BELAJAR MACHINE LEARNING

DENGAN PYTHON DAN LIBRARY SCIKIT-LEARN

Table of Contents

◆ Kata Pengantar

Kepada para pembaca yang terhormat,

Selamat datang di perjalanan yang menakjubkan ke dunia Machine Learning dengan python dan liblary Scikit-Learn! Saya sangat senang dapat berbagi pengetahuan dan pengalaman saya dalam bidang ini melalui E-Book ini kepada anda,sebagai panduan lengkap untuk mempelajari dasar-dasar Machine Learning dan menerapkannya dengan mudah dengan menggunakan bahasa pemrograman Python dan liblary Scikit-Learn yang kuat.

Machine Learning telah menjadi pusat perhatian dalam dunia teknologi modern,memungkinkan kitta untuk mengungkap pola-pola yang tersembunyi dalam data dan membuat prediksi yang cerdas.Dengan menggunakan Python,bahasa pemrograman yang populer dan mudah di pelajari,serta liblary Scikit-Learn yang kaya fitur.Anda akan memperoleh keterampilan yang di perlukan untuk membangun model Machine Learning yang efektif dan berkinerja tinggi.

Dalam E-Book ini,Anda akan diajak melalui perjalanan dari pemahaman dasar tentang apa itu Machine Learning hingga penerapan teknik-teknik kelas atas dalam liblary Scikit-Learn.Setiap konsep akan di jelaskan dengan jelas dan disertai dengan contoh kode yang praktis,seingga Anda dapat langsung mulai mengaplikasikan pengetahuan yang anda pelajari.

Saya yakin E-Book ini akan menjadi akan menjadi panduan yang berharga bagi siapa saja yang tertarik untuk memulai atau meningkatkan pemahaman mereka tentang Machine Learning.Mulailah petualangan Anda sekarang dan temukan potensi yang tak terbatas dari Machine Learning dengan Python dan liblary Scikit-Learn.

Terima kasih telah memilih E-Book ini sebagai sumber belajar Anda.Selamat membaca dan selamat mengeksplorasi dunia Machine Learning yang menakjubkan!

◆ Instalasi dan Persiapan Lingkungan

Python

Python adalah bahasa pemrograman yang populer dan serbaguna. Anda dapat mengunduh installer Python dari situs web resmi dan mengikuti petunjuk instalasinya.

- Situs web resmi Python: [python.org](https://www.python.org/)

Windows:

1. Unduh installer Python dari situs web resmi.

2. Buka installer dan ikuti petunjuk instalasinya.

3. Pastikan untuk menambahkan Python ke PATH.

MacOS dan Linux:

1. Sebagian besar distribusi Linux dan MacOS sudah menyertakan Python.

2. Jika tidak ada, Anda dapat menginstalnya menggunakan manajer paket yang sesuai dengan distribusi Anda.

Cara memeriksa instalasi Python Anda:

- Buka terminal atau command prompt.

- Ketikkan `***python --version***`. Anda seharusnya melihat versi Python yang terinstal.

Scikit-learn (SKLearn)

SKLearn adalah pustaka Machine Learning yang populer dalam Python. Anda dapat menginstalnya menggunakan pip, manajer paket Python.

Cara instalasi:

```
$  pip install scikit-learn
```

Matplotlib

Matplotlib digunakan untuk membuat visualisasi data. Anda dapat menginstalnya menggunakan pip.

Cara instalasi:

```
$ pip install matplotlib
```

Pandas

Pandas adalah pustaka Python yang berfungsi untuk manipulasi dan analisis data. Anda dapat menginstalnya menggunakan pip.

Cara instalasi:

```
$ pip install pandas
```

NumPy

NumPy adalah pustaka dasar untuk komputasi numerik dalam Python. Banyak pustaka lain, termasuk SKLearn, menggunakan NumPy di bawah kepanjangnya.

Cara instalasi:

```
$ pip install numpy
```

Jupyter Notebook (Opsional)

Jupyter Notebook adalah lingkungan pengembangan interaktif yang sangat berguna untuk eksplorasi data dan percobaan dengan kode Python. Anda dapat menginstalnya menggunakan pip.

Cara instalasi:

```
$ pip install jupyter
```

◆ Bab 1 : Pengenalan Machine Learning

◆ Apa itu Machine Learning?

Dalam bab ini, kita akan menjelajahi beberapa konsep dasar yang penting untuk dipahami dalam Machine Learning. Yuk, mari kita mulai sesi pembelajaran kita.

Topik pembahasan kita kali ini adalah pengenalan Machine Learning. Pertanyaan dasar yang sering muncul adalah, apa sebenarnya Machine Learning itu?

Terdapat beberapa definisi yang berkaitan dengan Machine Learning. Mari kita lihat satu per satu:

- “Machine learning is a study that gives computers the ability to learn without being explicitly programmed.” – **Arthur Samuel**

  definisi ini dikemukakan oleh seorang computer Scientist bernama arthur Samuel.

  Disini Arthur Samuel berpendapat bahwa Machine-Learning itu adalah suatu bidang studi atau bidang keilmuan yang memampukan suatu komputer untuk belajar tanpa secara eksplisit kita program.

- “Machine Learning is a program can be said to learn from experience ***E*** with respect to some class of tasks ***T*** and performance ***P***,if its performance tasks in ***T***,as measured by ***P***,improves with experience E.” - **Tom Mitchell**

  Definisi ini dikemukakan oleh seorang computer Scientist yang lain bernama Tom Mitchell disini Tom berpendapat bahwa yang namanya machine-learning itu adalah sebuah program yang belajar dari suatu pengalaman atau experience ***E*** terhadap suatu tasks atau suatu tugas tertentu, dimana performanya bisa diukur sebagai ***P*** ,dimana nilai performa ini akan terus meningkat seiring dengan peningkatan experience atau pengalaman dari si program tadi dalam menyelesaikan tugas tadi.

Definisi di atas merupakan definisi Machine-Learning yang cukup umum ditemui. Kalau kita mau Sederhanakan dalam bahasa Indonesia Kita juga bisa definisikan :

- Machine Learning adalah bidang ilmu yang memampukan suatu program komputer untuk belajar dari sekumpulan data.

◆ Learning Problem

Learning problem dalam konteks Machine Learning melibatkan suatu dataset yang terdiri dari sejumlah **n** sampel data untuk melakukan prediksi terhadap properti yang tidak diketahui pada dataset lain yang serupa.

Secara umum learning problem dapat dibagi menjadi dua yaitu **supervised learning** dan **unsupervised learning.**

Pada **supervised learning problem** program memprediksi suatu nilai output untuk suatu input berdasarkan proses **learning** dengan memanfaatkan sekumpulan data yang terbagi dalam dua label, yaitu input and output. Disini program lakukan **learning** dari sekumpulan contoh kasus (examples) yang menyertakan “jawaban benar”.

Pada and **unsupervised learning problem,** program tidak melakukan **learning** dari label data, melainkan mencoba menemukan pola (patterns) pada data. Di sini program melakukan **learning** dari sekumpulan contoh kasus (example) tanpa disertai “jawaban benar”.

Konsep terkait **learning problem** ini butuh waktu untuk mengendap agar benar-benar bisa dipahami nanti Seiring berjalannya waktu sembari kita mempelajari machine-learning lebih lanjut Adakalanya kalian butuh untuk merevisi kembali bab ini, tetapi untuk saat ini kita coba lanjut dulu pembelajaran selanjutnya.

◆ Terminologi Dasar pada Supervised Learning

Disini kita akan mempelajari tentang beberapa terminologi dasar pada **Supervised Learning**.

Output :

Istilah output ini dikenal dengan beberapa istilah lain beberapa diantaranya adalah:

1. label,
2. respons variabel,
3. dependent variable,
4. regressand,
5. criterion variabel,
6. measured variabel,
7. responding variabel,
8. explained variabel,
9. outcome variabel,
10. eksperimental variabel,
11. output variabel.

Jadi kalian gak usah bingung ya kalau kalian menemui istilah-istilah semacam ini istilah-istilahnya memang banyak tetapi semuanya ini mengacu ke satu hal yang sama.

Input :

Serupa dengan output, input juga mengenal beberapa istilah-istilah lain beberapa diantaranya adalah:

1. features,
2. explanatory variables ,
3. predictors,
4. regressors,

5. controlled variables,
6. exposure variables.

Jadi sama di sini istilahnya memang ada banyak sekali, tetapi semua istilah ini mengacu pada hal yang sama yaitu input.

Training Set :

Training set ini Merupakan sekumpulan “contoh” yang digunakan untuk proses **learning (training)** pada **supervised learning.**

Test Set :

Test set disini akan mengacu pada sekumpulan “contoh” yang digunakan untuk mengukur performa pada **supervised learning**.

Di sini terdapat suatu statement yang sangat penting pada supervised learning **respons variabel** bisa dianalogikan sebagai **kunci jawaban** dan **explanatory variables** bisa dianalogikan sebagai **pertanyaannya.**

◆ Supervised Machine Learning Tasks

Secara umum terdapat dua jenis Supervise Machine Learning Tasks yaitu, **Classification tasks** dan **Regression tasks**.

Classification tasks, menekankan pada proses **learning (training)** untuk melakukan prediksi nilai discrete (i.e.,category,class,label) dari sejumlah features. Nilai discrete ini biasanya akan mengacu pada category,class ataupun label.

Regression tasks, menekankan pada proses **learning (training)** untuk melakukan prediksi nilai continuos (floating-point), dari sejumlah features.

Atau dengan kata lain kalau kita mau Sederhanakan, **Classification tasks** ini makan berfokus pada proses pengelompokan kategori, sedangkan **Regression tasks** ini akan berfokus pada proses prediksi nilai floating-point atau continuos

◆ Unsupervised Machine Learning Tasks

Terdapat beberapa **Unsupervised Machine Learning Tasks,** dan kita akan berfokus pada dua tasks yaitu **Clustering** dan **Dimensionality Reduction**.

Clustering tasks, menekankan pada proses eksplorasi data untuk menemukan kelompok (grup) pada data berdasarkan kemiripan (kedekatan) karakteristik.

Dimensionality Reduction, menekankan pada proses eksplorasi data untuk menemukan sejumlah features yang paling berdampak terhadap response variable.

◆ Rekomendasi buku

Dalam pembelajaran Machine Learning ini kita mencoba untuk menyeimbangkan antara materi yang bersifat konseptual Dan juga materi yang sifatnya practical, hanya saja kita akan tetap lebih menitikberatkan pada aspek practical nya.

Ini bukan berarti aspek konseptualnya tidak penting dalam Machine Learning, sebenarnya baik aspek konseptual maupun aspek practical ini sangat penting tetapi disini untuk menyimbangkan keduanya tidaklah mudah.

Oleh karenanya untuk memperlengkapi pemahaman kalian akan Machine Learning kita merekomendasikan Buku yang menurut kita sangat bagus sekali, dan kita yakin sekali Buku ini akan dapat memperkaya pemahaman kalian terkait Machine Learning.

- **Machine Learning Yearning** (by **Andrew Ng**)

Buku pertama yang kita rekomendasikan adalah **Machine Learning Yearning** yang ditulis oleh Profesor **Andrew Ng,** buku ini termasuk buku yang sangat baik sekali untuk memberikan pengantar konseptual terkait Machine Learning,buku ini bisa diperoleh dengan gratis di https://info.deeplearning.ai/machine-learning-yearning-book .

Buku ini juga termasuk ramah bagi pemula dan setiap konsep Machine Learning yang disampaikan benar-benar dikemas dengan sangat sederhana sekali.

- **The Hundred-Page Machine Learning Book** (by **Andriy Burkov**)

Buku kedua yang bisa kita rekomendasikan adalah **The Hundred-Page Machine Learning Book** yang ditulis oleh **Andriy Burkov,** buku ini juga cukup menarik dan cukup padat hanya saja penekanannya sedikit berbeda dengan buku yang pertama, buku kedua ini mencoba untuk membahas beberapa model yang memang umum dan populer dalam bidang Machine Learning.

Karena penekanannya disini adalah pembahasan terhadap beberapa model Machine Learning maka buku ini juga sangat erat dengan notasi matematika, oleh karenanya untuk dapat memahami buku ini dengan baik setidaknya kalian membutuhkan dasar matematika yang cukup baik pula.

Buku ini masuk dalam daftar rekomendasikan kita, karena kita melihat bahwa buku ini mampu membahas setiap model Machine Learning yang populer dengan sangat sederhana dan to the point.

- **Learning From Data** (by **Yaser S.Abu-Mostafa**)

Buku ketiga yang kita rekomendasikan adalah **Learning From Data** yang ditulis oleh **Yaser S.Abu-Mostafa,** buku ini bisa dibilang sebagai buku terbaik bagi teman-teman yang memang sedang menjalani perkuliahan atau mengikuti perkuliahan terkait Machine Learning.

Pemaparan konsep Machine Learning yang disampaikan dalam buku ini benar-benar sangat step-by-step dan sangat bertahap sekali, buku ini juga banyak diadopsi oleh banyak Universitas di dunia sewaktu mereka menyelenggarakan perkuliahan Machine Learning.

Jadi kita sangat-sangat merekomendasikan sekali bagi kalian yang memang mau serius untuk mendalami dan mempelajari Machine Learning untuk belajar juga dari buku yang satu ini.

◆ Kemampuan Teknis Yang Perlu Dipersiapkan

Untuk bisa mengikuti materi yang disampaikan dalam seri pembelajaran ini terdapat

sejumlah kemampuan teknis yang perlu kalian persiapkan.

