

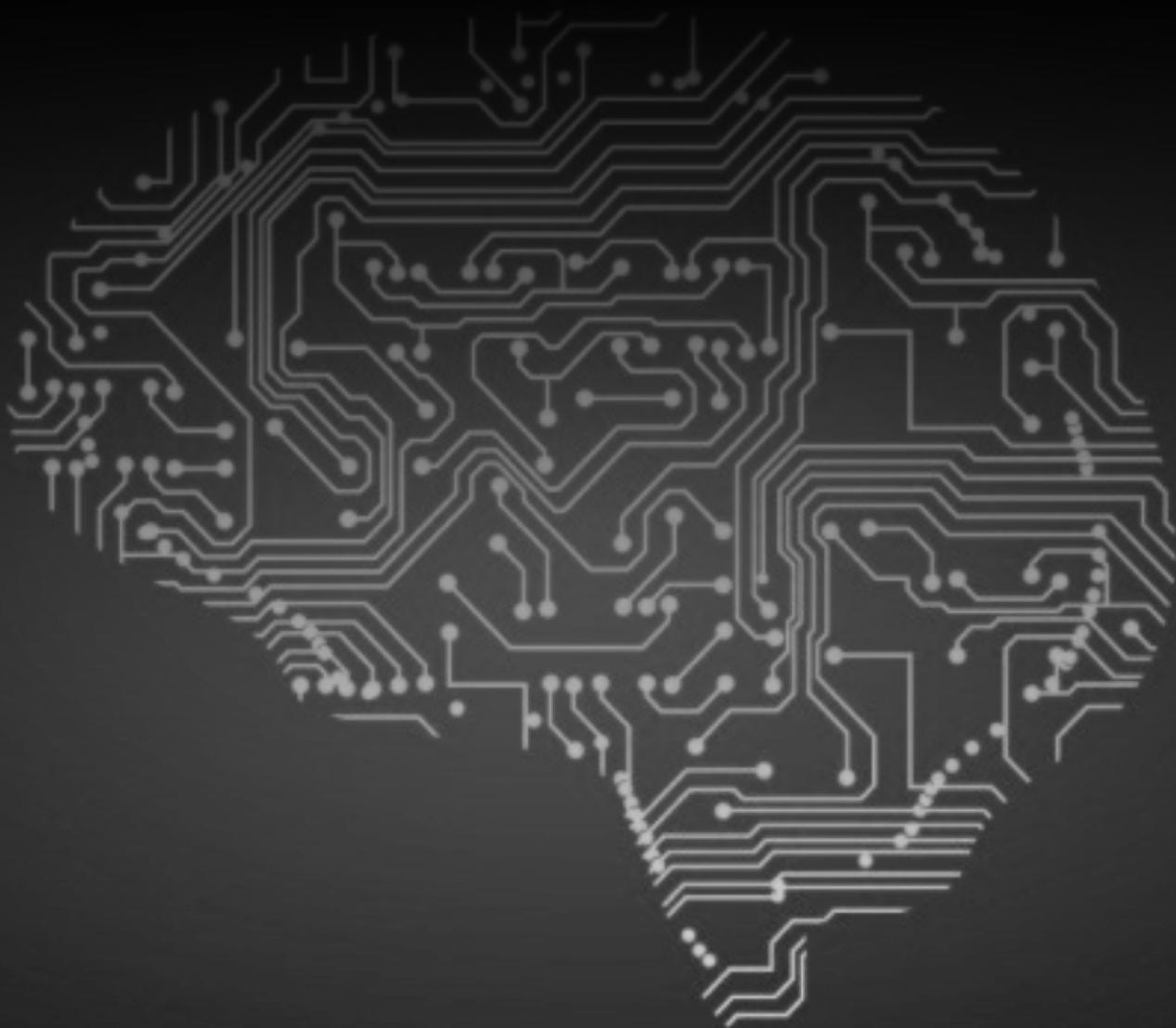
Dr. Joseph Teguh Santoso, S.Kom, M.Kom

Algoritma Machine Learning Dengan Python



Algoritma Machine Learning Dengan Python

Dr. Joseph Teguh Santoso, S.Kom, M.Kom



YAYASAN PRIMA AGUS TEKNIK

PENERBIT :

YAYASAN PRIMA AGUS TEKNIK

Jl. Majapahit No. 605 Semarang

Telp. (024) 6723456. Fax. 024-6710144

Email : penerbit_ypat@stekom.ac.id

Algoritma Machine Learning Dengan Python

Penulis :

Dr. Joseph Teguh Santoso, S.Kom., M.Kom

ISBN : 9 786235 734286

Editor :

Muhammad Sholikan, M.Kom

Penyunting :

Dr. Mars Caroline Wibowo. S.T., M.Mm.Tech

Desain Sampul dan Tata Letak :

Irdha Yunianto, S.Ds., M.Kom

Penerbit :

Yayasan Prima Agus Teknik Bekerja sama dengan
Universitas Sains & Teknologi Komputer (Universitas STEKOM)

Redaksi :

Jl. Majapahit no 605 Semarang

Telp. (024) 6723456

Fax. 024-6710144

Email : penerbit_ypat@stekom.ac.id

Distributor Tunggal :

Universitas STEKOM

Jl. Majapahit no 605 Semarang

Telp. (024) 6723456

Fax. 024-6710144

Email : info@stekom.ac.id

Hak cipta dilindungi undang-undang

Diarang memperbanyak karya tulis ini dalam bentuk dan dengan cara apapun tanpa ijin tertulis dari penerbit

KATA PENGANTAR

Puji syukur pada Tuhan Yang Maha Esa bahwa buku yang berjudul "*Algoritma Machine Learning Dengan Python*" ini dapat diselesaikan dengan baik. Pembelajaran Mesin merupakan faktor utama perubahan dunia, beranekaragam jenis aplikasi dan penelitian dilakukan dibidang akademi, industry maupun hiburan. Pembelajaran merubah paradigma setiap bagian dari kehidupan kita sehari-hari, dari cara pengambilan keputusan hingga prediksi dalam hal tertentu.

Buku ini dibagi menjadi 4 Bab, bab pertama menulis memaparkan bahwa penulis telah mempelajari banyak konsep yang berguna, beberapa konsep yang mungkin membuat Anda sedikit kebingungan. Pembelajaran mesin: ML mengacu pada membuat mesin bekerja lebih baik pada beberapa tugas, menggunakan data yang diberikan. Diantaranya pembelajaran mesin datang dalam berbagai jenis, seperti pembelajaran terawasi, batch, tanpa pengawasan, dan online. Untuk menjalankan proyek ML, mengumpulkan data dalam set pelatihan, lalu mengumpulkan set tersebut ke algoritma pembelajaran untuk mendapatkan keluaran, "prediksi". Jika ingin mendapatkan output yang tepat, sistem harus menggunakan data yang jelas, yang tidak terlalu kecil dan yang tidak memiliki fitur yang tidak relevan. Di bab kedua, mempelajari konsep baru yang berguna dan menerapkan banyak jenis algoritma klasifikasi. Juga konsep baru, seperti: ROC (karakteristik pengoperasian receiver, alat yang digunakan dengan pengklasifikasi biner); Analisis Kesalahan, Cara melatih pengklasifikasi acak menggunakan fungsi Scikit, Memahami Klasifikasi Multi-Output dan multi-Label.

Bab ketiga, mempelajari konsep baru, dan mempelajari cara melatih model menggunakan berbagai jenis algoritma, mempelajari kapan harus menggunakan setiap algoritma, termasuk: Penurunan gradien batch, Penurunan gradien mini-batch, Regresi polynomial, Model linier teratur, Regresi punggungan, Regresi Lasso.

Bab Terakhir akan membahas tentang algoritma pembelajaran mesin diantaranya regresi linier, kompleksitas komputasi, dan penurunan gradien. Akhir kata semoga buku ini bermanfaat bagi para pembaca.

Semarang, Januari 2022
Penulis

Dr. Joseph Teguh Santoso, M.Kom.

DAFTAR ISI

HALAMAN JUDUL	i
KATA PENGANTAR	iii
DAFTAR ISI	iv
BAB 1 PENGANTAR BELAJAR MESIN	1
1.1 Teori	1
1.2 Apa itu pembelajaran mesin?	1
1.3 Mengapa pembelajaran mesin?	1
1.4 Kapan sebaiknya Anda menggunakan pembelajaran mesin?	3
1.5 Jenis Sistem Pembelajaran Mesin	3
1.6 Pembelajaran yang diawasi dan tidak diawasi	3
1.7 Belajar Batch	6
1.8 Pembelajaran online	6
1.9 Pembelajaran berbasis instan	7
1.10 Pengujian	8
1.11 Melebihi Data	9
1.12 Mengelolakan Data	9
BAB 2 KLASIFIKASI	11
2.1 Instalasi	11
2.2 MNIST	11
2.3 Ukuran Kinerja	13
2.4 Matriks Konfusi	15
2.5 Mengingat	16
2.6 Ingat Tradeoff	16
2.7 ROC	18
2.8 Klasifikasi Multi-kelas	19
2.9 Melatih Pengklasifikasi Hutan Acak	20
2.10 Analisis Kesalahan	20
2.11 Klasifikasi Multi-label	22
2.12 Klasifikasi Multi-output	22
BAB 3 CARA MELATIH MODEL	24
3.1 Regresi linier	24
3.2 Kompleksitas Komputasi	26
3.3 Penurunan Gradien	26
3.4 Penurunan Gradien Batch	27
3.5 Penurunan Gradien Stokastik	28
3.6 Penurunan Gradien Batch Mini	30

3.7	Regresi Polinomial	30
3.8	Kurva Pembelajaran	31
BAB 4 KOMBINASI MODEL YANG BERBEDA	34	
4.1	Klasifikasi Pohon	34
4.2	Fungsi Massa Dari Distribusi Binominal	36
4.3	Menerapkan Pengklasifikasi Mayoritas Sederhana	37
4.4	Penggabungan Algoritma Berbeda Untuk Klasifikasi Suara Mayoritas	41
DAFTAR PUSTAKA	49	

BAB 1

PENGANTAR BELAJAR MESIN

1.1 TEORI

Jika saya bertanya tentang "Pembelajaran mesin", Anda mungkin akan membayangkan robot atau sesuatu seperti Terminator. Pada kenyataannya t, pembelajaran mesin terlibat tidak hanya dalam robotika, tetapi juga dalam banyak aplikasi lainnya. Anda juga dapat membayangkan sesuatu seperti filter spam sebagai salah satu aplikasi pertama dalam pembelajaran mesin, yang membantu meningkatkan kehidupan jutaan orang. Dalam bab ini, saya akan memperkenalkan Anda apa itu pembelajaran mesin, dan cara kerjanya.

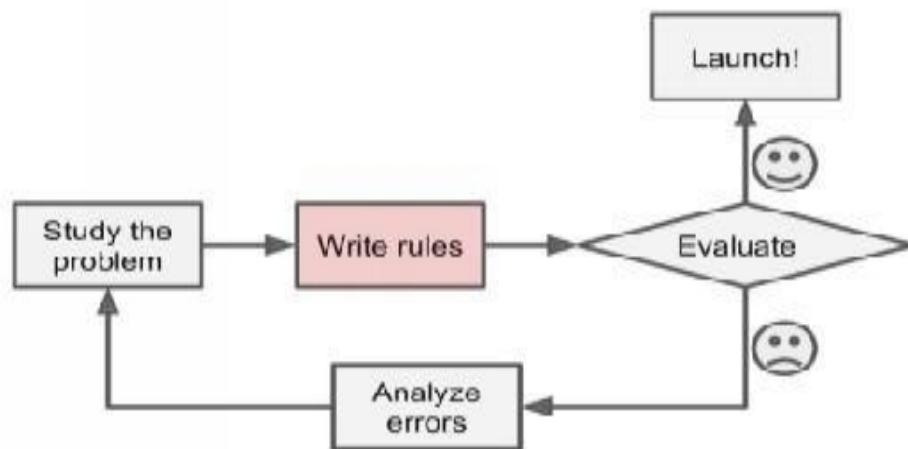
1.2 APA ITU PEMBELAJARAN MESIN?

Pembelajaran mesin adalah praktik pemrograman komputer untuk belajar dari data. Dalam contoh di atas, program akan dengan mudah dapat menentukan apakah yang diberikan penting atau "spam". Dalam machine learning, data disebut sebagai training set atau contoh.

1.3 MENGAPA PEMBELAJARAN MESIN?

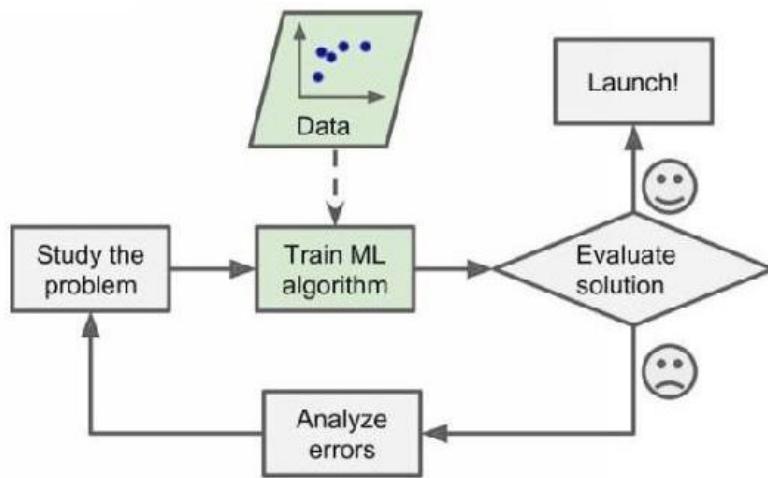
Mari kita asumsikan bahwa Anda ingin menulis program filter tanpa menggunakan metode pembelajaran mesin. Dalam hal ini, Anda harus melakukan langkah-langkah berikut:

- Pada awalnya, Anda akan melihat seperti apa email spam itu. Anda dapat memilihnya untuk kata atau frasa yang mereka gunakan, seperti "kartu debit", "gratis", dan seterusnya, dan juga dari pola yang digunakan dalam nama pengirim atau di badan email
- Kedua, Anda akan menulis algoritma untuk mendeteksi pola yang telah Anda lihat, dan kemudian perangkat lunak akan menandai email sebagai spam jika sejumlah pola tersebut terdeteksi.
- Terakhir, Anda akan menguji program, lalu ulangi dua langkah pertama lagi sampai hasilnya cukup baik.



Gambar 1.1 Pola Perencanaan Pembelajaran Mesin

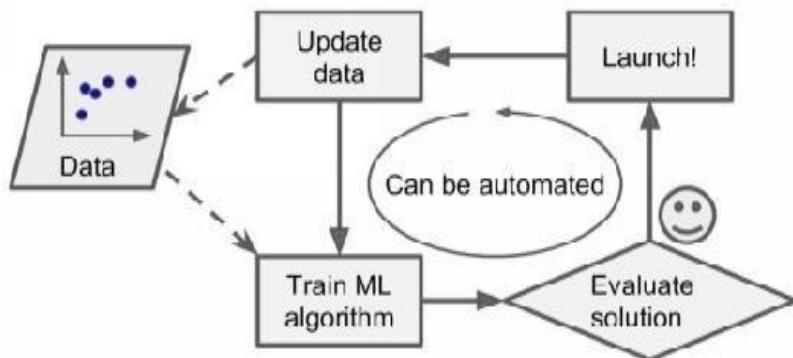
Karena program ini bukan perangkat lunak, program ini berisi daftar aturan yang sangat panjang yang sulit untuk dipelihara. Tetapi jika Anda mengembangkan perangkat lunak yang sama menggunakan ML, Anda akan dapat memeliharanya dengan baik.



Gambar 1.2 Flowchart Pembelajaran Mesin pengolahan Data

Selain itu, pengirim email dapat mengubah template email mereka sehingga kata seperti "4U" sekarang menjadi "untuk Anda", karena email mereka telah ditentukan sebagai spam. Program yang menggunakan teknik tradisional perlu diperbarui, yang berarti, jika ada perubahan lain, Anda perlu memperbarui kode Anda lagi dan lagi.

Di sisi lain, program yang menggunakan teknik ML akan secara otomatis mendeteksi perubahan ini oleh pengguna, dan mulai menandai mereka tanpa Anda memberi tahu secara manual.



Gambar 1.3 Alur Mesin Pembelajaran Otomatis

Selain itu, kita dapat menggunakan pembelajaran mesin untuk memecahkan masalah yang sangat kompleks untuk perangkat lunak pembelajaran non-mesin. Misalnya, pengenalan suara: ketika Anda mengatakan "satu" atau "dua", program harus dapat membedakan perbedaannya. Jadi, untuk tugas ini, Anda perlu mengembangkan algoritma yang mengukur suara.

Pada akhirnya, pembelajaran mesin akan membantu kita untuk belajar, dan algoritme pembelajaran mesin dapat membantu kita melihat apa yang telah kita pelajari.

1.4 KAPAN SEBAIKNYA ANDA MENGGUNAKAN PEMBELAJARAN MESIN?

- Ketika Anda memiliki masalah yang membutuhkan banyak daftar aturan yang panjang untuk menemukan solusinya. Dalam hal ini, teknik pembelajaran mesin dapat menyederhanakan kode Anda dan meningkatkan kinerja.
- Masalah yang sangat kompleks yang tidak ada solusi dengan pendekatan tradisional.
- Lingkungan yang tidak stabil: perangkat lunak pembelajaran mesin dapat beradaptasi dengan data baru.

