Penerapan Algoritma Machine Learning dengan Python

Dr. (c) Migunani, S.Kom, M.Kom.

Penerapan Algoritma Machine Learning dengan Python

Dr. (c) Migunani, S.Kom, M.Kom.

Penulis yang Lahir pada tahun 1976 ini memang memiliki minat yang tentang IT sejak lama. Penulis juga sudah banyak melakukan Seminar dan Workshop tentang perkembangan Ilmu Komputer. Penulis kelahiran semarang yang bergelar Magister ini sekarang sedang melakukan studinya untuk menyelesaikan pendidikan Kandidat Doktor Ilmu Komputer UKSW. Selain mengajar di Universitas Sains Dan Teknologi Komputer Fakultas Studi Akademik jurusan Teknik Informatika ini, kegiatan penulis memberkan pelatihan atau workshop dan seminar-seminar dibidang teknologi informasi. Beberapa buku yang sudah penulis buat digunakan untuk bahan ajar pengajar di Universitas Sains
Dan Teknologi Komputer. Penelitian pertama penulis tahun 2002 tentang Jurnal Teknik Informasi Dinamik sampai saat ini terhitung hingga puluhan Publikasi ilmiah yang sudah dikerjakan penulis. Penulis lulusan Magister Universitas Diponegoro ini mendapatkan penghargaan sebagai mahasiswa Cumlaude dan Terbaik tahun 2011 serta mendapatan pernghargaan The Best Of Fifth Paper Award ditahun yang sama.
Penulis juga menjadi beberapa narasumber dalam seminar nasional maupun internasional, seperti Visiting Lecturer from STEKOM University with Oles Honchar Dnipro National University (Ukraine) menjadi pembicara utama dalam seminar Internasional, dan mejadi narasumber webinar nasional "Peran Machine Learning Dalam Transformasi Digital, Teori Dan Penerapannya" dan masih banyak Seminar lagi yang penulis lakukan. Banyak Organisasi profesi ilmiah yang diikuti penulis, tahun 2016 Asosiasi Perguruan Tinggi Komputer (APTIKOM) sampai Sekarang, tahun 2017 hingga sekarang Ikatan Dosen Republik Indonesia, tahun 2010 Paguyuban Wakil Ketua / Rektor III, dan yang paling baru merupakan salah satu anggota Perkumpulan Dosen Perguruan Tinggi Nusantara (Pdptn) Provinsi Jawa Tengahpada tahun 2020 hingga sekarang. Cita-cita penulis adalah memajukan dunia IT melalui menulis dan mengajar. Motto hidup penulis “Hidup akan lebih berarti saat termotivasi, memiliki arah dan tujuan, serta mengejarnya”

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

Penerapan Algoritma Machine Learning dengan Python

Penulis :
Dr. (c) Migunani, S.Kom, M.Kom.

ISBN :

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

Penyunting :
Dr. Agus Wibowo, M.Kom, M.Si, MM.

Desain Sampul dan Tata Letak :
Irdha Yunianto, S.Ds., M.Kom

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

Anggota IKAPI No: 279 / ALB / JTE / 2023

Redaksi :
Jl. Majapahit no 605 Semarang
Telp. 08122925000
Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

Distributor Tunggal :
Universitas STEKOM
Jl. Majapahit no 605 Semarang
Telp. 08122925000
Fax. 024-6710144
Email : info@stekom.ac.id

KATA PENGANTAR

Puji syukur kita panjatkan ke hadirat Tuhan Yang Maha Esa, atas rahmat dan karunia-Nya, sehingga buku yang berjudul "***Penerapan Algoritma Machine Learning dengan Python***" ini dapat diselesaikan dengan baik. Buku ini disusun sebagai upaya untuk memberikan wawasan dan pemahaman lebih dalam mengenai dunia Machine Learning, sebuah bidang yang terus berkembang pesat dan memiliki peran yang sangat penting dalam kemajuan teknologi dan inovasi di berbagai sektor, mulai dari industri, kesehatan, hingga kehidupan sehari-hari.

Dalam era digital yang terus berkembang pesat ini, kemajuan teknologi memberikan dampak signifikan terhadap berbagai bidang, termasuk dalam dunia ilmu komputer. Salah satu cabang yang berkembang pesat dan mendapatkan perhatian besar adalah *Machine Learning* (Pembelajaran Mesin). Machine Learning merupakan salah satu cabang dari kecerdasan buatan yang memungkinkan komputer untuk belajar dan membuat keputusan atau prediksi berdasarkan data tanpa memerlukan program yang eksplisit. Teknologi ini memiliki potensi luar biasa untuk mengubah cara kita memproses informasi dan memecahkan masalah, mulai dari aplikasi sederhana hingga yang sangat kompleks.

Machine Learning kini telah merambah berbagai sektor, seperti kesehatan, keuangan, transportasi, dan e-commerce, memberikan solusi untuk tantangan yang semakin kompleks. Dari prediksi cuaca hingga rekomendasi produk dalam aplikasi belanja online, atau bahkan pengenalan wajah dalam perangkat pintar, Machine Learning semakin hadir dalam kehidupan sehari-hari kita. Dengan kemampuan untuk menganalisis data dalam jumlah besar dan menemukan pola tersembunyi, Machine Learning memungkinkan pengambilan keputusan yang lebih cepat dan lebih akurat.

Buku ini disusun untuk memberikan pemahaman dasar mengenai konsep, teknik, dan aplikasi Machine Learning, serta berbagai metode yang digunakan dalam proses pengembangan model pembelajaran mesin. Diharapkan pembaca dapat memperoleh wawasan yang lebih dalam mengenai cara kerja Machine Learning, serta bagaimana teknologi ini dapat diterapkan dalam berbagai kasus dunia nyata.

Kami berharap buku ini dapat menjadi panduan yang bermanfaat bagi siapa saja yang ingin memulai perjalanan mereka dalam dunia Machine Learning, baik bagi para pemula yang baru mengenal teknologi ini, maupun bagi para profesional yang ingin memperdalam pengetahuan mereka. Seiring dengan terus berkembangnya bidang ini, penting bagi kita untuk terus belajar dan beradaptasi dengan inovasi-inovasi terbaru di dalamnya. Semoga buku ini dapat memberikan kontribusi positif dalam pengembangan ilmu pengetahuan dan teknologi, khususnya dalam bidang Machine Learning. Terima Kasih.

Semarang, Desember 2024
Penulis

Dr. (c) Migunani, S.Kom, M.Kom

DAFTAR ISI

BAB 1
PENGANTAR PEMBELAJARAN MESIN

1.1 TEORI

Jika saya bertanya tentang "Pembelajaran mesin", Anda mungkin akan membayangkan robot atau sesuatu seperti Terminator. Kenyataannya, pembelajaran mesin tidak hanya digunakan 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 kepada Anda apa itu pembelajaran mesin, dan cara kerjanya.

Apa itu machine learning?

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

Machine learning (ML) memiliki kemampuan untuk memperoleh dan mempelajari data secara mandiri, memungkinkan sistem untuk melaksanakan berbagai tugas yang berbeda-beda berdasarkan apa yang telah dipelajarinya. Istilah machine learning pertama kali diperkenalkan oleh para ilmuwan matematika seperti Adrien Marie Legendre, Thomas Bayes, dan Andrey Markov pada 1920-an, yang menyusun dasar-dasar serta konsep-konsep ML. Sejak saat itu, perkembangan ML terus berlanjut. Salah satu contoh penerapan ML yang terkenal adalah Deep Blue, sebuah sistem yang dikembangkan oleh IBM pada 1996 untuk belajar dan bermain catur. Deep Blue diuji dengan bermain melawan juara catur profesional dan berhasil memenangkan pertandingan tersebut.

Machine learning memainkan peran penting dalam membantu manusia di berbagai bidang dan kini penerapannya semakin mudah dijumpai dalam kehidupan sehari-hari. Contohnya, saat menggunakan fitur face unlock di smartphone atau ketika menjelajah internet dan media sosial, kamu akan sering melihat iklan yang disesuaikan dengan preferensi pribadi. Iklan-iklan ini dihasilkan oleh algoritma ML yang memproses data untuk memberikan rekomendasi yang relevan.

Contoh penerapan ML lainnya juga banyak ditemukan di berbagai aspek kehidupan. Lalu, bagaimana ML dapat belajar? ML belajar dengan menganalisis data yang diberikan pada awal pengembangan dan selama penggunaan sistem. Proses ini bergantung pada teknik atau metode tertentu yang diterapkan selama pengembangan. Berikutnya, mari kita pelajari lebih lanjut tentang berbagai teknik yang digunakan dalam machine learning.

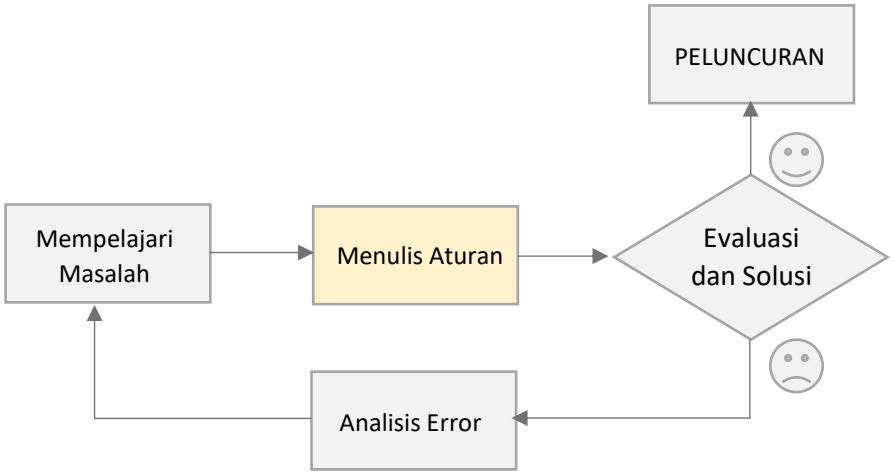
Mengapa pembelajaran mesin?

Misalnya Anda ingin menulis program filter tanpa menggunakan metode pembelajaran mesin. Dalam kasus ini, Anda harus melakukan langkah-langkah berikut:

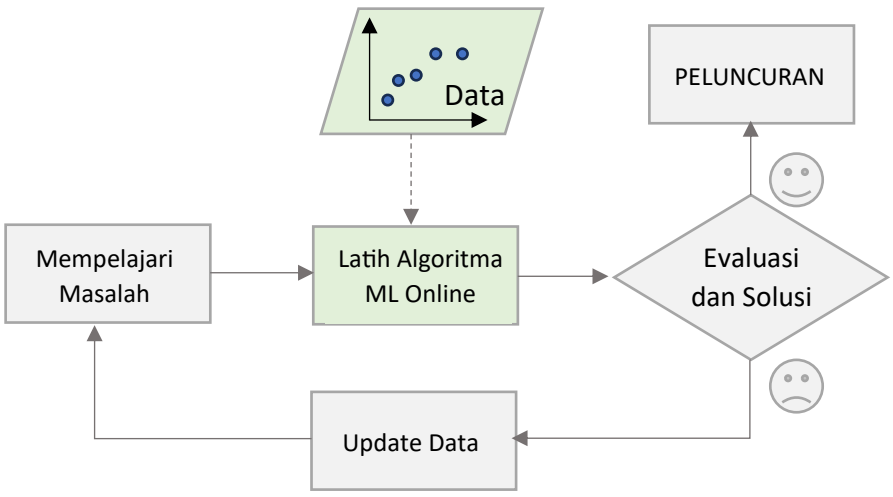
a. Pada awalnya, Anda akan melihat seperti apa tampilan email spam. Anda dapat memilihnya berdasarkan kata atau frasa yang digunakan, seperti "kartu debit",

"gratis", dan seterusnya, dan juga dari pola yang digunakan dalam nama pengirim atau di badan email

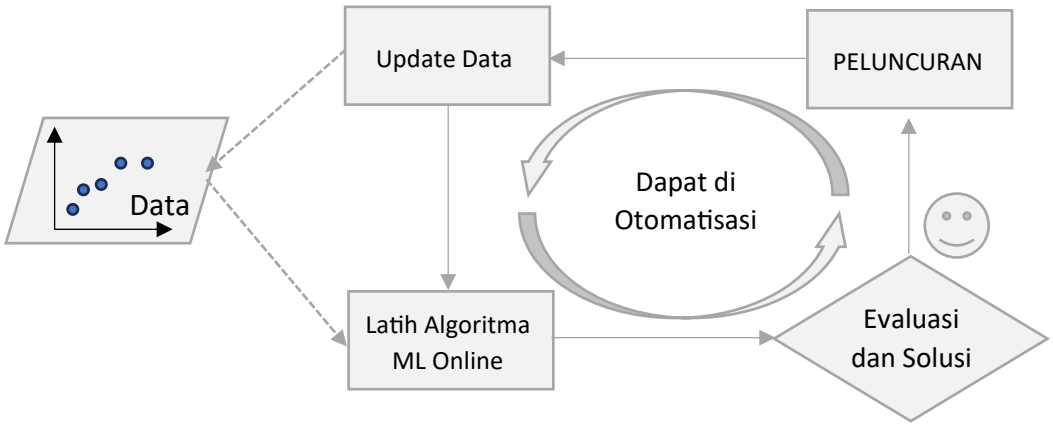
b. Kedua, Anda akan menulis algoritme untuk mendeteksi pola yang telah Anda lihat, lalu perangkat lunak akan menandai email sebagai spam jika sejumlah pola tersebut terdeteksi.
c. Terakhir, Anda akan menguji program, lalu mengulang dua langkah pertama lagi hingga hasilnya cukup baik.



Karena program tersebut bukan perangkat lunak, program tersebut berisi daftar aturan yang sangat panjang yang sulit untuk dipertahankan. Namun, jika Anda mengembangkan perangkat lunak yang sama menggunakan ML, Anda akan dapat mempertahankannya dengan baik.



Selain itu, pengirim email dapat mengubah templat email mereka sehingga kata seperti "4U" sekarang menjadi "untuk Anda," karena email mereka telah ditetapkan sebagai spam. Program yang menggunakan teknik tradisional perlu diperbarui, yang berarti bahwa, 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 menandainya tanpa Anda memberi tahu secara manual.



Selain itu, kita dapat menggunakan pembelajaran mesin untuk memecahkan masalah yang sangat rumit bagi perangkat lunak non-pembelajaran mesin. Misalnya, pengenalan ucapan: saat Anda mengatakan "satu" atau "dua", program tersebut harus dapat membedakan perbedaannya. Jadi, untuk tugas ini, Anda perlu mengembangkan algoritme yang mengukur suara. Pada akhirnya, pembelajaran mesin akan membantu kita belajar, dan algoritme pembelajaran mesin dapat membantu kita melihat apa yang telah kita pelajari.

Kapan sebaiknya Anda menggunakan pembelajaran mesin?

a. Ketika Anda memiliki masalah yang memerlukan banyak daftar aturan yang panjang untuk menemukan solusinya. Dalam kasus ini, teknik pembelajaran mesin dapat menyederhanakan kode Anda dan meningkatkan kinerja.
b. Masalah yang sangat rumit yang tidak dapat diselesaikan dengan pendekatan tradisional.
c. Lingkungan yang tidak stabil: perangkat lunak pembelajaran mesin dapat beradaptasi dengan data baru.

Jenis-jenis Sistem Pembelajaran Mesin

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

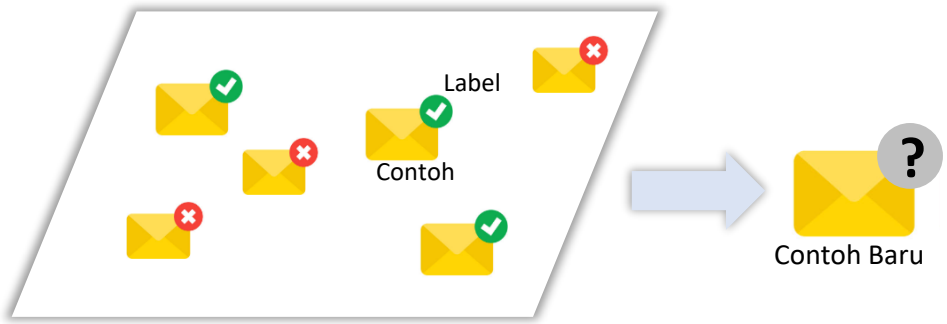
- Sistem tersebut telah dilatih dengan manusia atau tidak
  a. Diawasi
  b. Tidak Diawasi
  c. Semi-diawasi
  d. Pembelajaran Penguatan
- Apakah sistem tersebut dapat belajar secara bertahap
- Apakah sistem tersebut bekerja hanya dengan membandingkan titik-titik data baru untuk menemukan titik-titik data, atau dapat mendeteksi pola-pola baru dalam data, dan kemudian akan membangun sebuah model.

