respons aktual sama dengan prediksi yang sesuai.

Dalam beberapa situasi, ini mungkin yang Anda cari. Namun, dalam banyak kasus, ini adalah model yang terlalu pas. Ini cenderung memiliki perilaku yang buruk dengan data yang tidak terlihat, terutama dengan input yang lebih besar dari 50.

Sebagai contoh, diasumsikan, tanpa bukti apapun, bahwa terdapat penurunan yang signifikan dalam respon untuk $x > 50$ dan y mencapai nol untuk x mendekati 60. Perilaku tersebut merupakan konsekuensi dari usaha yang berlebihan untuk mempelajari dan menyesuaikan dengan data yang ada.

Menerapkan Regresi Linier dengan Python

A. Regresi Linier Sederhana Dengan scikit-learn

Langkah 1: Impor paket dan kelas

Langkah pertama adalah mengimpor paket numpy dan kelas LinearRegression dari sklearn.linear_model:

```
import numpy as np
from sklearn.linear_model import LinearRegression
```

Tipe data fundamental NumPy adalah tipe array yang disebut numpy.ndarray. Sisa artikel ini menggunakan istilah larik untuk merujuk ke contoh tipe numpy.ndarray.
Kelas sklearn.linear_model.LinearRegression akan digunakan untuk melakukan regresi linier dan polinomial dan membuat prediksi yang sesuai.

Langkah 2: Sediakan data

Langkah kedua adalah menentukan data untuk dikerjakan. Input (regressor, x) dan output (prediktor, y) harus berupa array (instance kelas numpy.ndarray) atau objek serupa. Ini adalah cara paling sederhana dalam menyediakan data untuk regresi:

```
x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
y = np.array([5, 20, 14, 32, 22, 38])
```

Sekarang, Anda memiliki dua array: input x dan output y. Anda harus memanggil .reshape () pada x karena array ini harus dua dimensi, atau lebih tepatnya, memiliki satu kolom dan baris sebanyak yang diperlukan. Persis seperti itulah yang ditentukan oleh argumen (-1, 1) .reshape ().

Langkah 3: Buat model dan sesuaikan

Langkah selanjutnya adalah membuat model regresi linier dan menyesuaikannya dengan data yang ada.

Mari buat instance kelas LinearRegression, yang akan mewakili model regresi:

Contoh ini menggunakan nilai default dari semua parameter.

Saatnya mulai menggunakan model. Pertama, Anda perlu memanggil .fit () pada model:

```
model.fit(x, y)
```

Dengan .fit (), Anda menghitung nilai optimal dari bobot b_0 dan b_1, menggunakan masukan dan keluaran yang ada (x dan y) sebagai argumennya. Dengan kata lain, .fit () cocok dengan modelnya. Ia mengembalikan diri, yang merupakan model variabel itu sendiri. Itulah mengapa Anda dapat mengganti dua pernyataan terakhir dengan yang ini:

```
model = LinearRegression().fit(x, y)
```

Langkah 4: Dapatkan hasil

Setelah model Anda dipasang, Anda bisa mendapatkan hasil untuk memeriksa apakah model bekerja dengan memuaskan dan menafsirkannya.

Anda bisa mendapatkan koefisien determinasi (R^2) dengan .score () disebut model:

```
r_sq = model.score(x, y)
print('coefficient of determination:', r_sq)
```

Saat Anda menerapkan .score (), argumennya juga merupakan prediktor x dan regressor y, dan nilai kembaliannya adalah R^2.

Atribut model adalah .intercept_, yang merepresentasikan koefisien, b_0 dan .coef_, yang merepresentasikan b_1:

```
print('intercept:', model.intercept_)


print('slope:', model.coef_)
```

Anda harus memperhatikan bahwa Anda juga dapat memberikan y sebagai larik dua dimensi. Dalam kasus ini, Anda akan mendapatkan hasil yang serupa. Seperti inilah tampilannya:

```
new_model = LinearRegression().fit(x, y.reshape((-1, 1)))
print('intercept:', new_model.intercept_)


print('slope:', new_model.coef_)
```

Seperti yang Anda lihat, contoh ini sangat mirip dengan yang sebelumnya, tetapi dalam kasus ini, .intercept_ adalah larik satu dimensi dengan satu elemen b_0, dan .coef_ adalah larik dua dimensi dengan satu elemen b_1.

Langkah 5: Prediksi respons

Setelah ada model yang memuaskan, Anda dapat menggunakannya untuk prediksi dengan data yang sudah ada atau yang baru.