1.5 JENIS SISTEM PEMBELAJARAN MESIN

Ada berbagai jenis sistem pembelajaran mesin. Kita dapat membaginya ke dalam kategori, tergantung pada apakah

- Mereka telah dilatih dengan manusia atau tidak
 - Diawasi
 - Tidak diawasi
 - Semi-diawasi
 - Pembelajaran Penguatan
- Jika mereka bisa belajar secara bertahap
- Jika mereka bekerja hanya dengan membandingkan titik data baru untuk menemukan titik data, atau dapat mendeteksi pola baru dalam data, dan kemudian akan membangun model.

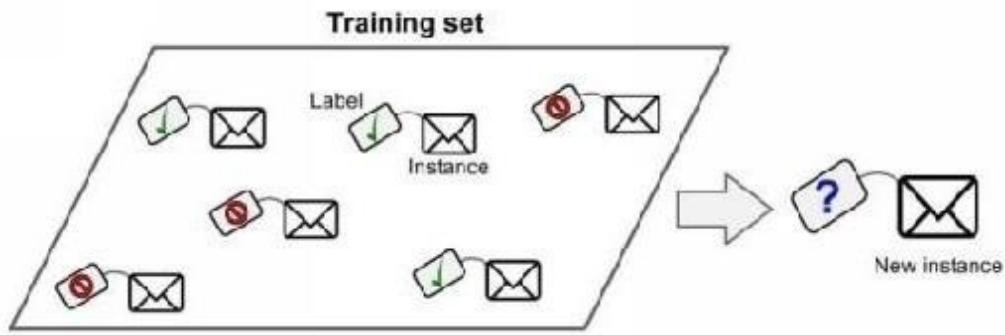
1.6 PEMBELAJARAN YANG DIAWASI DAN TIDAK DIAWASI

Kami dapat mengklasifikasikan sistem pembelajaran mesin sesuai dengan jenis dan jumlah pengawasan manusia selama pelatihan. Anda dapat menemukan empat kategori utama, seperti yang telah kami jelaskan sebelumnya.

- Pembelajaran yang diawasi
- Pembelajaran tanpa pengawasan
- Pembelajaran semi-diawasi
- Penguatan belajar

Pembelajaran Ya

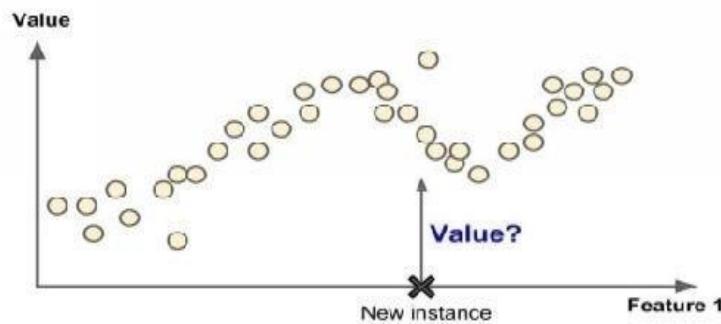
Dalam jenis sistem pembelajaran mesin ini, data yang Anda masukkan ke dalam algoritme, dengan solusi yang diinginkan, disebut sebagai "label."



Gambar 1.4 Data Training Set

- Mengawasi kelompok belajar bersama-sama tugas klasifikasi. Program di atas adalah contoh yang baik karena telah dilatih dengan banyak email pada saat yang sama dengan kelas mereka.

Contoh lain adalah untuk memprediksi nilai numerik seperti harga sebuah apartemen, diberikan serangkaian fitur (lokasi, jumlah kamar, fasilitas) yang disebut prediktor; jenis tugas ini disebut regresi.



Gambar 1.5 Kurva Regresi

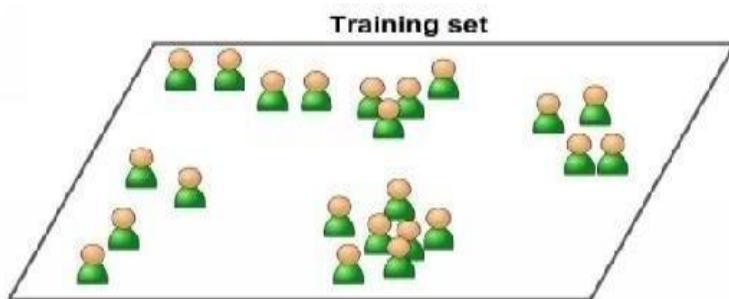
Anda harus ingat bahwa beberapa algoritma regresi dapat digunakan untuk klasifikasi juga, dan sebaliknya.

Algoritme terawasi yang paling penting

- K-tetangga dekat
- Regresi linier
- Jaringan saraf
- Mendukung mesin vektor
- Regresi logistik
- Pohon keputusan dan hutan acak

Pembelajaran tanpa pengawasan

Dalam sistem pembelajaran mesin jenis ini, Anda dapat menebak bahwa data tidak berlabel.

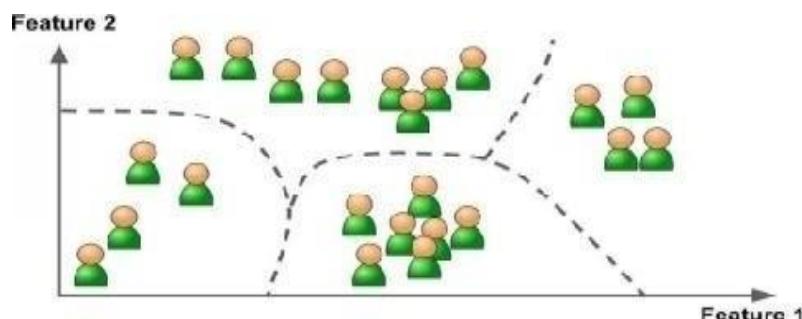


Gambar 1.6 Pembelajaran Tanpa Pengawasan

Algoritma tanpa pengawasan yang paling penting

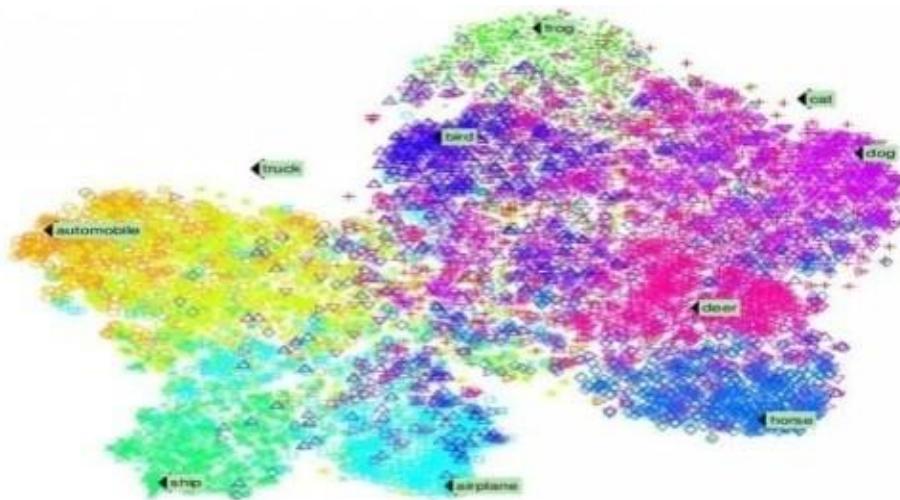
- Pengelompokan: k-means, analisis klaster hierarkis
- Pembelajaran aturan asosiasi: Eclat, apriori
- Visualisasi dan pengurangan dimensi: kernel PCA, t-distributed, PCA

Sebagai contoh, misalkan Anda memiliki banyak data tentang pengunjung. Menggunakan salah satu algoritma kami untuk mendeteksi grup dengan pengunjung serupa. Mungkin ditemukan bahwa 65% pengunjung Anda adalah pria yang suka menonton film di malam hari, sementara 30% menonton drama di malam hari; dalam hal ini, dengan menggunakan algoritma *clustering* akan membagi setiap grup menjadi sub-grup yang lebih kecil.



Gambar 1.7 Algoritma *clustering*

Ada beberapa algoritma yang sangat penting, seperti algoritma visualisasi; ini adalah algoritma pembelajaran tanpa pengawasan. Anda harus memberi mereka banyak data dan data tidak berlabel sebagai input, lalu Anda akan mendapatkan visualisasi 2D atau 3D sebagai output.



Gambar 1.8 Algoritma Visualisasi

Tujuannya di sini adalah untuk membuat output sesederhana mungkin tanpa kehilangan informasi apa pun. Untuk menangani masalah ini, itu akan menggabungkan beberapa fitur terkait menjadi satu fitur: misalnya, itu akan membuat mobil dengan modelnya. Ini disebut ekstraksi fitur.

Pembelajaran Penguatan

Pembelajaran penguatan adalah jenis lain dari sistem pembelajaran mesin. Agen "sistem AI" akan mengamati lingkungan, melakukan tindakan yang diberikan, dan kemudian menerima imbalan sebagai balasannya. Dengan tipe ini, agen harus belajar dengan sendirinya. Ikatan disebut kebijakan.

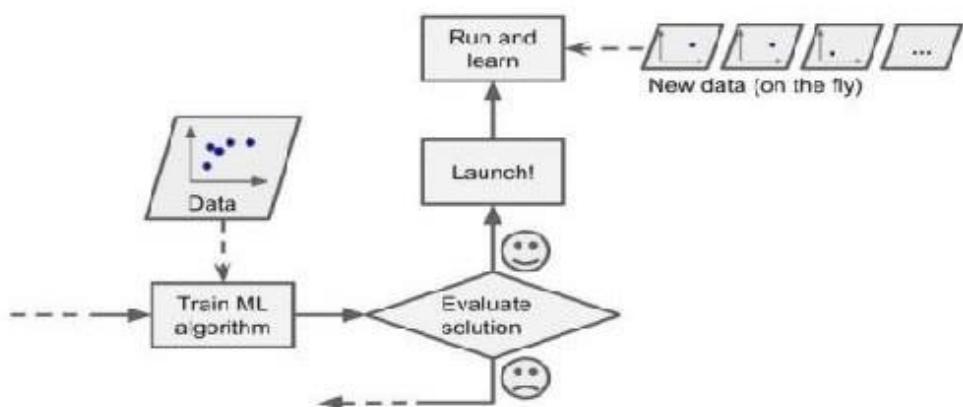
Anda dapat menemukan tipe pembelajaran ini di banyak aplikasi robotika yang mempelajari cara berjalan

1.7 BELAJAR BATCH

Dalam sistem pembelajaran mesin semacam ini, sistem tidak dapat belajar secara bertahap: sistem harus mendapatkan semua data yang diperlukan. Itu berarti akan membutuhkan banyak sumber daya dan banyak waktu, sehingga selalu dilakukan secara offline. Jadi, untuk bekerja dengan jenis pembelajaran ini, hal pertama yang harus dilakukan adalah melatih sistem, dan kemudian meluncurkannya tanpa pembelajaran apa pun.

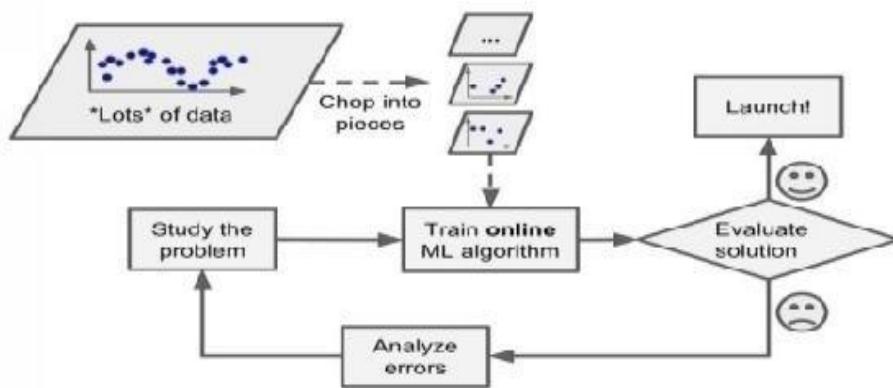
1.8 PEMBELAJARAN ONLINE

Pembelajaran seperti ini merupakan kebalikan dari pembelajaran batch. Maksud saya, di sini, sistem dapat belajar secara bertahap dengan menyediakan sistem dengan semua data yang tersedia sebagai instance (kelompok atau individu), dan kemudian sistem dapat belajar dengan cepat.



Gambar 1.9 Algoritma Pembelajaran Online

Anda dapat menggunakan sistem jenis ini untuk masalah yang memerlukan aliran data yang berkelanjutan, yang juga perlu beradaptasi dengan cepat terhadap perubahan apa pun. Selain itu, Anda dapat menggunakan sistem jenis ini untuk bekerja dengan kumpulan data yang sangat besar,

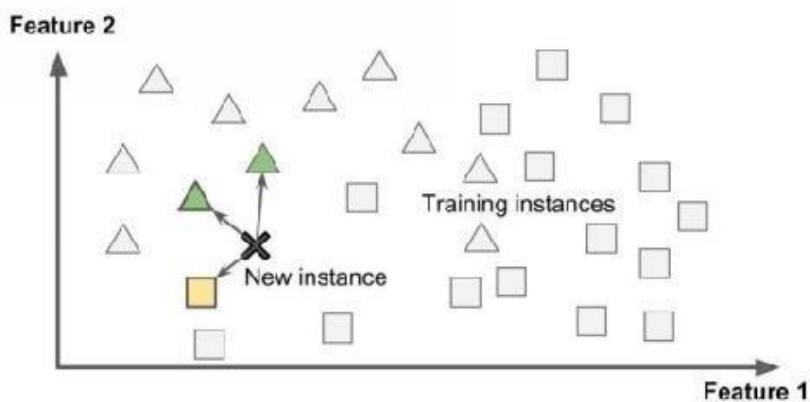


Gambar 1.10 Pengolahan Bigdata dalam Pembelajaran Online

Anda harus tahu seberapa cepat sistem Anda dapat beradaptasi dengan perubahan apa pun dalam "kecepatan pembelajaran" data. Jika kecepatannya tinggi, berarti sistem akan belajar cukup, cepat, tetapi juga akan cepat melupakan data lama.

1.9 PEMBELAJARAN BERBASIS INSTAN

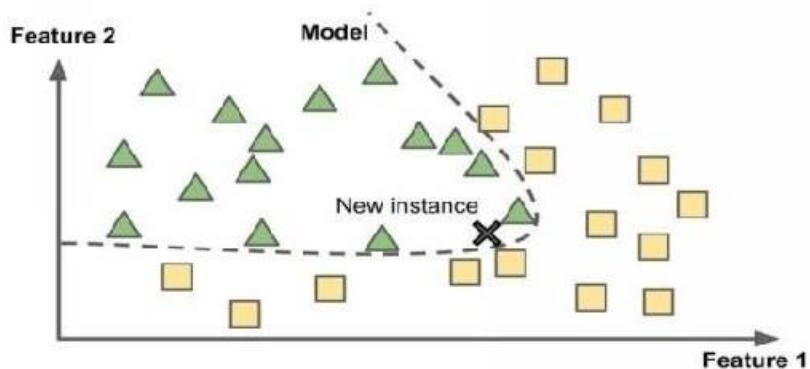
Ini adalah jenis pembelajaran paling sederhana yang harus Anda pelajari dengan hati. Dengan menggunakan jenis pembelajaran ini di program email kami, ini akan menandai semua email yang ditandai oleh pengguna.



Gambar 1.11 Ilustrasi Pembelajaran Berbasis Instan

Pembelajaran berbasis model

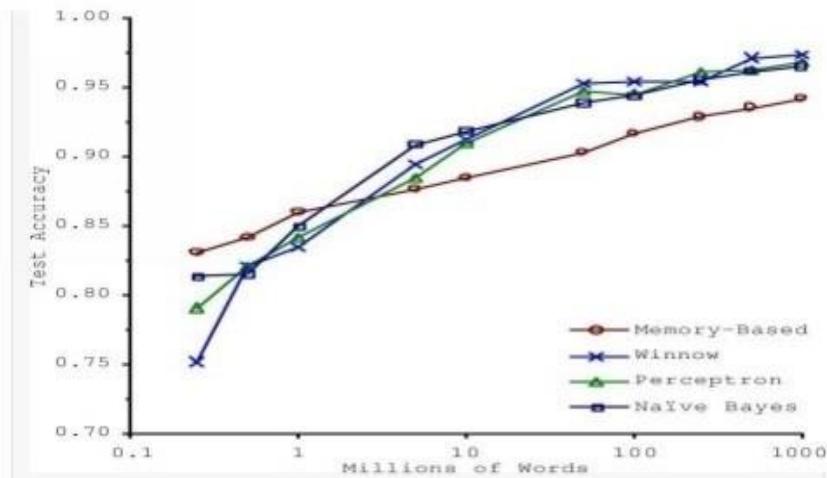
Ada jenis pembelajaran lain di mana belajar dari contoh memungkinkan konstruksi untuk membuat prediksi



Gambar 1.12 Pembelajaran Berbasis Model

Kuantitas Data Pelatihan yang Buruk dan Tidak Memadai

Sistem pembelajaran mesin tidak seperti anak-anak, yang dapat membedakan apel dan jeruk dalam berbagai warna dan bentuk, tetapi mereka membutuhkan banyak data untuk bekerja secara efektif, baik Anda bekerja dengan program dan masalah yang sangat sederhana, atau aplikasi kompleks seperti gambar pemrosesan dan pengenalan suara. Berikut adalah contoh efektivitas data yang tidak masuk akal, menunjukkan proyek MS, yang mencakup data sederhana dan masalah kompleks NLP.



Gambar 1.13 Grafik Kualitas data dengan beberapa metode

Data Berkualitas Buruk

Jika Anda bekerja dengan data pelatihan yang penuh dengan kesalahan dan outlier, ini akan membuat sistem sangat sulit untuk mendeteksi pola , sehingga tidak akan bekerja dengan benar. Jadi, jika Anda ingin program Anda bekerja dengan baik, Anda harus meluangkan lebih banyak waktu untuk membersihkan data pelatihan Anda.