1.2 PEMBELAJARAN TERBIMBING DAN TAK TERBIMBING

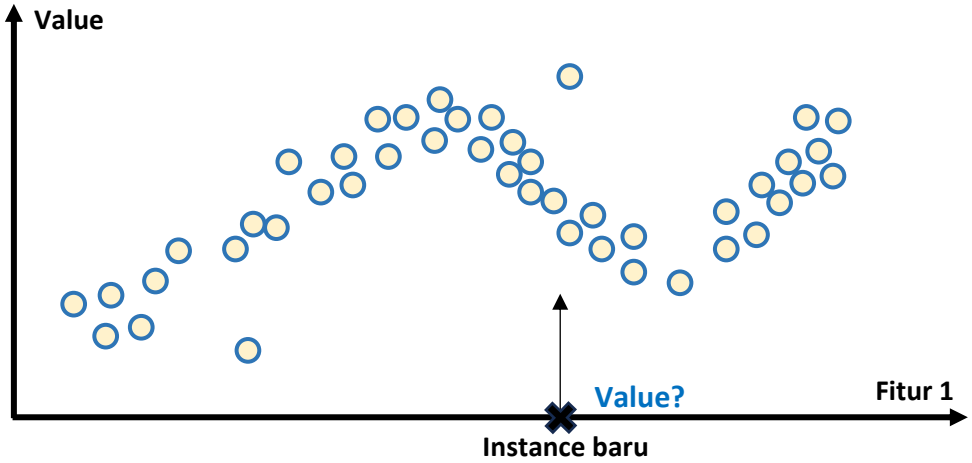
Kita dapat mengklasifikasikan sistem pembelajaran mesin menurut jenis dan jumlah pengawasan manusia selama pelatihan. Anda dapat menemukan empat kategori utama, seperti yang telah kami jelaskan sebelumnya.

Pembelajaran Terbimbing

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



Kelompok pembelajaran terbimbing mengelompokkan 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 memprediksi nilai numerik seperti harga flat, dengan serangkaian fitur (lokasi, jumlah kamar, fasilitas) yang disebut prediktor; jenis tugas ini disebut regresi.



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

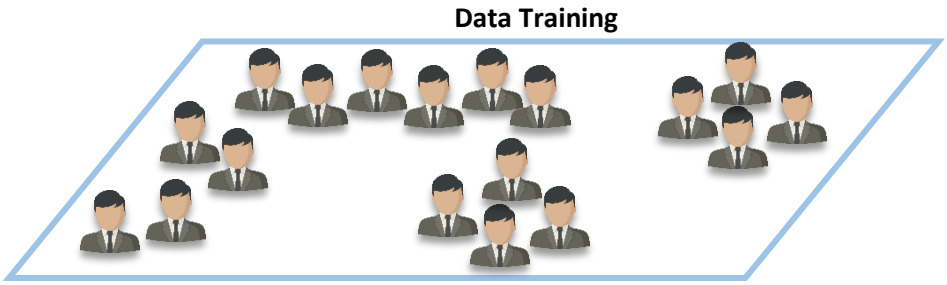
Algoritma terbimbing yang paling penting

a. K-nears neighbours
b. Regresi linier
c. Jaringan neural
d. Support vector machines

e. Regresi logistik
f. Pohon keputusan dan random forest

1.3 PEMBELAJARAN TANPA PENGAWASAN

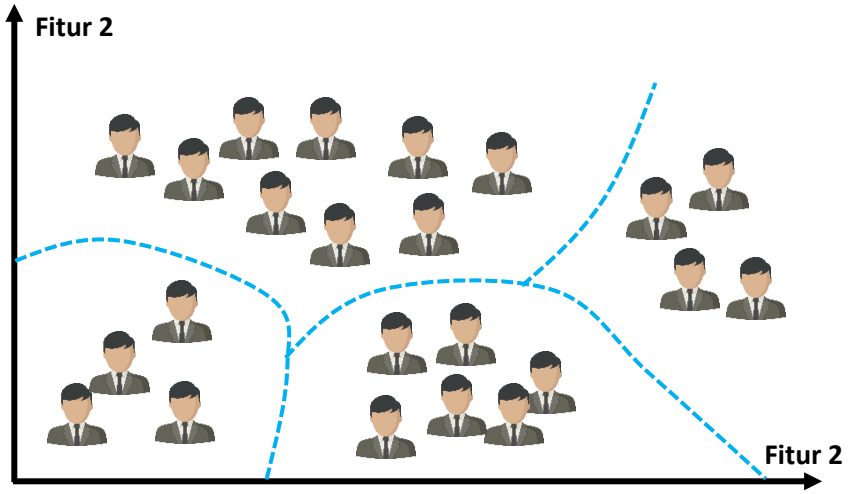
Dalam jenis sistem pembelajaran mesin ini, Anda dapat menebak bahwa data tidak berlabel.



Algoritme tanpa pengawasan yang paling penting

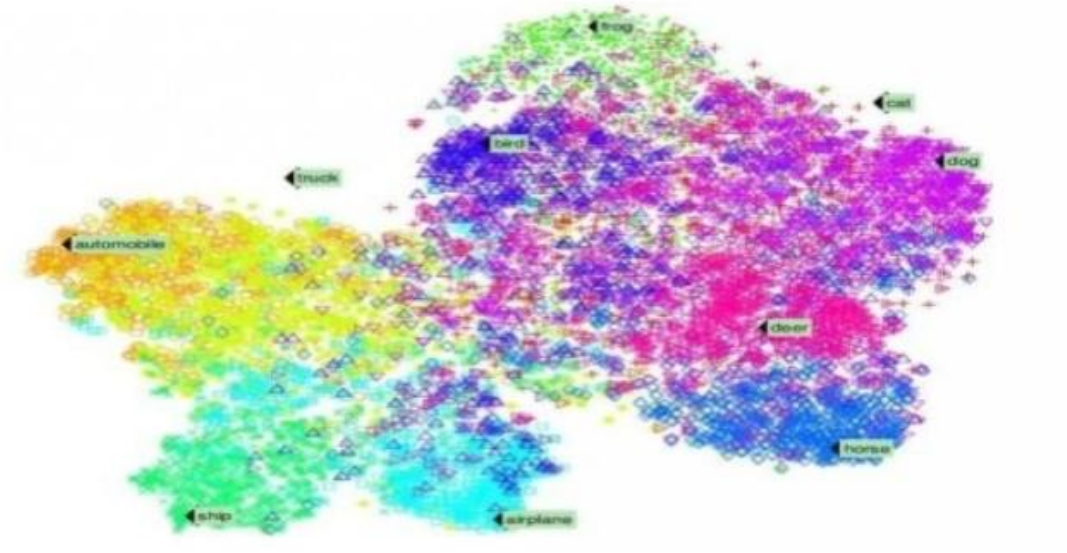
a. Pengelompokan: k-means, analisis kluster hierarkis
b. Pembelajaran aturan asosiasi: Eclat, apriori
c. Visualisasi dan pengurangan dimensionalitas: kernel PCA, t-distributed, PCA

Sebagai contoh, anggaplah Anda memiliki banyak data tentang pengunjung. Dengan menggunakan salah satu algoritme kami untuk mendeteksi grup dengan pengunjung yang serupa. Algoritme tersebut mungkin menemukan bahwa 65% pengunjung Anda adalah laki-laki yang suka menonton film di malam hari, sementara 30% menonton drama di malam hari; dalam kasus ini, dengan menggunakan algoritme pengelompokan, algoritme tersebut akan membagi setiap grup menjadi sub-grup yang lebih kecil.



Ada beberapa algoritma yang sangat penting, seperti algoritma visualisasi; ini adalah algoritma pembelajaran tanpa pengawasan. Anda perlu memberi mereka banyak data dan data yang

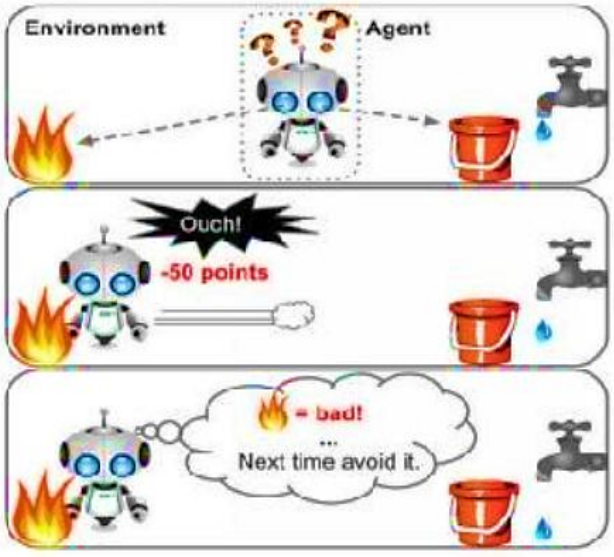
tidak berlabel sebagai input, lalu Anda akan mendapatkan visualisasi 2D atau 3D sebagai output.



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

1.4 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 jenis ini, agen harus belajar sendiri. Hal ini disebut kebijakan.



1. Observasi

2. Pilih tindakan menggunakan kebijakan

3. Aksi

4. Dapatkan Hadiah atau Pinalti

5. Update Kebijakan (Tahap Pembelajaran)

6. Ulangi hingga ditemukan kebijakan yang optimal

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

1.5 PEMBELAJARAN BATCH

Pembelajaran Batch adalah pendekatan dalam pembelajaran mesin di mana model hanya dilatih setelah seluruh dataset yang diperlukan tersedia. Semua data harus dikumpulkan terlebih dahulu sebelum proses pelatihan dimulai. Proses pelatihan ini memakan waktu yang lama karena model memproses seluruh dataset sekaligus, yang bisa sangat memakan banyak sumber daya, terutama jika dataset yang digunakan sangat besar. Selain itu, pembelajaran dilakukan secara offline, artinya tidak ada pembelajaran yang terjadi setelah model di-deploy dan digunakan di dunia nyata. Model hanya akan diperbarui jika dilakukan pelatihan ulang menggunakan data baru. Setelah pelatihan selesai dan model diterapkan, tidak ada pembelajaran lebih lanjut yang terjadi, dan model tidak bisa langsung beradaptasi dengan data baru kecuali dilakukan pelatihan ulang secara menyeluruh.

Pada proses pembelajaran batch, langkah pertama adalah mengumpulkan seluruh data yang diperlukan untuk melatih model. Setelah data terkumpul, model dilatih menggunakan dataset tersebut secara bersamaan. Proses pelatihan ini bertujuan untuk mencari pola-pola dalam data dan membuat prediksi. Setelah model dilatih, tahap selanjutnya adalah evaluasi untuk menguji kinerja model menggunakan data terpisah. Jika model memenuhi kriteria yang ditetapkan, model kemudian diterapkan untuk tugas dunia nyata. Pembaruan model hanya dilakukan jika pelatihan ulang diperlukan dengan data yang baru.

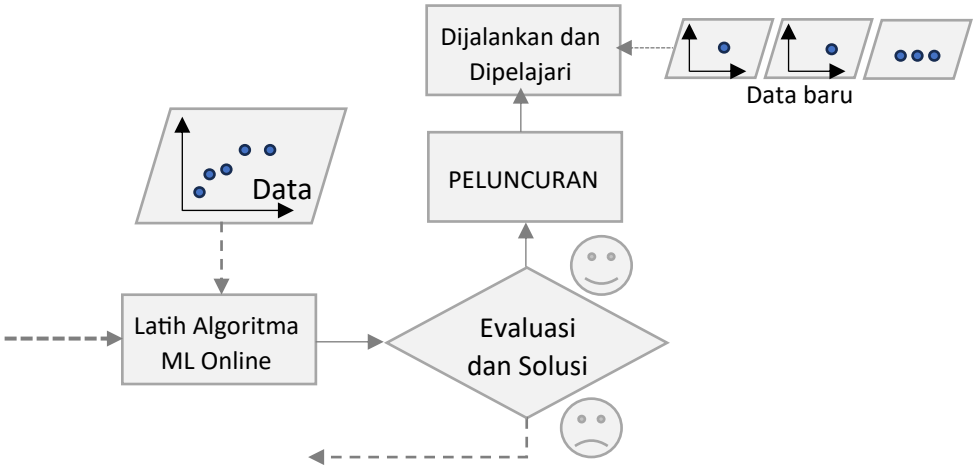
Kelebihan dari pembelajaran batch termasuk efisiensi dalam pengolahan data, di mana semua data diproses sekaligus, dan keakuratan yang tinggi karena model dilatih menggunakan seluruh dataset yang ada. Pembelajaran batch juga sangat cocok untuk aplikasi yang tidak membutuhkan pembaruan data secara real-time. Namun, ada beberapa kekurangan, seperti kebutuhan akan banyak waktu dan sumber daya komputasi, terutama untuk dataset besar. Selain itu, pembelajaran batch kurang fleksibel terhadap perubahan data dan tidak efisien untuk data yang terus berubah, karena model harus dilatih ulang secara menyeluruh setiap kali ada data baru.

Contoh aplikasi pembelajaran batch dapat ditemukan dalam model prediksi penjualan, di mana perusahaan melatih model menggunakan data historis yang lengkap untuk memprediksi penjualan masa depan. Model ini hanya diperbarui jika ada data baru yang cukup besar atau perubahan signifikan. Begitu juga dalam klasifikasi gambar, di mana model dilatih dengan dataset gambar yang lengkap dan hanya diperbarui jika ada dataset gambar baru yang besar yang memerlukan pelatihan ulang.

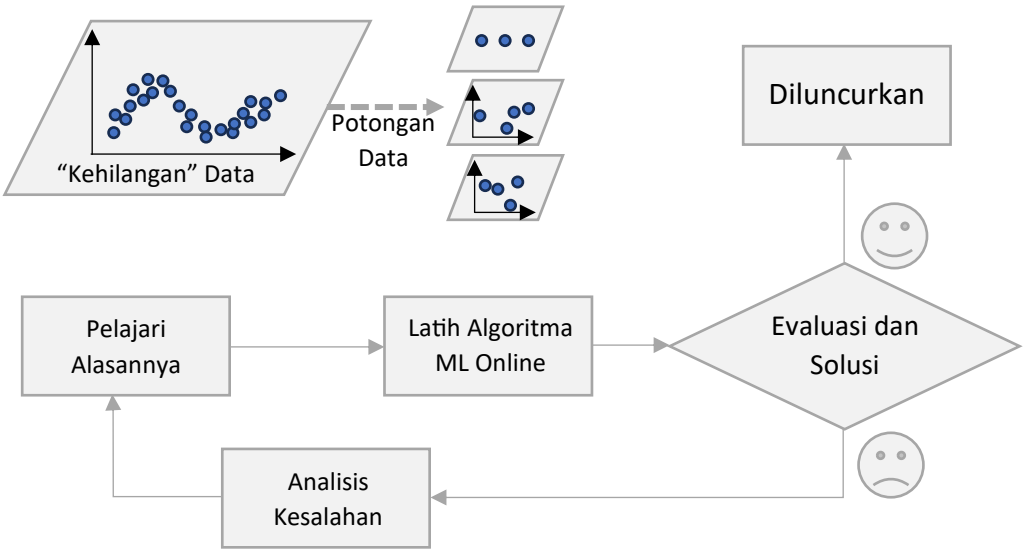
Secara keseluruhan, pembelajaran batch sangat cocok digunakan ketika data yang diperlukan untuk melatih model sudah lengkap dan tidak memerlukan pembaruan secara real-time. Namun, pendekatan ini kurang fleksibel dan memakan banyak sumber daya, sehingga lebih cocok untuk aplikasi yang tidak memerlukan pembelajaran berkelanjutan.

1.6 PEMBELAJARAN DARING

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



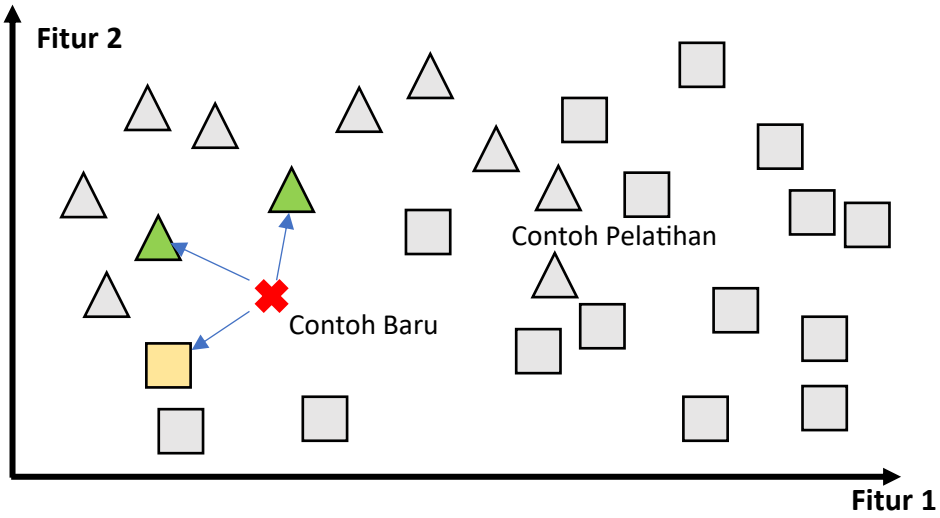
Anda dapat menggunakan jenis sistem ini untuk masalah yang memerlukan aliran data yang berkelanjutan, yang juga perlu beradaptasi dengan cepat terhadap perubahan apa pun. Anda juga dapat menggunakan jenis sistem ini untuk bekerja dengan kumpulan data yang sangat besar,



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

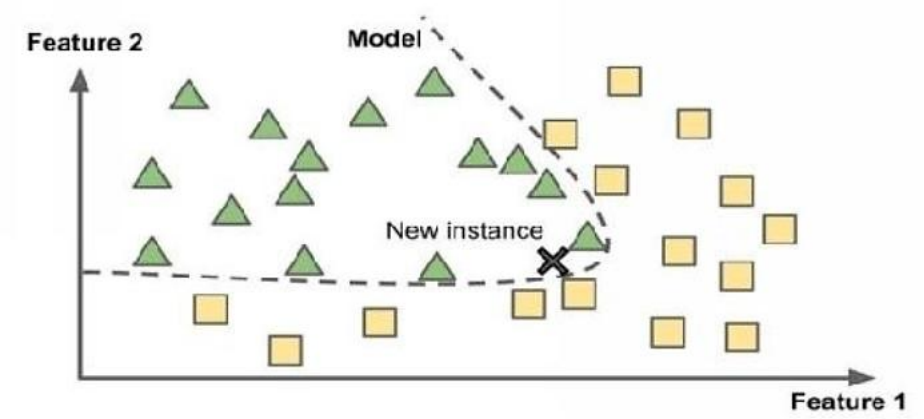
1.7 PEMBELAJARAN BERBASIS INSTANS

Ini adalah jenis pembelajaran paling sederhana yang harus Anda hafalkan. Dengan menggunakan jenis pembelajaran ini dalam program email kami, program akan menandai semua email yang ditandai oleh pengguna.