Untuk mendapatkan respons yang diprediksi, gunakan .predict ():

```
y_pred = model.predict(x)
print('predicted response:', y_pred, sep='\n')
```

Saat menerapkan .predict (), Anda meneruskan regressor sebagai argumen dan mendapatkan prediksi respons yang sesuai.

Ini adalah cara yang hampir identik untuk memprediksi respons:

Saat menerapkan .predict (), Anda meneruskan regressor sebagai argumen dan mendapatkan prediksi respons yang sesuai.

Ini adalah cara yang hampir identik untuk memprediksi respons:

```
>>> y_pred = model.intercept_ + model.coef_ * x
>>> print('predicted response:', y_pred, sep='\n')
predicted response:
[[ 8.33333333]
 [13.73333333]
 [19.13333333]
 [24.53333333]
 [29.93333333]
 [35.33333333]]
```

B. Advanced Linear Regression Dengan statsmodels

Langkah 1: Impor paket

Pertama, Anda perlu melakukan beberapa impor. Selain numpy, Anda perlu mengimpor statsmodels.api:

```
import numpy as np
import statsmodels.api as sm
```

Langkah 2: Sediakan data dan ubah input

Anda dapat memberikan input dan output dengan cara yang sama seperti yang Anda lakukan saat menggunakan scikit-learn:

```
x = [[0, 1], [5, 1], [15, 2], [25, 5], [35, 11], [45, 15], [55, 34], [60, 35]]
y = [4, 5, 20, 14, 32, 22, 38, 43]
x, y = np.array(x), np.array(y)
```

Langkah 3: Buat model dan sesuaikan
Model regresi berdasarkan kuadrat terkecil biasa adalah turunan dari statsmodels.regress.linear_model.OLS kelas. Beginilah cara Anda mendapatkannya:

```
model = sm.OLS(y, x)
```

Langkah 4: Dapatkan hasil
Hasil variabel mengacu pada objek yang memuat informasi rinci tentang hasil regresi linier. Menjelaskannya jauh di luar cakupan artikel ini, tetapi Anda akan mempelajari cara mengekstraknya di sini.
Anda bisa memanggil .summary () untuk mendapatkan tabel dengan hasil regresi linier:

```
>>> print(results.summary())
OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.862
Model:                            OLS   Adj. R-squared:                  0.806
Method:                 Least Squares   F-statistic:                     15.56
Date:                Sun, 17 Feb 2019   Prob (F-statistic):            0.00713
Time:                        19:15:07   Log-Likelihood:                -24.316
No. Observations:                   8   AIC:                             54.63
Df Residuals:                       5   BIC:                             54.87
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          5.5226      4.431      1.246      0.268      -5.867      16.912
x1             0.4471      0.285      1.567      0.178      -0.286       1.180
x2             0.2550      0.453      0.563      0.598      -0.910       1.420
==============================================================================
Omnibus:                        0.561   Durbin-Watson:                   3.268
Prob(Omnibus):                  0.755   Jarque-Bera (JB):                0.534
Skew:                           0.380   Prob(JB):                        0.766
Kurtosis:                       1.987   Cond. No.                         80.1
==============================================================================


Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
```

C. Regresi Komponen Utama vs Regresi Kuadrat Terkecil Sebagian

Contoh ini membandingkan Regresi Komponen Utama (PCR) dan Regresi Kuadrat Terkecil Sebagian (PLS) pada set data mainan. Tujuan kami adalah untuk menggambarkan bagaimana PLS dapat mengungguli PCR ketika target berkorelasi kuat dengan beberapa arah dalam data yang memiliki varian rendah.

PCR adalah regressor yang terdiri dari dua langkah: pertama, PCA diterapkan pada data pelatihan, kemungkinan melakukan reduksi dimensionalitas; kemudian, regressor (misalnya regressor linier) dilatih pada sampel yang diubah. Dalam PCA, transformasi murni tanpa pengawasan, artinya tidak ada informasi tentang target yang digunakan. Akibatnya, PCR mungkin berkinerja buruk di beberapa set data di mana target berkorelasi kuat dengan arah yang memiliki varian rendah. Memang, pengurangan dimensi PCA memproyeksikan data ke ruang dimensi yang lebih rendah di mana varians dari data yang diproyeksikan dimaksimalkan secara rakus di sepanjang setiap sumbu. Meskipun mereka memiliki kekuatan prediktif paling besar pada target, arah dengan varian yang lebih rendah akan dihilangkan, dan regressor akhir tidak akan dapat memanfaatkannya.