Pemrograman Python

Kemampuan teknis yang pertama adalah pemahaman terkait pemrograman Python.

Kalian harus memahami tentang pemerograman Python terutama dalam teknik penulisan idiomatik, penulisan kode Python idiomatik ini sangatlah penting untuk menjadikan Python code yang kita tulis menjadi lebih sederhana dan juga lebih readable.

Struktur Data Python

Yang perlu diperhatikan adalah terkait built-in data structure dalam pemrograman Python,disini setidaknya kalian harus punya pemahaman yang baik terkait list, tupple, dictionary, dan juga set.

Python Pandas

Kemampuan teknis berikutnya yang perlu dipersiapkan adalah pemahaman terkait Python Pandas data frame, dalam beberapa kasus di seri pembelajaran Machine Learning ini kita akan menggunakan Pandas Data Frame.

Jadi bagi kalian yang masih awam dengan Python Pandas Data Frame, kita sangat menyarankan kalian untuk mempersiapkan diri terlebih dahulu dan mempelajarinya.

Python Matplotlib

Kemampuan teknis berikutnya perlukan mempersiapkan adalah terkait visualisasi data dengan Python Matplotlib, karena dalam seri pembelajaran ini kita juga akan banyak sekali memanfaatkan Matplotlib sebagai modul dasar untuk melakukan visualisasi data dalam pemrograman Python.

Statistika Deskriptif

Karena Machine Learning ini sangat erat hubungannya dengan statistika dan probabilitas Kalian juga kita himbau untuk membenahi dasar statistika dan juga probabilitas.

Demikian beberapa kemampuan teknis yang perlu kalian persiapkan untuk dapat mengikuti materi pembelajaran Machine Learning ini dengan baik.

Demikian sesi pembelajaran terkait pemahaman dan beberapa konsep dasar dalam bidang Machine Learning yang perlu kalian ketahui, semoga materi pembelajaran yang disampaikan dapat membantu kalian dalam mempelajari dasar-dasar Machine Learning.

Bab 2 : Sample Dataset Pada Scikit-Learn

Pengenalan

Dalam bab ini, kita akan menjelaskan tentang sampel dataset yang disediakan oleh modul Scikit-Learn. Scikit-Learn adalah perpustakaan Python yang populer untuk Machine Learning, yang menyediakan beragam algoritma dan dataset untuk mempermudah pengembangan model.

Dataset Klasik pada Scikit-Learn

Scikit-Learn menyertakan sejumlah dataset klasik yang umum digunakan dalam pembelajaran statistika dan Machine Learning. Dataset ini berguna untuk menguji dan mengembangkan model, serta untuk mendemonstrasikan berbagai teknik analisis data.

Cara Mengakses Dataset

Salah satu cara untuk mengakses dataset pada Scikit-Learn adalah dengan menggunakan fungsi **load_iris()**,yang di sertakan untuk mengambil dataset iris.Berikut adalah langkah-langkah nya:

```
from sklearn.datasets import load_iris

iris = load_iris()
print(iris)
```

Pada contoh di atas,kita mengimport fungsi **load_irs()** dari modul **sklearn.datasets**,kemudian memanggilnya untuk mendapatkan dataset iris.Dataset tersebut kemudian di simpan dalam variabel **iris**.

Tampilan Dataset

Setelah kita memiliki dataset iris dalam variabel **iris**,kita datap menampilkan isi dari variabel tersebut untuk memeriksa informasi yang tersedia dalam dataset tersebut.

```
iris = load_iris()
```

```
print(iris)
```

Dengan menjalankan kode di atas, kita dapat melihat informasi mengenai dataset iris yang telah dimuat.

Berikutnya di sini kita coba tampilkan isi dari variabel **iris** nya, kita coba eksekusi kode tersebut dan hasilnya akan seperti ini

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
```

```
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
       [6.3, 3.3, 4.7, 1.6],
       [4.9, 2.4, 3.3, 1. ],
       [6.6, 2.9, 4.6, 1.3],
       [5.2, 2.7, 3.9, 1.4],
       [5. , 2. , 3.5, 1. ],
       [5.9, 3. , 4.2, 1.5],
       [6. , 2.2, 4. , 1. ],
       [6.1, 2.9, 4.7, 1.4],
       [5.6, 2.9, 3.6, 1.3],
       [6.7, 3.1, 4.4, 1.4],
       [5.6, 3. , 4.5, 1.5],
       [5.8, 2.7, 4.1, 1. ],
       [6.2, 2.2, 4.5, 1.5],
       [5.6, 2.5, 3.9, 1.1],
       [5.9, 3.2, 4.8, 1.8],
       [6.1, 2.8, 4. , 1.3],
       [6.3, 2.5, 4.9, 1.5],
       [6.1, 2.8, 4.7, 1.2],
       [6.4, 2.9, 4.3, 1.3],
       [6.6, 3. , 4.4, 1.4],
       [6.8, 2.8, 4.8, 1.4],
       [6.7, 3. , 5. , 1.7],
       [6. , 2.9, 4.5, 1.5],
       [5.7, 2.6, 3.5, 1. ],
       [5.5, 2.4, 3.8, 1.1],
       [5.5, 2.4, 3.7, 1. ],
       [5.8, 2.7, 3.9, 1.2],
       [6. , 2.7, 5.1, 1.6],
       [5.4, 3. , 4.5, 1.5],
```

```
       [6. , 3.4, 4.5, 1.6],
       [6.7, 3.1, 4.7, 1.5],
       [6.3, 2.3, 4.4, 1.3],
       [5.6, 3. , 4.1, 1.3],
       [5.5, 2.5, 4. , 1.3],
       [5.5, 2.6, 4.4, 1.2],
       [6.1, 3. , 4.6, 1.4],
       [5.8, 2.6, 4. , 1.2],
       [5. , 2.3, 3.3, 1. ],
       [5.6, 2.7, 4.2, 1.3],
       [5.7, 3. , 4.2, 1.2],
       [5.7, 2.9, 4.2, 1.3],
       [6.2, 2.9, 4.3, 1.3],
       [5.1, 2.5, 3. , 1.1],
       [5.7, 2.8, 4.1, 1.3],
       [6.3, 3.3, 6. , 2.5],
       [5.8, 2.7, 5.1, 1.9],
       [7.1, 3. , 5.9, 2.1],
       [6.3, 2.9, 5.6, 1.8],
       [6.5, 3. , 5.8, 2.2],
       [7.6, 3. , 6.6, 2.1],
       [4.9, 2.5, 4.5, 1.7],
       [7.3, 2.9, 6.3, 1.8],
       [6.7, 2.5, 5.8, 1.8],
       [7.2, 3.6, 6.1, 2.5],
       [6.5, 3.2, 5.1, 2. ],
       [6.4, 2.7, 5.3, 1.9],
       [6.8, 3. , 5.5, 2.1],
       [5.7, 2.5, 5. , 2. ],
       [5.8, 2.8, 5.1, 2.4],
       [6.4, 3.2, 5.3, 2.3],
       [6.5, 3. , 5.5, 1.8],
       [7.7, 3.8, 6.7, 2.2],
       [7.7, 2.6, 6.9, 2.3],
       [6. , 2.2, 5. , 1.5],
       [6.9, 3.2, 5.7, 2.3],
       [5.6, 2.8, 4.9, 2. ],
       [7.7, 2.8, 6.7, 2. ],
       [6.3, 2.7, 4.9, 1.8],
       [6.7, 3.3, 5.7, 2.1],
       [7.2, 3.2, 6. , 1.8],
       [6.2, 2.8, 4.8, 1.8],
       [6.1, 3. , 4.9, 1.8],
       [6.4, 2.8, 5.6, 2.1],
       [7.2, 3. , 5.8, 1.6],
       [7.4, 2.8, 6.1, 1.9],
       [7.9, 3.8, 6.4, 2. ],
       [6.4, 2.8, 5.6, 2.2],
       [6.3, 2.8, 5.1, 1.5],
```

```
       [6.1, 2.6, 5.6, 1.4],
       [7.7, 3. , 6.1, 2.3],
       [6.3, 3.4, 5.6, 2.4],
       [6.4, 3.1, 5.5, 1.8],
       [6. , 3. , 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
       [6.7, 3.1, 5.6, 2.4],
       [6.9, 3.1, 5.1, 2.3],
       [5.8, 2.7, 5.1, 1.9],
       [6.8, 3.2, 5.9, 2.3],
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]]), 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]), 'frame': None, 'target_names':
array(['setosa', 'versicolor', 'virginica'], dtype='<U10'), 'DESCR': '..
_iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data Set
Characteristics:**\n\n    :Number of Instances: 150 (50 in each of three
classes)\n    :Number of Attributes: 4 numeric, predictive attributes and the
class\n    :Attribute Information:\n        - sepal length in cm\n        - sepal
width in cm\n        - petal length in cm\n        - petal width in cm\n        -
class:\n                - Iris-Setosa\n                - Iris-Versicolour\n
- Iris-Virginica\n                \n    :Summary Statistics:\n\n
============== ==== ==== ======= ===== ====================\n
Min  Max   Mean    SD   Class Correlation\n    ============== ==== ==== =======
===== ====================\n    sepal length:   4.3  7.9   5.84   0.83    0.7826\
n    sepal width:    2.0  4.4   3.05   0.43   -0.4194\n    petal length:   1.0
6.9   3.76   1.76    0.9490  (high!)\n    petal width:    0.1  2.5   1.20   0.76
0.9565  (high!)\n    ============== ==== ==== ======= ===== ====================\
n\n    :Missing Attribute Values: None\n    :Class Distribution: 33.3% for each
of 3 classes.\n    :Creator: R.A. Fisher\n    :Donor: Michael Marshall (MARSHALL
%PLU@io.arc.nasa.gov)\n    :Date: July, 1988\n\nThe famous Iris database, first
used by Sir R.A. Fisher. The dataset is taken\nfrom Fisher\'s paper. Note that
it\'s the same as in R, but not as in the UCI\nMachine Learning Repository, which
has two wrong data points.\n\nThis is perhaps the best known database to be found
in the\npattern recognition literature.  Fisher\'s paper is a classic in the
field and\nis referenced frequently to this day.  (See Duda & Hart, for example.)
The\ndata set contains 3 classes of 50 instances each, where each class refers to
```

```
a\ntype of iris plant.  One class is linearly separable from the other 2; the\
nlatter are NOT linearly separable from each other.\n\n.. topic:: References\n\n
- Fisher, R.A. "The use of multiple measurements in taxonomic problems"\n
Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to\n
Mathematical Statistics" (John Wiley, NY, 1950).\n   - Duda, R.O., & Hart, P.E.
(1973) Pattern Classification and Scene Analysis.\n     (Q327.D83) John Wiley &
Sons.  ISBN 0-471-22361-1.  See page 218.\n   - Dasarathy, B.V. (1980) "Nosing
Around the Neighborhood: A New System\n     Structure and Classification Rule for
Recognition in Partially Exposed\n     Environments".  IEEE Transactions on
Pattern Analysis and Machine\n     Intelligence, Vol. PAMI-2, No. 1, 67-71.\n   -
Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions\n
on Information Theory, May 1972, 431-433.\n   - See also: 1988 MLC Proceedings,
54-64.  Cheeseman et al"s AUTOCLASS II\n     conceptual clustering system finds 3
classes in the data.\n   - Many, many more ...', 'feature_names': ['sepal length
(cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)'], 'filename':
'iris.csv', 'data_module': 'sklearn.datasets.data'}
```

kalau kita perhatikan strukturnya isi dari variabel **`iris`** atau data yang ditampung dalam variabel **`iris`** ini memiliki struktur dictionary.

Dictionary sendiri merupakan struktur data penting dalam pemrograman Python yang terdiri dari key ***n*** values. Untuk pemula, kita merekomendasikan untuk memahami konsep dictionary dengan mempelajari seri pembelajaran tentang struktur data Python.

Berikutnya kita akan mencoba mendata keys apa saja yang miliki dictionary ini,caranya kita tinggal panggil saja **`iris.keys()`**

```
from sklearn.datasets import load_iris

iris = load_iris()
print(iris..keys())
```

dan ini hasil nya

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names',
'filename', 'data_module'])
```

Kita bisa lihat keys yang tersedia adalah ***data,target,frame,target_names, feature_names, DESCR, filename, dan juga data_module***.

Keys ***data*** ini akan berasosiasi dengan data-data features,sedangkan ***target*** akan berasosiasi dengan data target,lalu ***frame*** ini untuk menandakan apakah data formatnya berupa data frame atau bukan,by default data formatnya adalah numpy array,di akhir sesi pembelajaran ini kita akan demokan juga cara untuk meload datasets ini dalam format python pandas data frame.

Berikutnya disini ada ***target_names*** dan ***feature_names***,ini akan berkolerasi dengan nama atau label untuk setip features dan juga untuk target nya.

Berikutnya ada ***DESCR,*** ini merupakan kependekan dari description atau deskripsi,dimana kita akan menggunakan keys ini untunk memgakses deskripsi dari datasets nya.

Kemudian ada keys ***filename,***ini akan berasosiasi dengan lokasi dari file datasets ini.

◆ Membaca Metadata dari Dataset

Langkah pertama setelah memperoleh dataset adalah membaca dan memahami deskripsi dari datasets tersebut. Berikut adalah cara untuk mengakses metadata atau deskripsi dari sampel dataset yang disediakan oleh Scikit-Learn.

Disini dataset yang kia gunakan adalah iris datasets,bagi kalian yang penasaran dengan iris datasets,kita juga akan menyertakan refrensi untuk mengakses dan mempelajari iris datasets ini lebih lanjut.