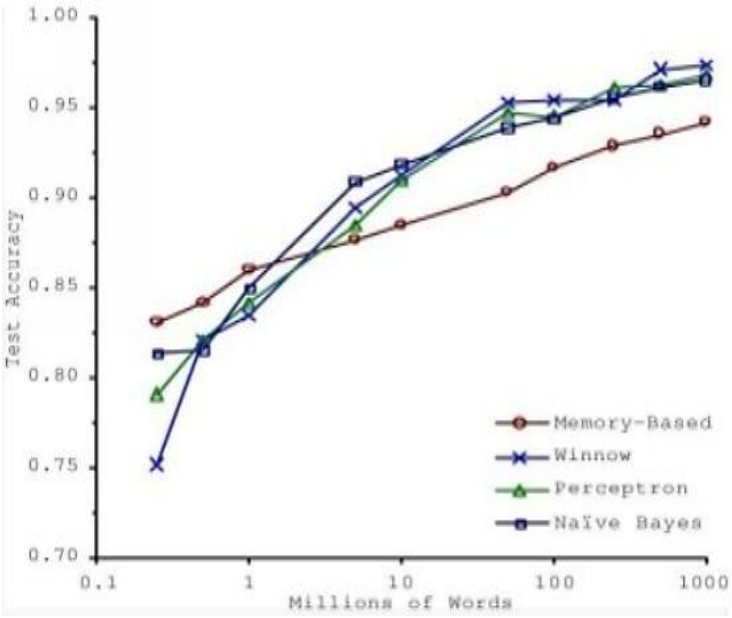
Pembelajaran berbasis model

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



1.8 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 memerlukan banyak data untuk bekerja secara efektif, baik Anda bekerja dengan program dan masalah yang sangat sederhana, atau aplikasi yang rumit seperti pemrosesan gambar dan pengenalan suara. Berikut adalah contoh efektivitas data yang tidak masuk akal, yang menunjukkan proyek MS, yang mencakup data sederhana dan masalah NLP yang rumit.



Data Berkualitas Buruk

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

Fitur yang Tidak Relevan

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

Fitur yang tidak relevan adalah variabel dalam dataset yang tidak memberikan informasi tambahan yang berguna untuk membuat prediksi. Sebagai contoh, dalam model yang memprediksi harga rumah, fitur seperti warna cat rumah atau preferensi pribadi penghuni tidak akan banyak berpengaruh terhadap harga rumah dan bisa dianggap tidak relevan. Fitur-fitur semacam ini justru dapat membuat model lebih rumit dan sulit diinterpretasi tanpa memberikan manfaat yang signifikan. Fitur yang tidak relevan dapat muncul karena beberapa alasan, seperti keterkaitan yang lemah dengan target yang diprediksi, informasi yang redundant (memberikan informasi yang sama dengan fitur lain), atau bahkan data acak (noise) yang tidak berguna untuk prediksi.

Fitur yang tidak relevan dapat memiliki dampak buruk pada model pembelajaran mesin. Salah satunya adalah overfitting, di mana model terlalu mempelajari detail atau pola yang tidak nyata dalam data, sehingga tampil sangat baik pada data pelatihan, namun buruk ketika diuji dengan data baru. Fitur yang tidak relevan juga dapat menambah kompleksitas model, yang memperlambat pelatihan dan menyulitkan model dalam generalisasi. Akibatnya, akurasi prediksi bisa menurun karena gangguan dari fitur yang tidak memiliki hubungan nyata dengan hasil yang diinginkan. Selain itu, penggunaan fitur yang tidak relevan juga merupakan

pemborosan waktu dan sumber daya, karena semakin banyak fitur yang ada, semakin besar pula waktu dan sumber daya komputasi yang diperlukan untuk melatih model.

Oleh karena itu, rekayasa fitur (feature engineering) sangat penting dalam proyek pembelajaran mesin. Tujuan utama dari rekayasa fitur adalah untuk memilih dan mengubah fitur yang relevan agar model dapat mempelajari informasi yang bermanfaat dalam memprediksi target. Beberapa langkah dalam rekayasa fitur antara lain seleksi fitur, yang dapat dilakukan menggunakan teknik statistik, model pembelajaran mesin, atau algoritma seperti *recursive feature elimination (RFE)*. Selain itu, transformasi fitur dilakukan untuk mengubah fitur yang ada menjadi bentuk yang lebih berguna, seperti mengubah fitur kategori menjadi variabel dummy atau mengubah data numerik yang tidak terdistribusi dengan baik. Terkadang, penciptaan fitur baru juga diperlukan, misalnya dengan menggabungkan beberapa fitur atau menggunakan teknik seperti *Principal Component Analysis (PCA)* untuk mereduksi dimensi.

Untuk mengelola fitur yang tidak relevan, beberapa strategi dapat diterapkan, seperti analisis korelasi, yang memungkinkan kita untuk mengevaluasi hubungan antara fitur dengan target yang diprediksi. Fitur yang tidak berkorelasi tinggi dengan target bisa dihapus. Selain itu, teknik regularisasi, seperti *Lasso* atau *Ridge Regression*, dapat digunakan untuk mengurangi pengaruh fitur yang tidak relevan dengan memberikan penalti terhadap koefisien fitur yang lebih kecil. Setelah seleksi fitur dilakukan, penting untuk menguji kinerja model pada data uji untuk memastikan bahwa hanya fitur yang relevan yang digunakan.

Secara keseluruhan, fitur yang tidak relevan dapat merugikan model pembelajaran mesin dengan menambah kompleksitas, mengurangi akurasi, dan meningkatkan risiko *overfitting*. Oleh karena itu, pengembangan dan seleksi fitur yang tepat sangat penting dalam menciptakan model yang efektif. Dengan melakukan rekayasa fitur yang cermat, kita dapat memastikan bahwa model hanya mempelajari informasi yang benar-benar berguna, meningkatkan efisiensi dan akurasi prediksi.

Rekayasa Fitur

Proses rekayasa fitur berjalan seperti ini:

a. Pemilihan fitur: memilih fitur yang paling berguna.
b. Ekstraksi fitur: menggabungkan fitur yang ada untuk menyediakan fitur yang lebih berguna.
c. Pembuatan fitur baru: pembuatan fitur baru, berdasarkan data.

Pengujian

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

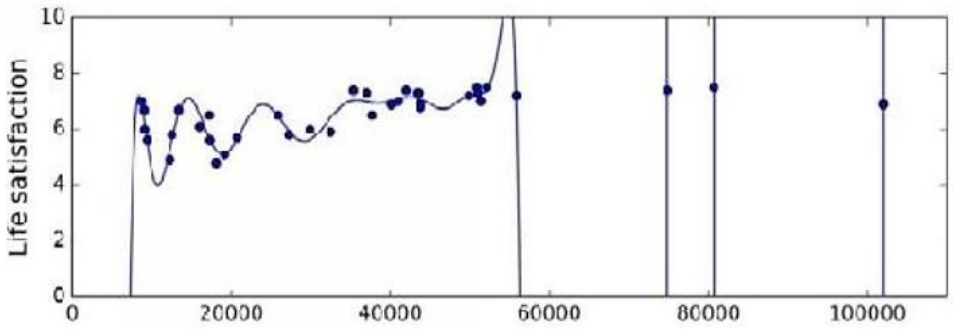
Anda harus membagi data Anda menjadi dua set, satu set untuk pelatihan dan yang kedua untuk pengujian, sehingga Anda dapat melatih model Anda menggunakan yang

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 model tersebut akan berfungsi dengan baik.

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

Overfitting Data

Jika Anda berada di negara asing dan seseorang mencuri sesuatu milik Anda, Anda mungkin mengatakan bahwa semua 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 mereka tidak dapat menggeneralisasinya dengan benar. Misalnya, pada gambar berikut Anda akan menemukan model kepuasan hidup tingkat tinggi yang overfitting data, tetapi bekerja dengan baik dengan data pelatihan.



Kapan hal ini terjadi?

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

Solusi

Untuk mengatasi masalah overfitting, Anda harus melakukan hal berikut:

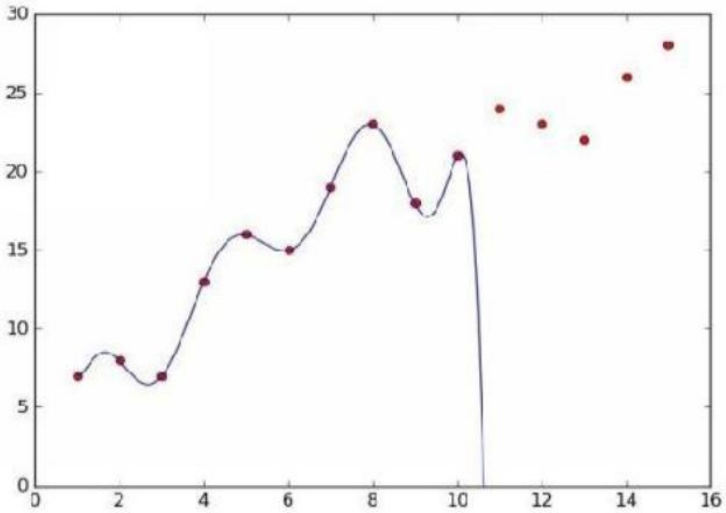
a. Kumpulkan lebih banyak data untuk "data pelatihan"
b. Kurangi tingkat gangguan
c. Pilih salah satu dengan parameter yang lebih sedikit

1.9 UNDERFITTING DATA

Dari namanya, underfitting adalah kebalikan dari overfitting, dan Anda akan mengalami hal ini saat model sangat mudah dipelajari. Misalnya, dengan menggunakan contoh kualitas hidup, kehidupan nyata lebih kompleks daripada model Anda, sehingga prediksi tidak akan menghasilkan hal yang sama, bahkan dalam contoh pelatihan. Dalam kasus

underfitting, model gagal untuk mempelajari informasi yang cukup dari data pelatihan, yang menyebabkan prediksi yang dihasilkan tidak akurat, bahkan pada data pelatihan itu sendiri. Dengan kata lain, model tidak dapat menangkap kompleksitas data dan sering kali menghasilkan prediksi yang buruk. Penyebab underfitting umumnya meliputi penggunaan model yang terlalu sederhana, seperti penerapan algoritma linier untuk data yang bersifat non-linier, serta kurangnya fitur yang relevan untuk melatih model. Faktor lainnya termasuk pengaturan parameter model yang tidak tepat, misalnya nilai regularisasi yang terlalu tinggi, dan data pelatihan yang terbatas atau tidak mencakup variasi yang cukup.

Dampak utama dari underfitting adalah prediksi yang tidak akurat, bahkan pada data pelatihan, yang menunjukkan bahwa model gagal menangkap pola yang relevan. Selain itu, model yang mengalami underfitting tidak dapat melakukan generalisasi dengan baik terhadap data uji, yang membuatnya kurang efektif pada data yang belum pernah dilihat sebelumnya. Salah satu tanda lainnya adalah tingginya kesalahan pada data pelatihan, yang membedakan kondisi ini dari overfitting, di mana kesalahan pada data pelatihan rendah, tetapi tinggi pada data uji. Untuk mengatasi underfitting, beberapa pendekatan dapat dilakukan, seperti menggunakan model yang lebih kompleks untuk menangkap pola data yang lebih rumit, menambahkan fitur yang relevan, menyesuaikan parameter model untuk mengurangi pengaruh regularisasi, atau mengumpulkan lebih banyak data pelatihan yang lebih variatif. Selain itu, meningkatkan proses pelatihan dengan memperpanjang waktu pelatihan atau menambah jumlah iterasi juga dapat membantu. Sebagai contoh, dalam model yang memprediksi kualitas hidup berdasarkan faktor-faktor seperti pendapatan dan pendidikan, jika hanya satu atau dua fitur yang digunakan, model mungkin gagal memahami hubungan yang lebih kompleks antara berbagai faktor tersebut, sehingga menghasilkan prediksi yang kurang akurat. Kesimpulannya, underfitting adalah masalah yang disebabkan oleh model yang terlalu sederhana untuk menangkap kompleksitas data. Untuk mengatasinya, model perlu lebih kompleks, dengan pemilihan fitur yang relevan, penyesuaian parameter yang tepat, serta data pelatihan yang cukup.



Solusi

Untuk memperbaiki masalah ini:

a. Pilih model yang paling kuat, yang memiliki banyak parameter.
b. Masukkan fitur terbaik ke dalam algoritme Anda. Di sini, saya mengacu pada rekayasa fitur.
c. Kurangi kendala pada model Anda.

Latihan Soal

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 terbimbing dan tak terbimbing.
4. Sebutkan tugas-tugas tak terbimbing.
5. Mengapa pengujian dan validasi penting?
6. Dalam satu kalimat, jelaskan apa itu pembelajaran daring.
7. Apa perbedaan antara pembelajaran batch dan luring?
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 bingung. Pembelajaran mesin: ML mengacu pada upaya membuat mesin bekerja lebih baik dalam beberapa tugas, menggunakan data yang diberikan.

a. Pembelajaran mesin hadir dalam berbagai jenis, seperti pembelajaran terbimbing, pembelajaran batch, pembelajaran tanpa pengawasan, dan pembelajaran daring.
b. Untuk menjalankan proyek ML, Anda perlu mengumpulkan data dalam set pelatihan, lalu memasukkan set tersebut ke algoritme pembelajaran untuk mendapatkan keluaran, "prediksi".
c. Jika Anda ingin mendapatkan keluaran 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 MNIST

Dalam bab ini, Anda akan mempelajari lebih dalam sistem klasifikasi, dan bekerja dengan set data MNIST. Ini adalah set yang terdiri dari 70.000 gambar digit yang ditulis tangan oleh siswa dan karyawan. Anda akan menemukan bahwa setiap gambar memiliki label dan digit 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 kodenya:

```
>>> from sklearn.data sets 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]], dataType=uint8),
'tar': array([ 0., 0., 0.,..., 9., 9., 9.])} de
```

a. Deskripsi adalah kunci yang menjelaskan kumpulan data.
b. Kunci data di sini berisi array dengan hanya satu baris misalnya, dan kolom untuk setiap fitur.
c. Kunci target ini berisi array dengan label. Mari kita kerjakan beberapa kode:

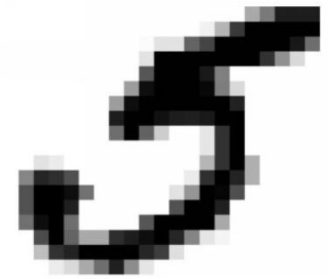
```
>>> X, y = mn["data"], mn["tar"]
>>> X.shape (70000, 784)
>>> y.shape (70000,)
. 7000
```

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, lalu menampilkannya menggunakan fungsi imshow:

```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
yourDigit = X[36000]
Your_image = your_image.reshape(26, 26) plt.imshow(Your_image, cmap = matplotlib.cm.binary, interpolation="nearest")
plt.axis("off") plt.show()
```

Seperti yang dapat Anda lihat pada gambar berikut, bentuknya seperti angka lima, dan kita dapat memberinya label yang memberi tahu kita bahwa angkanya lima.