Fitur yang Tidak Relevan

Sistem hanya akan dapat belajar jika data pelatihan berisi fitur yang cukup dan data yang tidak terlalu relevan. Bagian terpenting dari setiap proyek ML adalah mengembangkan fitur-fitur bagus "dari rekayasa fitur".

Rekayasa Fitur

Proses rekayasa fitur berjalan seperti ini:

- **Pemilihan fitur:** memilih fitur yang paling berguna.
- **Ekstraksi fitur:** menggabungkan fitur yang ada untuk menyediakan fitur yang lebih bermanfaat.
- **Pembuatan fitur baru:** pembuatan fitur baru, berdasarkan data.

1.10 PENGUJIAN

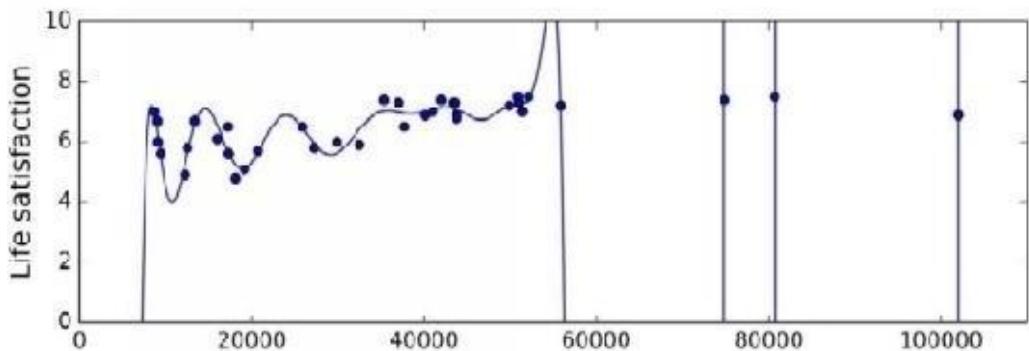
Jika Anda ingin memastikan bahwa model Anda bekerja dengan baik dan model tersebut dapat digeneralisasi dengan kasus baru, Anda dapat mencoba kasus baru dengannya dengan meletakkan model di lingkungan dan kemudian memantau bagaimana kinerjanya. Ini adalah metode yang baik, tetapi jika model Anda tidak memadai, pengguna akan mengeluh.

Anda harus membagi data Anda menjadi dua set, satu set untuk pelatihan dan yang kedua untuk pengujian, sehingga Anda dapat melatih model Anda menggunakan yang pertama dan mengujinya menggunakan yang kedua. Kesalahan generalisasi adalah tingkat kesalahan dengan mengevaluasi model Anda pada set pengujian. Nilai yang Anda dapatkan akan memberi tahu Anda apakah model Anda cukup baik, dan apakah itu akan berfungsi dengan baik.

Jika tingkat kesalahannya rendah, modelnya bagus dan akan bekerja dengan baik. Sebaliknya, jika tingkat Anda tinggi, ini berarti model Anda akan berkinerja buruk dan tidak berfungsi dengan baik. Saran saya untuk Anda adalah menggunakan 80% data untuk pelatihan dan 20% untuk tujuan pengujian, sehingga sangat mudah untuk menguji atau mengevaluasi model.

1.11 MELEBIHI DATA

Jika Anda berada di negara asing dan seseorang mencuri sesuatu milik Anda, Anda mungkin mengatakan bahwa setiap orang adalah pencuri. Ini adalah generalisasi yang berlebihan, dan, dalam pembelajaran mesin, disebut "*overfitting*". Ini berarti bahwa mesin melakukan hal yang sama: mereka dapat bekerja dengan baik saat bekerja dengan data pelatihan, tetapi tidak dapat menggeneralisasikannya dengan benar. Misalnya, pada gambar berikut, Anda akan menemukan model kepuasan hidup tingkat tinggi yang melampaui data, tetapi bekerja dengan baik dengan data pelatihan.



Gambar 1.14 Tingat Kepuasan Pelanggan

Kapan ini terjadi?

Overfitting terjadi ketika model sangat kompleks untuk jumlah data pelatihan yang diberikan.

Solusi

Untuk mengatasi masalah *overfitting*, Anda harus melakukan hal berikut:

- Kumpulkan lebih banyak data untuk "data pelatihan"
- Kurangi tingkat kebisingan
- Pilih satu dengan parameter lebih sedikit

1.12 MENGECLIKAN DATA

Dari namanya, *underfitting* adalah kebalikan dari *overfitting*, dan Anda akan menemukan ini ketika modelnya sangat sederhana untuk dipelajari. Misalnya, menggunakan contoh kualitas hidup, kehidupan nyata lebih kompleks daripada model Anda, sehingga prediksi tidak akan menghasilkan hal yang sama, bahkan dalam contoh pelatihan.

Solusi

Untuk memperbaiki masalah ini:

- Pilih model yang paling kuat, yang memiliki banyak parameter.
- Masukkan fitur terbaik ke dalam algoritme Anda. Di sini, saya mengacu pada rekayasa fitur.

- Kurangi batasan pada model Anda.

Latihan

Dalam bab ini, kita telah membahas banyak konsep pembelajaran mesin. Bab-bab berikut akan sangat praktis, dan Anda akan menulis kode, tetapi Anda harus menjawab pertanyaan-pertanyaan berikut hanya untuk memastikan Anda berada di jalur yang benar.

1. Definisikan pembelajaran mesin
2. Jelaskan empat jenis sistem pembelajaran mesin.
3. Apa perbedaan antara pembelajaran terawasi dan tidak terawasi.
4. Sebutkan tugas-tugas yang tidak diawasi.
5. Mengapa pengujian dan validasi penting?
6. Dalam satu kalimat, jelaskan apa itu pembelajaran online.
7. Apa perbedaan antara pembelajaran batch dan offline?
8. Jenis sistem pembelajaran mesin apa yang harus Anda gunakan untuk membuat robot belajar berjalan?

Ringkasan

Dalam bab ini, Anda telah mempelajari banyak konsep yang berguna, jadi mari kita tinjau beberapa konsep yang mungkin membuat Anda sedikit tersesat. Pembelajaran mesin: ML mengacu pada membuat mesin bekerja lebih baik pada beberapa tugas, menggunakan data yang diberikan.

- Pembelajaran mesin datang dalam berbagai jenis, seperti pembelajaran terawasi, batch, tanpa pengawasan, dan online.
- Untuk menjalankan proyek ML, Anda perlu mengumpulkan data dalam set pelatihan, lalu mengumpulkan set tersebut ke algoritme pembelajaran untuk mendapatkan keluaran, “prediksi”.
- Jika Anda ingin mendapatkan output yang tepat, sistem Anda harus menggunakan data yang jelas, yang tidak terlalu kecil dan yang tidak memiliki fitur yang tidak relevan.

BAB 2

KLASIFIKASI

2.1 INSTALASI

Anda harus menginstal Python, Matplotlib dan Scikit-belajar untuk bab ini. Cukup buka bagian referensi dan ikuti langkah-langkah yang ditunjukkan.

2.2 MNIST

Dalam bab ini, Anda akan masuk lebih dalam ke sistem klasifikasi, dan bekerja dengan kumpulan data MNIST. Ini adalah satu set 70.000 gambar angka yang ditulis tangan oleh siswa dan karyawan. Anda akan menemukan bahwa setiap gambar memiliki label dan angka yang mewakilinya. Proyek ini seperti contoh "Halo, dunia" dari pemrograman tradisional.

Jadi, setiap pemula dalam pembelajaran mesin harus memulai dengan proyek ini untuk mempelajari tentang algoritma klasifikasi. Scikit-Learn memiliki banyak fungsi, termasuk MNIST. Mari kita lihat kodennya:

```
>>> from sklearn.datasets import fetch_mldata
>>> mn= fetch_mldata('MNIST original')
>>> mn
{'COL_NAMES': ['label', 'data'],
'Description': 'mldata.org data set: mn-original',
'data': array([[0, 0, 0,..., 0, 0, 0],
[0, 0, 0,..., 0, 0, 0],
[0, 0, 0,..., 0, 0, 0],
...,
[0, 0, 0,..., 0, 0, 0],
[0, 0, 0,..., 0, 0, 0],
[0, 0, 0,..., 0, 0, 0]], dataType=int8),
'tar': array([ 0., 0., 0.,..., 9., 9., 9.])} de
```

- Deskripsi adalah kunci yang menggambarkan kumpulan data.
- Kunci data di sini berisi larik dengan hanya satu baris misalnya, dan kolom untuk setiap fitur.
- Kunci target ini berisi larik dengan label. Mari bekerja dengan beberapa kode:

```
>>> X, y = mn["data"], mn["tar"]
>>> X.shape
(70000, 784)
>>> y.shape
(70000,)
```

- 7000 di sini berarti ada 70.000 gambar, dan setiap gambar memiliki lebih dari 700 fitur: "784". Karena, seperti yang Anda lihat, setiap gambar berukuran 28 x 28 piksel, Anda dapat membayangkan bahwa setiap piksel adalah satu fitur.

Mari kita ambil contoh lain dari kumpulan data. Anda hanya perlu mengambil fitur instance, lalu membuatnya menjadi array 26 x 26, dan kemudian menampilkannya menggunakan fungsi imshow:

```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
yourDigit = X[36000]
Your_image = your_image.reshape(28, 28)
plt.imshow(Your_image, cmap = matplotlib.cm.binary,
interpolation="nearest")
plt.axis("off")
plt.show()
```

Seperti yang Anda lihat pada gambar berikut, sepertinya nomor lima, dan kami dapat memberikan label yang memberi tahu kami bahwa itu adalah lima.



Gambar 2.1 Angka 5

Pada gambar berikut, Anda dapat melihat tugas klasifikasi yang lebih kompleks dari kumpulan data MNIST.



Gambar 2.2 Kumpulan Angka

Selain itu, Anda harus membuat set pengujian dan membuatnya sebelum data Anda diperiksa. Kumpulan data MNIST dibagi menjadi dua set, satu untuk pelatihan dan satu untuk pengujian. $x_{\text{tr}}, x_{\text{tes}}, y_{\text{tr}}, y_{\text{te}} = x[:60000], x[60000:], y[:60000], y[60000:]$

Mari bermain dengan set latihan Anda sebagai berikut untuk membuat validasi silang menjadi serupa (tanpa ada angka yang hilang)

Import numpy as np

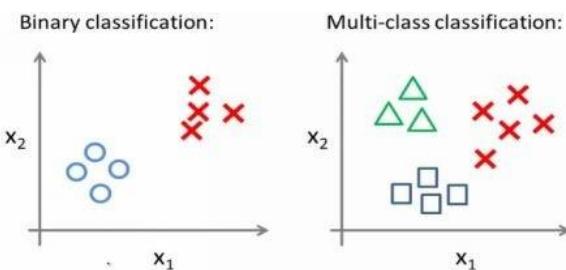
```
myData = np.random.permutation(50000)
x_tr, y_tr = x_tr[myData], y_tr[myData]
```

Sekarang saatnya untuk membuatnya cukup sederhana, kami akan mencoba mengidentifikasi satu digit saja, misal angka 6. “6-detektor” ini akan menjadi contoh pengklasifikasi biner, untuk membedakan antara 6 dan bukan 6, jadi kami akan membuat vektor untuk tugas ini:

```
Y_tr_6 = (y_tr == 6) // ini berarti benar untuk 6 detik, dan salah untuk nomor lainnya
Y_tes_6 = (Y_tes == 6)
```

Setelah itu, kita dapat memilih classifier dan melatihnya. Mulailah dengan pengklasifikasi SGD (*Stochastic Gradient Descent*). Kelas Scikit-Learn memiliki keuntungan menangani kumpulan data yang sangat besar. Dalam contoh ini, SGD akan menangani instance secara terpisah, sebagai berikut.

```
from sklearn.linear_model import SGDClassifier
mycl = SGDClassifier(random_state = 42)
mycl.fit(x_tr, y_tr_6)
menggunakannya untuk mendeteksi 6
>>>mycl.predict([any_digit])
```



Gambar 2.3 Pemetaan Klasifikasi Kelas

2.3 UKURAN KINERJA

Jika Anda ingin mengevaluasi *classifier*, ini akan lebih sulit daripada regressor, jadi mari kita jelaskan cara mengevaluasi *classifier*. Dalam contoh ini, kami akan menggunakan validasi silang untuk mengevaluasi model kami.

```

from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone
sf = StratifiedKFold(n=2, random_state = 40)
for train_index, test_index in sf.split(x_tr, y_tr_6):
    cl = clone(sgd_clf)
    x_tr_fd = x_tr[train_index]
    y_tr_fd = (y_tr_6[train_index])
    x_tes_fd = x_tr[test_index]
    y_tes_fd = (y_tr_6[test_index])
    cl.fit(x_tr_fd, y_tr_fd)
    y_p = cl.predict(x_tes_fd)
    print(n_correct / len(y_p))

```

Kami menggunakan kelas `StratifiedKFold` untuk melakukan pengambilan sampel bertingkat yang menghasilkan lipatan yang berisi jatah untuk setiap kelas. Selanjutnya, setiap iterasi dalam kode akan membuat tiruan dari pengklasifikasi untuk membuat prediksi pada lipatan uji. Dan akhirnya, itu akan menghitung jumlah prediksi yang benar dan rasionalnya.

Sekarang kita akan menggunakan fungsi `cross_val_score` untuk mengevaluasi `SGDClassifier` dengan validasi silang K-fold. Validasi silang k fold akan membagi training set menjadi 3 fold, kemudian akan dilakukan prediksi dan evaluasi pada setiap fold.

```

from sklearn.model_selection import cross_val_score
cross_val_score(sgd_clf, x_tr, y_tr_6, cv = 3, scoring = "accuracy")

```

Anda akan mendapatkan rasio akurasi "prediksi yang benar" di semua lipatan. Mari kita klasifikasikan setiap pengklasifikasi di setiap gambar di not-6

```

class never6Classifier(BaseEstimator):
    def fit(self, X, y=None):
        pass
    def predict(self, x):
        return np.zeros((len(X), 1), dtype=bool)

```

Mari kita periksa keakuratan model ini dengan kode berikut:

```

>>> never_6_cl = Never6Classifier()
>>> cross_val_score(never_6_cl, x_tr, y_tr_6, cv = 3, scoring = "accuracy")
Output: array ([“num”, “num”, “num”])

```

Untuk output, Anda akan mendapatkan tidak kurang dari 90%: hanya 10% dari gambar yang 6s, jadi kami selalu dapat membayangkan bahwa sebuah gambar bukan 6. Kami akan benar sekitar 90% dari waktu. Ingatlah bahwa akurasi bukanlah ukuran kinerja terbaik untuk pengklasifikasi, jika Anda bekerja dengan kumpulan data miring.

2.4 Matriks Konfusi

Ada metode yang lebih baik untuk mengevaluasi kinerja pengklasifikasi Anda: matriks kebingungan. Sangat mudah untuk mengukur kinerja dengan matriks konfusi, hanya dengan menghitung berapa kali instance kelas X diklasifikasikan sebagai kelas Y, misalnya. Untuk mendapatkan berapa kali pengklasifikasi gambar 6s dengan 2s, Anda harus mencari di baris ke-6 dan kolom ke-2 dari matriks konfusi. Mari kita hitung matriks konfusi menggunakan fungsi `cross_val_predict()` function.

```
from sklearn.model_selection import cross_val_predict
y_tr_pre = cross_val_predict(sgd_cl, x_tr, y_tr_6, cv = 3)
```

Fungsi ini, seperti fungsi `cross_val_score()`, melakukan validasi silang k fold, dan juga mengembalikan prediksi pada setiap fold. Ini juga mengembalikan prediksi bersih untuk setiap instance di set pelatihan Anda. Sekarang kita siap untuk mendapatkan matriks menggunakan kode berikut.