Refrensi: https://en.wikipedia.org/wiki/Iris_flower_data_set

Untuk mengakses metadata atau deskripsi dari sample datasets yang di sertakan oleh Scikit-Learn,caranya cukup mudah disini kita tinggal panggil saja,***iris.DESCR.***

Berikut kodenya :

```
from sklearn.datasets import load_iris

iris = load_iris()
print(iris.DESCR)
```

Kemudian kita coba print hasilnya nya.

```
.. _iris_dataset:

Iris plants dataset
--------------------
```

```
**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
```

```
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
- Many, many more ...
```

Output dari kode di atas akan berisi informasi tentang dataset iris, termasuk judul, jumlah baris, jumlah atribut, keterangan atribut, dan kelas-kelas yang tersedia.

Dibagian paling atas ada keterangan terkait dengan judul atau nama dari datasets nya,untuk kasus kita kali ini nama datasets nya adalah ***Iris plants dataset.***

Berikutnya disini juga terdapat beberapa informasi tambahan seperti ***Number of Instances: 150 (50 in each of three classes),***ini merepresentasikan jumlah barisnya,total disini tersedia ***150 Instances*** atau 150 baris data,disini juga ada keterangan ***50 in each of three classes,***berarti nanti akan terdapat tiga class,dimana untuk setiap class nya terdapat dari 50 baris data.

```
- class :
        - Iris-Setosa
        - Iris-Versicolour
        - Iris-Virginica
```

Untuk class ini nantinya akan di kenal sebagai ***target***,karena dataset ini merupakan datasets untuk klasifikasi.

Berikutnya ada ***Number of Attributes: 4 numeric, predictive attributes and the class,***disini terdapat 4 numeric attribute.

```
:Attribute Information:
    - sepal length in cm
    - sepal width in cm
    - petal length in cm
    - petal width in cm
```

Diatas merupakan keterangan dari attribute nya,dimana untuk ke empat attribute ini di ukur dalam satuan centimeter(cm).Attribute ini juga dikenal dengan istilah ***features***.

Kemudian berikutnya terkait ***Summary Statistics,***ini berbicara mengenai statistika deskriptif dari data-data numeric nya.

```
  :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================
```

Disini ada informasi terkait dengan nilai terkecil atau Min,nilai terbesar atau Max,Mean atau rata-rata,dan juga standard (SD) definition nya.Selain itu juga di sertakan informasi terkait correlation atau korelasi dari setiap atribut ini terhadap class nya.

Berikutnya terkait dengan ***Missing Attribute Values.***

```
:Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988
```

bisa kita lihat di sini missing value nya None,artinya semua datanya lengkap tidak ada data yang kosong.

Lalu ada keterangan ***Class Distribution ,*** disini ada 33,3% for each of 3 classes,jadi distribusi nya ada sekitar 33,3% untuk setiap class nya,dimana sebenarnya detailnya sudah dijelaskan sebelumnya, dimana untuk setiap class nya terdapat 50 data.

Kemudian ada ***Creator,*** atau orang yang membuat datasets ini dimana creator nya adalah R.A. Fisher.

Lalu ada juga keterangan ***Donor***,dan juga ada keterangan waktu atau ***Date,*** terkait kapan datasets ini di bentuk.Bisa kita lihat disini,datasets ini merupakan dataset klasik yang di bentuk pada bulan ***July***, tahun ***1988***.

Berikutnya disini ada narasi tambahan atau ***deskripsi*** terkait datasets nya.

```
The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.
```

Terakhir di dataset ini juga terdapat beberapa ***References*** yang bisa pelajari lebih lanjut kalau kita mau mencari tahu terkait datasets iris ini.

```
.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
- Many, many more ...
```

Itulah kurang lebih terkait metadata yang disediakan oleh Scikit-Learn untuk setiap sample datasets nya,disini bisa dibilang bahwa datasets nya sangat lengkap sekali.

◆ Explanatory & Response Variables (Features & Target)

Setelah mempelajari deskripsi dari dataset, tahapan selanjutnya adalah mengakses **Explanatory** dan juga **Respons variables** nya.

Explanatory variables seringkali dikenal dengan istilah features, sedangkan **Respons Variables** seringkali dikenal dengan istilah target.

Berikut akan kita demokan cara untuk mengakses F**eatures & Target** dari datasets iris.

Explanatory variables(Features)

```
X= iris.data
X.shape
```

Sebelumnya sudah kita load ke dalam variabel ***iris,*** untuk mengakses **Features** atau **Explanatory Variables** nya caranya cukup mudah Kita tinggal panggil saja ***iris.data,*** disini nilai nya akan kita tampung ke dalam variabel X besar, X besar ini merupakan variabel yang umum digunakan untuk menampung **Features,**berikutnya kita akan Panggil **X.shape,** ini digunakan untuk mengakses dimensi dari datanya.

Kita coba eksekusi kodenya

```
from sklearn.datasets import load_iris

iris = load_iris()

X= iris.data
X.shape

print(X.shape)
```

Bisa kita lihat disini outputnya.

```
(150 , 4)
```

Artinya disini terdapat 150 baris dan 4 buah kolom.Kalau kita mau melihat isi dari data nya,kita tinggal print X nya saja.

```
from sklearn.datasets import load_iris

iris = load_iris()

X= iris.data
X.shape

print(X)
```

Kita coba eksekusi ulang,dan hasilnya akan tampak seperti ini.

```
  array([[5.1 3.5 1.4 0.2]
   [4.9 3.  1.4 0.2]
   [4.7 3.2 1.3 0.2]
   [4.6 3.1 1.5 0.2]
   [5.  3.6 1.4 0.2]
   [5.4 3.9 1.7 0.4]
   [4.6 3.4 1.4 0.3]
   [5.  3.4 1.5 0.2]
   [4.4 2.9 1.4 0.2]
   [4.9 3.1 1.5 0.1]
   [5.4 3.7 1.5 0.2]
   [4.8 3.4 1.6 0.2]
   [4.8 3.  1.4 0.1]
   [4.3 3.  1.1 0.1]
   [5.8 4.  1.2 0.2]
   [5.7 4.4 1.5 0.4]
   [5.4 3.9 1.3 0.4]
   [5.1 3.5 1.4 0.3]
   [5.7 3.8 1.7 0.3]
   [5.1 3.8 1.5 0.3]
   [5.4 3.4 1.7 0.2]
   [5.1 3.7 1.5 0.4]
   [4.6 3.6 1.  0.2]
   [5.1 3.3 1.7 0.5]
   [4.8 3.4 1.9 0.2]
   [5.  3.  1.6 0.2]
   [5.  3.4 1.6 0.4]
   [5.2 3.5 1.5 0.2]
   [5.2 3.4 1.4 0.2]
```

```
    [4.7 3.2 1.6 0.2]
    [4.8 3.1 1.6 0.2]
    [5.4 3.4 1.5 0.4]
    [5.2 4.1 1.5 0.1]
    [5.5 4.2 1.4 0.2]
    [4.9 3.1 1.5 0.2]
    [5.  3.2 1.2 0.2]
    [5.5 3.5 1.3 0.2]
    [4.9 3.6 1.4 0.1]
    [4.4 3.  1.3 0.2]
    [5.1 3.4 1.5 0.2]
    [5.  3.5 1.3 0.3]
    [4.5 2.3 1.3 0.3]
    [4.4 3.2 1.3 0.2]
    [5.  3.5 1.6 0.6]
    [5.1 3.8 1.9 0.4]
    [4.8 3.  1.4 0.3]
    [5.1 3.8 1.6 0.2]
    [4.6 3.2 1.4 0.2]
    [5.3 3.7 1.5 0.2]
    [5.  3.3 1.4 0.2]
    [7.  3.2 4.7 1.4]
    [6.4 3.2 4.5 1.5]
    [6.9 3.1 4.9 1.5]
    [5.5 2.3 4.  1.3]
    [6.5 2.8 4.6 1.5]
    [5.7 2.8 4.5 1.3]
    [6.3 3.3 4.7 1.6]
    [4.9 2.4 3.3 1. ]
    [6.6 2.9 4.6 1.3]
    [5.2 2.7 3.9 1.4]
    [5.  2.  3.5 1. ]
    [5.9 3.  4.2 1.5]
    [6.  2.2 4.  1. ]
    [6.1 2.9 4.7 1.4]
    [5.6 2.9 3.6 1.3]
    [6.7 3.1 4.4 1.4]
    [5.6 3.  4.5 1.5]
    [5.8 2.7 4.1 1. ]
    [6.2 2.2 4.5 1.5]
    [5.6 2.5 3.9 1.1]
    [5.9 3.2 4.8 1.8]
    [6.1 2.8 4.  1.3]
    [6.3 2.5 4.9 1.5]
    [6.1 2.8 4.7 1.2]
    [6.4 2.9 4.3 1.3]
    [6.6 3.  4.4 1.4]
    [6.8 2.8 4.8 1.4]
    [6.7 3.  5.  1.7]
```

```
    [6.  2.9 4.5 1.5]
    [5.7 2.6 3.5 1. ]
    [5.5 2.4 3.8 1.1]
    [5.5 2.4 3.7 1. ]
    [5.8 2.7 3.9 1.2]
    [6.  2.7 5.1 1.6]
    [5.4 3.  4.5 1.5]
    [6.  3.4 4.5 1.6]
    [6.7 3.1 4.7 1.5]
    [6.3 2.3 4.4 1.3]
    [5.6 3.  4.1 1.3]
    [5.5 2.5 4.  1.3]
    [5.5 2.6 4.4 1.2]
    [6.1 3.  4.6 1.4]
    [5.8 2.6 4.  1.2]
    [5.  2.3 3.3 1. ]
    [5.6 2.7 4.2 1.3]
    [5.7 3.  4.2 1.2]
    [5.7 2.9 4.2 1.3]
    [6.2 2.9 4.3 1.3]
    [5.1 2.5 3.  1.1]
    [5.7 2.8 4.1 1.3]
    [6.3 3.3 6.  2.5]
    [5.8 2.7 5.1 1.9]
    [7.1 3.  5.9 2.1]
    [6.3 2.9 5.6 1.8]
    [6.5 3.  5.8 2.2]
    [7.6 3.  6.6 2.1]
    [4.9 2.5 4.5 1.7]
    [7.3 2.9 6.3 1.8]
    [6.7 2.5 5.8 1.8]
    [7.2 3.6 6.1 2.5]
    [6.5 3.2 5.1 2. ]
    [6.4 2.7 5.3 1.9]
    [6.8 3.  5.5 2.1]
    [5.7 2.5 5.  2. ]
    [5.8 2.8 5.1 2.4]
    [6.4 3.2 5.3 2.3]
    [6.5 3.  5.5 1.8]
    [7.7 3.8 6.7 2.2]
    [7.7 2.6 6.9 2.3]
    [6.  2.2 5.  1.5]
    [6.9 3.2 5.7 2.3]
    [5.6 2.8 4.9 2. ]
    [7.7 2.8 6.7 2. ]
    [6.3 2.7 4.9 1.8]
    [6.7 3.3 5.7 2.1]
    [7.2 3.2 6.  1.8]
    [6.2 2.8 4.8 1.8]
```

```
   [6.1 3.  4.9 1.8]
   [6.4 2.8 5.6 2.1]
   [7.2 3.  5.8 1.6]
   [7.4 2.8 6.1 1.9]
   [7.9 3.8 6.4 2. ]
   [6.4 2.8 5.6 2.2]
   [6.3 2.8 5.1 1.5]
   [6.1 2.6 5.6 1.4]
   [7.7 3.  6.1 2.3]
   [6.3 3.4 5.6 2.4]
   [6.4 3.1 5.5 1.8]
   [6.  3.  4.8 1.8]
   [6.9 3.1 5.4 2.1]
   [6.7 3.1 5.6 2.4]
   [6.9 3.1 5.1 2.3]
   [5.8 2.7 5.1 1.9]
   [6.8 3.2 5.9 2.3]
   [6.7 3.3 5.7 2.5]
   [6.7 3.  5.2 2.3]
   [6.3 2.5 5.  1.9]
   [6.5 3.  5.2 2. ]
   [6.2 3.4 5.4 2.3]
   [5.9 3.  5.1 1.8]])
```

Bisa kita lihat disini terdiri dari 4 buah kolom dan total jumlah barisnya ada 150 baris data.Bisa kita lihat juga disini tipe data nya adalah array,atau lebih tepatnya adalah numpy array.

Response Variables (Target)

```
y = iris.target
y .shape
```

Untuk cara mengakses **Response Variables (Target)** ini juga cukup mudah,kita tingaal panggil saja ***iris.target* .**

Penggunaan y kecil juga cukup umum di gunakan untuk merepresentsikan target,kita juga akan mencoba menampilkan dimensinya dengan memanggil **y.shape.**

Kita coba eksekusi kode nya

```
from sklearn.datasets import load_iris
```

```
iris = load_iris()
y = iris.target
y.shape
print (y.shape)
```

dan ini dia hasilnya

```
(150,)
```

Artinya datasets ini terdiri dari 150 baris dan 1 buah kolom saja.Yang perlu di perhatikan disini adalah jumlah baris untuk features dan target nya harus sama.

Berikutnya kita akan coba tampilkan nilainya atau isinya

```
from sklearn.datasets import load_iris

iris = load_iris()
y = iris.target
y.shape
print (y)
```

Kita coba eksekusi codenya,dan hasillnya akan tampak seperti di bawah ini.

```
array([0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2])
```

Features dan target ini saling berhubungan,disini untuk features yang paling awal atau paling pertama ini kan nilainya 5.1, 3.5, 1.4, 0.2,sekumpulan features ini akan berkolerasi dengan target bernilai 0,berikutnya untuk features yang kedua nilainya 4.9, 3. , 1.4 ,0.2,satu set features ini akan berkorelasi dengan target yang nilainya 0 1 juga,dan seterusnya.