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

Anda juga harus membuat set pengujian dan membuatnya sebelum data Anda diperiksa.
Set data MNIST dibagi menjadi dua set, satu untuk pelatihan dan satu untuk pengujian.

```
x_tr, x_tes, y_tr, y_te = x [:60000], x[60000:], y[:60000], y[60000:]
```

Mari kita bermain-main dengan set pelatihan Anda sebagai berikut untuk membuat validasi silang menjadi serupa (tanpa ada digit yang hilang)

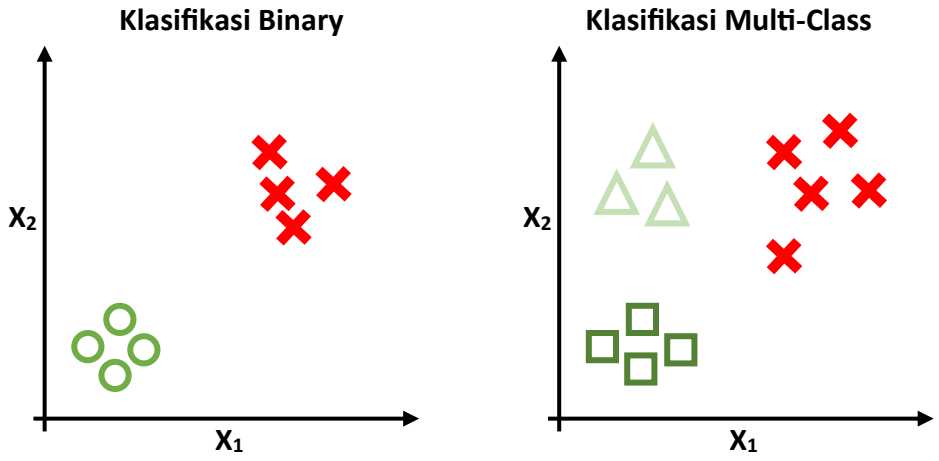
Import numpy as np
myData = np.radom.permutaion(50000) x_tr, y_tr = x_tr[myData], y_tr[myData]

Sekarang saatnya untuk membuatnya cukup sederhana, kita akan mencoba untuk mengidentifikasi satu digit saja, misalnya angka 6. "Detektor 6" ini akan menjadi contoh pengklasifikasi biner, untuk membedakan antara 6 dan bukan 6, jadi kita akan membuat vektor untuk tugas ini:

Y_tr_6 = (y_tr == 6) // ini berarti akan benar untuk angka 6, dan salah untuk angka lainnya
Y_tes_6 = (Y_tes == 6)

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

dari sklearn.linear_model import SGDClassifier mycl = SGDClassifier (random_state = 42)
mycl.fit(x_tr, y_tr_6)
untuk menggunakannya guna mendeteksi 6
>>>mycl.prdict([any_digit)]



2.2 UKURAN KINERJA

Jika Anda ingin mengevaluasi pengklasifikasi, ini akan lebih sulit daripada regresor, jadi mari kita jelaskan cara mengevaluasi pengklasifikasi. Dalam contoh ini, kita akan menggunakan validasi silang untuk mengevaluasi model kita.

dari sklearn.model_selection impor StratifiedKFold
formulir sklearn.base impor klon
sf = StratifiedKFold(n=2, ran_state = 40)
untuk train_index, test_index dalam sf.split(x_tr, y_tr_6):

```
cl = klon(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) cetak(n_correct / len(y_p))
```

Kami menggunakan kelas StratifiedFold untuk melakukan pengambilan sampel bertingkat yang menghasilkan lipatan yang berisi rasio untuk setiap kelas. Selanjutnya, setiap iterasi dalam kode akan membuat klon pengklasifikasi untuk membuat prediksi pada lipatan pengujian. Dan terakhir, ia akan menghitung jumlah prediksi yang benar dan rasionya.

Sekarang kita akan menggunakan fungsi cross_val_score untuk mengevaluasi SGDClassifier dengan validasi silang K-fold. Validasi silang k-fold akan membagi set pelatihan menjadi 3 lipatan, dan kemudian akan membuat prediksi dan evaluasi pada setiap lipatan.
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” pada semua lipatan.

Mari klasifikasikan setiap pengklasifikasi pada setiap gambar tunggal dalam not-6 dari

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

Mari 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% gambar yang merupakan angka 6, jadi kita selalu dapat membayangkan bahwa gambar tersebut bukan angka 6. Kita akan benar sekitar 90% dari waktu. Ingatlah bahwa akurasi bukanlah ukuran kinerja terbaik untuk pengklasifikasi, jika Anda bekerja dengan kumpulan data yang bias.

2.3 MATRIKS KEBINGUNGAN

Ada metode yang lebih baik untuk mengevaluasi kinerja pengklasifikasi Anda: matriks kebingungan. Mudah untuk mengukur kinerja dengan matriks kebingungan, cukup dengan menghitung berapa kali contoh kelas X diklasifikasikan sebagai kelas Y, misalnya. Untuk mendapatkan berapa kali pengklasifikasi gambar 6 dengan 2, Anda harus melihat pada baris ke-6 dan kolom ke-2 matriks kebingungan. Mari kita hitung matriks kebingungan menggunakan fungsi cross_val_predict ().

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

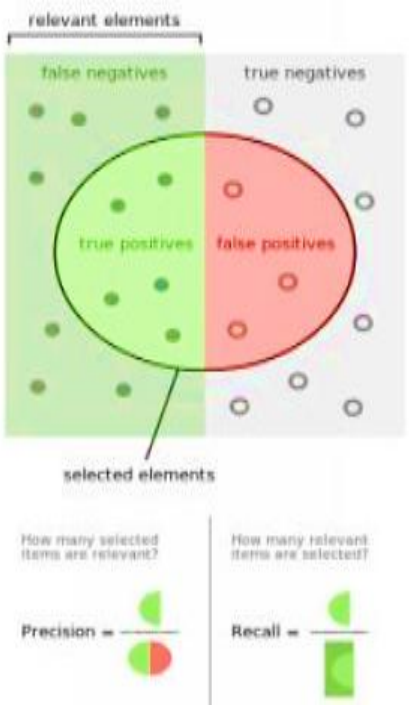
Fungsi ini, seperti fungsi cross_val_score(), melakukan validasi silang k fold, dan juga mengembalikan prediksi pada setiap fold. Ia juga mengembalikan prediksi bersih untuk setiap contoh dalam set pelatihan Anda. Sekarang kita siap untuk mendapatkan matriks menggunakan kode berikut.

dari sklearn.metrics import confused_matrix confused_matrix (y_tr_6, y_tr_pred)
Anda akan mendapatkan array berisi 4 nilai, "angka".
Setiap baris mewakili kelas dalam matriks, dan setiap kolom mewakili kelas yang diprediksi.
Baris pertama adalah baris negatif: yang "berisi gambar non-6". Anda dapat belajar banyak dari matriks tersebut.

Namun, ada juga baris yang bagus, yang menarik untuk dikerjakan jika Anda ingin mendapatkan akurasi prediksi positif, yaitu presisi pengklasifikasi menggunakan persamaan ini.

Presisi = (TP)/ (TP+FP)
TP: jumlah positif benar
FP: jumlah positif salah

Recall = (TP) /(TP+FN) “sensitivitas”: mengukur rasio kejadian positif.



Ingat
>>> 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 satu metrik saja, yaitu skor F1.

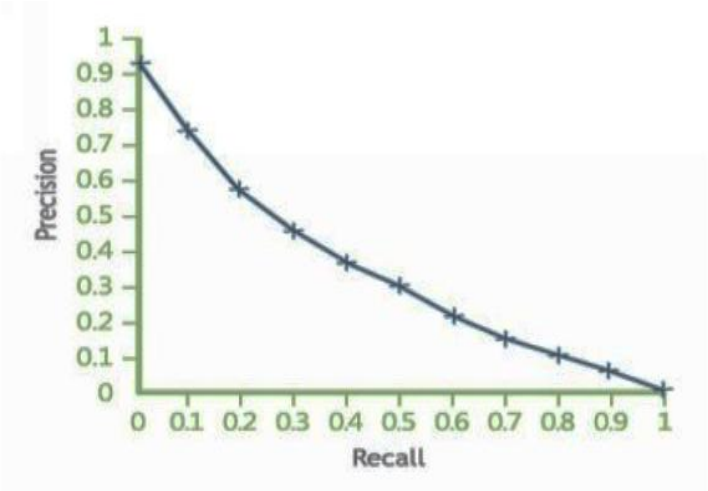
F1 adalah rata-rata presisi dan ingatan. Kita dapat menghitung skor F1 dengan persamaan berikut:

F1 = 2 / ((1/presisi) + (1)/ingat)) = 2 * (presisi * ingat) / (presisi + ingat) = (TP) / ((TP) + (FN+FP)/2)

Untuk menghitung skor F1, cukup gunakan fungsi berikut:

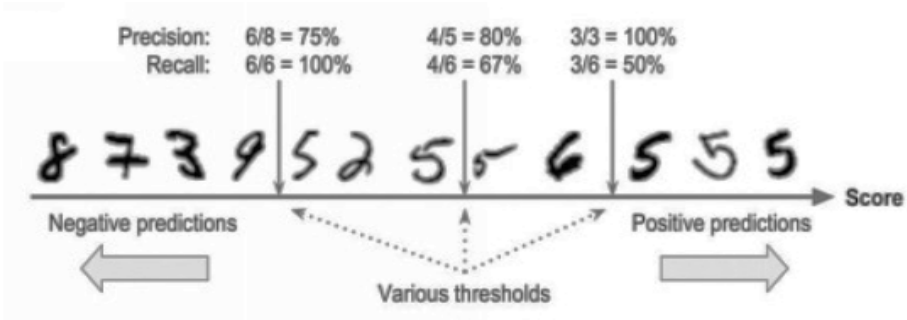
>>> from sklearn.metrics import f1_score

>>>f1_score (y_tr_6, y_pre)



Penarikan Kembali Tradeoff

Untuk sampai ke titik ini, Anda harus melihat SGDClassifier dan bagaimana ia membuat keputusan terkait klasifikasi. Ia menghitung skor berdasarkan fungsi keputusan, lalu membandingkan skor dengan ambang batas. Jika lebih besar dari skor ini, ia akan menetapkan contoh ke kelas "positif atau negatif". Misalnya, jika ambang batas keputusan berada di tengah, Anda akan menemukan 4 benar + di sisi kanan ambang batas, dan hanya satu salah. Jadi rasio presisi hanya akan menjadi 80%.



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

```
>>> y_sco = sgd_clf.decision_funciton([angka apa pun])
>>> y_sco
>>> threshold = 0
>>>y_any_digit_pre = (y_sco > threshold)
```

Dalam kode ini, SGDClassifier berisi ambang batas, = 0, untuk mengembalikan hasil yang sama dengan fungsi predict ().

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

Kode ini akan mengonfirmasi bahwa ketika ambang batas meningkat, penarikan kembali akan menurun.

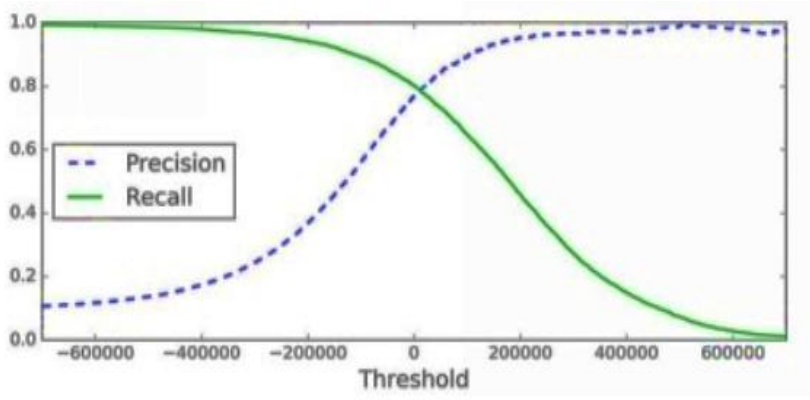
```
y_sco = cross_val_predict (sgd_cl, x_tr, y_tr_6, cv =3, method=”decision function)
```

Saatnya menghitung semua presisi dan recall yang mungkin untuk ambang batas dengan memanggil fungsi precision_recall_curve()

```
from sklearn.metrics import precision_recall_curve
precisions, recalls, threshold = precision_recall_curve (y_tr_6, y_sco)
```

dan sekarang mari kita plot presisi dan recall 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(pra, ulang, thr)
plt.tampilkan
```



2.4 ROC

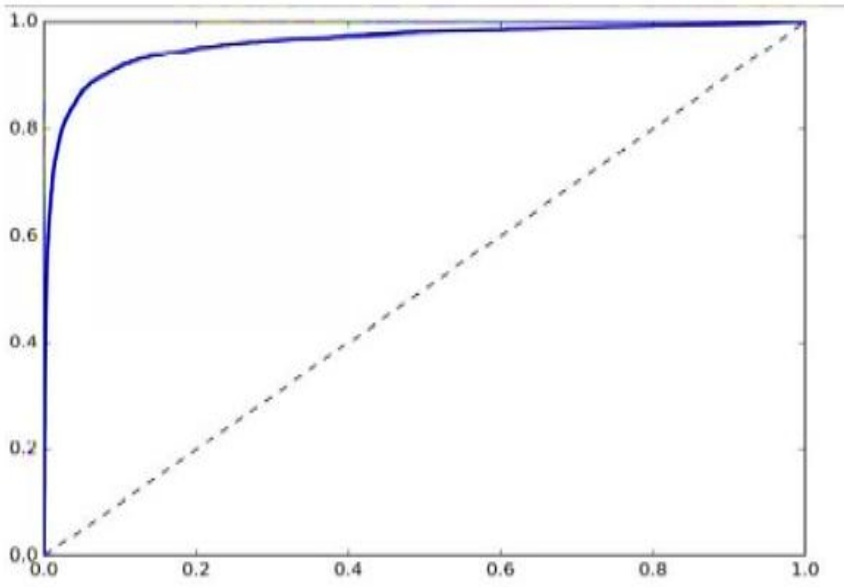
ROC adalah singkatan dari receiver operating Characteristics dan merupakan alat yang digunakan dengan pengklasifikasi biner. Alat ini mirip dengan kurva recall, tetapi tidak memetakan presisi dan recall: alat ini memetakan rasio positif dan rasio salah. Anda juga akan bekerja dengan FPR, yang merupakan rasio sampel negatif. Anda dapat membayangkan jika seperti (1 – rasio negatif. Konsep lainnya adalah TNR dan itu adalah spesifisitas. Recall = 1 – spesifisitas.

Mari bermain dengan Kurva ROC. Pertama, kita perlu menghitung TPR dan FPR, cukup dengan memanggil fungsi roc-curve (),

dari 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 petunjuk 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(‘Ini adalah rasio palsu’)
plt.ylabel(‘Ini adalah nilai sebenarnya’)
roc_plot (fp, tp)
plt.show
```



Klasifikasi Multikelas

Kami menggunakan pengklasifikasi biner untuk membedakan antara dua kelas, tetapi bagaimana jika Anda ingin membedakan lebih dari dua kelas? Anda dapat menggunakan sesuatu seperti pengklasifikasi hutan acak atau pengklasifikasi Bayes, yang dapat membandingkan lebih dari dua kelas. 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 perlu melatih 12 pengklasifikasi biner, dan membuat satu untuk setiap pengklasifikasi (seperti 4 – detektor, 5-detektor, 6-detektor, dan seterusnya), dan kemudian Anda perlu mendapatkan DS, "skor keputusan," dari setiap pengklasifikasi untuk gambar tersebut. Kemudian, Anda akan memilih pengklasifikasi dengan skor tertinggi. Kami menyebutnya strategi OvA: "satu lawan semua".

Metode lainnya adalah melatih pengklasifikasi biner untuk setiap pasangan digit; misalnya, satu untuk 5 dan 6 dan satu lagi untuk 5 dan 7. — kami menyebut metode ini OvO, "satu lawan satu" — untuk menghitung berapa banyak pengklasifikasi yang Anda perlukan, berdasarkan jumlah kelas yang menggunakan persamaan berikut: "N = 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 fungsi_keputusan () 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"]])
```

2.5 MELATIH PENGKLASIFIKASI RANDOM FOREST

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

Seperti yang Anda lihat, melatih pengklasifikasi random forest hanya dengan dua baris kode sangatlah mudah. Scikit-Learn tidak menjalankan fungsi OvA atau OvO apa pun karena algoritme semacam ini — "pengklasifikasi random forest" — dapat secara otomatis menjalankan beberapa kelas. Jika Anda ingin melihat daftar kemungkinan pengklasifikasi, Anda dapat memanggil fungsi predict_oroba().

```
>>> 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 dapat Anda lihat pada output; ada 0,8 pada indeks nomor 5.
Mari kita evaluasi pengklasifikasi 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 pada lipatan. Saat menggunakan pengklasifikasi acak, Anda akan mendapatkan, dalam kasus ini, 10% untuk skor akurasi. Perlu diingat bahwa semakin tinggi nilai ini, semakin baik.