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_tr_6, y_tr_pred)
```

Anda akan mendapatkan array 4 nilai, "angka".

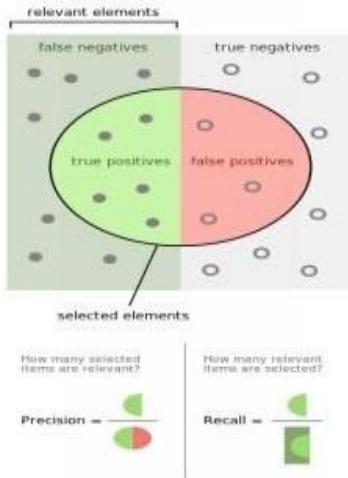
Setiap baris mewakili kelas dalam matriks, dan setiap kolom mewakili kelas yang diprediksi. Baris pertama adalah yang negatif: yang "berisi non-6 gambar". Anda dapat belajar banyak dari matriks. Tapi ada juga yang bagus, menarik untuk dikerjakan jika Anda ingin mendapatkan keakuratan prediksi positif, yaitu ketepatan pengklasifikasi menggunakan persamaan ini.

$$\text{Presisi} = (TP) / (TP+FP)$$

TP: jumlah positif benar

FP: jumlah positif palsu

Recall = (TP) / (TP+FN) "sensitivitas": mengukur rasio contoh positif.



Gambar 2.4 Perhitungan Sensitifitas

2.5 MENGINGAT

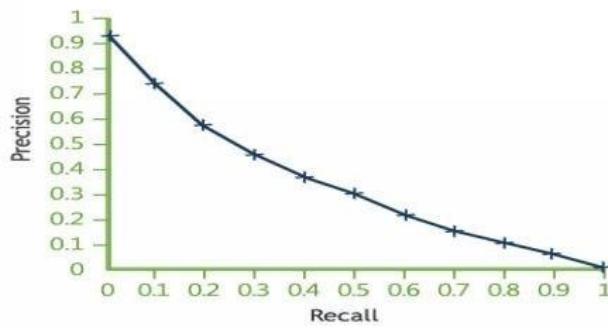
```
>>> from sklearn.metrics import precision_score, recall_score
>>> precision_score(y_tr_6, y_pre)
>>> recall_score(y_tr_6, y_tr_pre)
```

Sangat umum untuk menggabungkan presisi dan ingatan menjadi hanya satu metrik, yaitu skor F1. F1 adalah mean dari presisi dan recall. Kita dapat menghitung skor F1 dengan persamaan berikut:

$$F1 = \frac{2}{((1/\text{precision}) + (1/\text{recall}))} = \frac{2 * (\text{precision} * \text{recall})}{(\text{precision} + \text{recall})} = \frac{(\text{TP})}{((\text{TP}) + (\text{FN}+\text{FP})/2)}$$

Untuk menghitung skor F1, cukup gunakan fungsi berikut:

```
>>> from sklearn.metrics import f1_score
>>> f1_score (y_tr_6, y_pre)
```

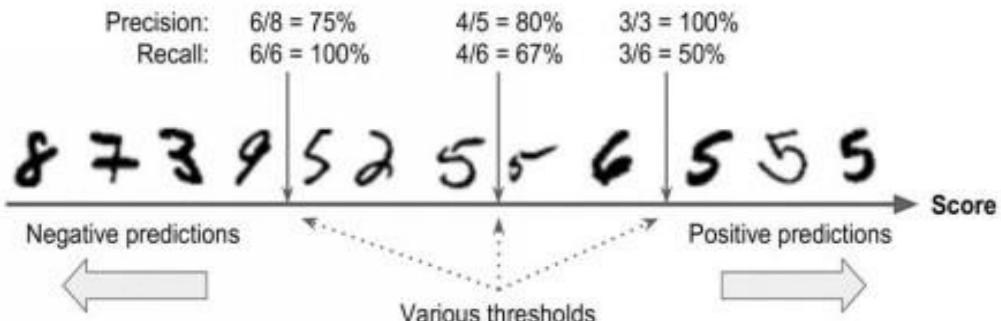


Gambar 2.5 Grafik Sensitifitas

2.6 INGAT TRADEOFF

Untuk sampai ke titik ini, Anda harus melihat SGDClassifier dan bagaimana SGDClassifier membuat keputusan terkait klasifikasi. Ini menghitung skor berdasarkan fungsi keputusan, dan kemudian membandingkan skor dengan ambang batas. Jika lebih besar dari

skor ini, itu akan menetapkan instance ke "positif atau negatif". Misalnya, jika ambang batas keputusan berada di tengah, Anda akan menemukan 4 benar + di sisi kanan ambang batas, dan hanya satu yang salah. Jadi rasio presisi akan hanya 80%.



Gambar 2.6 Penetapan hasil Prediksi

Di Scikit-Learn, Anda tidak dapat menetapkan ambang batas secara langsung. Anda harus mengakses skor keputusan, yang menggunakan prediksi, dan dengan memanggil fungsi keputusan, ().

```
>>> y_sco = sgd_clf.decision_funciton([any digit])
>>> y_sco
>>> threshold = 0
>>>y_any_digit_pre = (y_sco > threshold)
```

Dalam kode ini, SGDClassifier berisi ambang, = 0, untuk mengembalikan hasil yang sama seperti fungsi prediksi () .

```
>>> threshold = 20000
>>>y_any_digit_pre = (y_sco > threshold)
>>>y_any_digit_pre
```

Kode ini akan mengkonfirmasi bahwa, ketika ambang batas meningkat, penarikan berkurang.

```
y_sco = cross_val_predict (sgd_cl, x_tr, y_tr_6, cv =3, method="decision function")
```

Saatnya untuk menghitung semua kemungkinan presisi dan penarikan kembali untuk ambang batas dengan memanggil fungsi precision_recall_curve()

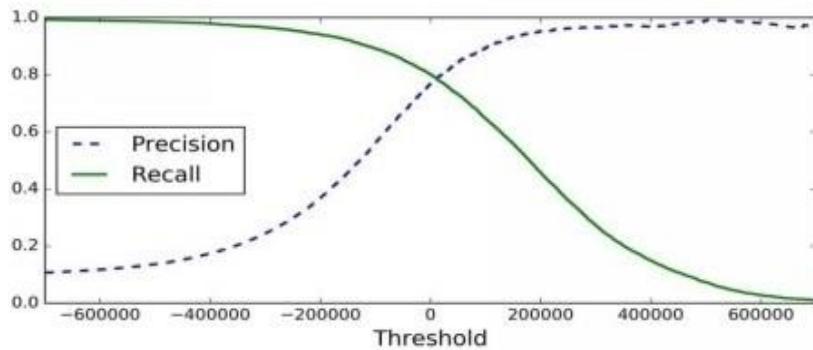
```
from sklearn.metrics import precision_recall_curve
```

presisi, penarikan, ambang batas = precision_recall_curve (y_tr_6, y_sco) dan sekarang mari kita plot presisi dan penarikan kembali menggunakan Matplotlib

```

def plot_pre_re(pre, re, thr):
    plt.plot(thr, pre[:-1], "b-", label = "precision")
    plt.plot(thr, re[1:], "g-", label="Recall")
    plt.xlabel("Threshold")
    plt.legend(loc="left")
    plt.ylim([0,1])
    plot_pre_re(pre, re, thr)
    plt.show

```



Gambar 2.7 Kurva penetapan Treshold

2.7 ROC

ROC singkatan dari karakteristik operasi penerima dan itu adalah alat yang digunakan dengan pengklasifikasi biner. Alat ini mirip dengan kurva recall, tetapi tidak memplot presisi dan recall: memplot tingkat positif dan tingkat palsu. Anda juga akan bekerja dengan FPR, yang merupakan rasio sampel negatif. Bisa dibayangkan jika seperti $(1 - \text{rate negatif})$. Konsep lain adalah TNR dan spesifisitasnya. Recall = $1 - \text{spesifisitas}$. Mari bermain dengan Kurva ROC. Pertama, kita perlu menghitung TPR dan FPR, cukup dengan memanggil fungsi roc_curve (),

```

from sklearn.metrics import roc_curve
fp,tp,thers = roc_curve (y_tr_6, y_sco)

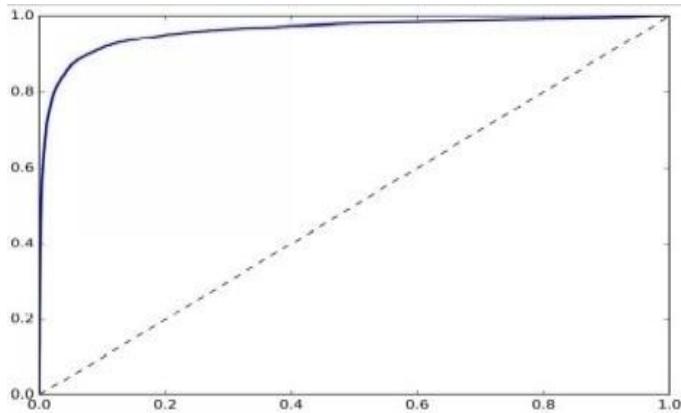
```

Setelah itu, Anda akan memplot FPR dan TPR dengan Matplotlib sesuai dengan instruksi berikut.

```

def roc_plot (fp, tp, label=none):
    plt.plot(fp, tp, linewidth=2, label = label)
    plt.plot([0,1], [0,1], "k--")
    plt.axis([0,1,0,1])
    plt.xlabel('This is the false rate')
    plt.ylabel('This is the true rate')
    roc_plot (fp, tp)
    plt.show

```



Gambar 2.8 Kurva Hasil Perhitungan

2.8 KLASIFIKASI MULTI-KELAS

Kami menggunakan pengklasifikasi biner untuk membedakan antara dua kelas, tetapi bagaimana jika Anda ingin membedakan lebih dari dua?

Anda dapat menggunakan sesuatu seperti pengklasifikasi hutan acak atau pengklasifikasi Bayes, yang dapat membandingkan antara lebih dari dua. Namun, di sisi lain, SVM (*Support Vector Machine*) dan pengklasifikasi linier berfungsi seperti pengklasifikasi biner. Jika Anda ingin mengembangkan sistem yang mengklasifikasikan gambar digit ke dalam 12 kelas (dari 0 hingga 11), Anda harus melatih 12 pengklasifikasi biner, dan membuatnya untuk setiap pengklasifikasi (seperti 4 – detektor, 5-detektor, 6-detektor dan seterusnya), dan kemudian Anda harus mendapatkan DS, "skor keputusan", dari setiap pengklasifikasi untuk gambar. Kemudian, Anda akan memilih pengklasifikasi skor tertinggi. Kami menyebutnya strategi OvA: "satu lawan semua."

Metode lainnya adalah melatih pengklasifikasi biner untuk setiap pasangan digit; misalnya, satu untuk 5s dan 6s dan satu lagi untuk 5s dan 7s. — kami menyebut metode ini OvO, "satu lawan satu" — untuk menghitung berapa banyak pengklasifikasi yang Anda perlukan, berdasarkan jumlah kelas yang menggunakan persamaan berikut: " $N = \text{jumlah kelas}$ ".

$N * (N-1)/2$. Jika Anda ingin menggunakan teknik ini dengan MNIST $10 * (10-1)/2$, outputnya akan menjadi 45 pengklasifikasi, "pengklasifikasi biner".

Di Scikit-Learn, Anda menjalankan OvA secara otomatis saat Anda menggunakan algoritma klasifikasi biner.

```
>>> sgd_cl.fit(x_tr, y_tr)
>>> sgd_cl.Predict([any-digit])
```

Selain itu, Anda dapat memanggil `decision_function()` untuk mengembalikan skor "10 skor untuk satu kelas"

```
>>> any_digit_scores = sgd_cl.decision_function([any_digit])
>>> any_digit_scores
Array(["num", "num", "num", "num", "num", "num", "num", "num", "num",
,"num"])
```

2.9 MELATIH PENGKLASIFIKASI HUTAN ACAK

```
>>> forest.clf.fit(x_tr, y_tr)
>>> forest.clf.predict([any-digit])
array([num])
```

Seperti yang Anda lihat, melatih pengklasifikasi hutan acak dengan hanya dua baris kode sangat mudah. Scikit-Learn tidak menjalankan fungsi OvA atau OvO karena algoritme semacam ini — “pengklasifikasi hutan acak” — dapat secara otomatis mengerjakan beberapa kelas. Jika Anda ingin melihat daftar kemungkinan pengklasifikasi, Anda dapat memanggil fungsi predict_proba().

```
>>> forest_cl.predict_proba([any_digit])
array([[0.1, 0, 0, 0.1, 0, 0.8, 0, 0, 0]])
```

Pengklasifikasi sangat akurat dengan prediksinya, seperti yang Anda lihat di output; ada 0,8 pada indeks nomor 5. Mari kita evaluasi classifier menggunakan fungsi cross_val_score().

```
>>> cross_val_score(sgd_cl, x_tr, y_tr, cv=3, scoring = "accuracy")
array([0.84463177, 0.859668, 0.8662669])
```

Anda akan mendapatkan 84% lebih banyak di lipatan. Saat menggunakan pengklasifikasi acak, Anda akan mendapatkan, dalam hal ini, 10% untuk skor akurasi. Perlu diingat bahwa semakin tinggi nilai ini, semakin baik.

2.10 ANALISIS KESALAHAN

Pertama-tama, saat mengembangkan proyek pembelajaran mesin:

1. Tentukan masalahnya;
2. Kumpulkan data Anda;
3. Kerjakan data Anda dan jelajahi;
4. Bersihkan data
5. Bekerja dengan beberapa model dan pilih yang terbaik;
6. Gabungkan model Anda ke dalam solusi;
7. Tunjukkan solusi Anda;
8. Jalankan dan uji sistem Anda.

Pertama, Anda harus bekerja dengan matriks konfusi dan membuat prediksi dengan fungsi cross-val. Selanjutnya, Anda akan memanggil fungsi matriks kebingungan:

```

>>> y_tr_pre = cross_val_predict(sgd_cl, x_tr_scaled, y_tr, cv=3)
>>> cn_mx = confusion_matrix(y_tr, y_tr_pre)
>>> cn_mx

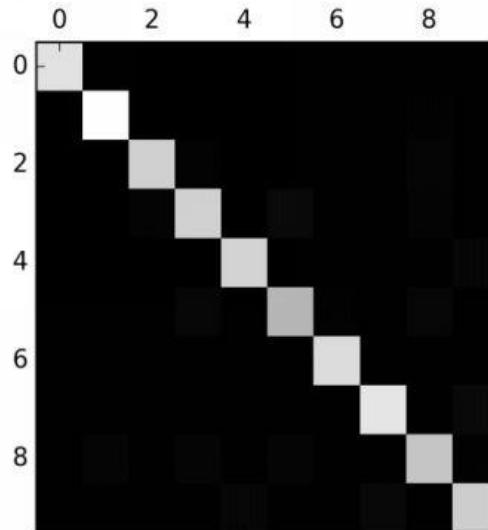
array([[5625, 2, 25, 8, 11, 44, 52, 12, 34, 6],
       [2, 2415, 41, 22, 8, 45, 10, 10, 9],
       [52, 43, 7443, 104, 89, 26, 87, 60, 166, 13],
       [47, 46, 141, 5342, 1, 231, 40, 50, 141, 92],
       [19, 29, 41, 10, 5366, 9, 56, 37, 86, 189],
       [73, 45, 36, 193, 64, 4582, 111, 30, 193, 94],
       [29, 34, 44, 2, 42, 85, 5627, 10, 45, 0],
       [25, 24, 74, 32, 54, 12, 6, 5787, 15, 236],
       [52, 161, 73, 156, 10, 163, 61, 25, 5027, 123],
       [50, 24, 32, 81, 170, 38, 5, 433, 80, 4250]])

```

```

plt.matshow(cn_mx, cmap=plt.cm.gray)
plt.show()

```



Gambar 2.9 Hasil Array hasil yang muncul

Pertama, Anda harus membagi setiap nilai dalam matriks dengan jumlah gambar di kelas, dan kemudian Anda akan membandingkan tingkat kesalahannya.

```

rw_sm = cn_mx.sum(axis=1, keepdims=True)
nm_cn_mx = cn_mx / rw_sm

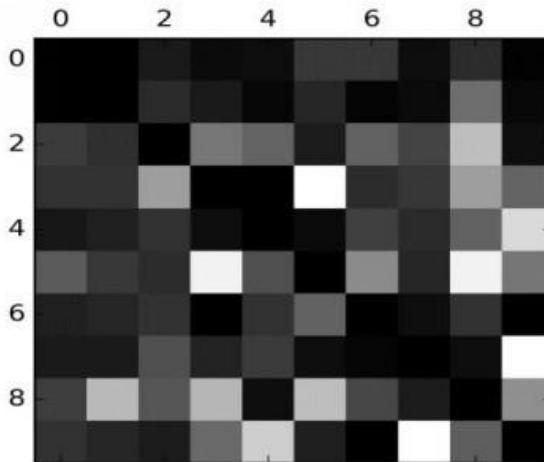
```

Langkah selanjutnya adalah membuat semua nol pada diagonal, dan itu akan mencegah kesalahan terjadi.

```

np.fill_diagonal(nm_cn_mx, 0)
plt.matshow(nm_cn_mx, cmap=plt.cm.gray)
plt.show()

```



Gambar 2.10 Hasil Array hasil yang muncul

2.11 KLASIFIKASI MULTI-LABEL

Dalam contoh di atas, setiap kelas hanya memiliki satu instance. Tetapi bagaimana jika kita ingin menetapkan instance ke beberapa kelas — pengenalan wajah, misalnya. Misalkan Anda ingin menemukan lebih dari satu wajah di foto yang sama. Akan ada satu label untuk setiap wajah. Mari kita berlatih dengan contoh sederhana.

```
y_tr_big = (y_tr >= 7)
y_tr_odd = (y_tr %2 ==1)
y_multi = np.c [y_tr_big, y_tr_odd]
kng_cl = KNeighborsClassifier()
kng_cl.fit (x_tr, y_m,ulti)
```

Dalam instruksi ini, kami telah membuat larik `y_mullti` yang berisi dua label untuk setiap gambar. Dan yang pertama berisi informasi apakah angkanya “besar” (8,9,.), dan yang kedua memeriksa ganjil atau tidak. Selanjutnya, kita akan membuat prediksi menggunakan rangkaian instruksi berikut.

```
>>>kng_cl.predict([any-digit])
Array([false, true], dataType=bool)
```

Benar di sini berarti ganjil dan salah, tidak besar.

2.12 KLASIFIKASI MULTI-OUTPUT

Pada titik ini, kita dapat membahas jenis tugas klasifikasi terakhir, yaitu klasifikasi multi-output. Ini hanya kasus umum klasifikasi multi-label, tetapi setiap label akan memiliki multiclass. Dengan kata lain, itu akan memiliki lebih dari satu nilai. Mari kita perjelas dengan contoh ini, menggunakan gambar MNIST, dan menambahkan beberapa noise ke gambar dengan fungsi NumPy.