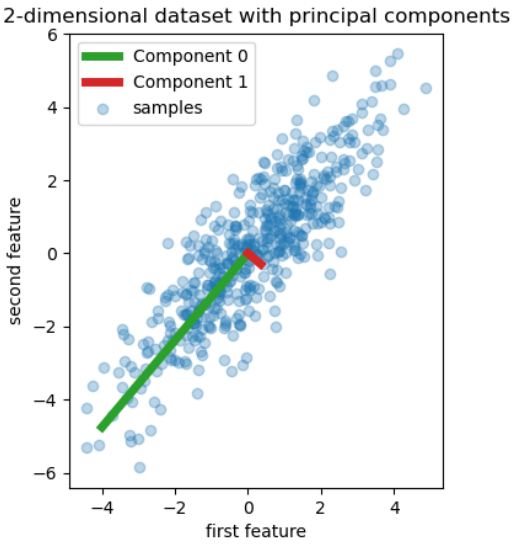
PLS adalah transformator dan regressor, dan sangat mirip dengan PCR: PLS juga menerapkan pengurangan dimensi pada sampel sebelum menerapkan regressor linier ke data yang diubah. Perbedaan utama dengan PCR adalah transformasi PLS diawasi. Oleh karena itu, seperti yang akan kita lihat dalam contoh ini, ia tidak mengalami masalah yang baru saja kita sebutkan.

Kami mulai dengan membuat kumpulan data sederhana dengan dua fitur. Bahkan sebelum kami menyelami PCR dan PLS, kami memasang penduga PCA untuk menampilkan dua komponen utama dari kumpulan data ini, yaitu dua arah yang menjelaskan varian paling besar dalam data.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

rng = np.random.RandomState(0)
n_samples = 500
cov = [[3, 3],
       [3, 4]]
X = rng.multivariate_normal(mean=[0, 0], cov=cov, size=n_samples)
pca = PCA(n_components=2).fit(X)


plt.scatter(X[:, 0], X[:, 1], alpha=.3, label='samples')
for i, (comp, var) in enumerate(zip(pca.components_,
pca.explained_variance_)):
    comp = comp * var  # scale component by its variance explanation power
    plt.plot([0, comp[0]], [0, comp[1]], label=f"Component {i}",
linewidth=5,
             color=f"C{i + 2}")
plt.gca().set(aspect='equal',
              title="2-dimensional dataset with principal components",
              xlabel='first feature', ylabel='second feature')
plt.legend()
plt.show()
```

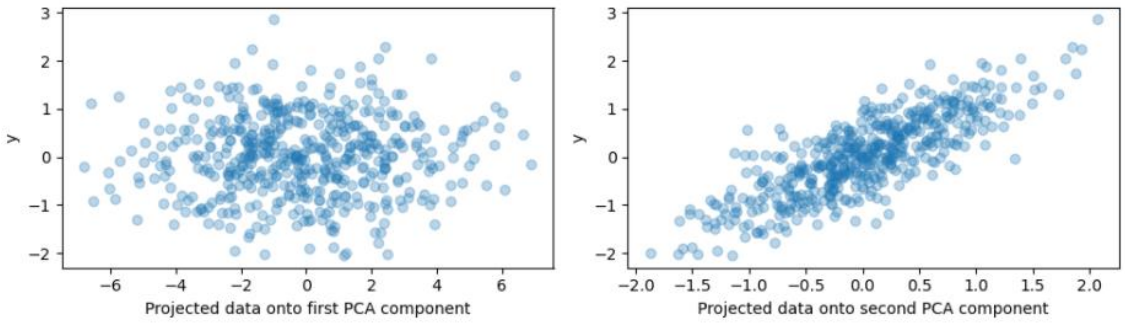


Untuk tujuan contoh ini, sekarang kita mendefinisikan target y sedemikian rupa sehingga sangat berkorelasi dengan arah yang memiliki varians kecil. Untuk tujuan ini, kami akan memproyeksikan X ke komponen kedua, dan menambahkan beberapa noise padanya.

```
y = X.dot(pca.components_[1]) + rng.normal(size=n_samples) / 2

fig, axes = plt.subplots(1, 2, figsize=(10, 3))

axes[0].scatter(X.dot(pca.components_[0]), y, alpha=.3)
axes[0].set(xlabel='Projected data onto first PCA component', ylabel='y')
axes[1].scatter(X.dot(pca.components_[1]), y, alpha=.3)
axes[1].set(xlabel='Projected data onto second PCA component', ylabel='y')
plt.tight_layout()
plt.show()
```



D. Proyeksi pada satu komponen dan daya prediksi

Kami sekarang membuat dua regressor: PCR dan PLS, dan untuk tujuan ilustrasi kami, kami menetapkan jumlah komponen ke 1. Sebelum memasukkan data ke langkah PCA PCR, pertama-tama kami membakukannya, seperti yang direkomendasikan oleh praktik yang baik. Estimator PLS memiliki kemampuan penskalaan bawaan.

Untuk kedua model, kami memplot data yang diproyeksikan ke komponen pertama terhadap target. Dalam kedua kasus tersebut, data yang diproyeksikan inilah yang akan digunakan para regressor sebagai data pelatihan.