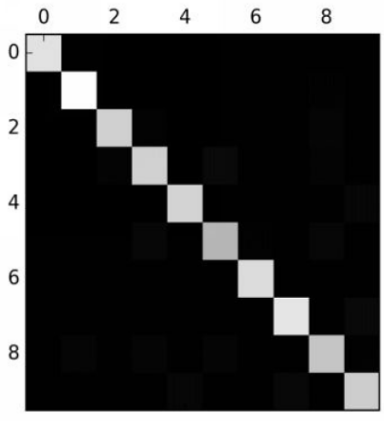
Analisis Kesalahan

Pertama-tama, saat mengembangkan proyek pembelajaran mesin:

1. Tentukan masalahnya;
2. Kumpulkan data Anda;
3. Kerjakan data Anda dan jelajahi;
4. Bersihkan data
5. Kerjakan 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 kebingungan dan membuat prediksi dengan fungsi cross-val. Selanjutnya, Anda akan memanggil fungsi matriks kebingungan:

```
>>> y_tr_pre = cross_val_prediciton(sgd_cl, x_tr_scaled, y_tr, cv=3)
>>> cn_mx = chaotic_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.abu-abu)
Plt.show()
```

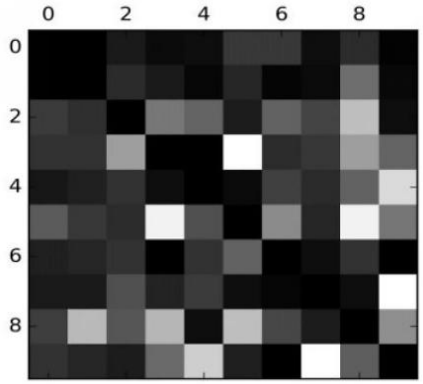


Pertama, Anda harus membagi setiap nilai dalam matriks dengan jumlah gambar dalam kelas, lalu Anda akan membandingkan tingkat kesalahannya.

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

Langkah berikutnya adalah membuat semua angka nol pada diagonal, dan itu akan mencegah terjadinya kesalahan. np.fill_diagonal (nm_cn_mx, 0)

```
plt.matshow(nm_cn_mx, cmap=plt.cm.gray)
plt.show()
```



Kesalahan mudah dikenali dalam skema di atas. Satu hal yang perlu diingat adalah bahwa baris mewakili kelas dan kolom mewakili nilai yang diprediksi.

Klasifikasi Multilabel

Dalam contoh di atas, setiap kelas hanya memiliki satu contoh. Namun, bagaimana jika kita ingin menetapkan contoh tersebut ke beberapa kelas misalnya pengenalan wajah? Misalkan Anda ingin menemukan lebih dari satu wajah dalam foto yang sama. Akan ada satu label untuk setiap wajah. Mari 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 petunjuk ini, kami telah membuat array y_mullti yang berisi dua label untuk setiap gambar. Label pertama berisi informasi apakah digit tersebut "besar" (8,9,.), dan label kedua memeriksa apakah digit tersebut 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.

Klasifikasi Multi-output

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

```
Tidak = rnd.randint (0, 101, (len(x_tr), 785)))
Tidak = rnd.randint(0, 101, (len(x_tes), 785))
x_tr_mo = x_tr + tidak
x_tes_mo = x_tes + tidak
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[indeks apa pun]])
plot_digit(cl_digit)
```

Latihan soal

1. Buat pengklasifikasi untuk set data MNIST. Cobalah untuk mendapatkan akurasi lebih dari 96% pada set pengujian Anda.
2. Tulis metode untuk menggeser gambar dari MNIST (kanan atau kiri) sebesar 2 piksel.
3. Kembangkan program anti-spam atau pengklasifikasi Anda sendiri.
   a. Unduh contoh spam dari Google.
   b. Ekstrak set data.
   c. Bagi set data menjadi pelatihan untuk set pengujian.
   d. Tulis program untuk mengubah setiap email menjadi vektor fitur.
   e. Bermain dengan pengklasifikasi, dan cobalah untuk membuat yang terbaik, dengan nilai yang tinggi untuk mengingat dan presisi.

Ringkasan

Dalam bab ini, Anda telah mempelajari konsep-konsep baru yang bermanfaat dan menerapkan banyak jenis algoritme klasifikasi. Anda juga telah bekerja dengan konsep-konsep baru, seperti:

a. ROC: karakteristik operasi penerima, alat yang digunakan dengan pengklasifikasi biner.
b. Analisis Kesalahan: mengoptimalkan algoritme Anda.
c. Cara melatih pengklasifikasi hutan acak menggunakan fungsi hutan di Scikit-Learn.

d. Memahami Klasifikasi Multi-Output.
e. Memahami klasifikasi multi-Label.

BAB 3
CARA MELATIH MODEL

3.1 PENDAHULUAN

Setelah bekerja dengan banyak model pembelajaran mesin dan algoritme pelatihan, yang tampak seperti kotak hitam yang tidak dapat dipahami, kami mampu mengoptimalkan sistem regresi, juga bekerja dengan pengklasifikasi gambar. Namun, kami mengembangkan sistem ini tanpa memahami apa yang ada di dalamnya dan cara kerjanya, jadi sekarang kami perlu mempelajarinya lebih dalam sehingga kami dapat memahami cara kerjanya dan memahami detail implementasinya.

Memperoleh pemahaman yang mendalam tentang detail ini akan membantu Anda dengan model yang tepat dan memilih algoritme pelatihan terbaik. Selain itu, ini akan membantu Anda dengan debugging dan analisis kesalahan. Dalam bab ini, kami akan bekerja dengan regresi polinomial, yang merupakan model kompleks yang berfungsi untuk set data nonlinier. Selain itu, kami akan bekerja dengan beberapa teknik regularisasi yang mengurangi pelatihan yang mendorong overfitting.

Regresi Linier

Sebagai contoh, kita akan mengambil l_S = θ0 + θ1 × GDP_per_cap. Ini adalah model sederhana untuk fungsi linier fitur input, “GPD_per_cap”. (θ0 dan θ1) adalah parameter model, secara umum, Anda akan menggunakan model linier untuk membuat prediksi dengan menghitung jumlah tertimbang fitur input, dan juga “bias” konstan, seperti yang dapat Anda lihat dalam persamaan berikut.

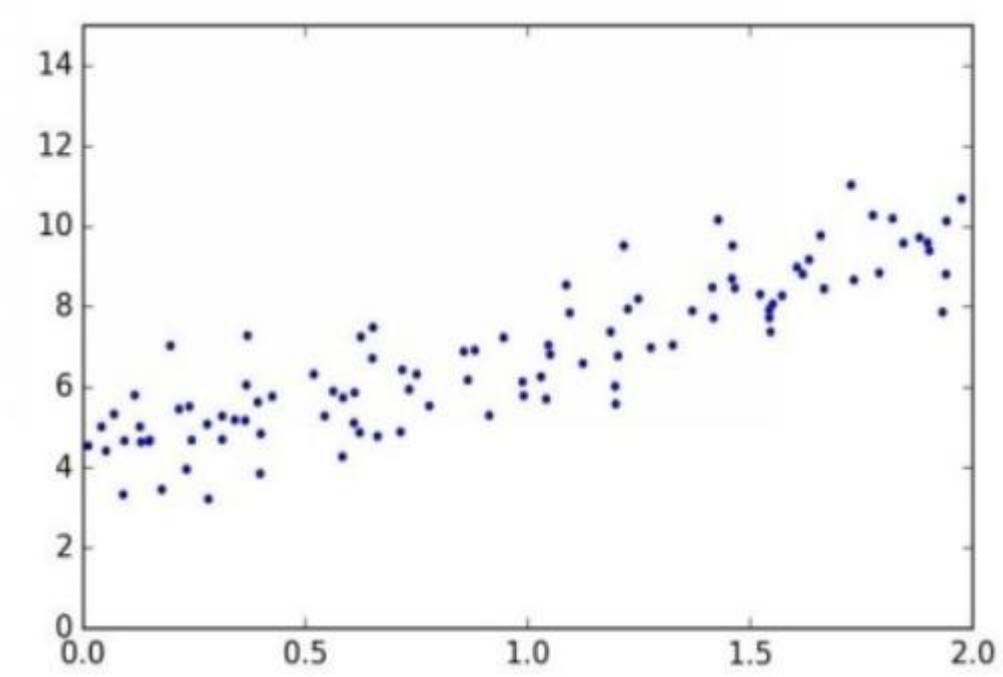
$$\hat{y} = 0_0 + 0_1 x_1 + 0_2 x_2 + \ldots + 0_n x_n$$

a. Y adalah nilai prediktor.
b. N mewakili fitur-fitur
c. X1 adalah nilai fitur.
d. Θj adalah parameter model j theta.

Kita juga dapat menulis persamaan dalam bentuk vektor, seperti pada contoh berikut:

$$\hat{y} = h_\theta(x) = \theta^T . x$$

Θ adalah nilai yang meminimalkan biaya.
Y berisi nilai y (1) hingga y (m).
Mari kita tulis beberapa kode untuk berlatih. Import numpy sebagai np
V1_x = 2 * np.random.rand (100, 1)
V2_y = 4 + 3 * V1_x + np.random.randn (100, 1)



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

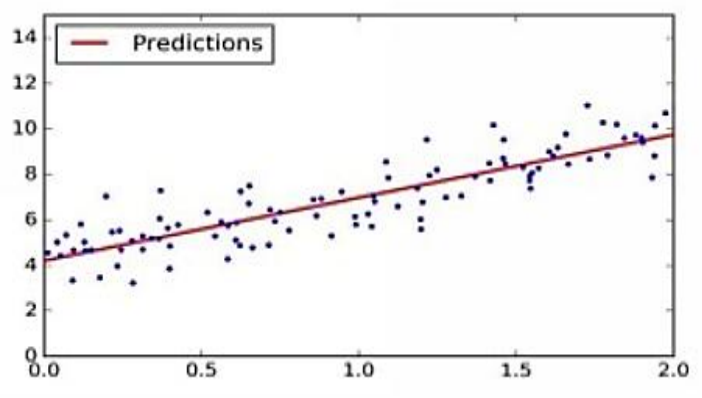
```
Value1 = np.c_[np.ones((100, 1)), V1_x]
myTheta = np.linalg.inv(Value1.T.dot(Value1)).dot(Value1.T).dot(V2_y)
>>>myTheta Array([[num], [num]])
```

Fungsi ini menggunakan persamaan berikut — y = 4 + 3x + 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_predicit = V1_new_2.dot(myTheta)
>>>V2_predict
Array([[ 4.219424], [9.74422282]])
```

Sekarang, saatnya untuk memplot model.

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



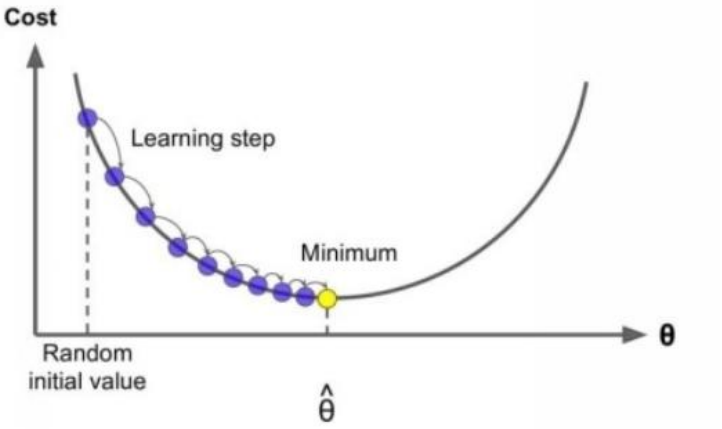
Kompleksitas Komputasi

Dengan rumus normal, kita dapat menghitung kebalikan dari M^T. M — yaitu, matriks n^n (n = jumlah fitur). Kompleksitas inversi ini kira-kira seperti O(n^2,5) hingga O(n^3,2), yang didasarkan pada implementasi. Sebenarnya, jika Anda membuat jumlah fitur menjadi dua kali lipat, Anda akan membuat waktu komputasi mencapai antara 2^2,5 dan 2^3,2. Berita bagusnya di sini adalah persamaan tersebut adalah persamaan linier.

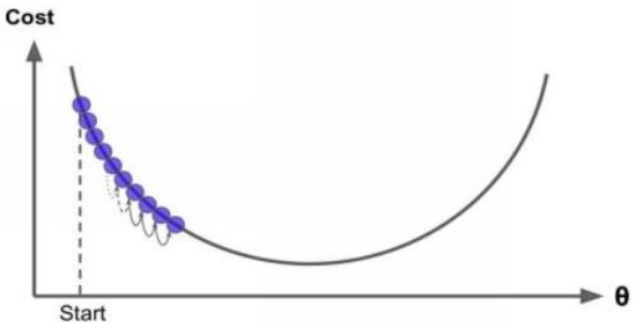
Ini berarti persamaan tersebut dapat dengan mudah menangani set pelatihan yang besar dan menyesuaikan dengan memori. Setelah melatih model Anda, prediksi tidak akan lambat, dan kompleksitasnya akan sederhana, berkat model linier. Saatnya untuk mempelajari lebih dalam metode pelatihan model regresi linier, yang selalu digunakan ketika ada sejumlah besar fitur dan instans dalam memori.

3.2 GRADIENT DESCENT

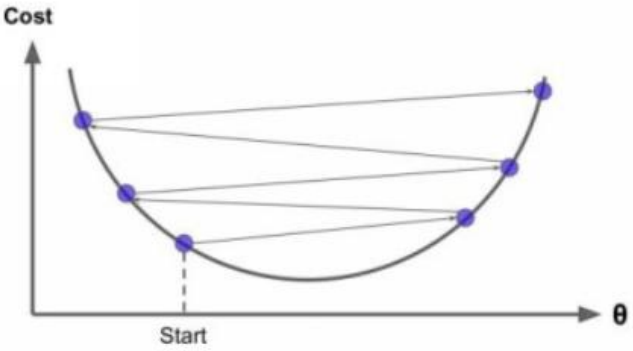
Algoritma ini adalah algoritma umum yang digunakan untuk optimasi dan untuk menyediakan solusi optimal untuk berbagai masalah. Ide dari algoritma ini adalah untuk bekerja dengan parameter secara berulang, untuk membuat fungsi biaya sesederhana mungkin. Algoritma gradient descent menghitung gradien kesalahan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



Selain itu, Anda harus ingat bahwa ukuran langkah sangat penting untuk algoritma ini, karena jika ukurannya sangat kecil – yang berarti laju pembelajarannya lambat – maka akan butuh waktu lama untuk mencakup semua hal yang dibutuhkan.

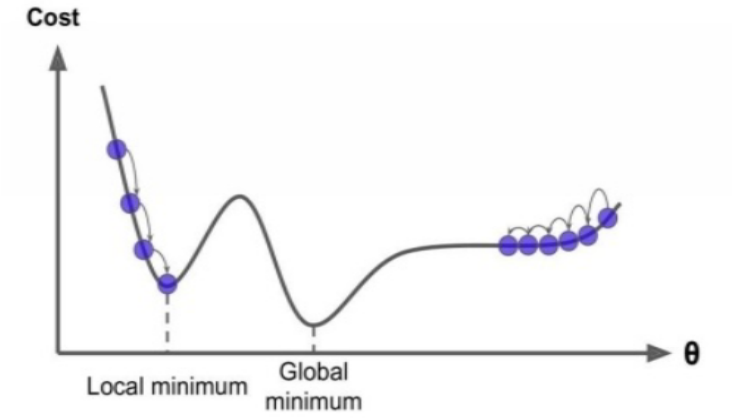


Namun ketika laju pembelajarannya tinggi, akan membutuhkan waktu yang singkat untuk mencakup apa yang dibutuhkan, dan akan memberikan solusi yang optimal.

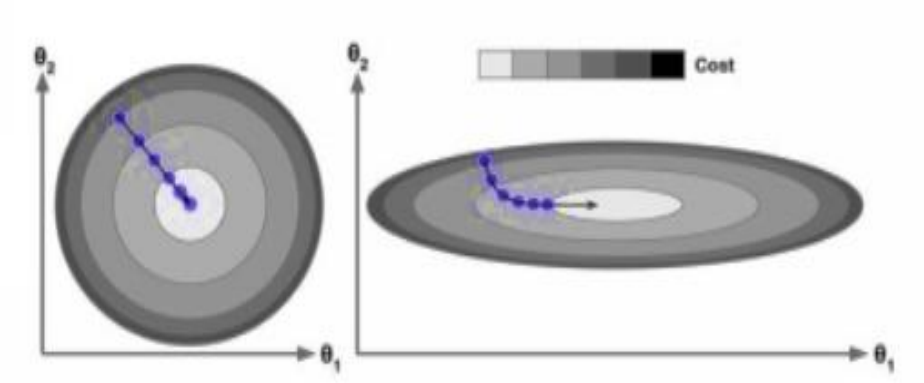


Pada akhirnya, Anda tidak akan selalu menemukan bahwa semua fungsi biaya mudah, seperti yang dapat Anda lihat, tetapi Anda juga akan menemukan fungsi tidak teratur yang membuat

perolehan solusi optimal menjadi sangat sulit. Masalah ini terjadi ketika minimum lokal dan minimum global terlihat seperti pada gambar berikut.



Jika Anda menetapkan sembarang ke dua titik mana pun pada kurva Anda, Anda akan menemukan bahwa segmen garis tidak akan menghubungkan keduanya pada kurva yang sama. Fungsi biaya ini akan terlihat seperti mangkuk, yang akan terjadi jika fitur memiliki banyak skala, seperti pada gambar berikut



3.3 PENURUNAN GRADIEN BATCH

Jika Anda ingin menerapkan algoritma ini, pertama-tama Anda harus menghitung gradien fungsi biaya Anda menggunakan parameter theta. Jika nilai parameter theta telah berubah, Anda perlu mengetahui laju perubahan fungsi biaya Anda. Kita dapat menyebut perubahan ini dengan turunan parsial. Kita dapat menghitung turunan parsial menggunakan persamaan berikut:

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

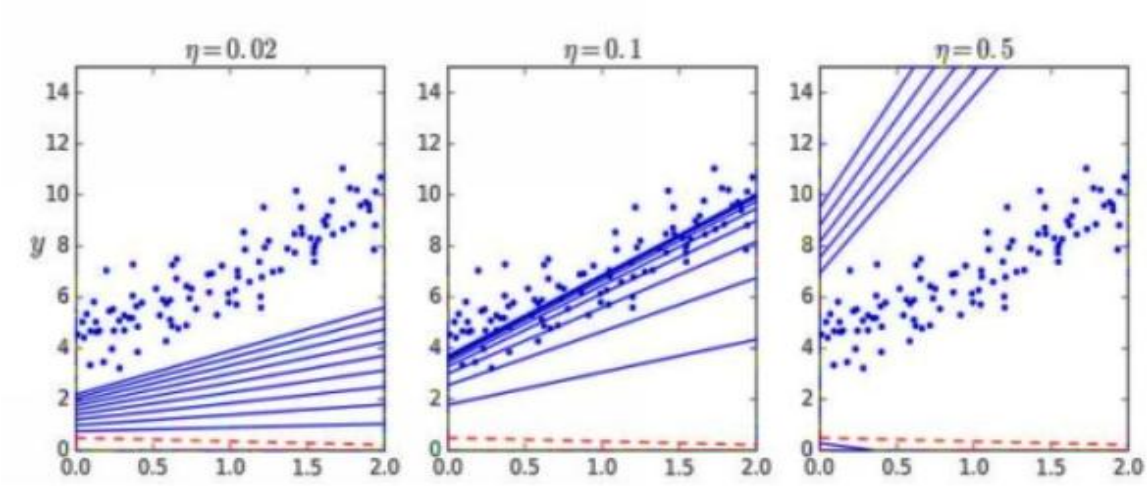
Namun kita juga akan menggunakan persamaan berikut untuk menghitung turunan parsial dan vektor gradien secara bersamaan.

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

Mari kita terapkan algoritmanya.

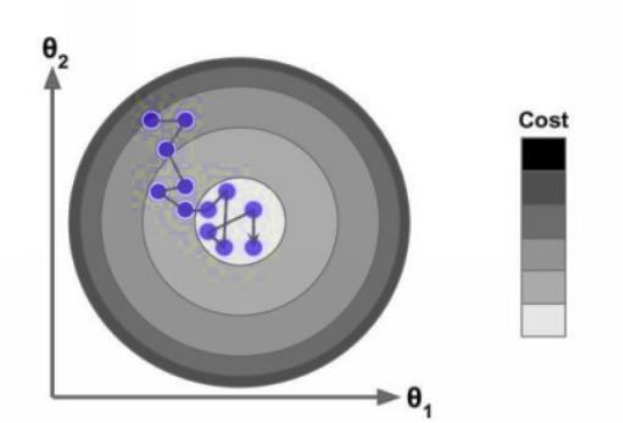
```
Lr = 1 # Lr untuk laju pembelajaran
Num_it = 1000 # jumlah iterasi
L = 100
myTheta = np.random.randn (2,1)
for it in range(Num_it):
gr = 2/L * Value1.T.dot(Value1.dot(myTheta) – V2_y)
myTheta = myTheta – Lr * gr
>>> myTheta
Array([[num],[num]])
```

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



3.4 PENURUNAN GRADIEN STOKASTIK

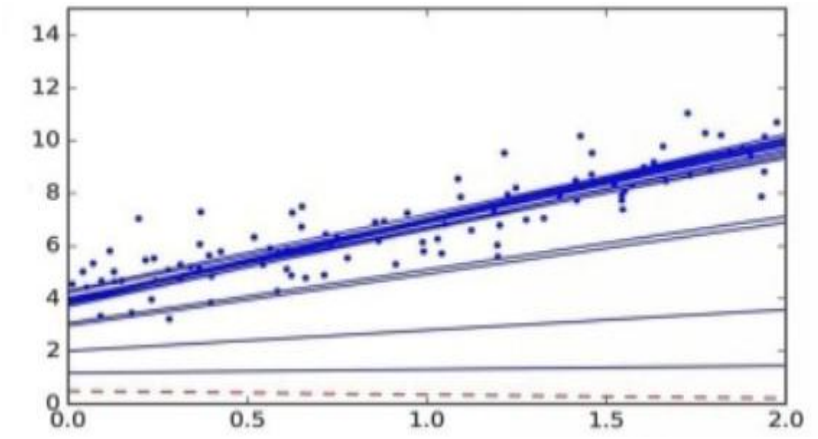
Anda akan menemukan masalah saat menggunakan penurunan gradien batch: ia perlu menggunakan seluruh set pelatihan untuk menghitung nilai pada setiap langkah, dan itu akan memengaruhi "kecepatan" performa.



Namun saat menggunakan penurunan gradien stokastik, algoritme akan secara acak memilih satu contoh dari set pelatihan Anda di setiap langkah, lalu akan menghitung nilainya. Dengan cara ini, algoritme akan lebih cepat daripada penurunan gradien batch, karena tidak perlu menggunakan seluruh set untuk menghitung nilai. Di sisi lain, karena keacakan metode ini, ia akan menjadi tidak teratur jika dibandingkan dengan algoritme batch.
Mari kita terapkan algoritmanya.

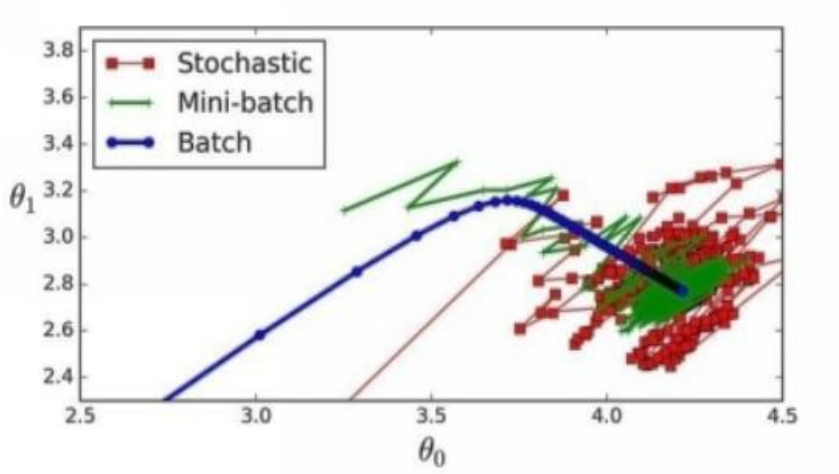
```
Nums = 50
L1, L2 = 5, 50
Def lr_sc(s):
  return L1 / (s + L2)
myTheta = np.random.randn(2,1)