◆ Feature & Target Names

kalau kita perhatikan dari features & target yang kita akses sebelumnya,kita hanya dapat melihat sekumpulan nilai yang terkumpul dalam satu array.

Tetapi kita tidak mengetahui makna dari nilai-nilai tersebut,untuk memahami makna dari nilai-nilai tersebut,kita butuh untuk mengakses **Feature & Target Names.**

Berikut akan kita demokan cara nya.

Untuk mengakses **Feature Names** cukup mudah,kita tinggal panggil saja **iris.feature_names,** dan kita akan tampung kedalam variabel **feature_names.**

```
feature_names = iris.feature_names
featurenames
```

Kita akan coba eksekusi kodenya

```
from sklearn.datasets import load_iris

iris = load_iris()

feature_names = iris.feature_names
print(feature_names)
```

Dan ini dia hasilnya.

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

Urutan disini sangat penting,berarti untuk urutan pertama nya adalah sepal length, sepal width,baru di ikuti dengan petal length ,petal width.

Ini adak berkorelasi dengan data yang ada di **Explanatory Variable (features).** Jadi untuk data yang 5.1, 3.5, 1.4, 0.2,ini berarti 5.1 adalah sepal length,3.5 adalah sepal width,1.4 adalah petal length dan 0.2 adalah petal width.

Dengan cara demikian kita mulai bisa memahami nilai dari setiap kolom dari features nya.

Demikian juga dengan nilai dari target,untuk mendapatkan keterangan dari targetnya,kita bisa penggil **iris.target_names.**

```
target_names = iris.target_names
target_names
```

Kita coba eksekusi kodenya

```
from sklearn.datasets import load_iris

iris = load_iris()

target_names = iris.target_names
print(target_names)
```

dan ini hasilnya

```
['setosa' 'versicolor' 'virginica']
```

Kalau kita lihat disini,index 0 nya adalah sentosa,index 1 nya adalah versicolor dan index ke 2 nya adalah virginica.Karena ini merupakan array maka index nya dimulai dari 0.

Ini juga akan berkolerasi dengan **Response Variables (Target).**

◆ Visualisasi Data

Apabila di butuhkan,kita juga bisa memanfaatkan matplotlib untuk melakukan visualisasi data,kali ini kita akam demokan proses visualisasi data Sepal Length dan Sepal Width dalam bentuk scatter plot.

```
# Import library matplotlib.pyplot dengan alias plt
import matplotlib.pyplot as plt

# Melakukan pemotongan data untuk mengambil hanya dua fitur pertama
# X adalah matriks fitur, dengan baris-baris yang mewakili sampel dan kolom-kolom yang mewakili fitur-fitur
# Pemotongan ini dilakukan untuk memvisualisasikan hanya sepal length dan sepal width
X = X[:, :2]

# Menghitung nilai minimum dan maksimum dari setiap fitur untuk menentukan batas-batas sumbu x dan y
x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5

# Menggambar scatter plot dari data
# plt.scatter digunakan untuk membuat scatter plot, dengan X[:, 0] sebagai sumbu x dan X[:, 1] sebagai sumbu y
# c=y digunakan untuk memberikan warna pada setiap titik berdasarkan label kelas (y)
```

```
plt.scatter(X[:, 0], X[:, 1], c=y)

# Memberikan label pada sumbu x dan y
plt.xlabel('sepal length')  # Label sumbu x
plt.ylabel('sepal width')   # Label sumbu y

# Mengatur batas sumbu x dan y
plt.xlim(x_min, x_max)  # Batas sumbu x
plt.ylim(y_min, y_max)  # Batas sumbu y

# Menampilkan grid pada plot
plt.grid(True)

# Menampilkan plot
plt.show()
```

Di sini, kita pertama-tama mengimpor modul matplotlib.pyplot yang penting untuk membuat visualisasi data. kita mengimpornya sebagai `plt`, mempersingkat penulisan kode selanjutnya.

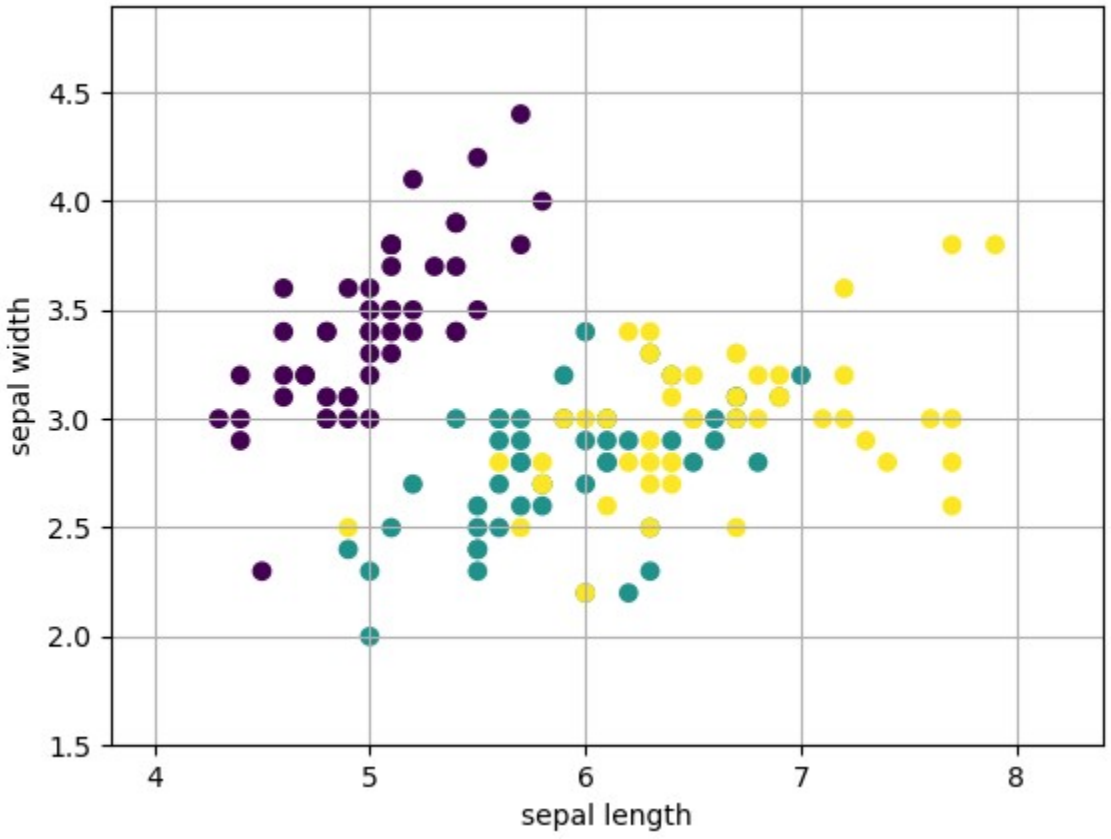
Selanjutnya, karena kita akan memvisualisasikan hanya dua fitur pertama dari data, yaitu sepal length dan sepal width, kita menggunakan slicing `X = X[:, :2]`. Ini berarti kita hanya mempertahankan dua kolom pertama dari variabel `X`.

Untuk lebih jelasnya kita coba eksekusi kodenya

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_iris

# Load iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Ambil hanya dua fitur pertama
X = X[:, :2]

# Hitung batas-batas sumbu x dan y
x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5

# Plot data
plt.scatter(X[:, 0], X[:, 1], c=y)
plt.xlabel('sepal length')
```

```
plt.ylabel('sepal width')

 # Atur batas sumbu x dan y
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)

 # Tampilkan grid
plt.grid(True)
plt.show()
```

Dan hasilnya bisa kita lihat pada gambar di bawah ini.



Disini kita bisa lihat,hanya terdapat tiga kode warna,yaitu ungu,hijau,dan kuning.Dimana masing-masing kode warna ini akan merepresentasikan class atau species dari iris nya.

Jika kalian masih merasa awam dalam memvisualisasi data dengan matplotlib,kalian bisa belajar dulu matplotlib agar bisa lebih paham lagi dengan pembelajaran ini.

◆ Training & Testing Dataset

Sebelum melakukan proses training model,umumnya dataset akan dibagi menjadi 2 bagian,yang biasa di kenal dengan **Training Set & Testing Set.**

Proses pembagiannya di lakukan secara acak,modul scikit-learn sudah memfasilitasi kita untuk melakukan proses ini dengan lebih mudah.

Berikut akan kita demokan caranya.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.3,
                                                    random_state=1 )
print(f'X train : {X_train.shape}')
print(f'X test : {X_test.shape}')
print(f'y train : {y_train.shape}')
print(f'y test : {y_test.shape}')
```

Untuk melakukan spliting atau pembagian dataset menjadi Training dan Testing set,disini pertama-tama kita import dulu function ***train_test_split***,caranya kita panggil saja ***from sklearn.model_selection import train_test_split.***

Untuk melakukan spliting dataset,caranya cukup mudah kita tinggal saja fungsinya **train_test_split,**fungsi ini akan membutuhkan 4 parameter.

Parameter pertama adalah feature nya,kedua adalah targetnya,ketiga adalah test_size nya,ini akan merepresentasikan ukuran dari testing set nya,ukutan ini mulai dari 0-1,dan ini merepresentasikan persentase,disini test_size kita set sebagai 0.3.Artinya proporsi dari test set nya adalah 30% sedangkan train set nya adalah 70%.Seperti pernah di jelaskan sebelumnya,bahwa proses spliting ini dilkukan dengan menggunakan pengacakan,tetapi kita juga mau agar proses pengacakan nya konsisten,sehingga kita bisa melakukan replikasi,oleh karena nya disini kita perlu tentukan random state number nya.

Pada scikit-learn,random state number ini di kenal dengan istilah random_state.Untuk kasus kita kali ini,kita set nilai random_state nya adalah 1.

Yang perlu kita perhatikan berikutnya adalah fungsi train_test_split ini akan menghasilkan 4 kelompok data,oleh karenanya kita disini perlu menampungnya ke dalam 4 variable.Kelompok data tersebut adalah **x_train, x_test, y_train, y_test.**

Dimana **x_train** ini akan menampung features untuk training set,lalu **x_test** ini akan menampung features untuk testing set,**y_train** akan menampung target untuk training set dan **y_test** akan menampung target untuk testing set.

Disini kita juga akan coba tampilkan **shape** atau dimensi untuk setiap dataset nya.

Kita coba eksekusi kodenya

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
iris = load_iris()

X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.3,
                                                    random_state=1 )
print(f'X train : {X_train.shape}')
print(f'X test : {X_test.shape}')
print(f'y train : {y_train.shape}')
print(f'y test : {y_test.shape}')
```

Outpunya akan tampak seperti dibawah ini.

```
X train : (105, 4)
X test : (45, 4)
y train : (105,)
y test : (45,)
```

Untuk train dataset jumlah baris nya ada 105,dan untuk testing dataset,jumlah testing nya ada 45 baris data.Dan kalau kita lihat itu sudah memenuhi proporsi,dimana 30% untuk testing dataset dan 70% untuk training dataset.

◆ Load Iris Dataset sebagai Pandas DataFrame

Diawal sesi pembelajaran ini,sempat di singgung bahwa kita bisa melakukan load sample dataset yang di sertakan oleh scikit-learn ini dalam format **Pandas DataFrame.**

Berikut akan kita demokan caranya.