```
from sklearn.model_selection import train_test_split
```

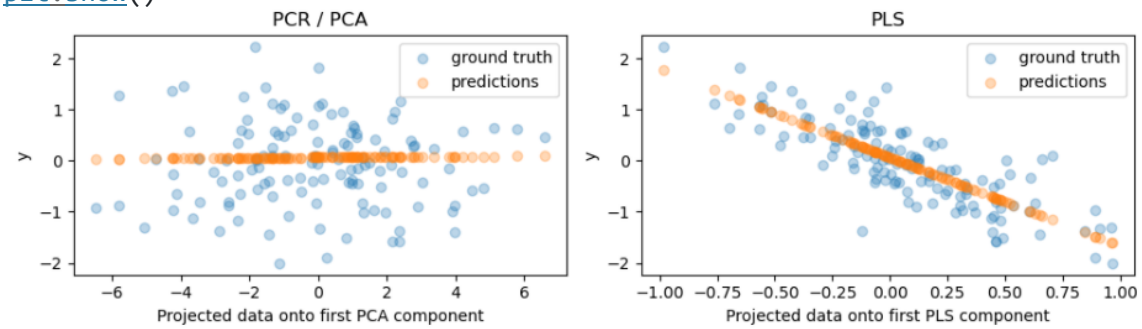
```
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import PLSRegression

X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=rng)

pcr = make_pipeline(StandardScaler(), PCA(n_components=1),
LinearRegression())
pcr.fit(X_train, y_train)
pca = pcr.named_steps['pca']  # retrieve the PCA step of the pipeline

pls = PLSRegression(n_components=1)
pls.fit(X_train, y_train)

fig, axes = plt.subplots(1, 2, figsize=(10, 3))
axes[0].scatter(pca.transform(X_test), y_test, alpha=.3, label='ground
truth')
axes[0].scatter(pca.transform(X_test), pcr.predict(X_test), alpha=.3,
                label='predictions')
axes[0].set(xlabel='Projected data onto first PCA component',
            ylabel='y', title='PCR / PCA')
axes[0].legend()
axes[1].scatter(pls.transform(X_test), y_test, alpha=.3, label='ground
truth')
axes[1].scatter(pls.transform(X_test), pls.predict(X_test), alpha=.3,
                label='predictions')
axes[1].set(xlabel='Projected data onto first PLS component',
            ylabel='y', title='PLS')
axes[1].legend()
plt.tight_layout()
plt.show()
```



Seperti yang diharapkan, transformasi PCA tanpa pengawasan dari PCR telah menghilangkan komponen kedua, yaitu arah dengan varian terendah, meskipun itu merupakan arah yang paling prediktif. Ini karena PCA adalah transformasi yang sama sekali tidak diawasi, dan menghasilkan data yang diproyeksikan memiliki daya prediksi yang rendah pada target.

Di sisi lain, regressor PLS berhasil menangkap efek arah dengan varian terendah, berkat penggunaan informasi target selama transformasi: ia dapat mengenali bahwa arah ini sebenarnya

paling prediktif. Kami mencatat bahwa komponen PLS pertama berkorelasi negatif dengan target, yang berasal dari fakta bahwa tanda vektor eigen berubah-ubah.

Kesimpulan

Sekarang tahu apa itu regresi linier dan bagaimana Anda dapat mengimplementasikannya dengan Python dan tiga paket open-source: NumPy, scikit-learn, dan statsmodels.

Anda dapat menggunakan NumPy untuk menangani array.

Regresi linier diimplementasikan dengan berikut ini:

- scikit-learn jika Anda tidak memerlukan hasil yang mendetail dan ingin menggunakan pendekatan yang konsisten dengan teknik regresi lainnya
- statsmodels jika Anda membutuhkan parameter statistik lanjutan dari suatu model

Kedua pendekatan tersebut layak dipelajari bagaimana menggunakan dan mengeksplorasi lebih jauh. Tautan dalam artikel ini bisa sangat berguna untuk itu.