untuk Num dalam rentang (Nums):
untuk l dalam rentang (f)
myIndex = np.random.randint(f)
V1_Xi = Value1[myIndex:myIndex+1]
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]])
```



3.5 MINI-BATCH GRADIENT DESCENT

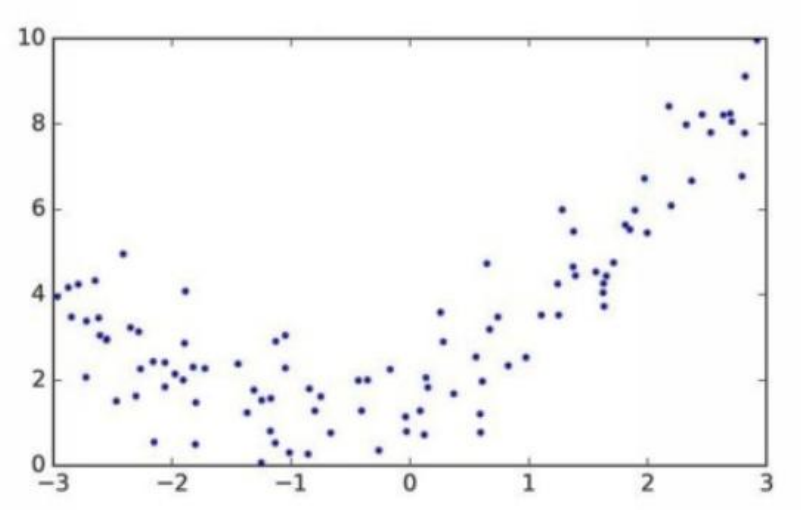
Karena Anda sudah mengetahui algoritma batch dan stokastik, jenis algoritma ini sangat mudah dipahami dan digunakan. Seperti yang Anda ketahui, kedua algoritma menghitung nilai gradien, berdasarkan seluruh set pelatihan atau hanya satu contoh. Namun, mini-batch menghitung algoritmanya berdasarkan set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



3.6 REGRESI POLINOMIAL

Kita akan menggunakan teknik ini saat bekerja dengan data yang lebih kompleks, khususnya, dalam kasus data linear dan nonlinier. Setelah kita menambahkan pangkat setiap fitur, kita 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)
```

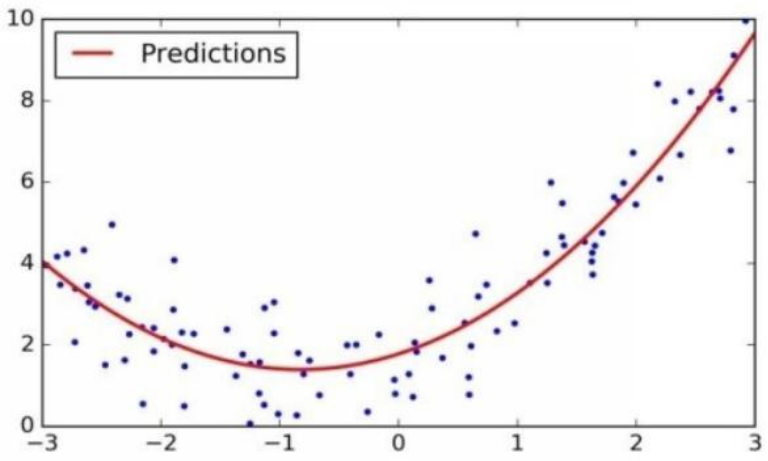


Seperti yang Anda lihat, garis lurus tidak akan pernah merepresentasikan data dengan cara yang paling efisien. Jadi, kita akan menggunakan metode polinomial untuk mengerjakan masalah ini.

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

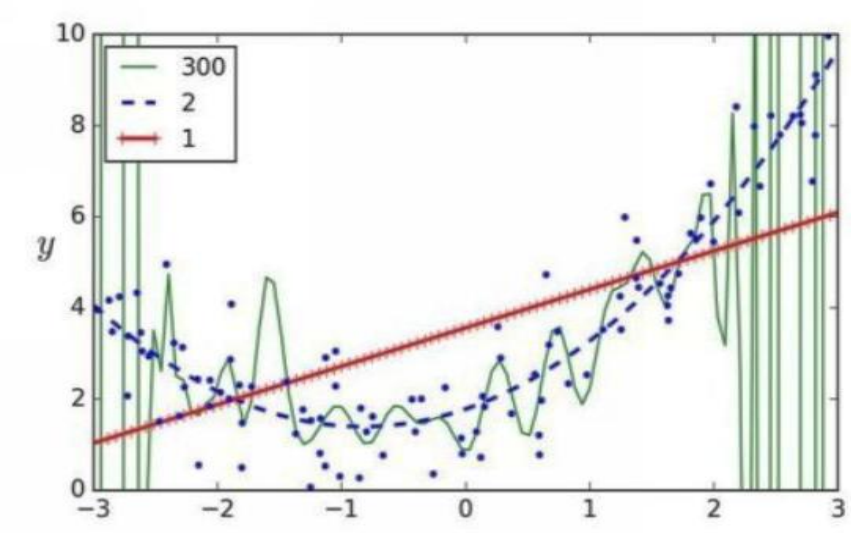
Sekarang, mari kita buat fungsi ini dengan benar menggunakan data kita, dan ubah garis lurusnya.

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



3.7 KURVA PEMBELAJARAN

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



Pada gambar di atas, Anda dapat melihat overfitting data saat Anda menggunakan polinomial. Di sisi lain, dengan polinomial linier, Anda dapat melihat bahwa data jelas-jelas mengalami underfitting.

Model Linier yang Diregulasi

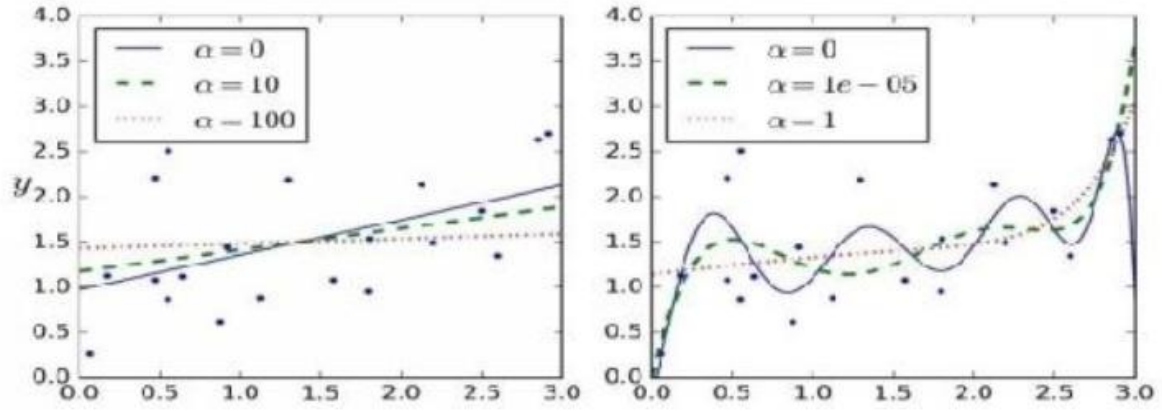
Kita telah membahas, pada bab pertama dan kedua, tentang cara mengurangi overfitting dengan sedikit meregulasi model, sebagai contoh, jika Anda ingin meregulasi model polinomial. Dalam kasus ini, untuk memperbaiki masalah, Anda harus mengurangi jumlah derajat.

Regresi Ridge

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

$$\mathrm{J}(\theta) = \mathrm{MSE}(\theta) + \alpha \frac{1}{2} \sum_{I=1}^{n} \theta^{2_I}$$

Sebagai contoh regresi punggungan, lihat saja gambar berikut.

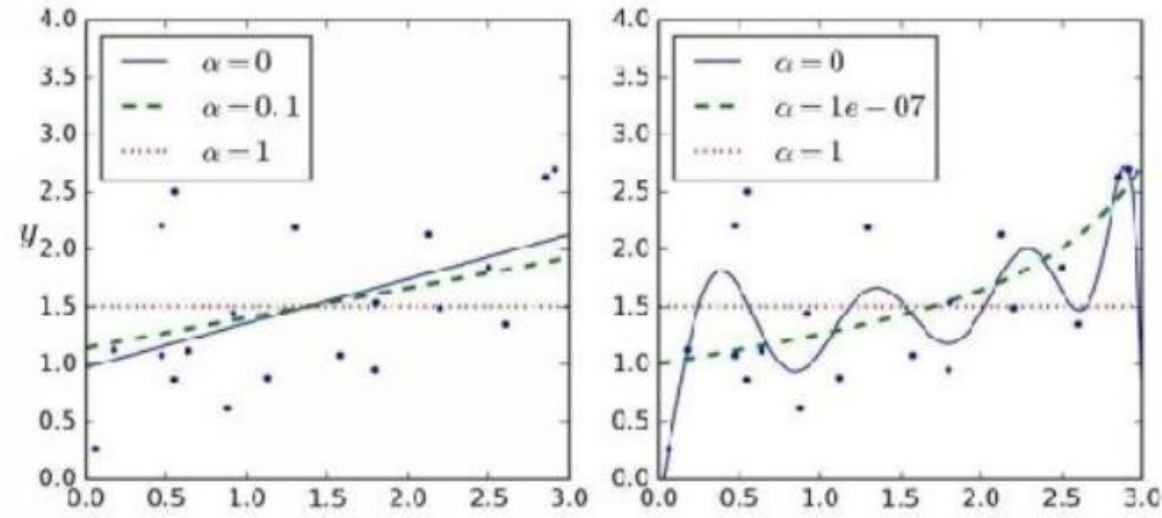


Regresi Lasso

Regresi "Lasso" merupakan singkatan dari regresi "Least Absolute Shrinkage and Selection Operator". Ini merupakan jenis lain dari versi regresi linier yang teregulasi. Regresi ini tampak seperti regresi ridge, tetapi dengan sedikit perbedaan dalam persamaan, seperti pada gambar berikut Fungsi biaya regresi laso:

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

Seperti yang dapat Anda lihat pada gambar berikut, regresi laso menggunakan nilai yang lebih kecil daripada punggungan.



Latihan Soal

1. Jika Anda memiliki satu set yang berisi sejumlah besar fitur (jutaan), 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 akan memperbaiki masalah ini?
3. Apa yang harus Anda lakukan jika Anda melihat bahwa kesalahan menjadi lebih besar saat Anda menggunakan metode mini-batch? Mengapa?
4. Dari pasangan ini, metode mana yang lebih baik? Mengapa? :
   a. Regresi ridge dan regresi linier?
   b. Regresi laso dan regresi ridge?
5. Tulis algoritma penurunan gradien batch.