```
iris = load_iris(as_frame=True)

iris_feature_df = iris.data
iris_feature_df
```

Disini kita akan coba load data yang sama,hanya saja kali ini kita akan load sebagai Pandas DataFrame.Caranya cukup mudah,kita tinggal panggil saja ***load_iris,*** hanya saja kali ini kita menyaertakan parameter tambahan yaitu ***as_frame=True,***lalu datanya kita tampung ke dalam variable ***iris,***dan berikutnya kita akan coba ambil nilai feature nya,disini nilai feature nya kita ambil dengan cara ***iris.data,*** dan kita tampung ke dalam variable **iris_feature_df.**

Sekarang kita coba eksekusi kodenya

```
from sklearn.datasets import load_iris
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

iris = load_iris()

X = iris.data
y = iris.target

iris = load_iris(as_frame=True)

iris_feature_df = iris.data
print(iris_feature_df)
```

Bisa kita lihat disini,format datanya adalah pandas dataframe dan bukan lagi numpy array.

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
..	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

[150 rows x 4 columns]

Demikian sesi pembelajaran terkait memanfaatkan sample dataset yang di sertakan oleh scikit-learn,semoga materi pembelajaran yang di sampaikan dapat membantu kalian dalam mempelajari Machine Learning.

Bab 3 : Machine Learning Workflow dengan Scikit-Learn

Dalam bab ini kita akan mengenal dan mempelajari **Machine Learning Workflow dengan Scikit-Learn.**

Persiapan Dataset

Salah satu fase yang sangat krusial dalam machine learning adalah persiapan dataset,ketika kita bekerja dengan real data di lapangan,fase ini adalah fase yang membutuhkan paling banyak waktu dan effort.

Untuk kasus disini kita coba sederhanakan prosesnya dengan memanfaatkan sample dataset yang sudah di sediakan oleh scikit-learn.

Berikut akan kita demokan prosesnya.

Load Sample Dataset Iris Dataset

```
from sklearn.datasets import load_iris

iris = load_iris()

X = iris.data
Y= iris.target
```

kita eksekusi dulu kodenya,setelah dataset nya kita load,selanjutnya kita akan bagi dataset ini menjadi dua bagian,atau istilahnya adalah kita melakukan splitting dataset.Dataset ini kita akan split menjadi Training Set & Testing Set.

Di sesi pembelajaran sebelumnya,kita juga sudah mengetahui bahwa sckit-learn memfasilitasi kita dalam proses spliting,testing dan training dataset ini.Disini kita tinggal import aja `from sklearn.model_selection import train_test_split.`

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.4,
                                                    random_state=1 )
```

Kita coba eksekusi kodenya

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

iris = load_iris()

X = iris.data
y= iris.target

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.4,
                                                    random_state=1 )


print(f'X train : {X_train.shape}')
print(f'X test : {X_test.shape}')
print(f'y train : {y_train.shape}')
print(f'y test : {y_test.shape}')
```

dan hasilnya akan tampak seperti ini

```
X train : (90, 4)
X test : (60, 4)
y train : (90,)
y test : (60,)
```

Bagi kalian yang mengalami kesulitan untuk memahami proses periapan dataset ini,kamai sangat menyarankan untuk mempelajari sesi pembelajaran sebelumnya.

Karena pada sesi pembelajaran sesi sebelumnya kita sudah membahas ini semua dengan cukup mendetail.

◆ Training Model

Setelah training dan testing set terbentuk, tahapan selanjutnya kita perlu menentukan model machine learning yang akan kita gunakan.

Pada scikit-learn model machine learning dibentuk dari kelas yang dikenal dengan istilah **estimator** atau estimator class.
Setiap estimator akan mengimplementasikan dua method utama yaitu ***fit()*** dan ***predict().*** Method **fit,** digunakan untuk melakukan training model, sedangkan metode **predict,** digunakan untuk melakukan estimasi atau prediksi dengan memanfaatkan trained model, atau model yang sudah di training sebelumnya.

Untuk kasus kita disini kita akan menggunakan ***KNeighborsClassifier*** sebagai mesin learning model,model ini akan kita training dengan memanfaatkan training set yang sudah kita persiapkan sebelumnya.

Berikut kita akan demokan caranya.

```
from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train, y_train)
```

Disini kita perlu import modelnya adat estimeter class nya,untuk kasus kita kali ini,kita akan menggunakan ***KNeighborsClassifier,*** sebagai machine learning model.

Tahapan berikutnya kita akan membentuk objek dari **class KNeighbors** ini, proses pembentukan object model Ini membutuhkan satu buah parameter yaitu **n_neighbors,** parameter ini dibutuhkan karena objek model yang mau kita bentuk ini berasal dari ***KNeighborsClassifier,*** dimana kita perlu spesifikasikan jumlah **neighbors** nya bersih sejumlah tetangganya.

Untuk kasus kita kali ini, kita set jumlah **neighbors** nya adalah 3, lalu berikutnya object model yang terbentuk akan kita tampung ke dalam variabel **model,** selanjutnya object model ini akan kita training dengan menggunakan metode **fit.**

Proses training nya nanti akan memanfaatkan proses training set, oleh karenanya di sini kita Panggil **X_train** dan **y_train.** Dimana **X_train** ini berisi sekumpulan nilai features untuk training set dan **y_train** ini akan berisi sekumpulan nilai target untuk training set .

Setelah modelnya menjalani proses training, maka model ini dikenal dengan istilah train model atau model yang sudah di training.

Kita coba eksekusi kodenya.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

iris = load_iris()

X = iris.data
y= iris.target

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.4,
                                                     random_state=1 )


model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train, y_train)

print(model)
```

outputnya akan terlihat seperti dibawah ini.

```
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.4,
                                                   random_state=1 )
```

Sebagian dari antara kalian Mungkin bertanya-tanya terkait Apa yang dimaksud dengan **KNeighbors**, kalian gak perlu kawatir karena kita akan bahas lebih detail terkait kini **KNeighbors** ini pada beberapa sesi pembelajaran selanjutnya.

◆ Evaluasi Model

Sebelumnya kita telah membagi datasest yang kita miliki menjadi dua bagian yaitu training set,dan testing set, sesuai dengan namanya training set digunakan untuk melakukan proses training model, sedangkan testing set digunakan untuk melakukan proses evaluasi atau testing performa dari model yang kita training sebelumnya.

Terdapat beberapa **metrics** yang bisa kita gunakan untuk melakukan proses evaluasi, dan disini kita akan gunakan **accuracy** sebagai metrics nya,

berikut akan kita demokan prosesnya.

```
from sklearn.metrics import accuracy_score

y_pred = model.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print(f'Accuracy: {acc}')
```

Disini sklearn atau scikit-learn sudah menyatakan sejumlah metrics untuk melakukan proses evaluasi dari suatu modelz salah satunya adalah ***accuracy_score.***

Untuk melakukan evaluasi model disini tentunya kita perlu impor dulu **accuracy_score nya,** caranya cukup panggil saja ***from sklearn.metrics import accuracy_score***, setelah metode pengukurannya kita impor, berikutnya kita akan melakukan prediksi terhadap nilai features yang ada di dalam testing dataset.
Di sini kita Panggil ***model.predict(X_test),*** dan hasil prediksinya kita akan tampung ke dalam variabel ***y_pred.*** *D*i sini kita tahu pada testing set itu selain terdapat nilai features, juga terdapat nilai targetnya, nilai target itu kan disimpan dalam variable **y_test.**

proses evaluasi ini pada dasarnya akan membandingkan nilai target yang terdapat dalam variabel **y_test** dibandingkan dengan nilai prediksi yang kita tampung ke dalam variabel **y_pred** ini,

Proses pengukurannya atau proses evaluasinya bisa bermacam-macam, tetapi kali ini kita akan demokan dengan menggunakan ***accuracy_score,*** disini kita panggil saja ***accuracy_score(y_test, y_pred),*** lalu nilai akurasinya kita tampung ke dalam variabel **acc,** berikutnya di sini kita coba print out nilai akurasinya.

Kita coba eksekusi kode nya

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

iris = load_iris()

X = iris.data
y= iris.target

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.4,
                                                    random_state=1 )


model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print(f'Accuracy: {acc}')
```

Disini ini dia ya nilai akurasi adalah 0.9833333333333333, nah disini artinya nilai akurasi dari model kita adalah 98% dan ini termasuk nilai akurasi yang sangat baik.

```
Accuracy: 0.9833333333333333
```

◆ Pemanfaatan Trained Model

Setelah train model ini dinilai cukup baik berdasarkan hasil evaluasi, maka model yang telah di training ini dapat kita gunakan untuk melakukan prediksi terhadap data baru.

Berbeda dengan testing set pada data baru, kita hanya memiliki sekumpulan nilai features tetapi tidak memiliki nilai target, dan kita akan memanfaatkan model yang sudah kita training sebelumnya untuk melakukan prediksi nilai target dari sekumpulan nilai features yang ada.

berikut akan kita demokan prosesnya.

```
data_baru = [[5, 5, 3, 2],
             [2, 4, 3, 5]]

preds = model.predict(data_baru)
```

```
preds
```

dDsini pertama-tama kita akan bentuk dulu dataset yang baru disini data saya terdiri dari dua instansi atau dua row atau dua baris, dimana setiap instan atau setiap barisnya terbagi dalam empat nilai features.Disini nilai nya adalah 5,5,3,2 untuk instansi pertama, dan untuk instansi kedua nilainya adalah 2,4,3,5, karena untuk konteks Irish memang terdapat empat buah features.

Berikutnya untuk data baru ini kita akan melakukan prediksi, caranya disini kita akan Panggil train modelnya atau model yang sudah kita train sebelumnya, lalu kita Panggil method ***predict*** dan prediksinya akan kita kenakan terhadap **data_baru** ini,lalu berikutnya hasil prediksi kita tampung ke dalam variabel **preds.**

Kita eksekusi kode nya

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

iris = load_iris()

X = iris.data
y= iris.target

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.4,
                                                    random_state=1 )


model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train, y_train)

data_baru = [[5, 5, 3, 2],
             [2, 4, 3, 5]]

preds = model.predict(data_baru)

print(preds)
```

dan hasilnya akan tampak seperti di bawah ini

```
array([1 2])
```

Disini hasil prediksinya, ini artinya untuk instans yang pertama dengan nilai features, 5,5,3,2, diprediksi memiliki nilai target 1, sedangakan untuk instans yang kedua dengan nilai features 2,4,3,5, ini diprediksi untuk memiliki nilai target 2.

Nilai target 1 dan 2 ini tentunya perlu kita konfirmasikan dengan target names nya, oleh karena nya disini kita akan lakukan proses untuk memanggil target names yang kita mapping kan dengan nilai hasil prediksi ini.

```
pred_species = [iris.target_names[p] for p in preds]
print(f'Hasil Prediksi : {pred_species}')
```

Kita coba eksekusi kode nya

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

iris = load_iris()

X = iris.data
y= iris.target

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.4,
                                                    random_state=1 )


model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train, y_train)

data_baru = [[5, 5, 3, 2],
             [2, 4, 3, 5]]

preds = model.predict(data_baru)

pred_species = [iris.target_names[p] for p in preds]
print(f'Hasil Prediksi : {pred_species}')
```

Dan ini dia hasilnya

```
Hasil Prediksi : ['versicolor', 'virginica']
```

Disini Kita bisa bilang bahwa untuk data yang pertama dengan nilai features 5,5,3,2, ini diprediksi masuk ke dalam klasifikasi spesies iris **versicolor** ,Sedangkan untuk instens yang kedua dengan nilai feaures 2,4,3,5 ini diprediksi masuk ke dalam kategori spesies irisi **virginica**.

◆ Dump & Load Trained Model

Train model yang sudah siap ini pada akhirnya akan kita default di production, untuk kebutuhan semacam ini maka train model perlu kita ekspor atau kita Dump sebagai suatu file model,di sini kita bisa memanfaatkan modul joblib.

Berikut akan kita demokan prosesnya

```
import joblib

joblib.dump(model, 'iris_classifier_knn.joblib')
```

Untuk menggunakan joblib, pertama-tama kita akan ***import joblib*** nya, setelah joblib di import maka kita bisa mulai melakukan proses dumpling ataupun loading model machine learning.

Disini pertama-tama kita akan dumpling dulu model machine learning yang baru saja kita training tadi ke dalam suatu file joblib, caranya cukup mudah di sini Kita tinggal panggil j***oblib.dump,*** dan proses ini akan membutuhkan dua parameter,parameter pertama adalah train ***model*** yang mau kita dump, dan parameter kedua adalah nama file joblib nya, di sini nama file joblib nya kita set sebagai ***iris_classifier_knn.joblib.***

Kita coba eksekusi kode nya

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import joblib


iris = load_iris()

X = iris.data
y= iris.target

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
```

```
                                                test_size=0.4,
                                                random_state=1 )


model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train, y_train)

#y_pred = model.predict(X_test)
#acc = accuracy_score(y_test, y_pred)

data_baru = [[5, 5, 3, 2],
             [2, 4, 3, 5]]

preds = model.predict(data_baru)
pred_species = [iris.target_names[p] for p in preds]

joblib.dump(model, 'iris_classifier_knn.joblib')
print(joblib.dump)
```

dan outputnya

```
['iris_classifier_knn.joblib']
```

Disini bisa kita lihat ada suatu file baru dengan nama ***iris_classifier_knn.joblib.*** File ini terbentuk dari hasil dumpling yang kita lakukan di sini.

Selanjutnya proses deployment machine learning model ini dapat dilakukan dengan mudah yaitu dengan menempatkan file jonlib ini ke server yang ada di production, lalu berikutnya di production server kita akan load file joblib ini menjadi machine learning model yang siap digunakan.

Prosesnya lebih kurang seperti ini

```
production_model = joblib.load('iris_classifier_knn.joblib')
```

Kita tinggal panggil saja ***joblib.load,*** lalu diikuti dengan nama file joblib nya , begitu file joblib ini kita load,maka akan dihasilkan suatu machine learning model yang sudah kita training sebelumnya.