```
No = rnd.randint (0, 101, (len(x_tr), 785))
No = rnd.randint(0, 101, (len(x_tes), 785))
```

```
x_tr_mo = x_tr + no  
x_tes_mo = x_tes + no  
y_tr_mo = x_tr  
y_tes_mo = x_tes  
kng_cl.fit(x_tr_mo, y_tr_mo)  
cl_digit = kng_cl.predict(x_tes_mo[any-index]))  
plot_digit(cl_digit)
```



Gambar 2.11 Hasil Output Klasifikasi Multi Output

Latihan

1. Buat pengklasifikasi untuk kumpulan data MNIST . Cobalah untuk mendapatkan akurasi lebih dari 96% pada set pengujian Anda.
2. Tulis metode untuk menggeser gambar dari MNIST (kanan atau kiri) sebanyak 2 piksel.
3. Kembangkan program atau pengklasifikasi anti-spam Anda sendiri.
 - Download contoh spam dari Google.
 - Ekstrak kumpulan data.
 - Bagilah kumpulan data menjadi pelatihan untuk kumpulan pengujian.
 - Tulis program untuk mengonversi setiap email menjadi vektor fitur.
 - Mainkan dengan pengklasifikasi, dan coba buat yang terbaik, dengan nilai tinggi untuk ingatan dan presisi.

BAB 3

CARA MELATIH MODEL

Setelah bekerja dengan banyak model pembelajaran mesin dan algoritma pelatihan, yang tampak seperti kotak hitam yang tak terduga. Kami dapat mengoptimalkan sistem regresi, juga bekerja dengan pengklasifikasi gambar. Tetapi kami mengembangkan sistem ini tanpa memahami apa yang ada di dalamnya dan bagaimana cara kerjanya, jadi sekarang kami perlu mempelajari lebih dalam sehingga kami dapat memahami cara kerjanya dan memahami detail implementasinya.

Memperoleh pemahaman mendalam tentang detail ini akan membantu Anda dengan model yang tepat dan dengan memilih algoritma pelatihan terbaik. Juga, ini akan membantu Anda dengan debugging dan analisis kesalahan.

Dalam bab ini, kita akan bekerja dengan regresi polinomial, yang merupakan model kompleks yang bekerja untuk kumpulan data nonlinier. Selain itu, kami akan bekerja dengan beberapa teknik regularisasi yang mengurangi pelatihan yang mendorong *overfitting*.

3.1 REGRESI LINIER

Sebagai contoh, kita akan mengambil $I_S = \theta_0 + \theta_1 \times \text{GDP_per_cap}$. Ini adalah model sederhana untuk fungsi linier dari fitur input , "GDP_per_cap". (θ_0 dan θ_1) adalah parameter model.

Secara umum, Anda akan menggunakan model linier untuk membuat prediksi dengan menghitung jumlah bobot dari fitur input, dan juga "bias" konstan seperti yang Anda lihat dalam persamaan berikut.

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

Y adalah nilai prediktor.

N mewakili fitur

X_1 adalah nilai fitur.

θ_j adalah parameter model dari j theta.

Juga, kita dapat menulis persamaan dalam bentuk vektor, seperti pada contoh berikut:

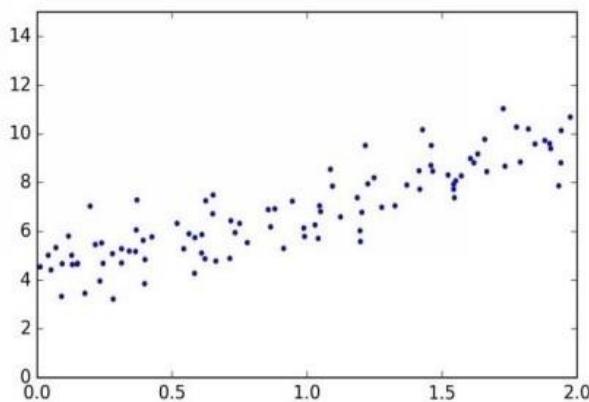
$$\hat{y} = h_{\theta}(\mathbf{x}) = \boldsymbol{\theta}^T \cdot \mathbf{x}$$

Θ adalah nilai yang meminimalkan biaya.

Y berisi nilai y (1) hingga y (m).

Mari kita menulis beberapa kode untuk berlatih. Impor numpy sebagai np

```
V1_x = 2 * np.random.rand(100, 1)
V2_y = 4 + 3 * V1_x + np.random.randn(100, 1)
```



Gambar 3.1 Hasil Regresi linier perhitungan diatas

Setelah itu, kita akan menghitung Θ nilai menggunakan persamaan kita. Saatnya menggunakan fungsi `inv()` dari modul aljabar linier numpy (`np.linalg`) untuk menghitung invers matriks apa pun, dan juga, fungsi `dot()` untuk mengalikan matriks kita

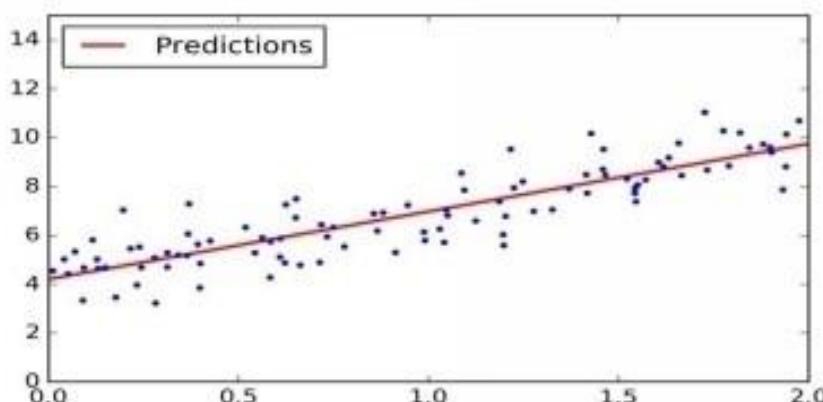
```
Value1 = np.c_[np.ones((100, 1)), V1_x]
myTheta = np.linalg.inv(Value1.T.dot(Value1)).dot(Value1.T).dot(V2_y)
>>>myTheta Array([[num], [num]])
```

Fungsi ini menggunakan persamaan berikut — $y = 4 + 3x + \text{noise}$ “Gaussian” — untuk menghasilkan data kita. Sekarang mari kita buat prediksi kita.

```
>>>V1_new = np.array([[0],[2]])
>>>V1_new_2 = np.c_[np.ones((2,1)), V1_new]
>>>V2_predict = V1_new_2.dot(myTheta)
>>>V2_predict
Array([[ 4.219424], [9.74422282]])
```

Sekarang, saatnya untuk memplot modelnya.

```
Plt.plot(V1_new, V2_predict, "r-")
Plt.plot(V1_x, V2_y, "b.")
Plt.axis([0,2,0,15])
Plt.show()
```



Gambar 3.2 Grafik Predikso Regresi Linier

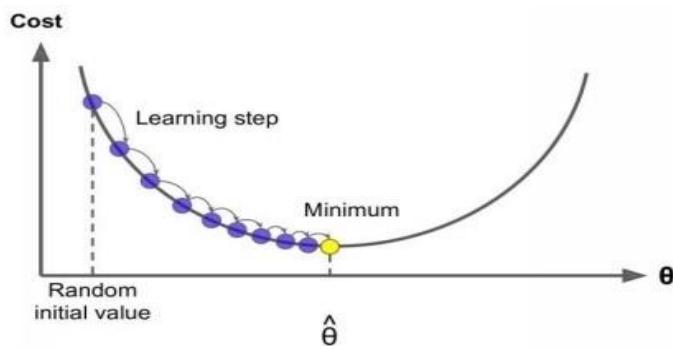
3.2 KOMPLEKSITAS KOMPUTASI

Dengan rumus normal, kita dapat menghitung invers dari $M^T M$ — yaitu, matriks $n \times n$ ($n = \text{jumlah fitur}$). Kompleksitas invers ini kira-kira seperti $O(n^{2.5})$ hingga $O(n^{3.2})$, yang didasarkan pada implementasinya. Sebenarnya, jika Anda membuat jumlah fitur menjadi dua kali, Anda akan membuat waktu komputasi mencapai antara $2^{2.5}$ dan $2^{3.2}$.

Kabar baiknya di sini adalah bahwa persamaan tersebut adalah persamaan linier. Ini berarti ia dapat dengan mudah menangani set pelatihan besar dan memasukkan memori. Setelah melatih model Anda, prediksinya tidak akan lambat, dan kerumitannya akan sederhana, berkat model linier. Saatnya untuk masuk lebih dalam ke metode pelatihan model regresi linier, yang selalu digunakan ketika ada banyak fitur dan instance dalam memori.

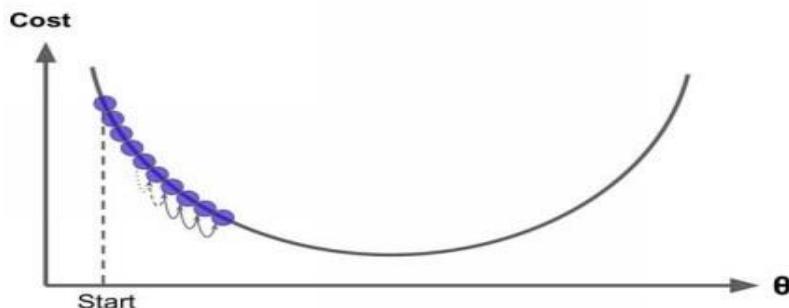
3.3 PENURUNAN GRADIENT

Algoritma ini merupakan algoritma umum yang digunakan untuk optimasi dan untuk memberikan solusi optimal untuk berbagai masalah. Ide dari algoritma ini adalah untuk bekerja dengan parameter secara iteratif, untuk membuat fungsi biaya sesederhana mungkin. Algoritma penurunan gradien menghitung gradien kesalahan menggunakan parameter theta, dan bekerja dengan metode gradien menurun. Jika gradien sama dengan nol, Anda akan mencapai minimum.



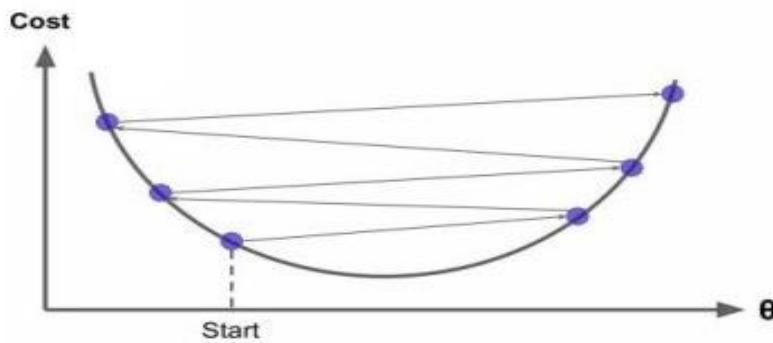
Gambar 3.3 Penurunan Gradien

Selain itu, Anda harus ingat bahwa ukuran langkah sangat penting untuk algoritma ini, karena jika sangat kecil — “artinya laju pembelajaran” lambat, akan memakan waktu lama untuk mencakup semua yang diperlukan.



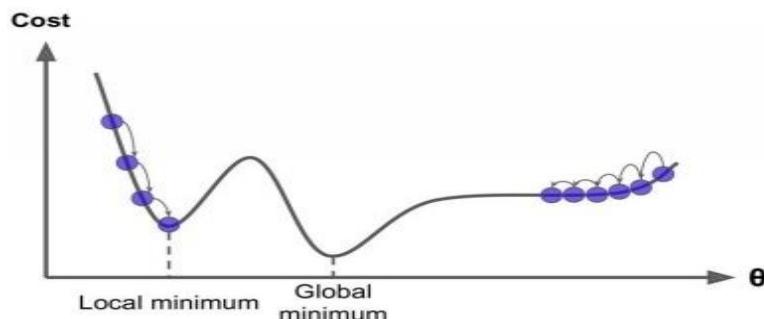
Gambar 3.4 Gradien Laju Pembelajaran Lambat

Tetapi ketika tingkat pembelajaran tinggi, itu akan memakan waktu singkat untuk menutupi apa yang dibutuhkan, dan itu akan memberikan solusi yang optimal.



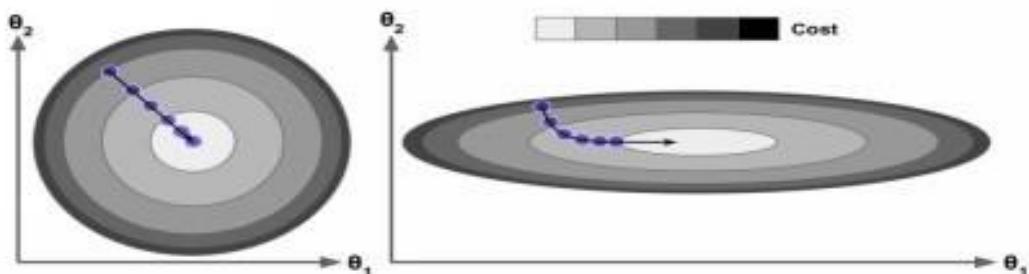
Gambar 3.5 Gradien Laju Pembelajaran Tinggi

Pada akhirnya, Anda tidak akan selalu menemukan bahwa semua fungsi biaya itu mudah, seperti yang Anda lihat, tetapi Anda juga akan menemukan fungsi tidak beraturan yang membuat mendapatkan solusi optimal menjadi sangat sulit. Masalah ini terjadi ketika minimum lokal dan minimum global terlihat seperti pada gambar berikut.



Gambar 3.6 Gradien Tidak Beraturan

Jika Anda menetapkan salah satu untuk dua titik pada kurva Anda, Anda akan menemukan bahwa segmen garis tidak akan bergabung dengan mereka pada kurva yang sama. Fungsi biaya ini akan terlihat seperti mangkuk, yang akan terjadi jika fitur memiliki banyak skala, seperti pada gambar berikut:



Gambar 3.7 Multi Skala Penurunan Gradien

3.4 PENURUNAN GRADIENTE BATCH

Jika Anda ingin menerapkan algoritme ini, Anda harus terlebih dahulu menghitung gradien fungsi biaya Anda menggunakan parameter theta. Jika nilai parameter theta telah berubah, Anda harus mengetahui tingkat perubahan fungsi biaya Anda. Kita dapat menyebut

perubahan ini dengan turunan parsial. Kita dapat menghitung turunan parsial menggunakan persamaan berikut:

$$\frac{\partial}{\partial \theta_j} \text{MSE}(\theta) = \frac{2}{m} \sum_{i=1}^m (\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)}) x_j^{(i)}$$

Tapi kita juga akan menggunakan persamaan berikut untuk menghitung turunan parsial dan vektor gradien bersama-sama.

$$\nabla_{\theta} \text{MSE}(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \text{MSE}(\theta) \\ \frac{\partial}{\partial \theta_1} \text{MSE}(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \text{MSE}(\theta) \end{pmatrix} = \frac{2}{m} \mathbf{X}^T \cdot (\mathbf{X} \theta - \mathbf{y})$$

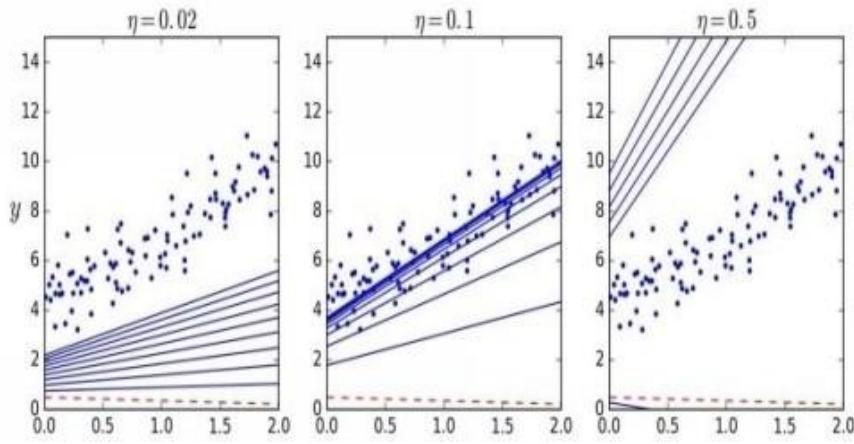
Mari kita terapkan algoritmanya.