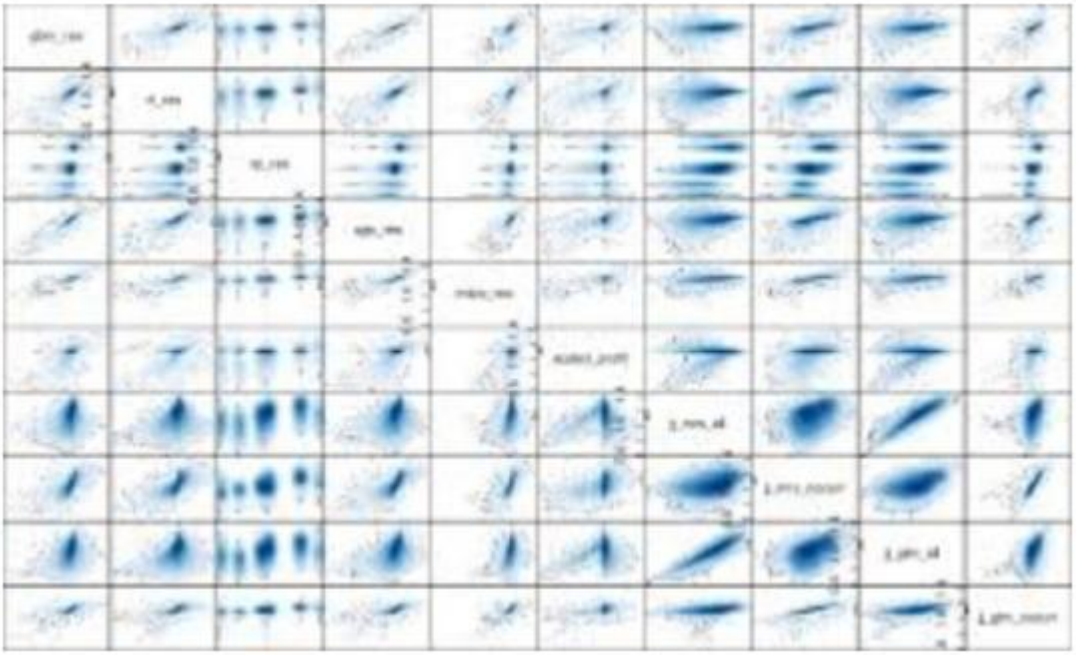
Ringkasan

Dalam bab ini, Anda telah mempelajari konsep baru, dan telah mempelajari cara melatih model menggunakan berbagai jenis algoritme. Anda juga telah mempelajari kapan harus menggunakan setiap algoritme, termasuk yang berikut:

a. Penurunan gradien batch
b. Penurunan gradien mini-batch
c. Regresi polinomial
d. Model linier terregulasi
   - Regresi punggungan
   - Regresi laso

Selain itu, Anda sekarang mengetahui arti dari istilah-istilah tertentu: regresi linier, kompleksitas komputasi, dan penurunan gradien.

BAB 4
KOMBINASI BERBAGAI MODEL



Klasifikasi pohon

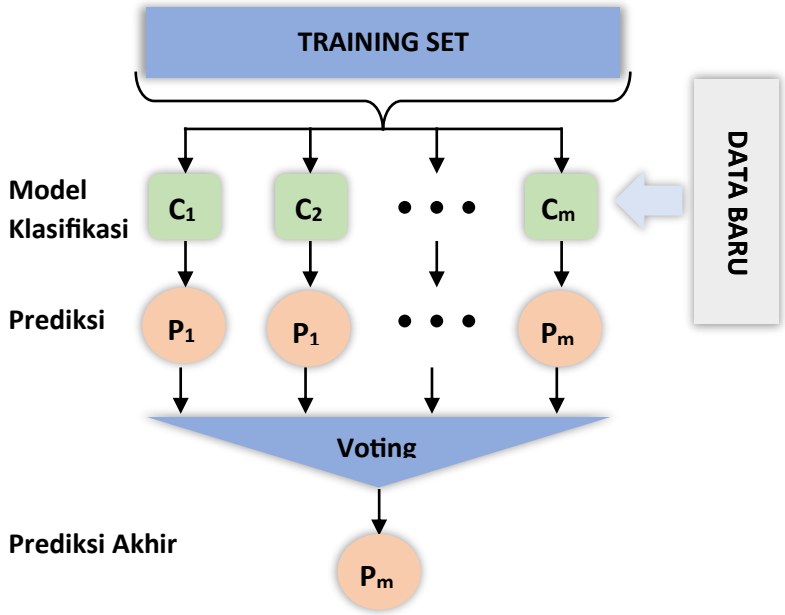
Gambar berikutnya akan mengilustrasikan definisi target umum fungsi pengumpulan, yaitu menggabungkan berbagai klasifikasi menjadi Satu Klasifikasi yang memiliki kinerja generalisasi yang lebih baik daripada masing-masing klasifikasi secara terpisah. Sebagai contoh, asumsikan Anda mengumpulkan prediksi dari banyak pakar. Metode ensemble akan memungkinkan kita untuk menggabungkan prediksi ini oleh banyak pakar untuk mendapatkan prediksi yang lebih tepat dan kuat daripada prediksi masing-masing pakar. Seperti yang dapat Anda lihat nanti di bagian ini, ada banyak metode berbeda untuk membuat ensemble klasifikasi.

Di bagian ini, kami akan memperkenalkan persepsi dasar tentang cara kerja ensemble dan mengapa ensemble biasanya dikenal menghasilkan kinerja generalisasi yang baik. Di bagian ini, kami akan bekerja dengan metode ensemble paling populer yang menggunakan prinsip pemungutan suara mayoritas. Pemungutan suara banyak berarti bahwa kami memilih label yang telah diprediksi oleh mayoritas klasifikasi; yaitu, menerima lebih dari 50 persen suara. Sebagai contoh, istilah di sini seperti pemungutan suara yang hanya merujuk pada pengaturan kelas biner saja.

Namun, tidak sulit untuk menghasilkan prinsip pemungutan suara mayoritas untuk pengaturan multikelas, yang disebut pemungutan suara pluralitas. Setelah itu, kita akan memilih label kelas yang menerima suara terbanyak. Diagram berikut menggambarkan

konsep pemungutan suara mayoritas dan pluralitas untuk ansambel 10 pengklasifikasi di mana setiap simbol unik (segitiga, persegi, dan lingkaran) mewakili label kelas yang unik:

Dengan menggunakan set pelatihan, kita mulai dengan melatih m pengklasifikasi yang berbeda (C C 1, , … m). Berdasarkan metode tersebut, ansambel dapat dibangun dari banyak algoritma klasifikasi; misalnya, pohon keputusan, mesin vektor pendukung, pengklasifikasi regresi logistik, dan sebagainya. Bahkan, Anda dapat menggunakan algoritma klasifikasi dasar yang sama yang sesuai dengan subset yang berbeda dari set pelatihan. Contoh dari metode ini adalah algoritma hutan acak, yang menggabungkan banyak cara ansambel keputusan menggunakan pemungutan suara mayoritas.



Untuk memprediksi label kelas melalui pemungutan suara mayoritas atau pluralitas sederhana, kami menggabungkan label kelas yang diprediksi dari setiap pengklasifikasi C j dan memilih label kelas yˆ yang menerima suara terbanyak:

y m ˆ = ode{C C 1 2 ( ) x x , , ( ) …, Cm ( ) x }

Misalnya, dalam tugas klasifikasi biner di mana class1 1 = − dan class2 1 = +, kami dapat menulis prediksi suara mayoritas. Untuk mengilustrasikan mengapa metode ensemble dapat bekerja lebih baik daripada pengklasifikasi individual saja, mari terapkan konsep kombinatori 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 bersifat independen dan tingkat kesalahan tidak berkorelasi. Seperti yang Anda lihat, kami dapat menjelaskan statistik kesalahan ensemble pengklasifikasi dasar sebagai probabilitas.

4.1 FUNGSI MASSA DARI DISTRIBUSI BINOMIAL:

Di sini, n. k adalah koefisien binomial n pilih k. Seperti yang dapat Anda lihat, Anda dapat menghitung probabilitas bahwa prediksi ensembel tersebut salah. Sekarang, mari kita lihat contoh yang lebih konkret dari 11 pengklasifikasi dasar (n = 11) dengan tingkat kesalahan 0,25 (ε = 0,25):

Anda dapat melihat bahwa tingkat kesalahan ensembel (0,034) lebih kecil daripada tingkat kesalahan setiap pengklasifikasi individual (0,25) jika semua asumsi terpenuhi. Perhatikan bahwa dalam gambar yang disederhanakan ini, pembagian 50-50 dengan jumlah pengklasifikasi genap n diperlakukan sebagai kesalahan, sedangkan ini hanya benar setengah dari waktu. Untuk membandingkan pengklasifikasi ensemble idealis tersebut dengan pengklasifikasi basis pada rentang rasio kesalahan basis yang berbeda, mari terapkan fungsi massa probabilitas dalam Python:

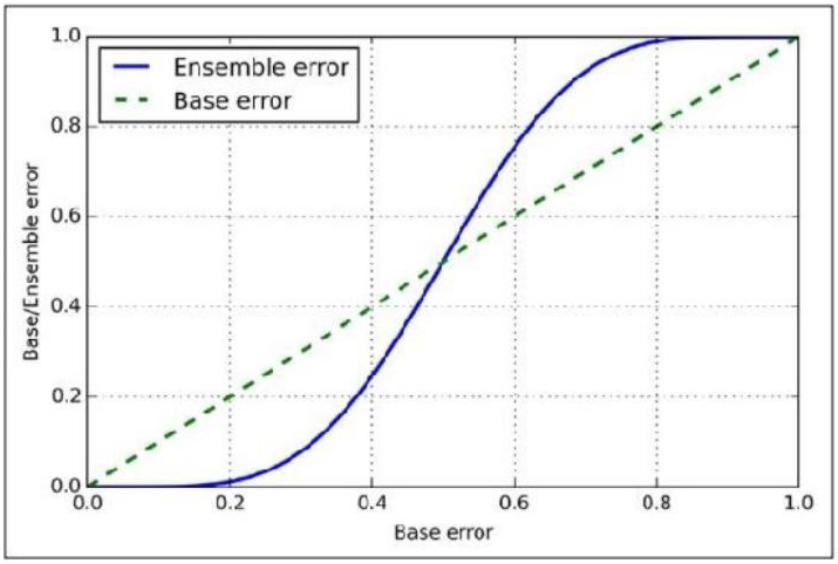
```
>>> import math
>>> def ensemble_error(n_classifier, error):
... q_start = math.ceil(n_classifier / 2.0)
... Probability = [comb(n_class, q) *
... error**q *
... (1-error)**(n_classifier - q)
... for q in range(q_start, l_classifier + 2)]
... return sum(Probability)
>>> ensemble_error(n_classifier=11,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

Mari tulis beberapa kode untuk menghitung rasio kesalahan yang berbeda. Visualisasikan hubungan antara kesalahan ensemble dan basis 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)
... untuk er di er_range]
>>> impor matplotlib.pyplot sebagai plt
>>> plt.plot(er_range, en_error,
... label='Kesalahan kumpulan',
... lebar garis=2)
>>> plt.plot(er_range, er_range,
... ls='--', label='B_ er',
... lebar garis=2)
>>> plt.xlabel('B_ er')
>>> plt.label('B/En_er')
>>> plt.legend(loc='kiri atas')
```

```
>>> plt.grid()
>>> plt.tampilkan()
```

Seperti yang bisa kita lihat pada plot yang dihasilkan, probabilitas kesalahan suatu ansambel selalu lebih baik daripada kesalahannya pengklasifikasi dasar individual selama pengklasifikasi dasar tersebut berkinerja lebih baik daripada tebakan acak ($\varepsilon < 0{,}5$). Anda harus memperhatikan bahwa sumbu y menggambarkan kesalahan dasar serta kesalahan ansambel (garis kontinu):



4.2 MENERAPKAN PENGKLASIFIKASI MAYORITAS SEDERHANA

Seperti yang kita lihat dalam pengantar pembelajaran gabungan di bagian terakhir, kita akan bekerja dengan pelatihan pemanasan dan kemudian mengembangkan pengklasifikasi sederhana untuk pemungutan suara mayoritas dalam pemrograman Python. Seperti yang Anda lihat, algoritme berikutnya akan bekerja pada pengaturan multikelas melalui pemungutan suara pluralitas; Anda akan menggunakan istilah pemungutan suara mayoritas untuk kesederhanaan seperti yang juga sering dilakukan dalam literatur.

Dalam program berikut, kita akan mengembangkan dan juga menggabungkan berbagai program klasifikasi yang terkait dengan bobot individual untuk keyakinan. Sasaran kita adalah membangun meta-klasifikasi yang lebih kuat yang menyeimbangkan kelemahan pengklasifikasi individual 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 yang praktis dari NumPy:

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

Di sini, pij adalah probabilitas yang diprediksi dari pengklasifikasi j untuk label kelas i. Untuk melanjutkan contoh kita sebelumnya, mari kita asumsikan bahwa kita memiliki masalah klasifikasi biner dengan label kelas i ∈ { } 0,1 dan ensembel tiga pengklasifikasi C j (j ∈ { } 1,2,3). Mari kita asumsikan bahwa pengklasifikasi C j mengembalikan probabilitas keanggotaan kelas berikut untuk sampel tertentu x:

C1 ( ) x → [ ] 0.9,0.1 , C2 ( ) x → [ ] 0.8,0.2 , C3 ( ) x → [ ] 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)
```

Setelah menggabungkan semuanya, mari kita terapkan MajorityVoteClassifier dalam Python:

```
from sklearn.base import ClassifierMixin
from sklearn.pre_processing import Label_En
from sklearn.ext import six
from sklearn.ba import clone
from sklearn.pipeline
import _name_estimators
import numpy as np
import operator

class MVClassifier(BaseEstimator, ClassifierMixin):
‘’’’Pengklasifikasi Ansambel Dengan Suara Mayoritas
Parameter
------
cl : mirip array, bentuk = [n_classifiers]
Klasifikasi yang berbeda untuk pemungutan suara ensemble: str, {'cl_label', 'prob'}
Default: 'cl_label'
Jika 'cl_label' prediksi didasarkan pada argmax label kelas. Elif 'prob', arg dari total probs digunakan untuk memprediksi label kelas (disarankan untuk klasifikasi yang dikalibrasi).

w : arr-like, s = [n_cl]
```

```
Opsional, default: None
Jika daftar nilai `int` atau `float` disediakan, klasifikasi diberi bobot oleh """
def init (s, cl,
v='cl_label', w=None): s.cl = cl
s.named_cl = {kunci: nilai untuk
kunci, nilai dalam
_nama_penaksir(cl)}
s.v = v
s.w = w
def fit_cl(s, X, y):
“”” Fit_cl.
Parameter
----------
X : {matriks jarang mirip array},
s = [n_sampel, n_fitur]
Matriks sampel pelatihan.
y : arr_like, sh = [n_samples]
Vektor label kelas target.
Returns
---------
s : objek """
# Gunakan LabelEncoder untuk memastikan label kelas dimulai # dengan 0, yang penting untuk np.argmax
# panggil 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 pada kode untuk lebih memahami masing-masing bagian. Namun, sebelum kita menerapkan metode yang tersisa, mari kita istirahat sejenak dan membahas beberapa kode yang mungkin tampak membingungkan pada awalnya. Kami menggunakan kelas induk BaseEstimator dan ClassifierMixin untuk mendapatkan beberapa fungsi dasar secara gratis, termasuk metode get_params dan set_params untuk menetapkan dan mengembalikan parameter classifer serta metode score untuk menghitung

akurasi prediksi, masing-masing. Perhatikan juga bahwa kami mengimpor enam untuk membuat MajorityVoteClassifier kompatibel dengan Python 2.7.