Untuk kasus kita disini machine learning modelnya kita tampung kedalam suatu variabel yaitu **production_model.**

Kita coba eksekusi kode nya

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import joblib


iris = load_iris()

X = iris.data
y= iris.target

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.4,
                                                    random_state=1 )


model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train, y_train)


data_baru = [[5, 5, 3, 2],
             [2, 4, 3, 5]]

preds = model.predict(data_baru)
pred_species = [iris.target_names[p] for p in preds]
production_model = joblib.load('iris_classifier_knn.joblib')
print(production_model)
```

dan hasilnya akan tampak seperti ini

```
KNeighborsClassifier(n_neighbors=3)
```

Selanjutnya kita bisa memanfaatkan production_model ini untuk melakukan prediksi terhadap data-data baru yang kita temui di production.

◆ Bab 4 : Data Preprocessing dengan Scikit-Learn

Sesi pembelajaran kita kali ini adalah terkait data Processing dengan menggunakan scikit-learn, pada sesi pembelajaran Sebelumnya, kita pernah menyampaikan bahwa fase mempersiapkan dataset bisa jadi tahapan yang paling banyak membutuhkan waktu dan effort dalam suatu Project data scient maupun machine learning .

Dalam sesi pembelajaran kali ini kita akan berkenalan dengan beberapa teknik data preprocessing, modul scikit-learn juga sudah menyertakan sejumlah teknik data preprocessing yang umum ditemui,disini kita akan mempelajari beberapa diantaranya, pada sesi kita akan persiapkan terlebih dahulu sampel dataset yang akan kita olah.

◆ Sample Data

```
import numpy as np
from sklearn import preprocessing

sample_data = np.array([[2.1, -1.9, 5.5],
                        [-1.5, 2.4, 3.5],
                        [0.5, -7.9, 5.6],
                        [5.9, 2.3, -5.8],
                        ])

sample_data
```

disini pertama-tama kita akan import numpy sebagai np terlebih dahulu, karena dataseet nya akan kita generate sebagai numpy array, lalu terkait dengan modul preprocessing ,mudol preprocessing ini bisa kita impor dengan memanggil **from sklearn import** preprocessing, ini adalah dua modul yang harus kita import, lalu berikutnya ya kalau kita lihat di sini nah disini terdapat dataset dumpy yang kita generate untuk keperluan sesi pembicaraan kita kali ini.

kita coba eksekusi kodenya

dan ini hasilnya

```
[[ 2.1 -1.9  5.5]
 [-1.5  2.4  3.5]
```

```
 [ 0.5 -7.9  5.6]
 [ 5.9  2.3 -5.8]]
```

Dataset ini kita bentuk sebagai numpy array, kalau kita lihat ya ini merupakan hasilnya Nah kalau kita lihat kembali materi yang disampaikan dalam sesi pengajaran sebelumnya, maka dataset ini bisa kita panel sebagai dataset features, dan kalau kita lihat disini jumlah instansi yang ada 4 intense yang bisa terlihat dari jumlah barisnya ada 4.

Lalu terkait jumlah features dataset ini terbagi dalam 3 features.Dataset yang terbentuk ini kita tampung ke dalam variabel yang kita beri nama **sample_data,** untuk lebih jelasnya lagi Kita juga bisa menampilkan dimensi dari dataset ini dengan memanggil **sample_date.shape.**

coba kita eksekusi dulu kodenya

```
import numpy as np
from sklearn import preprocessing

sample_data = np.array([[2.1, -1.9, 5.5],
                    [-1.5, 2.4, 3.5],
                    [0.5, -7.9, 5.6],
                     [5.9, 2.3, -5.8],
                          ])


print(sample_data.shape)
```

hasilnya tampak seperti ini

```
(4, 3)
```

Disini bisa kita lihat shape nya adalah 4,3 atau dengan kata lain jumlah instance atau jumlah barisnya 24 dan jumlah kolom atau jumlah feature ada 3.

◆ Binarisaton

Teknik data preprossesing yang pertama yang akan kita pelajari disini adalah Binarisation, tujuan utama dari Teknik ini adalah untuk menghasilkan suatu data yang terdiri dari dua nilai numerik saja yaitu 0 dan 1, disini kita coba tampilkan terlebih dahulu dataset yang telah kita bentuk sebelumnya.

```
sample_data

preprocessor = preprocessing.Binarizer(threshold=0.5)
binarised_data = preprocessor.transform(sample_data)
binarised_data
```

Disini kita coba tampilkan kembali data yang ditampung dalam variabel **sampel_data,**ini merupakan dataset yang kita bentuk sebelumnya, bisa nampak disini dataset ini terdiri dari sekumpulan nilai floating-point yang cukup beragam, semisal saja di sini kita dihadapkan pada kebutuhan untuk mengkonversikan setiap nilai numerik yang lebih besar dari 0,5 menjadi satu dan sisanya kita konversikan menjadi nol, kebutuhan semacam ini bisa kita penuhi dengan memanfaatkan class **binarised.**

berikut kita akan demokan caranya

```
import numpy as np
from sklearn import preprocessing

sample_data = np.array([[2.1,-1.9, 5.5],
                        [-1.5, 2.4, 3.5],
                               [0.5, -7.9, 5.6],
                               [5.9, 2.3, -5.8],
                              ])


preprocessor = preprocessing.Binarizer(threshold=0.5)
binarised_data = preprocessor.transform(sample_data)
print(binarised_data)
```

Disini pertama-tama kita akan bentuk dulu objek binarised nya, caranya cukup mudah Kita tinggal panggil saja **`preprocessing.Binarizer,`** dan kita akan menyertakan satu buah parameter yaitu

parameter **threshold,** dan parameter ini kita set di 0,5. Pemanggilan semacam ini akan menghasilkan sebuah objek binarizer yang kemudian kita tampung dalam variabel **preprocessor.**

Pada tahapan selanjutnya kita akan mengenakan proses transformasi terhadap sampel data yang kita miliki tadi caranya cukup mudah kita panggil saja **preprocessor.transform,** lalu kita Panggil sampel dataset nya, hasil transformasinya lalu kita tampung ke dalam variabel **binarised_data** untuk selanjutnya kita tampilkan isinya.

dan ini hasilnya

```
[[1. 0. 1.]
 [0. 1. 1.]
 [0. 0. 1.]
 [1. 1. 0.]]
```

Disini bisa nampak hasil dari proses binarisation ini adalah suatu dataset yang hanya terdiri dari dua nilai yaitu 1 dan 0, dimana batasannya itu ditentukan dari nilai threshold, kita bisa lihat disini nilai treshold adalah 0,5, artinya setiap nilai yang lebih kecil atau sama dengan 0,5 akan dikonversikan menjadi nilai 0.

Sebagai contoh disini untuk nilai -1,9, karena dia lebih kecil dari 0,5 maka dia akan dikonversikan menjadi nilai 0, sedangkan nilai-nilai yang lebih besar dari 0,5 ,misal di sini 2,1 dan 5,5 akan dikonversikan menjadi nilai 1.Yang perlu diperhatikan disini adalah, untuk nilai yang sama persis dengan nilai threshold nya maka dia akan dikonversikan menjadi nilai 0, atau kalau kita mau Sederhanakan nilai 1 ini akan kita peroleh dari suatu nilai yang lebih besar dari nilai treshold nya, dan sisanya akan dikonversikan menjadi nilai 0.

Pada kasus yang kita hadirkan disini, merupakan bentuk paling sederhana dari teknik binarisation pada data preprocessing ,dalam beberapa sesi pembelajaran selanjutnya kita akan bertemu dengan bentuk binarisation lainnya.

◆ Scaling

Teknik data preprocessing yang kedua yang akan kita pelajari disini adalah **scaling,** ini sesuai dengan namanya, tujuan utama dari teknik scaling ini untuk menghasilkan suatu data numerik yang berada dalam rentang skala tertentu.

Disini kita coba tampilkan kembali dataset Yang sudah kita bentuk sebelumnya, oleh karenanya di sini kita coba tampilkan isi dari variabel sampel data, karena variabel sampel data ini yang kita gunakan untuk menampung dataser yang kita bentuk sebelumnya.

```
import numpy as np
from sklearn import preprocessing

sample_data = np.array([[2.1, -1.9, 5.5],
                        [-1.5, 2.4, 3.5],
                               [0.5, -7.9, 5.6],
                               [5.9, 2.3, -5.8],
                              ])

preprocessor = preprocessing.MinMaxScaler(feature_range=(0, 1))
preprocessor.fit(sample_data)
scaled_data = preprocessor.transform(sample_data)
print(scaled_data)
```

Bisa nampak disini, dataset kita terdiri dari sekumpulan nilai floating point dengan rentang nilai mulai dari -7,9 sebagai nilai terkecil, sampai dengan nilai 5,9 sebagai nilai terbesarnya. Misal nya kita dihadapkan pada kebutuhan untuk mengkonversikan sekumpulan nilai numerik ini kedalam rentang nilai mulai dari 0 sampai dengan 1, kebutuhan semacam ini bisa kita penuhi dengan memanfaatkan class scaler.

Teknik scaling sendiri ada berbagai macam, disini kita akan gunakan scaling yang paling sederhana yaitu MinMaxScaler, disini caranya juga cukup mudah ya Kita tinggal panggil saja **`preprocessing.MinMaxScaler,`** lalu di sini kita sertakan satu parameter penting yaitu **`feature_range,`** parameter feature ini akan membutuhkan data bertipe tuple yang akan memuat 2 nilai. untuk kasus kita kali ini nilainya kita set sebagai 0 dan 1, nilai 0 ini akan mengindikasikan nilai terkecil dari skala yang baru, dan nilai 1 ini mengindikasikan nilai terbesar dari skala yang baru nanti.

Begitu kita memanggil **`preprocessing.MinMaxScaler`** dengan nilai **`feature_range`** nya 0,1 ini maka akan terbentur objek scaler yang kita tampung ke dalam variabel **`preprocessor.`**

Selanjutnya objek scaler ini akan kita fit terhadap sampel data yang kita miliki, caranya **`preprocessor.fit,`** lalu kita masukkan **`sample_data`** sebagai parameternya,Setelah scaler nya kita fut terhadap sampel data kita maka scaler ini bisa kita gunakan untuk melakukan proses transformasi data.

Untuk kasus kita kali ini transformasi Datanya juga akan kita kenakan pada sampel data yang kita miliki tadi, oleh karenanya di sini kita Panggil **`preprocessor.fit_transform,`** lalu kita masukkan **sampel_data** nya.

Hasil dari proses transformasi data ini kita tampung ke dalam variabel **`scaled_data,`** lalu berikutnya kita tampilkan isi dari scaled data nya.

dan ini hasilnya

```
[[0.48648649 0.58252427 0.99122807]
 [0.         1.         0.81578947]
 [0.27027027 0.         1.        ]
 [1.         0.99029126 0.        ]]
```

Hasilnya bisa nampak di sini , nilai terkecil nya adalah 0, dan nilai terbesarnya adalah 1, dan nilai ini merupakan hasil Transformasi dari sampel data.

Jadi kalau kita simpulkan disini proses **`MinMaxScaler`** ini di digunakan untuk mengubah skala nilai terkecil dan nilai terbesar dari dataset yang kita miliki ini ke skala tertentu.Untuk kasus kita kali ini skalanya nilai terkecil nya 0 dan nilai terbesarnya adalah 1, dan output ini merupakan hasil transformasinya.

kalau kita lihat disini, proses fiting dan proses transformasi data nya dikenakan terhadap data yang sama, yaitu **sampel_data**, dalam scikit-learn proses ini bisa kita satukan dengan memanfaatkan method fit transform, caranya cukup sederhana di sini kita panggil saja objek **`preprocessor`** nya atau objek scaler nya, lalu kita Panggil method **`fit_transform,`** lalu kita sertakan **`sampel_data`** parameternya .

```
scaled_data = preprocessor.fit_transform(sample_data)
print(scaled_data)
```

Disini artinya adalah proses fitting dan transformasi dikenakan pada data yang sama dan selanjutnya hasil dari proses transformasinya kita akan tampung ke dalam variabel **`scaled_data,`** untuk selanjutnya kita tampilkan nilai dari scale daya nya,

kita coba eksekusi akaya scriptnya

```
import numpy as np
from sklearn import preprocessing

sample_data = np.array([[2.1, -1.9, 5.5],
                        [-1.5, 2.4, 3.5],
                              [0.5, -7.9, 5.6],
                              [5.9, 2.3, -5.8],
                             ])

preprocessor = preprocessing.MinMaxScaler(feature_range=(0, 1))
preprocessor.fit(sample_data)
scaled_data = preprocessor.fit_transform(sample_data)
print(scaled_data)
```

dan outputnya.

```
[[0.48648649 0.58252427 0.99122807]
 [0.         1.         0.81578947]
 [0.27027027 0.         1.        ]
 [1.         0.99029126 0.        ]]
```

Kalau kita perhatikan di sini, keduanya menghasilkan hasil ya sama, bedanya kalau di Cara yang pertama proses fitting dan proses transform nya ini terpisah atau dikerjakan dengan menggunakan dua metode yang berbeda,sedangkan di sini proses fitting dan proses transformasi datanya dilakukan dengan menggunakan satu kali pemanggilan method yaitu method **`fit_transform.`**

◆ L1 normalisation: Least Absolute Deviations

Teknik data preprocessing yang ke 3 dan ke 4 yang akan kita pelajari disini adalah **normalisation**, sesuai dengan namanya tujuan utama dari Teknik ini adalah untuk melakukan normalisasi terhadap data numerik yang kita miliki.

Disini kita coba tampilkan kembali dataset yang telah kita bentuk sebelumnya, pada kasus normalization pertama ini kita akan menerapkan List Absolute Deviations, bagi kalian yang tertarik untuk mempelajari lebih lanjut mengenai List Absolute Deviations, kita juga menyertakan referensi yang bisa kita gunakan yang kita ambil dari Wikipedia, jadi bagi kalian yang tertarik kalian bisa mempelajari lebih lanjut dari halaman Wiki ini.

Refrensi : https://en.wikipedia.org/wiki/Least_absolute_deviations

Selanjutnya kita akan coba demokan proses untuk menerapkan teknik normalisation ini.

```
sample_data

l1_normalised_data = preprocessing.normalize(sample_data, norm='l1')
l1_normalised_data
```

Disini pertama-tama kita Panggil **`preprocessing.normalize,`** ini akan membutuhkan dua parameter,parameter pertama adalah datanya atau **`sample_data`** nya, lalu parameter kedua adalah parameter **`norm,`** untuk kasus kita kali ini parameter norm nya kita beri nilai string **`l1`**,normalisasi l1 ini akan berasosiasi dengan normalisasi yang menerapkan List Absolute Definition.Pemanggilan fungsi preprocessing normalise sini akan menghasilkan data yang ternormalisasi.

Kita coba eksekusi kode nya.