```
Lr = 1 # Lr untuk kecepatan belajar
Jumlah_it = 1000 # jumlah iterasi L = 100
myTheta = np.random.randn (2,1)
```

untuk itu dalam jangkauan (Num_it):

```
gr = 2/L * Value1.T.dot(Value1.dot(myTheta) – V2_y)
myTheta = myTheta – Lr * gr
>>> myTheta
Array([[num],[num]])
```

Jika Anda mencoba mengubah nilai learning rate, Anda akan mendapatkan bentuk yang berbeda, seperti pada gambar berikut.

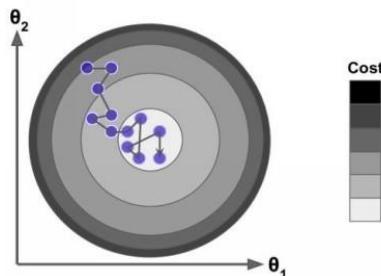


Gambar 3.8 Mengubah nilai learning rate

3.5 PENURUNAN GRADIENT STOKASTIK

Anda akan menemukan masalah saat menggunakan penurunan gradien batch: perlu menggunakan seluruh rangkaian pelatihan untuk menghitung nilai pada setiap langkah, dan itu akan memengaruhi "kecepatan" kinerja.

Tetapi saat menggunakan penurunan gradien stokastik, algoritme akan secara acak memilih instance dari set pelatihan Anda di setiap langkah, dan kemudian akan menghitung nilainya. Dengan cara ini, algoritma akan lebih cepat daripada penurunan gradien batch, karena tidak perlu menggunakan seluruh rangkaian untuk menghitung nilainya. Di sisi lain, karena keacakan metode ini, maka akan menjadi tidak teratur jika dibandingkan dengan algoritma batch.



Gambar 3.9 penggunaan metode penurunan gradient stokastik

Mari kita terapkan algoritmanya.

Bilangan = 50

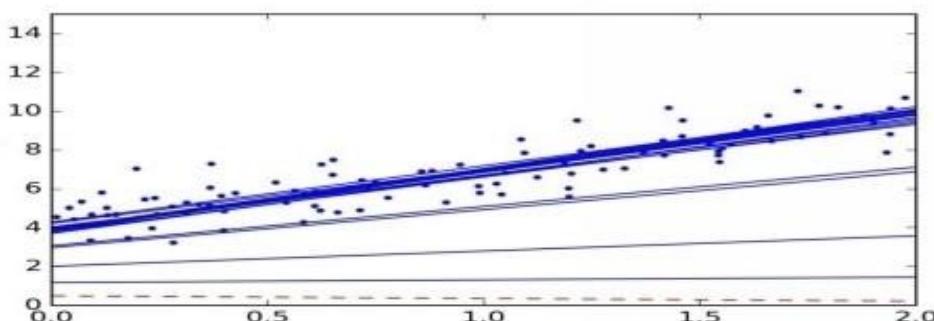
L1, L2 = 5, 50

Def lr_sc(s):

```
return L1 / (s + L2)
myTheta = np.random.randn(2,1)
```

untuk Num dalam jangkauan (Bilangan):

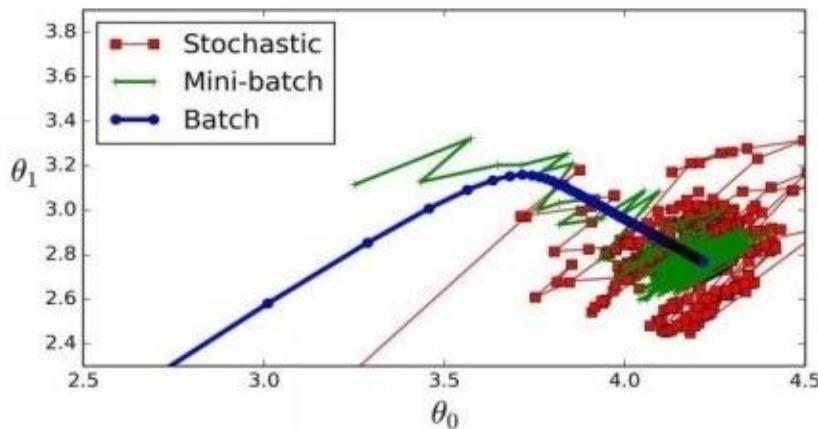
```
for l in range (f)
    myIndex = np.random.randint(f)
    V1_Xi = Value1[myIndex:myIndex+1] V
    V2_yi = V2_y[myIndex:myIndex+1]
    gr = 2 * V1_xi.T.dot(V1_xi.dot(myTheta) - V2_yi)
    Lr = lr_sc(Num * f + i)
    myTheta = myTheta - Lr * gr
    >>> myTheta
    Array ([[num], [num]])
```



3.10 hasil penerapan algoritmanya

3.6 PENURUNAN GRADIEN BATCH MINI

Karena Anda sudah mengetahui kumpulan dan algoritma stokastik, algoritma semacam ini sangat mudah dipahami dan digunakan. Seperti yang Anda ketahui, kedua algoritma menghitung nilai gradien, berdasarkan seluruh rangkaian pelatihan atau hanya satu contoh. Namun, mini-batch menghitung algoritmanya berdasarkan set kecil dan acak, dan berkinerja lebih baik daripada dua algoritma lainnya.



Gambar 3.11 penghitungan algoritma mini batch berdasarkan set kecil dan acak

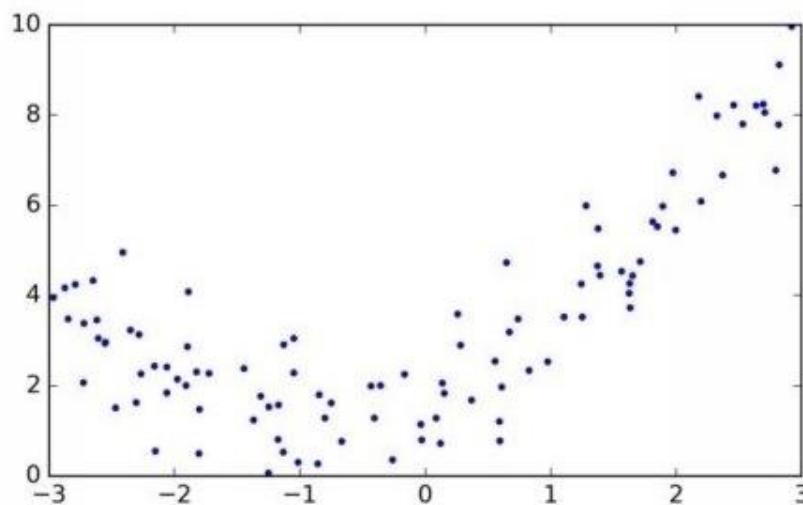
3.7 REGRESI POLINOMIAL

Kami akan menggunakan teknik ini ketika bekerja dengan data yang lebih kompleks, terutama, dalam kasus data linier dan nonlinier. Setelah kami menambahkan kekuatan setiap fitur, kami dapat melatih model dengan fitur baru. Ini dikenal sebagai regresi polinomial.

Sekarang, mari kita tulis beberapa kode.

$L = 100$

```
V1 = 6*np.random.rand(L, 1) - 3
V2 = 0,5 * V1**2 + V1 + 2 + np.random.randn(L, 1)
```



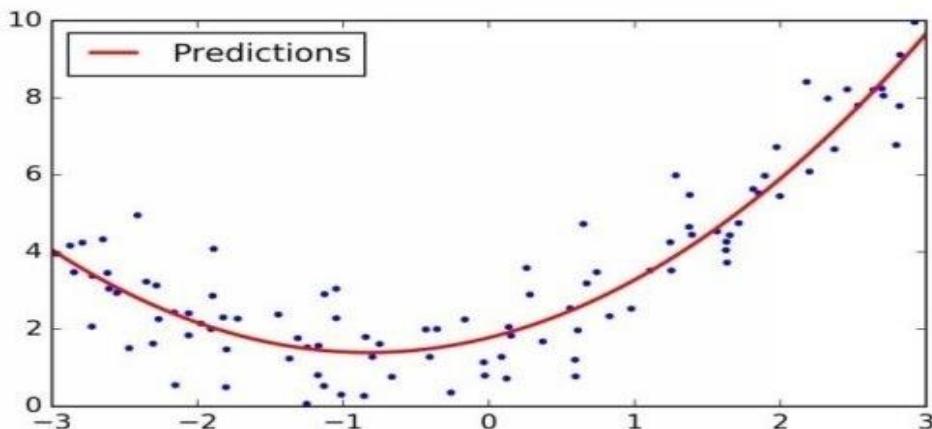
Gambar 3.12 Penggunaan teknik regresi polinomial membuat bekerja dengan data yang lebih kompleks

Seperti yang Anda lihat, lurus tidak akan pernah mewakili data dengan cara yang paling efisien. Jadi kita akan menggunakan metode polinomial untuk mengatasi masalah ini.

```
>>>from sklearn.preprocessing import PolynomialFeatures
>>>P_F = PolynomialFeatures(degree = 2, include_bias=False)
>>>V1_P = P_F.fit_transform(V1)
>>>V1[0]
Array([num])
>>>V1_P[0]
```

Sekarang, mari kita buat fungsi ini dengan data kita benar, dan ubah garis lurusnya.

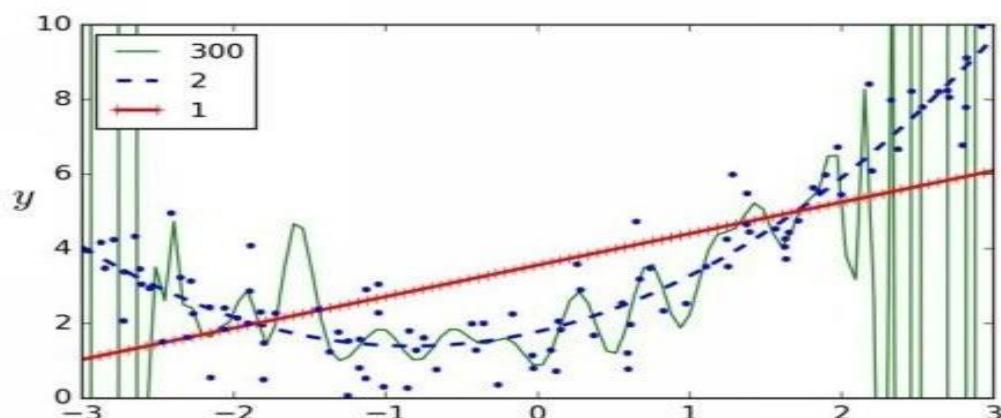
```
>>>ln_r = LinearRegression()
>>>ln_r.fit(V1_P, V2)
>>>ln_r.intercept_, ln_r.coef
```



Gambar 3.13 membuat fungsi dan mengubah garis lurusnya

3.8 KURVA PEMBELAJARAN

Asumsikan bahwa Anda bekerja dengan regresi polinomial, dan Anda ingin agar data lebih cocok daripada regresi linier. Pada gambar berikut, Anda akan menemukan model 300 derajat. Kita juga dapat membandingkan hasil akhir dengan jenis regresi lainnya: "linier normal".



Gambar 3.14 Overfitting data menggunakan polinomial

Pada gambar di atas, Anda dapat melihat *overfitting* data saat Anda menggunakan polinomial. Di sisi lain, dengan yang linier, Anda dapat melihat bahwa datanya jelas kurang pas.

Model Linier Teratur

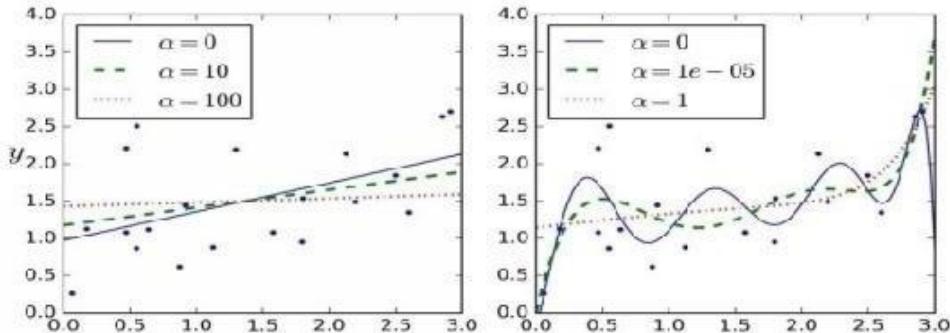
Kami telah bekerja, di bab pertama dan kedua, tentang cara mengurangi *overfitting* dengan sedikit mengatur model, sebagai contoh, jika Anda ingin mengatur model polinomial. Dalam hal ini, untuk memperbaiki masalah, Anda harus mengurangi jumlah derajat.

Regresi punggungan

Regresi punggungan adalah versi lain dari regresi linier, tetapi, setelah mengurnya dan menambahkan bobot pada fungsi biaya, ini membuatnya sesuai dengan data, dan bahkan membuat bobot model sesederhana mungkin. Berikut adalah fungsi biaya dari regresi ridge:

$$J(\theta) = \text{MSE}(\theta) + \alpha \frac{1}{2} \sum_{i=1}^n \theta_i^2$$

Sebagai contoh regresi ridge, lihat saja gambar berikut.



Gambar 3.15 Contoh regresi ridge

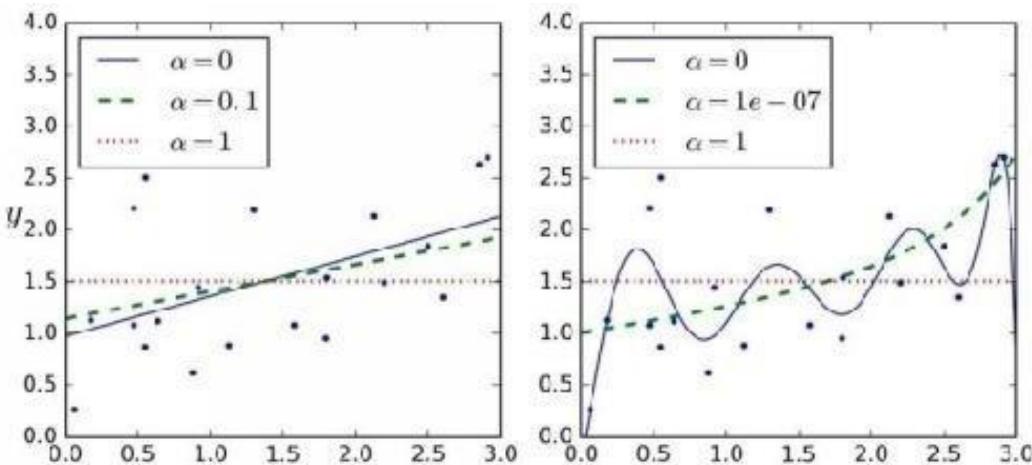
Regresi Lasso

Regresi "Lasso" adalah singkatan dari regresi "Penyusutan Terkecil dan Operator Seleksi". Ini adalah jenis lain dari versi reguler regresi linier. Ini terlihat seperti regresi punggungan, tetapi dengan perbedaan kecil dalam persamaan, seperti pada gambar berikut:

Fungsi biaya dari regresi lasso:

$$J(\theta) = \text{MSE}(\theta) + \alpha \sum_{i=1}^n |\theta_i|$$

Seperti yang Anda lihat pada gambar berikut, regresi lasso menggunakan nilai yang lebih kecil daripada ridge.



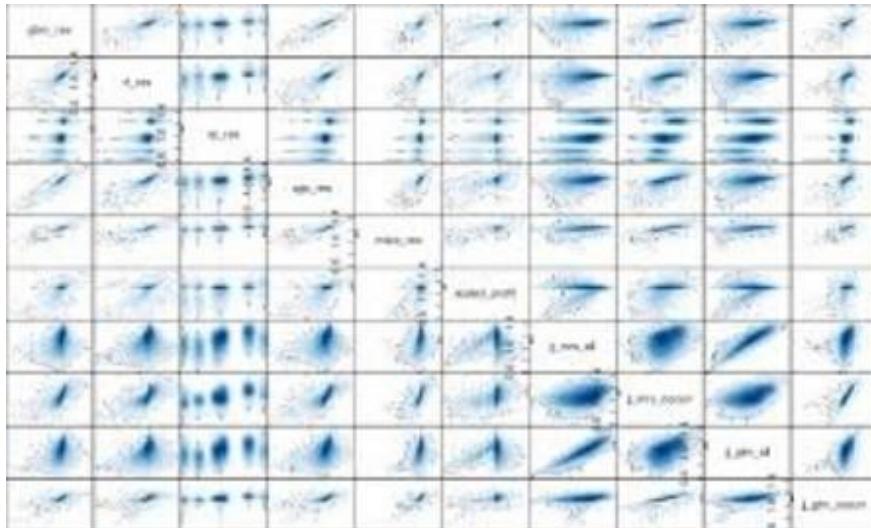
Gambar 3.16 Regresi Lasso Mempunyai Nilai Yang Lebih Kecil Dibandingkan Ridge

Latihan

1. Jika Anda memiliki satu set yang berisi sejumlah besar fitur (jutaan fitur), algoritma regresi mana yang harus Anda gunakan, dan mengapa?
2. Jika Anda menggunakan penurunan gradien batch untuk memplot kesalahan pada setiap periode, dan tiba-tiba tingkat kesalahan meningkat, bagaimana Anda memperbaiki masalah ini?
3. Apa yang harus Anda lakukan jika Anda melihat kesalahan menjadi lebih besar saat Anda menggunakan metode mini-batch? Mengapa?
4. Dari pasangan ini, metode mana yang lebih baik? Mengapa?
 - Regresi punggungan dan regresi linier?
 - Regresi Lasso dan regresi ridge?
5. Tulis algoritma penurunan gradien batch.

BAB 4

KOMBINASI MODEL YANG BERBEDA



Gambar 4.1 Kombinasi model yang berbeda

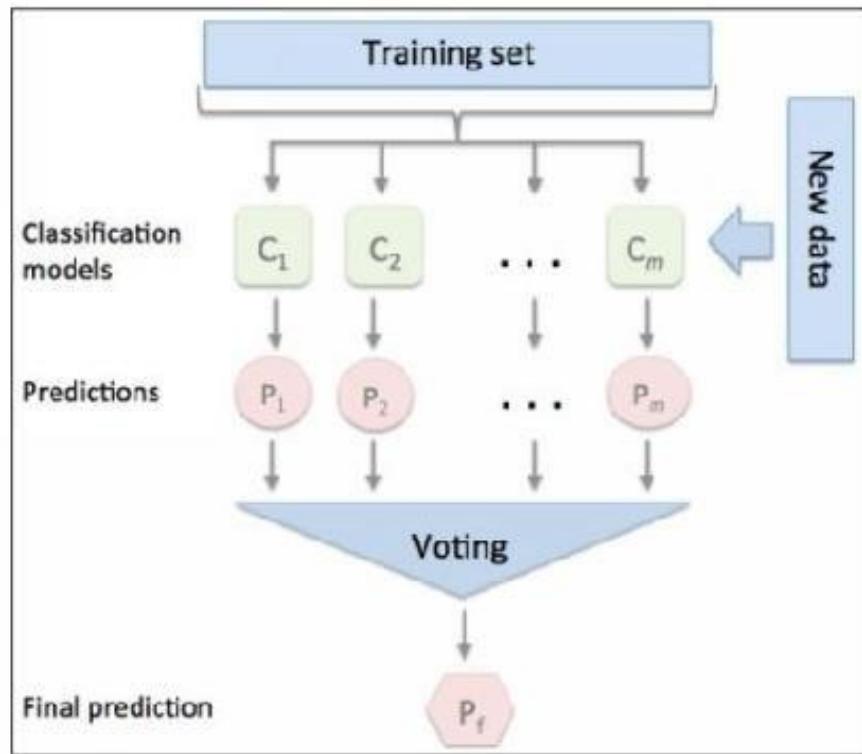
4.1 KLASIFIKASI POHON

Gambar berikut akan mengilustrasikan definisi dari target umum dari fungsi pengumpulan yaitu hanya untuk menggabungkan classifier yang berbeda menjadi satu classifier yang memiliki kinerja generalisasi yang lebih baik daripada masing-masing classifier saja.

Sebagai contoh, asumsikan bahwa Anda mengumpulkan prediksi dari banyak ahli. Metode ensemble akan memungkinkan kita untuk menggabungkan prediksi ini oleh banyak ahli untuk mendapatkan prediksi yang lebih tepat dan kuat daripada prediksi masing-masing pakar individu. Seperti yang dapat Anda lihat nanti di bagian ini, ada banyak metode berbeda untuk membuat ansambel pengklasifikasi. Pada bagian ini, kami akan memperkenalkan persepsi dasar tentang bagaimana ansambel bekerja dan mengapa mereka biasanya dikenal untuk menghasilkan kinerja generalisasi yang baik.

Pada bagian ini, kita akan bekerja dengan metode ensemble paling populer yang menggunakan prinsip voting mayoritas. Banyak voting berarti kita memilih label yang telah diprediksi oleh mayoritas pengklasifikasi; yaitu, menerima lebih dari 50 persen suara. Sebagai contoh, istilah di sini seperti memilih hanya mengacu pada pengaturan kelas biner saja. Namun, tidaklah sulit untuk membangkitkan prinsip pemungutan suara mayoritas ke dalam pengaturan multi-kelas, yang disebut pemungutan suara pluralitas. Setelah itu, kita akan memilih label kelas yang mendapat suara terbanyak. Diagram berikut mengilustrasikan konsep pemungutan suara mayoritas dan pluralitas untuk ansambel 10 pengklasifikasi di mana setiap simbol unik (segitiga, persegi, dan lingkaran) mewakili label kelas yang unik: Menggunakan set pelatihan, kita mulai dengan melatih m pengklasifikasi yang berbeda (C_1, \dots, C_m). Berdasarkan metodenya, ensemble dapat dibangun dari banyak algoritma klasifikasi; misalnya, pohon keputusan, mesin vektor pendukung, pengklasifikasi regresi logistik, dan *Algoritma Machine Learning Dengan Python (Dr. Joseph Teguh Santoso, M.Kom)*

sebagainya. Bahkan, Anda dapat menggunakan algoritma klasifikasi dasar yang sama untuk menyesuaikan subset yang berbeda dari set pelatihan. Contoh dari metode ini adalah algoritma hutan acak, yang menggabungkan banyak cara ansambel keputusan menggunakan suara mayoritas.



Gambar 4.2 Metode algoritma hutan acak dengan menggabungkan banyak cara ansambel suara mayoritas

Untuk memprediksi label kelas melalui pemungutan suara mayoritas atau pluralitas sederhana, kami menggabungkan label kelas yang diprediksi dari masing-masing classifier individu C_j dan memilih label kelas \hat{y} yang menerima suara terbanyak:

$$\hat{y} = \text{mode}\{C_1(x), C_2(x), \dots, C_m(x)\}$$

Misalnya, dalam tugas klasifikasi biner di mana kelas 1 = + dan kelas 2 = -, kita dapat menulis prediksi suara mayoritas.

Untuk mengilustrasikan mengapa metode ansambel dapat bekerja lebih baik daripada pengklasifikasi individual saja, mari kita terapkan konsep kombinasi yang sederhana. Untuk contoh berikut, kami membuat asumsi bahwa semua n pengklasifikasi dasar untuk tugas klasifikasi biner memiliki tingkat kesalahan yang sama, ϵ . Selain itu, kami berasumsi bahwa pengklasifikasi independen dan tingkat kesalahan tidak berkorelasi. Seperti yang Anda lihat, kami hanya dapat menjelaskan statistik kesalahan dari ensemble pengklasifikasi dasar sebagai probabilitas.

4.2 FUNGSI MASSA DARI DISTRIBUSI BINOMINAL

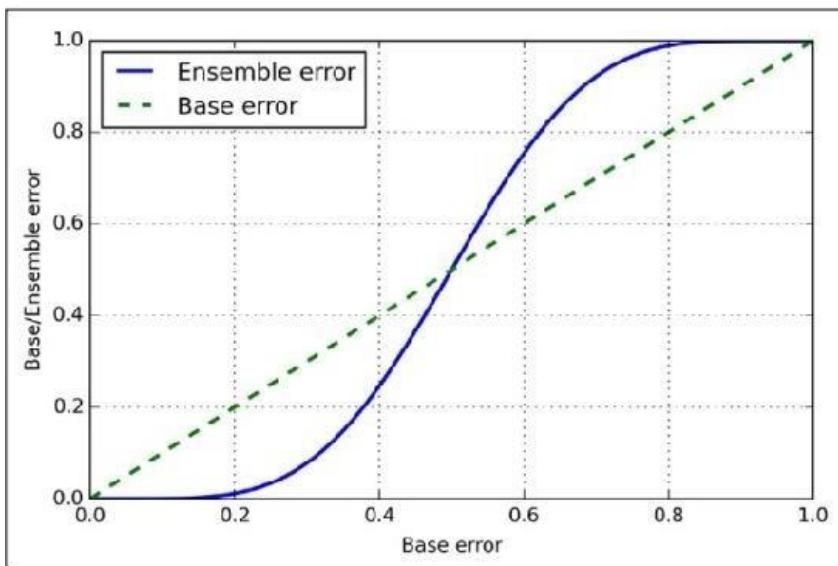
Di sini, n . k adalah koefisien binomial n pilih k . Seperti yang Anda lihat, Anda dapat menghitung probabilitas bahwa prediksi ansambel salah. Sekarang, mari kita lihat contoh yang lebih konkret dari 11 pengklasifikasi dasar ($n = 11$) dengan tingkat kesalahan $0,25$ ($\epsilon = 0,25$): Anda dapat melihat bahwa tingkat kesalahan ensemble ($0,034$) lebih kecil dari tingkat kesalahan masing-masing classifier ($0,25$) jika semua asumsi terpenuhi. Perhatikan bahwa dalam gambar yang disederhanakan ini, pembagian 50-50 dengan jumlah pengklasifikasi n yang genap dianggap sebagai kesalahan, sedangkan ini hanya benar separuh dari waktu. Untuk membandingkan pengklasifikasi ansambel idealis dengan pengklasifikasi dasar pada rentang tingkat kesalahan dasar yang berbeda, mari kita terapkan fungsi massa probabilitas dengan Python:

```
>>> import math
>>> def ensemble_error(n_classifier, error):
... q_start = math.ceil(n_classifier / 2.0)
... Probability = [comb(n_classifier, q) *
... error**q *
... (1-error)**(n_classifier - q)
... for q in range(q_start, n_classifier + 2)]
... return sum(Probability)
>>> ensemble_error(n_classifier=11, error=0.25)
0.034327507019042969
```

Mari kita menulis beberapa kode untuk menghitung tingkat kesalahan yang berbeda memvisualisasikan hubungan antara kesalahan ensemble dan dasar dalam grafik garis:

```
>>> import numpy as np
>>> error_range = np.arange(0.0, 1.01, 0.01)
>>> en_error = [en_er(n_classifier=11, er=er)
... for er in error_range]
>>> import matplotlib.pyplot as plt
>>> plt.plot(error_range, en_error,
... label='Ensemble error',
... linewidth=2)
>>> plt.plot(error_range, en_error,
... ls='--', label='B_en_error',
... linewidth=2)
>>> plt.xlabel('B_en_error')
>>> plt.ylabel('B/En_error')
>>> plt.legend(loc='upper left')
>>> plt.grid()
>>> plt.show()
```

Seperti yang dapat kita lihat di plot yang dihasilkan, probabilitas kesalahan suatu ensemble selalu lebih baik daripada kesalahan pengklasifikasi dasar individu selama pengklasifikasi dasar berkinerja lebih baik daripada tebakan acak ($\epsilon < 0.5$). Anda harus memperhatikan bahwa sumbu y menggambarkan kesalahan dasar serta kesalahan ensemble (garis kontinu):



Gambar 4.3 Kesalahan ensemble dan dasar dalam grafik garis

4.3 MENERAPKAN PENGKLASIFIKASI MAYORITAS SEDERHANA

Seperti yang kita lihat dalam pengantar untuk menggabungkan pembelajaran di bagian terakhir, kita akan bekerja dengan pelatihan pemanasan dan kemudian mengembangkan classifier sederhana untuk pemungutan suara mayoritas dalam pemrograman Python. Seperti yang Anda lihat, algoritma berikutnya akan bekerja pada pengaturan multi-kelas melalui pemungutan suara pluralitas; Anda akan menggunakan istilah suara mayoritas untuk penyederhanaan seperti yang juga sering dilakukan dalam literatur.

Dalam program berikut, kami akan mengembangkan dan juga menggabungkan berbagai program klasifikasi yang terkait dengan bobot individu untuk kepercayaan. Tujuan kami adalah membangun meta-classifier yang lebih kuat yang menyeimbangkan kelemahan masing-masing classifier pada kumpulan data tertentu. Dalam istilah matematika yang lebih tepat, kita dapat menulis suara mayoritas tertimbang.

Untuk menerjemahkan konsep suara mayoritas tertimbang ke dalam kode Python, kita dapat menggunakan fungsi argmax dan bincount NumPy yang nyaman:

```
>>> import numpy as np
>>> np.argmax(np.bincount([0, 0, 1],
... weights=[0.2, 0.2, 0.6]))
1
```

Di sini, p_{ij} adalah probabilitas yang diprediksi dari classifier ke- j untuk label kelas i . Untuk melanjutkan contoh sebelumnya, mari kita asumsikan bahwa kita memiliki masalah klasifikasi biner dengan label kelas $i \in \{0, 1\}$ dan ansambel tiga pengklasifikasi C_j ($j \in \{1, 2, 3\}$). Mari

kita asumsikan bahwa classifier C j mengembalikan probabilitas keanggotaan kelas berikut untuk sampel tertentu x:

$$C1(x) \rightarrow [0.9, 0.1], C2(x) \rightarrow [0.8, 0.2], C3(x) \rightarrow [0.4, 0.6]$$

Untuk menerapkan suara mayoritas tertimbang berdasarkan probabilitas kelas, kita dapat kembali menggunakan NumPy menggunakan `numpy.average` dan `np.argmax`:

```
>>> ex = np.array([[0.9, 0.1],
... [0.8, 0.2],
... [0.4, 0.6]])
>>> p = np.average(ex, axis=0, weights=[0.2, 0.2, 0.6])
>>> p
array([0.58, 0.42])
>>> np.argmax(p)
0
```

Menyatukan semuanya, sekarang mari kita implementasikan `MajorityVoteClassifier` dengan Python:

```
from sklearn.base import ClassifierMixin
from sklearn.preprocessing import LabelEncoder
from sklearn.ext import six
from sklearn.base import clone
from sklearn.pipeline import _name_estimators
import numpy as np
import operator
class MVClassifier(BaseEstimator,
ClassifierMixin):
    """ A majority vote ensemble classifier
```

Parameters

`cl` : array-like, shape = [n_classifiers]

Different classifiers for the ensemble vote: str, {'cl_label': 'prob'}

Default: 'cl_label'

Jika 'cl_label' prediksi didasarkan pada argmax label kelas. Elif 'prob', argumen dari total probs digunakan untuk memprediksi label kelas (disarankan untuk pengklasifikasi yang dikalibrasi).
`w` : arr-like, s = [n_cl]

Opsional, default: Tidak ada

Jika daftar nilai `int` atau `float` disediakan, pengklasifikasi diberi bobot oleh """"

`Def_init_(s, cl,`

```
v='cl_label', w=None): s.cl = cl
s.named_cl = {key: value for
key, value in
_name_estimators(cl)}
s.v = v
```

```

s.w= w
def fit_cl(s, X, y):
    """ Fit_cl.

    Parameters
    X : {array-like, sparse matrix},
    s = [n_samples, n_features]

```

Matriks sampel pelatihan.

```

y: arr_like, sh = [n_samples]
Vector of target class labels.

Returns
s: object
"""

# Use LabelEncoder to ensure class labels start
# with 0, which is important for np.argmax
# call in s.predict
s.l_ = LabelEncoder()
s.l_.fit(y)
s.cl_ = self.lablenc_.classes_
s.cl_ = []
for cl in s.cl:
    fit_cl = clone(cl).fit(X,
                           s.la_.transform(y))
    s.cl_.append(fit_cl)
return s

```

Saya menambahkan banyak komentar ke kode untuk lebih memahami bagian-bagian individu. Namun, sebelum kita menerapkan metode lainnya, mari kita istirahat sejenak dan mendiskusikan beberapa kode yang mungkin terlihat membingungkan pada awalnya. Kami menggunakan kelas induk BaseEstimator dan ClassifierMixin untuk mendapatkan beberapa fungsionalitas dasar secara gratis, termasuk metode `get_params` dan `set_params` untuk mengatur dan mengembalikan parameter pengklasifikasi serta metode `skor` untuk menghitung akurasi prediksi, masing-masing. Juga, perhatikan bahwa kami mengimpor enam untuk membuat `MajorityVoteClassifier` kompatibel dengan Python 2.7.

Selanjutnya, kita akan menambahkan metode prediksi untuk memprediksi label kelas melalui suara mayoritas berdasarkan label kelas jika kita menginisialisasi objek `MajorityVoteClassifier` baru dengan `vote='classlabel'`. Atau, kita akan dapat menginisialisasi classifier ensemble dengan `vote='probability'` untuk memprediksi label kelas berdasarkan probabilitas keanggotaan kelas. Selanjutnya, kami juga akan menambahkan metode `predict_proba` untuk mengembalikan probabilitas rata-rata, yang berguna untuk menghitung area Karakteristik Operator Penerima di bawah kurva (ROC AUC).

```

def pre(s, X):
    """ Pre class labels for X.
    Parameters
    X : {arr-like, spar mat},
    Sh = [n_samples, n_features]
    Matrix of training samples.
    Returns
    ma_v : arr-like, sh = [n_samples]
    Predicted class labels.
    """
    if se.v == 'probability':
        ma_v = np.argmax(spredict_prob(X),
                          axis=1)
    else: # 'cl_label' v
        predictions = np.asarr([cl.predict(X)
                               for cl in
                               s.cl_]).T
        ma_v = np.ap_al_ax(
            lambda x:
            np.argmax(np.bincount(x, weights=s.w)),
            axis=1,
            arr=predictions)
        ma_v = s.l_.inverse_transform(ma_v)
    return ma_v
def predict_proba(self, X):
    """ Prediction for X.
    Parameters
    X : {arr-like, sp mat},
    sh = [n_samples, n_features]

```

Vektor pelatihan, di mana n_samples adalah jumlah sampel dan n_features adalah jumlah fitur.

Returns

```

av_prob : array-like,
sh = [n_samples, n_classes]

```

Probabilitas rata-rata tertimbang untuk setiap kelas per sampel.

```

"""
probs = np.asarr([cl.predict_prob(X)
                  for cl in s.cl_])
av_prob = np.av(probs,
                axis=0, weights=s.w)

```

```

    return av_prob
def get_ps(self, deep=True):
    """ Get classifier parameter names for GridSearch"""
    if not deep:
        return super(MVC,
                     self).get_ps(deep=False)
    else:
        ou = s.n_cl.copy()
        for n, step in \
            six.iteritems(s.n_cl):
            for k, value in six.iteritems(
                step.get_ps(deep=True)):
                ou['%s      %s' % (n, k)] = value
        return ou

```

4.4 PENGGABUNGAN ALGORITMA BERBEDA UNTUK KLASIFIKASI SUARA MAYORITAS

Sekarang, sudah waktunya untuk menerapkan MVC yang kita implementasikan di bagian sebelumnya ke dalam tindakan. Anda harus terlebih dahulu menyiapkan kumpulan data yang dapat Anda uji. Karena kita sudah terbiasa dengan teknik untuk memuat kumpulan data dari file CSV, kita akan mengambil jalan pintas dan memuat kumpulan data Iris dari modul kumpulan data scikit-learn.

Selanjutnya, kami hanya akan memilih dua fitur, lebar sepal dan panjang kelopak, untuk membuat tugas klasifikasi lebih menantang. Meskipun MajorityVoteClassifier, atau MVC kami, digeneralisasikan ke masalah multikelas, kami hanya akan mengklasifikasikan sampel bunga dari dua kelas, Ir-Versicolor dan Ir-Virginica, untuk menghitung ROC AUC. Kodennya adalah sebagai berikut:

```

>>> import sklearn as sk
>>> import sklearn.cross_validation as cv
>>> ir = datasets.load_ir()
>>> X, y = ir.data[50:, [1, 2]], ir.target[50:]
>>> le = LabelEncoder()
>>> y = le.fit_transform(y)

```

Selanjutnya kami membagi sampel Iris menjadi 50 persen pelatihan dan 50 persen data uji:

```

>>> X_train, X_test, y_train, y_test =\
... train_test_split(X, y,
... test_size=0.5,
... random_state=1)

```

Dengan menggunakan dataset pelatihan, sekarang kita akan melatih tiga pengklasifikasi yang berbeda — pengklasifikasi regresi logistik, pengklasifikasi pohon keputusan, dan pengklasifikasi k-nearest neighbor — dan melihat kinerja masing-masing melalui validasi

silang 10 pada dataset pelatihan sebelum kita menggabungkan mereka menjadi satu ansambel:

import the following

```

sklearn.cross_validation
sklearn.linear_model
sklearn.tree
sklearn.pipeline
Pipeline
numpy as np
>>> clf1 = LogisticRegression(penalty='l2',
... C=0.001,
... random_state=0)
>>> clf2 = DTCl(max_depth=1,
... criterion='entropy',
... random_state=0)