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

```
def pre(s, X):
""" Label kelas pra untuk X.
Parameter
------
X : {arr-like, spar mat},
Sh = [n_samples, n_features]
Matriks sampel pelatihan.

Pengembalian
ma_v : arr-like, sh = [n_samples]
Label kelas yang diprediksi.
"""
if se.v == 'probability':
ma_v = np.argmax(spredict_prob(X), axis=1)
yang lain: # 'cl_label' v
prediksi = np.asarr([cl.predict(X)
untuk cl di
s.cl_]).T
ma_v = np.ap_al_ax(lambda x:
np.argmax(np.bincount(x, bobot=s.w)),
axis=1,
arr=prediksi)
ma_v = s.l_.inverse_transform(ma_v)
return ma_v
def predict_proba(self, X):
"""
Prediksi untuk X.
Parameter
------
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.
Mengembalikan
------
av_prob : seperti array,
sh = [n_sampel, n_kelas]

Probabilitas rata-rata tertimbang untuk setiap kelas per sampel.
probs = np.asarr([cl.predict_prob(X) untuk cl dalam s.cl_])
av_prob = np.av(probs, sumbu=0, bobot=s.w)
return av_prob
def get_ps(self, deep=True):
""" Dapatkan nama parameter pengklasifikasi untuk GridSearch""" jika tidak deep:
return super(MVC, self).get_ps(deep=False) yang lain:
ou = s.n_cl.copy() untuk n, langkah di\ six.iteritems(s.n_cl):
untuk k, nilai di six.iteritems(step.get_ps(deep=True)):
ou['%s %s' % (n, k)] = nilai return ou

4.3 PENGGABUNGAN ALGORITME UNTUK KLASIFIKASI SUARA MAYORITAS

Sekarang, saatnya untuk menerapkan MVC yang telah kita terapkan di bagian sebelumnya. Pertama-tama, Anda harus menyiapkan kumpulan data yang dapat Anda uji. Karena kita sudah familier 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.

Lebih jauh, kita hanya akan memilih dua fitur, lebar sepal dan panjang petal, untuk membuat tugas klasifikasi lebih menantang. Meskipun MajorityVoteClassifier, atau MVC, kita menggeneralisasi ke masalah multikelas, kita hanya akan mengklasifikasikan sampel bunga dari dua kelas, Ir-Versicolor dan Ir-Virginica, untuk menghitung ROC AUC. Kodenya 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 kita membagi sampel Iris menjadi 50 persen data pelatihan dan 50 persen data pengujian:

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

Dengan menggunakan dataset pelatihan, kita sekarang akan melatih tiga pengklasifikasi yang berbeda pengklasifikasi regresi logistik, pengklasifikasi pohon keputusan, dan pengklasifikasi k-nearest neighbor dan melihat kinerja masing-masing melalui 10 validasi silang pada dataset pelatihan sebelum kita menggabungkannya menjadi satu kesatuan:
impor yang berikut

```
sklearn.cross_validation
sklearn.linear_model
sklearn.tree
sklearn.pipeline
Pipeline
numpy sebagai np

>>> clf1 = LogisticRegression(penalti='l2',
... C=0,001,
... random_state=0)
>>> clf2 = DTCl(kedalaman_maks=1,
... kriteria='entropi',
... 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 = ['Regresi Logistik', 'Pohon Keputusan', 'KNN']
>>> print('validasi silang 10 kali lipat:\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 masing-masing pengklasifikasi hampir sama:
validasi silang 10 kali lipat:

```
ROC AUC: 0,92 (+/- 0,20) [Regresi Logistik]
ROC AUC: 0,92 (+/- 0,15) [Pohon Keputusan]
ROC AUC: 0,93 (+/- 0,10) [KNN]
```

Anda mungkin bertanya-tanya mengapa kami melatih regresi logistik dan pengklasifikasi k-tetangga terdekat sebagai bagian dari alur kerja. Penyebabnya di sini adalah, seperti yang kami katakan, algoritma regresi logistik dan k-tetangga terdekat (menggunakan metrik jarak Euclidean) tidak invarian skala, berbeda dengan pohon keputusan. Namun, semua keuntungan Ir diukur pada skala yang sama; merupakan kebiasaan yang baik untuk bekerja dengan fitur-fitur yang terstandarisasi.

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

```
>>> mv_cl = M_V_C(
... cl=[pipe1, clf2, pipe3])
>>> cl_labels += ['Pemungutan Suara Mayoritas']
>>> all_cl = [pipe1, clf2, pipe3, mv_clf]
>>> 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) [Regresi Logistik]
R_AUC: 0,92 (+/- 0,15) [D_T]
R_AUC: 0,93 (+/- 0,10) [KNN]
R_AUC: 0,97 (+/- 0,10) [Pemungutan Suara Mayoritas]
```

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

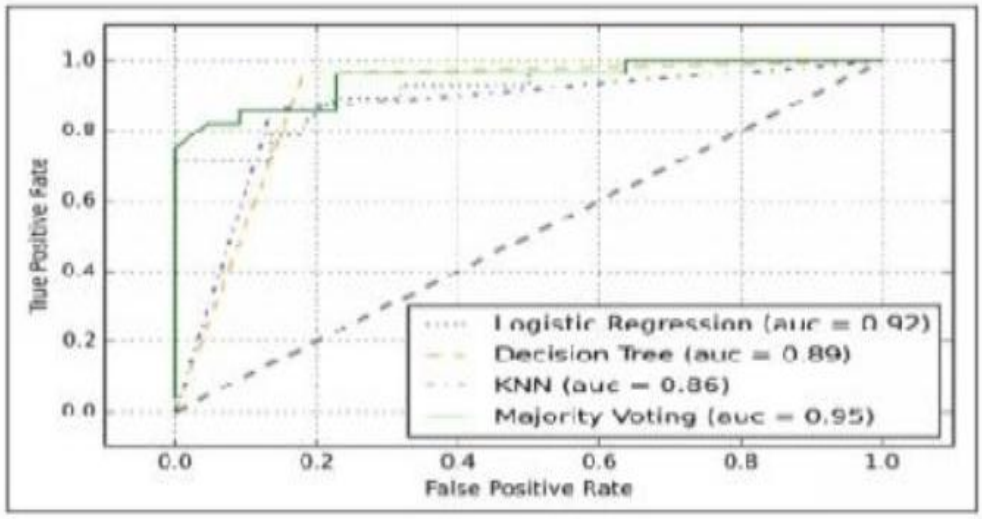
4.4 PENGKLASIFIKASI

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

```
import roc_curve dari sklearn.metrics import auc cls = ['hitam', 'oranye', 'biru', 'hijau']

ls = [':', '--', '-.', '-']
untuk cl, label, cl, l \
... dalam zip(all_cl, cl_labels, cls, ls):
... y_pred = clf.fit(X_train,
... y_train).predict_proba(X_test)[:, 1]
... fpr, tpr, ambang batas = rc_curve(y_t=y_tes,
... y_sc=y_pr)
... rc_auc = ac(x=fpr, y=tpr)
... plt.plot(fpr, tpr,
... warna=clr,
... gaya garis=ls,
... la='%s (ac = %0.2f)' % (la, rc_ac))
>>> plt.lg(lc='kanan bawah')
>>> plt.plot([0, 1], [0, 1],
... linestyle='--',
... color='abu-abu',
... 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, pengklasifikasi ensemble juga berkinerja baik pada set pengujian (ROC AUC = 0,95), sedangkan pengklasifikasi k-tetangga terdekat tampaknya melakukan overfitting pada data pelatihan (pelatihan ROC AUC = 0,93, pengujian ROC AUC = 0,86):



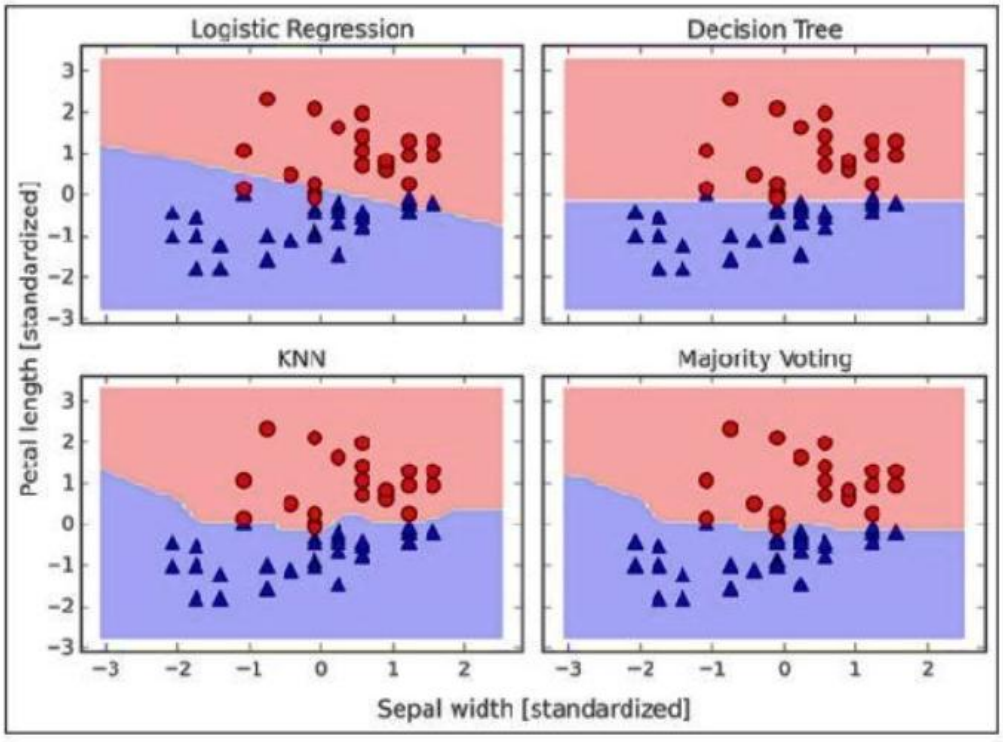
Anda hanya memilih dua fitur untuk tugas klasifikasi. Akan menarik untuk menunjukkan seperti apa sebenarnya wilayah keputusan dari pengklasifikasi ensemble. Meskipun tidak perlu untuk menstandardisasi fitur pelatihan sebelum model agar sesuai karena jalur regresi logistik dan k-tetangga terdekat kita akan secara otomatis mengurus ini, Anda akan membuat set pelatihan sehingga wilayah keputusan dari pohon keputusan akan berada pada skala yang sama untuk tujuan visual.

Mari kita lihat:

```
>>> sc = SS()
X_tra_std = sc.fit_transform(X_train)
Dari 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.subplot(nrows=2, ncols=2, sharex='col',
sharey='baris', gambar=(7, 5))
untuk ix, cl, tt di zip(produk([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.resh(xx.bentuk)
... 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='biru',
... tandai='^',
```

```
...s=50)
... axarr[idx[0], idx[1]].scatt(X_tra_std[y_tra==1, 0],
... X_tra_std[y_tra==1, 1],
... c='merah',
... 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 ensemble tampaknya merupakan hibrida dari wilayah keputusan dari pengklasifikasi individual. Sekilas, batas keputusan suara mayoritas tampak sangat mirip dengan batas keputusan dari pengklasifikasi k-tetangga terdekat. Akan tetapi, kita dapat melihat bahwa hal tersebut ortogonal terhadap sumbu y untuk lebar sepal ≥1, sama seperti tunggul pohon keputusan:



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

```
>>> mv_clf.get_params()
{'decisiontreeclassifier': DecisionTreeClassifier(class_weight=None,
criteria='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 criteria': 'entropy',
[...]
'decisiontreeclassifier random_state': 0,
'decisiontreeclassifier splitter': 'terbaik',
'pipeline-1': Pipeline(langkah=[('sc', StandardScaler(salin=Benar, dengan_ rata-rata=Benar,
dengan_std=Benar)), ('clf', LogisticRegression(C=0,001, bobot_kelas=Tidak_ada, dual=Salah,
fit_intercept=Benar,
intercept_scaling=1, max_iter=100, multi_kelas='ovr',
penalti='l2', random_state=0, solver='liblinear', tol=0,0001,
verbose=0))]),
'pipeline-1 clf': LogisticRegression(C=0,001, bobot_kelas=Tidak_ada,
dual=Salah, fit_intercept=Benar,
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': Tidak Ada, 'pipeline-1 clf dual': Salah,
[...]
'pipeline-1 sc with_std': Benar,
'pipeline-2': Pipeline(langkah=[('sc', StandardScaler(salin=Benar,
dengan_mean=Benar, dengan_std=Benar)), ('clf',
KNeighborsClassifier(algoritma='au to', ukuran_daun=30, metrik='minkowski',
parameter_metrik=Tidak Ada, 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, kini Anda tahu cara mengakses atribut pengklasifikasi individual. Mari bekerja dengan parameter regularisasi terbalik C dari pengklasifikasi regresi logistik dan kedalaman pohon keputusan melalui pencarian grid untuk tujuan demonstrasi. Mari kita lihat:

```
>>> dari 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, kita dapat mencetak berbagai kombinasi nilai parameter hiper dan skor R_C AC rata-rata yang dihitung melalui validasi silang 10 kali lipat. Kodenya adalah sebagai berikut:

```
>>> untuk params, mean_sc, skor dalam grid.grid_sc_:
... cetak("%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('Parameter terbaik: %s' % gd.best_ps_) Parameter terbaik: {'p1 cl C': 100,0,
'dtreeclassifier ma_depth': 1}
>>> print('Akurasi: %.2f' % gd.best_sc_)
Akurasi: 1,00
```

Latihan Soal

1. Jelaskan cara menggabungkan berbagai model secara rinci.
2. Apa tujuan dan manfaat menggabungkan model?

DAFTAR PUSTAKA

Aha, D. W. (1997). *Lazy learning*. Kluwer Academic Publishers.

Alpaydin, E. (2020). *Introduction to machine learning* (4th ed.). MIT Press.

Bakar, A. A., & Ibrahim, H. (2017). *Introduction to machine learning with Python*. Springer.

Benz, A. (2018). *Hands-on machine learning with scikit-learn, keras, and tensorflow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5-32. https://doi.org/10.1023/A:1010933404324

Chollet, F. (2017). *Deep learning with Python*. Manning Publications.

Chollet, F., & Allaire, J. J. (2018). *Deep learning with R*. Manning Publications.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification* (2nd ed.). Wiley.

Fei-Fei, L., & Li, J. (2016). *Computer vision: Algorithms and applications*. Springer.

Fitting, M. (2013). *Foundations of machine learning*. MIT Press.

Goldstein, M., & Taleb, N. (2012). *Machine learning in risk analysis and management*. Wiley.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning* (2nd ed.). Springer.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786), 504-507. https://doi.org/10.1126/science.1127647

Horvitz, E. J., & Breese, J. S. (1993). *Decision theory for artificial intelligence*. Elsevier.

Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245), 255-260. https://doi.org/10.1126/science.aaa8415

Joulin, A., Grave, E., Mikolov, T., & Ranzato, M. (2017). Bag of tricks for efficient text classification. *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 427-431.

Karpathy, A. (2016). *CS231n: Convolutional neural networks for visual recognition*. Stanford University.

Kelleher, J. D., & Tierney, B. (2015). *Machine learning: An applied approach*. Springer.

Kelleher, J. D., Mac Carthy, M., & Namee, B. M. (2015). *Fundamentals of machine learning for predictive data analytics: Algorithms, worked examples, and case studies*. MIT Press.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. *Proceedings of the International Conference on Learning Representations (ICLR)*, 1-15.

Kolassa, J. E. (2016). *Machine learning in business: An introduction to the fundamentals*. Apress.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Proceedings of Neural Information Processing Systems (NIPS)*, 1097-1105.

Landwehr, N., & Pfahringer, B. (2003). *Ensemble methods: A review of ensemble methods in machine learning*. Springer.

Li, X., & Hoi, S. C. (2014). *Machine learning in big data*. Springer.

Liu, Y., & Lee, W. S. (2018). *Machine learning: A constraint-based approach*. Elsevier.

Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2018). *Foundations of machine learning* (2nd ed.). MIT Press.

Murphy, K. P. (2012). *Machine learning: A probabilistic perspective*. MIT Press.

Ng, A. Y. (2004). Feature selection, L1 vs. L2 regularization, and rotational invariance. *Proceedings of the Twenty-First International Conference on Machine Learning*, 78.

Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10), 1345-1359. https://doi.org/10.1109/TKDE.2009.191

Patterson, J. (2019). *Deep learning: A practitioner's approach*. O'Reilly Media.

Patton, M. Q. (2015). *Machine learning methods in economics and finance*. Wiley.

Pereira, F., & Glaser, J. (2016). Machine learning and computational biology. *Nature Reviews Genetics*, 17(7), 451-463. https://doi.org/10.1038/nrg.2016.50

Quionero-Candela, J., Sugiyama, M., & Cortes, C. (2009). *Dataset shift in machine learning*. Springer.

Rani, P., & Sahu, S. (2021). *Hands-on machine learning with R: A practical guide for beginners*. Wiley.

Rojas, R. (1996). *Neural networks: A systematic introduction*. Springer.

Russell, S., & Norvig, P. (2016). *Artificial intelligence: A modern approach* (3rd ed.). Pearson Education.

Shalev-Shwartz, S. (2012). *Online learning and online convex optimization*. Foundations and Trends in Machine Learning, 4(2), 109-264. https://doi.org/10.1561/2200000033

Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge University Press.

Sun, X., & Zhang, J. (2018). *Deep learning for time-series analysis*. Springer.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press.

Tibshirani, R., Hastie, T., & Friedman, J. (2001). *The elements of statistical learning: Data mining, inference, and prediction*. Springer.

Van Der Maaten, L., & Hinton, G. (2008). Visualizing high-dimensional data using t-SNE. *Journal of Machine Learning Research*, 9, 2579-2605.

Vapnik, V. (1995). *The nature of statistical learning theory*. Springer.

Vellido, A., Martìn-Guerrero, J. D., & Ocampo, P. (2012). Machine learning in bioinformatics. *Briefings in Bioinformatics*, 13(3), 336-344. https://doi.org/10.1093/bib/bbr029

Witten, I. H., Frank, E., & Hall, M. A. (2016). *Data mining: Practical machine learning tools and techniques* (4th ed.). Morgan Kaufmann.

Zhang, C., & Ma, Y. (2012). *Ensemble machine learning: Methods and applications*. Springer.

Zhang, T., & Oles, F. J. (2001). Text classification based on a discriminative model. *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, 1001-1008.

Zhang, Z., & Zheng, Y. (2016). *Deep learning: A comprehensive overview*. Springer.

Dr. (c) Migunani, S.Kom, M.Kom.

Penerapan Algoritma Machine Learning dengan Python

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