```
import numpy as np
from sklearn import preprocessing

sample_data = np.array([[2.1, -1.9, 5.5],
                        [-1.5, 2.4, 3.5],
                    [0.5, -7.9, 5.6],
```

```
                          [5.9, 2.3, -5.8],
                         ])

l1_normalised_data = preprocessing.normalize(sample_data, norm='l1')
print(l1_normalised_data)
```

Untuk kasus kita disini, datan yang sudah dinormalisasi kita tampung ke dalam variabel **l1_normalised_data,** untuk berikutnya kita tampilkan datanya.

dan ini merupakan hasilnya.

```
[[ 0.22105263 -0.2         0.57894737]
 [-0.2027027   0.32432432  0.47297297]
 [ 0.03571429 -0.56428571  0.4       ]
 [ 0.42142857  0.16428571 -0.41428571]]
```

kalau kita perhatikan di sini,

```
sample_data = np.array([[2.1, -1.9, 5.5],
                        [-1.5, 2.4, 3.5],
                         [0.5, -7.9, 5.6],
                        [5.9, 2.3, -5.8],
                         ])
```

ini merupakan dataset sampel sebelum proses normalisasi l1,

Sedangkan output ini merupakan data hasil normalisasi l1

```
[[ 0.22105263 -0.2         0.57894737]
 [-0.2027027   0.32432432  0.47297297]
 [ 0.03571429 -0.56428571  0.4       ]
 [ 0.42142857  0.16428571 -0.41428571]]
```

◆ L2 normalisation: Least Squares

Untuk kasus normalisation kedua kita akan menerapkan **Least Squares** .

Bagi kalian yang ingin mempelajari lebih lanjut mengenai Least Swuare, kita juga menyertakan referensi sederhana yang bisa kita peroleh dari halaman Wikipedia, ini merupakan halaman Wikipedia yang memberikan penjelasan sederhana mengenai Least square.

Refrensi: https://en.wikipedia.org/wiki/Least_squares

```
sample_data

l2_normalised_data = preprocessing.normalize(sample_data, norm='l2')
l2_normalised_data
```

Disini juga Sama, kita coba tampilkan dulu sampel data sebelum kita mengenakan proses normalisasi l2, lalu berikutnya kita akan demokan proses untuk menerapkan teknik normalisation kedua ini.

Disini caranya juga cukup sederhana, kita panggil saja **`preprocessing.normalize`** ,lalu di sini kita menyertakan dua parameter, parameter pertama adalah **`sample_data`** nya, dan parameter kedua adalah parameter **`norm,`** kali ini parameter **`norm`** nya kita beri nilai **`l2.`**

Normalisasi `l2` ini akan berasosiasi dengan teknik yang namanya Least Squares, lalu berikutnya pemanggilan fungsi **`preprocessing.normalize`** ini akan menghasilkan data yang sudah ternormalisasi, data tersebut Lalu kita tampung ke dalam variabel **`l2_normalised_data`** untuk selanjutnya kita tampilkan.

Kita coba eksekusi kodenya

```
import numpy as np
from sklearn import preprocessing

sample_data = np.array([[2.1, -1.9, 5.5],
                        [-1.5, 2.4, 3.5],
                    [0.5, -7.9, 5.6],
                        [5.9, 2.3, -5.8],
                       ])
```

```
l2_normalised_data = preprocessing.normalize(sample_data, norm='l2')
print(l2_normalised_data)
```

dan ini hasilnya

```
[[ 0.33946114 -0.30713151  0.88906489]
 [-0.33325106  0.53320169  0.7775858 ]
 [ 0.05156558 -0.81473612  0.57753446]
 [ 0.68706914  0.26784051 -0.6754239 ]]
```

Ini merupakan data setelah proses normalisasi l2 atau proses normalisasi dengan menerapkan Least Squares.

◆ Bab 5 : Simple Linear Regression

Pembelajaran kali ini kita akan mempelajari salah satu model machine learning paling sederhana yaitu **Simple Linear Regression,** model mesin learning yang satu ini juga umum diperkenalkan sebagai model yang pertama kali dipelajari dikelas machine learning.

Untuk sesi pembelajaran kita kali ini akan lebih menekankan pada sisi practical dari simple linear regression, bagi kalian yang membutuhkan referensi lebih lanjut kalian bisa memanfaatkan halaman Wikipedia.

Refrensi : https://en.wikipedia.org/wikiSimple_linear-regression

Ini merupakan penjelasan lebih lanjut yang juga masih sederhana terkait simple linear regression, jadi bagi kalian yang butuh untuk mempelajari lebih lanjut kalian bisa mempelajarinya melalui halaman Wikipedia ini.

◆ Sample Dataset

Kita akan mempersiapkan terlebih dahulu dataset yang akan kita pergunakan dalam sesi pembelajaran kita kali ini,disini kita akan membentuk dataset yang berisi daftar diameter pizza beserta harganya rasa sini akan kita tambah kedalam format Pandas dataframe.

Berikut akan kita demokan caranya.

```
import pandas as pd

pizza = {'diameter': [6, 8, 10, 14, 18],
      'harga': [7, 9, 13, 17.5, 18]}

pizza_df = pd.DataFrame(pizza)
print(pizza_df)
```

Pertama-tama kita akan import dulu modul pandas nya, di sini kita Panggil **import pandas as pd,** lalu berikutnya di sini kita siapkan satu dictionary yang terdiri dari dua buah keys, yaitu **diameter** dan **harga,**dan setiap keys nya akan berasosiasi dengan list, list untuk meter dan list untuk harga, lalu berikutnya dictionary ini akan kita tampung ke dalam variabel **pizza,** untuk selanjutnya kita bentuk sebagai objek **DataFrame,** disini proses pembentukan **DataFrame** bisa dilakukan dengan cara

pd.DataFrame, lalu kita menyatakan pizza sebagai parameternya, dan objek telah frame yang berbentuk disini akan kita tampung ke dalam variabel **pizza_df,** untuk selanjutnya kita tampilkan isinya.

Kita eksekusi kodenya.

Dan ini hasilnya.

	diameter	harga
0	6	7.0
1	8	9.0
2	10	13.0
3	14	17.5
4	18	18.0

Hasilnya ini merupakan DataFrame berbentuk, kalau kita lihat dataset kita kali ini terdiri dari dua buah kolom, yaitu kolom **diameter** dan kolom **harga**, dalam sesi pembelajaran kita kali ini kita akan membentuk suatu model machine learning sederhana yang dapat digunakan untuk memprediksi harga pizza bila diketahui ukuran diameternya.

Dengan kata lain disini nilai **diameter** akan berperan sebagai **feature** sedangkan harga **pizza** akan berperan sebagai **target.**

◆ Visualisasi Data

Disini kita akan coba visualisasikan dataset **pizza** yang telah kita bentuk sebelumnya ke dalam scatter plot, dimana sumbu x nya akan berasosiasi dengan diameter pizza sedangkan sumbu y nya akan berasosiasi dengan harga pizza.

Berikut kita demokan cara nya.

```
import matplotlib.pyplot as plt

pizza_df.plot(kind='scatter', x='diameter', y='harga')
```

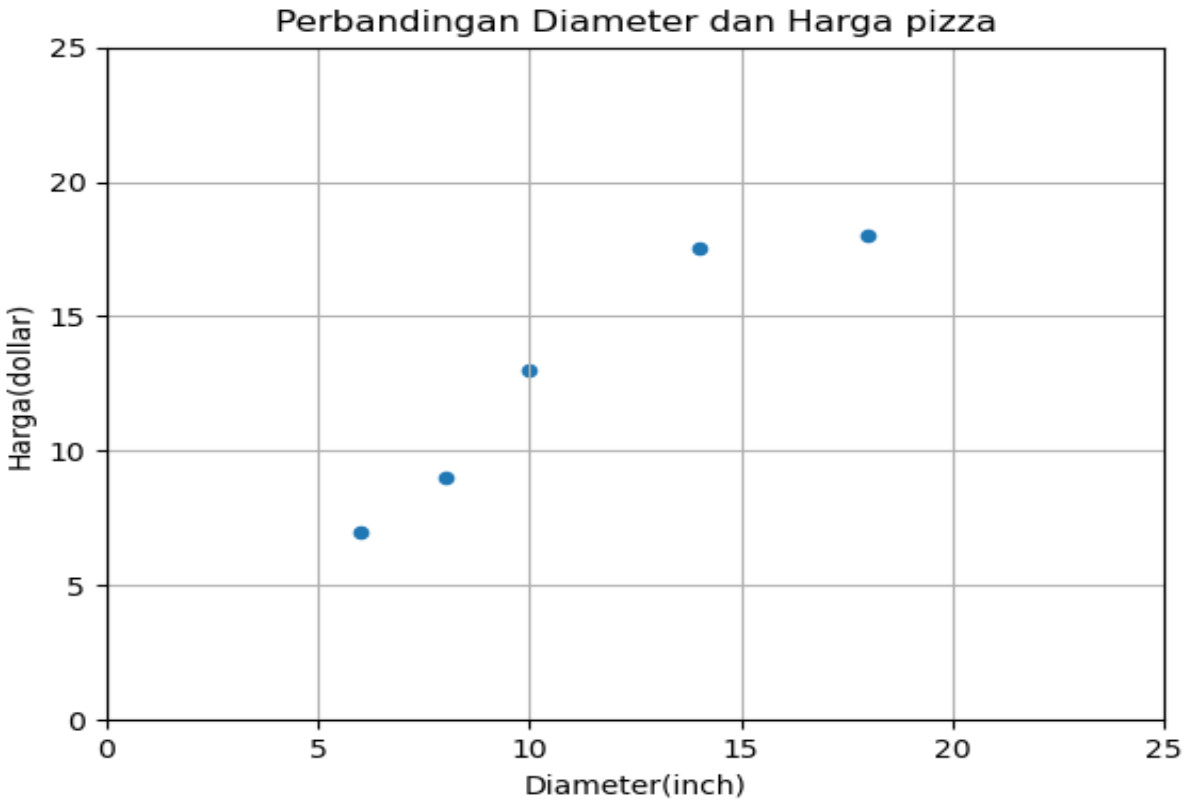
```
plt.title('Perbandingan Diameter dan Harga pizza')
plt.xlabel('Diameter(inch)')
plt.ylabel('Harga(dollar)')
plt.xlim(0, 25)
plt.ylim(0, 25)
plt.grid(True)
plt.show()
```

Disini karena kita ingin melakukan visualisasi data maka kita akan impor dulu modul penting kita yaitu **matplotlib.pyplot,** di sini kita Panggil **import matplotlib.pyplot as plt** dan kita alias kan sebagai **plt,** lalu berikutnya karena kita ingin melakukan visualisasi berupa scatter plot dari data yang disimpan dalam **pizza_df**, maka disini kita Panggil **pizza_df.plot,** lalu kita akan sertakan tiga parameter.Parameter yang pertama adalah parameter **kind** yang kita beri nilai **scatter,** karena floating yang mau kita hasilkan kali ini adalah scatter plot, lalu berikutnya untuk sumbu **x** akan kita asosiasikan dengan kolom diameter sedangkan sumbu **y** yang akan kita asosiasikan dengan kolom harga.

Kita coba eksekusi kodenya

```
import pandas as pd
import matplotlib.pyplot as plt

pizza = {'diameter': [6, 8, 10, 14, 18],
         'harga': [7, 9, 13, 17.5, 18]}

pizza_df = pd.DataFrame(pizza)

pizza_df.plot(kind='scatter', x='diameter', y='harga')

plt.title('Perbandingan Diameter dan Harga pizza')
plt.xlabel('Diameter(inch)')
plt.ylabel('Harga(dollar)')
plt.xlim(0, 25)
plt.ylim(0, 25)
plt.grid(True)
plt.show()
```

dan ini hasilnya



Lalu berikutnya ada beberapa hal lain yang perlu kita perhatikan, disini **plt.title,plt.xlabel,plt.ylabel** akan mengatur tampilan untuk judul dan juga label pada sumbu-x maupun sumbu-y nya, untuk judul disini diatur dengan menggunakan **plt.title** Sedangkan untuk label sumbu-x dan sumbu-y kita atur dengan **plt.xlabel,** dan **plt.ylabel.**

Selanjutnya kita juga mengatur **xlim** dan **ylim** nya,ini digunakan untuk mengatur jangkauan sumbu-x dan juga jangkauan sumbu-y nya,kita lihat di sini ya 0,25 artinya jangkauan untuk sumbu x adalah 0 sampai dengan 25, demikian juga jangkauan untuk sumbu-y nya, makanya disini kita juga set **ylim** nya 0,25.

Jadi disini nilai sumbu y nya dimulai dari 0 sampai dengan 25,lalu berikutnya disini kita juga mengaktifkan gridnya, di sini kita Panggil **plt.grid** lalu kita beri nilai True, dan diakhir disini kita akan Tampilkan floating kita dengan memanggil **plt.show(),** Jadi lebih kurang seperti itulah proses visualisasi data nya.

Berikutnya disini kita akan pelajari lebih lanjut mengenai hasil floating,kalau kita perhatikan output, setiap data point atau setiap marker ini akan merepresentasikan satu baris dari data frame yang kita miliki. kalau kita lihat data kita disini terdiri dari 5 Instance atau terdiri dari lima baris,dimana setiap barisnya ini akan direpresentasikan dengan satu marker yang ada di output, oleh karenanya disini kita bisa melihat adanya lima marker atau lima titik berwarna biru, dan kalau kita perhatikan di sini ini seolah-olah akan membentuk suatu garis lurus.

Ini merupakan kasus yang sangat tepat sekali untuk menerapkan linear regression karena model linier regression ini akan berkaitan dengan persamaan garis lurus, dan disini kita juga perlu ulas lagi terkait dengan tujuan dari eksplorasi, kita tujuannya disini adalah kita mencoba untuk memprediksi harga pizza bila diketahui diameternya, dan kita akan pecahkan permasalahan ini dengan memanfaatkan modal machine learning yaitu simple linear regression.

◆ Simple Regression Linear Model

Setelah kita memahami konteks dataset dan juga permasalahan, kita akan coba menerapkan **simple linear regression model** sebagai solusi, sesuai dengan namanya model machine learning yang satu ini memang digunakan untuk menyelesaikan dari presentase, bagi kalian yang masih bingung dengan istilah regression tasks kita sangat menyarankan untuk mempelajari kembali seri belajar machine learning di seri sebelumnya.

Disini kita akan melakukan beberapa penyesuaian terhadap data pizza yang kita miliki,langkah pertama yang akan kita lakukan adalah mengelompokkan feature dengan target kedalam dua variabel terpisah, selain itu kita juga akan melakukan konversi struktur data dari yang semula menggunakan pandas dataframe menjadi numpy array.

Berikut kita demokan caranya.