>>> cl = KNC(n_nb=1,
... p=2,
... met='minsk')
>>> pipe1 = Pipeline([('sc', StandardScaler()),
... ['clf', clf1]])
>>> pipe3 = Pipeline([('sc', StandardScaler()),
... ['clf', clf3]])
>>> clf_labels = ['Logistic Regression', 'Decision Tree', 'KNN']
>>> print('10-fold cross validation:\n')
>>> for clf, label in zip([pipe1, clf2, pipe3], clf_labels):
... sc = crossVSc(estimator=clf,
... X=X_train,
... y=y_train,
... cv=10,
... scoring='roc_auc')
... print("ROC AUC: %0.2f (+/- %0.2f) [%s]"
... % (scores.mean(), scores.std(), label))

```

Output yang kami terima, seperti yang ditunjukkan dalam cuplikan berikut, menunjukkan bahwa kinerja prediktif dari pengklasifikasi individu hampir sama:

10- lipat validasi silang:

```

ROC AUC: 0.92 (+/- 0.20) [Logistic Regression]
ROC AUC: 0.92 (+/- 0.15) [Decision Tree]
ROC AUC: 0.93 (+/- 0.10) [KNN]

```

Anda mungkin bertanya-tanya mengapa kami melatih regresi logistik dan pengklasifikasi k-nearest neighbor sebagai bagian dari pipeline. Penyebabnya di sini adalah, seperti yang kami katakan, regresi logistik dan algoritma k-nearest neighbor (menggunakan metrik jarak Euclidean) tidak invariant skala dibandingkan dengan pohon keputusan. Namun, keuntungan Ir semuanya diukur pada skala yang sama; itu adalah kebiasaan yang baik untuk bekerja dengan fitur standar.

Sekarang, mari kita beralih ke bagian yang lebih menarik dan menggabungkan pengklasifikasi individu untuk pemungutan suara aturan mayoritas di M_V_C kami:

```
>>> mv_cl = M_V_C()
... cl=[pipe1, clf2, pipe3])
>>> cl_labels += ['Majority Voting']
>>> all_cl = [pipe1, clf2, pipe3, mv_cl]
>>> for cl, label in zip(all_clf, clf_labels):
... sc = cross_val_score(est=cl,
... X=X_train,
... y=y_train,
... cv=10,
... scoring='roc_auc')
... % (scores.mean(), scores.std(), label))
R_AUC: 0.92 (+/- 0.20) [Logistic Regression]
R_AUC: 0.92 (+/- 0.15) [D_T]
R_AUC: 0.93 (+/- 0.10) [KNN]
R_AUC: 0.97 (+/- 0.10) [Majority Voting]
```

Selain itu, output dari MajorityVotingClassifier telah meningkat secara substansial dibandingkan pengklasifikasi individu dalam evaluasi validasi silang 10 kali lipat.

Penggolong

Di bagian ini, Anda akan menghitung kurva R_C dari set pengujian untuk memeriksa apakah MV_Classifier menggeneralisasi dengan baik ke data yang tidak terlihat. Kita harus ingat bahwa test set tidak akan digunakan untuk pemilihan model; satu-satunya tujuan adalah untuk melaporkan perkiraan keakuratan sistem classifier. Mari kita lihat metrik Impor.

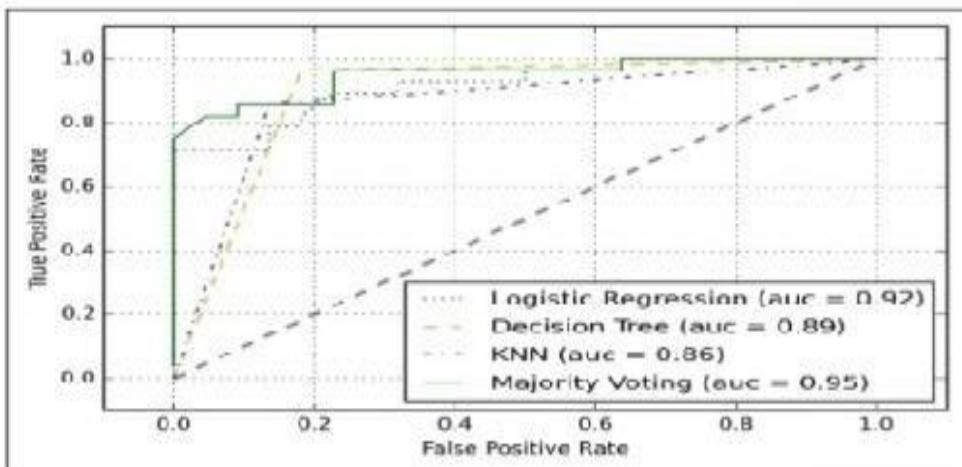
```
import roc_curve from sklearn.metrics import auc
cls = ['black', 'orange', 'blue', 'green']
ls = [':', '--', '-.', '-']
for cl, label, cl, l \
... in zip(all_clf, clf_labels, cls, ls):
... y_pred = clf.fit(X_train,
... y_train).predict_proba(X_test)[:, 1]
... fpr, tpr, thresholds = rc_curve(y_t=y_tes,
... y_sc=y_pr)
... rc_auc = ac(x=fpr, y=tpr)
... plt.plot(fpr, tpr,
```

```

... color=clr,
... linestyle=ls,
... la='%s (ac = %0.2f)' % (la, rc_ac))
>>> plt.lg(lc='lower right')
>>> plt.plot([0, 1], [0, 1],
... linestyle='--',
... color='gray',
... linewidth=2)
>>> plt.xlim([-0.1, 1.1])
>>> plt.ylim([-0.1, 1.1])
>>> plt.grid()
>>> plt.xlb('False Positive Rate')
>>> plt.ylb('True Positive Rate')
>>> plt.show()

```

Seperti yang dapat kita lihat pada ROC yang dihasilkan, classifier ensemble juga bekerja dengan baik pada set pengujian (ROC AUC = 0,95), sedangkan classifier tetangga k-terdekat tampaknya terlalu pas dengan data pelatihan (pelatihan ROC AUC = 0,93, pengujian ROC AUC = 0,86):



Gambar 4.4 Wilayah keputusan dari classifier ensemble

Anda hanya memilih dua fitur untuk tugas klasifikasi. Akan menarik untuk menunjukkan seperti apa sebenarnya wilayah keputusan dari classifier ensemble itu.

Meskipun tidak perlu menstandardisasi fitur pelatihan sebelum model agar sesuai karena regresi logistik dan pipa k-nearest tetangga kami akan secara otomatis menangani ini, Anda akan membuat set pelatihan sehingga wilayah keputusan dari pohon keputusan akan berada di skala yang sama untuk tujuan visual.

Mari lihat:

```

>>> sc = SS()
X_tra_std = sc.fit_transform(X_train)

```

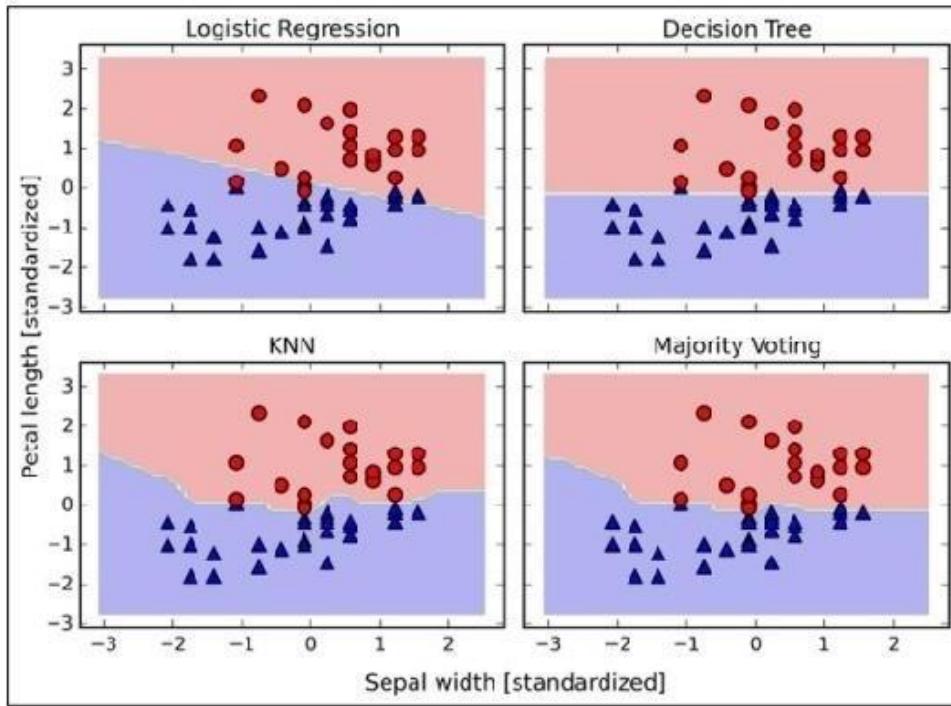
From itertools import product

```

x_mi= X_tra_std[:, 0].mi() - 1
x_ma = X_tra_std[:, 0].ma() + 1
y_mi = X_tra_std[:, 1].mi() - 1
y_ma = X_tra_std[:, 1].ma() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
... np.arange(y_mi, y_ma, 0.1))
f, axarr = plt.subplots(nrows=2, ncols=2,
sharex='col',
sharey='row',
figze=(7, 5))
for ix, cl, tt in zip(product([0, 1], [0, 1]),
all_cl, cl_lb):
... cl.fit(X_tra_std, y_tra)
... Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
... Z = Z.reshape(xx.shape)
... axarr[idx[0], idx[1]].contou(_xx, _yy, Z, alph=0.3)
... axarr[idx[0], idx[1]].scatter(X_tra_std[y_tra==0, 0],
... X_tra_std[y_tra==0, 1],
... c='blue',
... mark='^',
... s=50)
... axarr[idx[0], idx[1]].scatt(X_tra_std[y_tra==1, 0],
... X_tra_std[y_tra==1, 1],
... c='red',
... marker='o',
... s=50)
... axarr[idx[0], idx[1]].set_title(tt)
>>> plt.text(-3.5, -4.5,
... z='Sl wid [standardized]',
... ha='center', va='center', ftsize=12)
>>> plt.text(-10.5, 4.5,
... z='P_length [standardized]',
... ha='center', va='center',
... f_size=13, rotation=90)
>>> plt.show()

```

Menariknya, tetapi juga seperti yang diharapkan, wilayah keputusan dari pengklasifikasi ansambel tampaknya merupakan hibrida dari wilayah keputusan dari pengklasifikasi individu. Sepintas, batas keputusan suara mayoritas sangat mirip dengan batas keputusan pengklasifikasi k-nearest neighbor. Namun, kita dapat melihat bahwa itu ortogonal terhadap sumbu y untuk lebar sepal 1, seperti tampilan pohon keputusan:



Gambar 4.5 Wilayah keputusan pengklasifikasian ansambel merupakan hibrida dari keputusan pengklasifikasian individu

Sebelum Anda mempelajari cara menyetel parameter classifier individual untuk klasifikasi ensemble, mari panggil metode `get_ps` untuk menemukan ide penting tentang bagaimana kita dapat mengakses parameter individual di dalam objek GridSearch:

```
>>> mv_clf.get_params()
{'decisiontreeclassifier':DecisionTreeClassifier(class_weight=None,
criterion='entropy', max_depth=1,
max_features=None, max_leaf_nodes=None, min_samples_
leaf=1,
min_samples_split=2,min_weight_fraction_leaf=0.0,
random_state=0, splitter='best'),
'decisiontreeclassifier class_weight': None,
'decisiontreeclassifier criterion': 'entropy',
[...]
'decisiontreeclassifier random_state': 0,
'decisiontreeclassifier splitter': 'best',
'pipeline-1': Pipeline(steps=[('sc', StandardScaler(copy=True, with_
mean=True, with_std=True)), ('clf', LogisticRegression(C=0.001, class_
weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, max_iter=100, multi_class='ovr',
penalty='l2', random_state=0, solver='liblinear',
tol=0.0001,
verbose=0))]),
'pipeline-1_clf': LogisticRegression(C=0.001, class_weight=None,
```

```

dual=False, fit_intercept=True,
intercept_scaling=1, max_iter=100, multi_class='ovr',
penalty='l2', random_state=0, solver='liblinear', tol=0.0001,
verbose=0),
'pipeline-1_clf_C': 0.001,
'pipeline-1_clf_class_weight': None, 'pipeline-1mclf dual': False,
[...]
'pipeline-1_sc_with_std': True,
'pipeline-2': Pipeline(steps=[('sc', StandardScaler(copy=True, with_
mean=True, with_std=True)), ('clf', KNeighborsClassifier(algorithm='au
to', leaf_size=30, metric='minkowski',
metric_params=None, n_neighbors=1, p=2, w='uniform'))]),
'p-2_cl': KNC(algorithm='auto', leaf_size=30, met='miski',
met_ps=None, n_neighbors=1, p=2, w='uniform'),
'p-2_cl algorithm': 'auto',
[...]
'p-2_sc with_std': T}

```

Bergantung pada nilai yang dikembalikan oleh metode get_ps, Anda sekarang tahu cara mengakses atribut pengklasifikasi individu. Mari kita bekerja dengan parameter regularisasi terbalik C dari pengklasifikasi regresi logistik dan kedalaman pohon keputusan melalui pencarian grid untuk tujuan demonstrasi. Mari kita lihat:

```

>>> from sklearn.grid_search import GdSearchCV
>>> params = {'dtreecl_max_depth': [0.1, .02],
... 'p-1 clf C': [0.001, 0.1, 100.0]}
>>> gd = GdSearchCV(estimator=mv_cl,
... param_grid=params,
... cv=10,
... scoring='roc_auc')
>>> gd.fit(X_tra, y_tra)

```

Setelah pencarian grid selesai, kami dapat mencetak kombinasi nilai parameter hiper yang berbeda dan skor R_C AC rata-rata yang dihitung melalui validasi silang 10 kali lipat. Kodennya adalah sebagai berikut:

```

>>> for params, mean_sc, scores in grid.grid_sc_:
... print("%0.3f+/-%0.2f %r"
... % (mean_sc, sc.std() / 2, params))
0.967+/-0.05 {'p-1_cl_C': 0.001, 'dtreeclassifier_ma_depth': 1}
0.967+/-0.05 {'p-1_cl_C': 0.1, 'dtreeclassifier_ma_depth': 1}
1.000+/-0.00 {'p-1_cl_C': 100.0, 'dtreeclassifier_ma_depth': 1}
0.967+/-0.05 {'p-1_cl_C': 0.001, 'dtreeclassifier_ma_depth': 2}
0.967+/-0.05 {'p-1_cl_C': 0.1, 'dtreeclassifier_ma_depth': 2}

```

```
1.000+-0.00 {'p-1_cl_C': 100.0, 'dtreeclassifier_ma_depth': 2}
>>> print('Best parameters: %s' % gd.best_ps_)
Best parameters: {'p1_cl_C': 100.0,
'dtreeclassifier_ma_depth': 1}
>>> print('Accuracy: %.2f' % gd.best_sc_)
Accuracy: 1.00
```

Pertanyaan

1. Jelaskan bagaimana menggabungkan model yang berbeda secara rinci.
2. Apa tujuan dan manfaat dari menggabungkan model?

DAFTAR PUSTAKA

Fahriza Azwar Muhammad, Rizky Arif Windiator,Yuridi Bintang Pratama, "pemrograman socket untuk koneksi Abtara Raspberry Pi dengan Referee Box",Universitas Islam Indonesia, 2016.

<http://scikit-learn.org/stable/install.html>

<https://www.python.org>

<https://matplotlib.org/2.1.0/users/installing.html>

<http://yann.lecun.com/exdb/mnist/>

Kadir, A. 2018. Dasar Logika Pemrograman Komputer. Cetakan Kedua. Elexmedia Komputindo.

Panduan KRSBI beroda 2017,

<http://kontesrobotindonesia.org/datakontes/2017/PanduanKRSBIBeroda2017.pdf>, diakses pada 12 Desember, 2020.

Yuliza, IncomTech, Jurnal Telekomunikasi dan Komputer, vol.4, no.1,2013.

Algoritma Machine Learning Dengan Python

Dr. Joseph Teguh Santoso, S.Kom, M.Kom

BIODATA PENULIS



Dr. Joseph Teguh Santoso, S.Kom, M.Kom adalah Rektor dari Universitas Sains & Teknologi Komputer (Universitas STEKOM) Semarang yang memiliki banyak pengalaman praktis dalam bidang e-commerce sejak Tahun 2002. Beliau mempunyai 3 (tiga) toko *Official Online Store* di China untuk merek Sepeda Raleigh, dengan omzet tahunan pada Tahun 2019 mencapai lebih dari Rp. 35 Miliar rupiah dan terus meningkat. Dr. Joseph T.S memiliki lisensi tunggal sepeda merek “Raleigh” untuk penjualan *Online* di seluruh China. Di samping itu beliau juga memiliki pabrik sepeda dan sepeda listrik merek “Fengjiu”, yaitu Pabrik Sepeda Listrik yang masih tergolong kecil di China. Pengalaman beliau malang melintang di dunia *online store* di China seperti Alibaba, Tmall, Taobao, JD, Aliexpress sangat membantu mahasiswa untuk memiliki pengalaman teknis dan praktis untuk membuka toko *online* bersama beliau.



YAYASAN PRIMA AGUS TEKNIK

PENERBIT :
YAYASAN PRIMA AGUS TEKNIK
Jl. Majapahit No. 605 Semarang
Telp. (024) 6723456. Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

ISBN 978-623-5734-28-6 (PDF)



9 786235 734286