```
import numpy as np

X = np.array(pizza_df['diameter'])
y = np.array(pizza_df['harga'])

print(f'X: {X}')
```

```
print(f'y: {y}')
```

Pertama-tama kita akan import dulu numpy nya, disini numpy nya kita import **`import numpy as np,`** berikutnya untuk diameter, karena diameter ini merupakan feature akan kita seleksi dulu, lalu kita konversikan menjadi suatu numpy array, yang berikutnya akan kita tampung ke dalam variabel **`X`**, dimana variabel **`X`** ini akan kita pandang sebagai variabel feature, lalu berikutnya untuk kolom **`harga`** juga akan kita konversikan menjadi numpy array, dan nilainya akan kita tampung ke dalam variabel **`y,`** variabel **`y`** ini akan kita perlakukan sebagai variabel target ,berikutnya kita akan coba print out dulu data yang disimpan dalam **`X`** maupun **`y`** nya.

kita eksekusi kode nya

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

pizza = {'diameter': [6, 8, 10, 14, 18],
         'harga': [7, 9, 13, 17.5, 18]}

pizza_df = pd.DataFrame(pizza)

X = np.array(pizza_df['diameter'])
y = np.array(pizza_df['harga'])

print(f'X: {X}')
print(f'y: {y}')
```

dan ini hasilnya

```
X: [ 6  8 10 14 18]
y: [ 7.   9.  13.  17.5 18. ]
```

Hasil **x** nya ini merupakan kumpulan nilai features dan yang **y** ini merupakan kumpulan nilai harga, kalau kita perhatikan di sini, variabel **x** yang digunakan untuk menampung nilai feature hanya terdiri dari satu dimensi saja, padahal dari sesi pembelajaran sebelumnya kita mengetahui bahwa model machine learning yang disertakan dalam scikit-learn membutuhkan sekumpulan nilai feature yang diramu dalam suatu array 2 dimensi.Oleh karenanya di sini kita perlu melakukan proses receiving terhadap array yang digunakan untuk menampung nilai feature ini .

Berikut kita demokan caara nya

```
X: X.reshape(-1, 1)
X.shape
```

Disini array yang ditampung dalam **X** nya akan kita reshape, caranya cukup mudah ya di sini Kita tinggal panggil saja **X.reshape** dan kita beri nilai parameter **-1, 1** ,lalu berikutnya kita akan Tampilkan siap atau dimensi dari **X,** proses reshape ini.

kita coba eksekusi kodenya

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

pizza = {'diameter': [6, 8, 10, 14, 18],
         'harga': [7, 9, 13, 17.5, 18]}

pizza_df = pd.DataFrame(pizza)

X = np.array(pizza_df['diameter'])
y = np.array(pizza_df['harga'])

X: X.reshape(-1, 1)
print(X.shape)
```

dan ini hasilnya

```
(5,)
```

Di sini setelah proses reshape, dimensinya menjadi 5,1, dan kalau kita tampilkan **print**(X), hasilnya kali ini variabel **X** akan menampung 1 hari 2 dimensi dimana jumlah barisnya ada lima dan jumlah kolomnya ada satu.

```
[ 6  8 10 14 18]
```

Proses transformasi dari kondisi nilai **X** seperti ini dimungkinkan atau difasilitasi dengan memanggil method **reshape.**

◆ Training Simple Linear Regression Model

Kita akan melakukan proses training model machine learning dimana model yang akan kita pilih adalah **Linear Regression**.

berikut akan kita demokan prosesnya

```
from sklearn.linear_model import LinearRegression

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

Pertama-tama kita perlu import dulu estimator kelasnya dan estimeter class yang mau kita gunakan kali ini adalah **linearRegression,** caranya kita cukup panggil saja **from sklearn.linear_model import linearRegression**, berikutnya kita akan membuat object **model** dari estimeter class ini, kita panggil saja **LinearRegression(),** lalu objek yang terbentuk akan kita tampung ke dalam variabel **model,**selanjutnya objek model ini akan kita training Dengan cara memanggil method fit, metode ini nantinya akan membutuhkan dua buah parameter yaitu sekumpulan nilai features beserta sekumpulan nilai targetnya ,dimana sekumpulan nilai feature dan nilai targetnya sudah kita tempatkan ke dalam variabel **X** dan variabel **y** .

kita coba eksekusi kodenya.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression

pizza = {'diameter': [6, 8, 10, 14, 18],
         'harga': [7, 9, 13, 17.5, 18]}

pizza_df = pd.DataFrame(pizza)

X = np.array(pizza_df['diameter'])
y = np.array(pizza_df['harga'])

X = X.reshape(-1, 1)

model = LinearRegression()
model.fit(X, y)

print(model)
```

Dan ini hasilnya

```
LinearRegression()
```

Sekarang ini object model kita statusnya adalah train model atau objek model yang sudah di training.

◆ Visualisasi Simple Linear Regression Model

Karena model machine learning yang kita gunakan adalah linear model yang notabene terbilang masih sangat sederhana, maka disini kita juga bisa memvisualisasikan model ini, karena pada dasarnya linear model yang dihasilkan ini akan berupa garis lurus.

Untuk dapat memvisualisasikan persamaan garis ini, pertama-tama kita akan Siapkan dua buah nilai untuk X nya atau untuk feature nya yaitu nilai 0 dan juga nilai 25, kedua nilai ini kita Tentukan karena disini kita ingin melakukan ploting pada area dengan ukuran 25 kali 25 dan disini kita mengambil nilai terkecil dan nilai terbesarnya yaitu 0 dan 25,

untuk lebih jelasnya berikut akan kita demokan prosesnya

```
X_vis = np.array([0, 25]).reshape(-1, 1)
y_vis = model.predict(X_vis)

plt.scatter(X, y)
plt.plot(X_vis, y_vis, '-r')

plt.title('Perbandingan Diameter dan Harga pizza')
plt.xlabel('Diameter(inch)')
plt.ylabel('Harga(dollar)')
plt.xlim(0, 25)
plt.ylim(0, 25)
plt.grid(True)
plt.show()
```

Disini kita akan Siapkan dua buah nilai untuk feature nya, yaitu 0 dan 25,kedua nilai ini akan kita bentuk sebagai numpy array dan karena numpy array ini akan digunakan sebagai feature, maka perlu kita reshape agar menghasilkan array 2 dimensi.

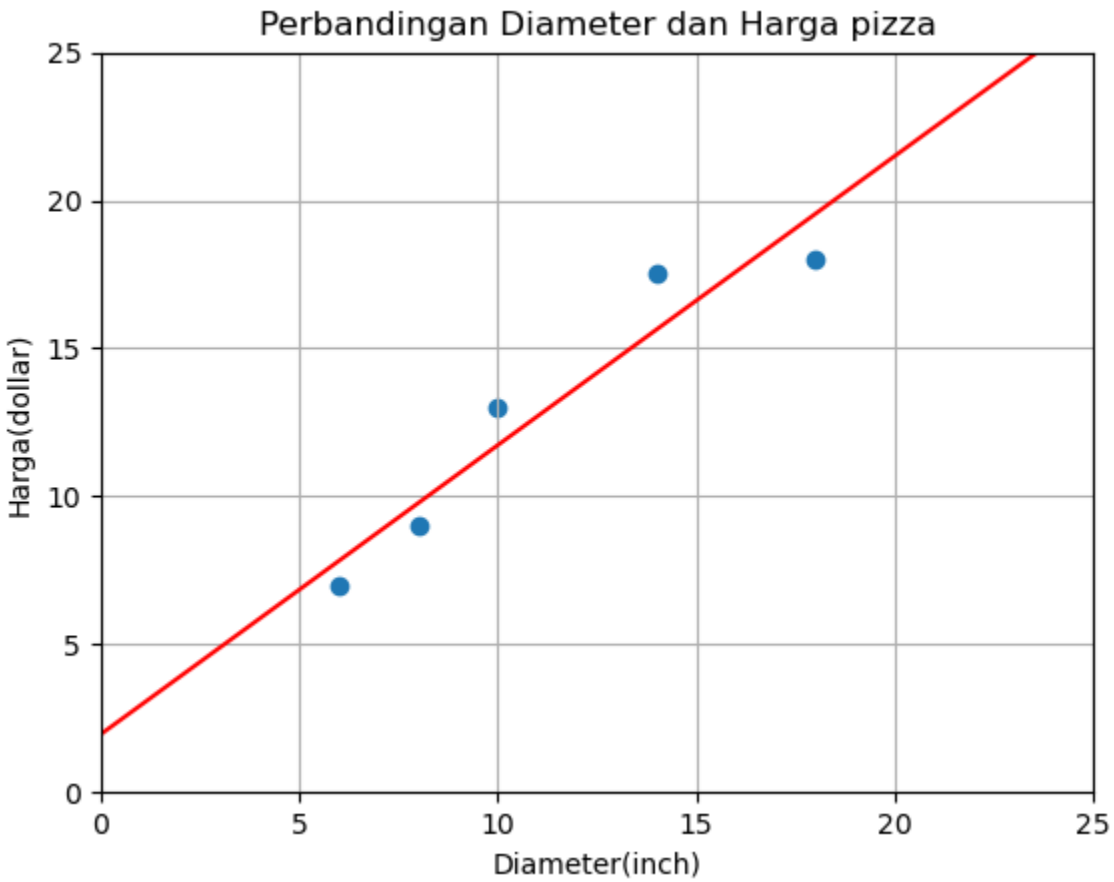
Oleh karenanya kita Panggil juga **reshape** dengan nilai parameter -**1**, **1,**nilai feature ini lalu kita tampung ke dalam variabel **X_vis,**karena pada dasarnya nilai ini mau kita gunakan sebagai nilai feature yang digunakan untuk melakukan visualisasi garis linier.

Lalu berikutnya berdasarkan nilai **X_vis** ini kita akan melakukan prediksi nilai **y** nya makanya di sini kita Panggil **model.predict** ,lalu kita masukkan nilai **X_vis** nya, hasil prediksinya kita tampung ke dalam variabel **y_vis.**

kita coba eksekusi kode untuk proses visualisasinya nya

```
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

pizza = {'diameter': [6, 8, 10, 14, 18],
         'harga': [7, 9, 13, 17.5, 18]}

pizza_df = pd.DataFrame(pizza)

X = np.array(pizza_df['diameter'])
y = np.array(pizza_df['harga'])

X = X.reshape(-1, 1)


model = LinearRegression()
model.fit(X, y)


X_vis = np.array([0, 25]).reshape(-1, 1)
y_vis = model.predict(X_vis)

plt.scatter(X, y)
plt.plot(X_vis, y_vis, '-r')

plt.title('Perbandingan Diameter dan Harga pizza')
plt.xlabel('Diameter(inch)')
plt.ylabel('Harga(dollar)')
plt.xlim(0, 25)
plt.ylim(0, 25)
plt.grid(True)
plt.show()
```

ini merupakan hasil visualisasinya,dan kita akan coba pelajari dulu prosesnya.



Pertama-tama kita akan Panggil dulu **plt.scatter** lalu kita sertakan nilai **X** dan **y,** perlu di ingat bawa **X** dan **y** disini bukan **X_vis** dan **y_vis**,tetapi **X** dan **y** yang kita miliki dari dataset sebelumnya, dan ini digunakan untuk melakukan floating marker berwarna biru, bisa kita di sini ya jumlah markernya ada 5 Yang merepresentasikan 5 instance atau 5 datapoint dari dataset yang kita milik.

Lalu selanjutnya kita juga melakukan floating plot untuk melakukan floating garis merah, ini dihasilkan dari proses floating ada di **plt.plot(X_vis, y_vis, '-r'),** yang kita floating adalah nilai **X** dan nilai **y,** yang digunakan melakukan proses visualisasi,dimana nilai x itu ada dua yaitu 0 dan 25, dari dataponit yang pertama dan datapoint yang ke-2.

Lalu nilai **y,** nya ini dihasilkan dari proses prediksi **model.predict,** dari setiap nilai **X_vis** nya, dan disini hasilnya kita tampung adalah variabel **y_vis.** Lalu visualisasinya kita gunakan **-r,**ini akan membentuk suatu garis lurus, dan **-r** ini nantinya akan berkorelasi dengan garis warna merah.

Lalu berikutnya terkait label -nya baik untuk judul dan juga pengaturan **X_lim** dan **y_lim** nya, ini masih Sama persis dengan visualisasi yang kita hasilkan sebelumnya.

Disini kita akan berfokus pada garis linearnya yaitu garis yang berwarna merah ini, garis linear yang terbentuk di sini akan memiliki formula persamaan yang mengacu pada formula Linear Regression sebagai berikut:

Formula Linear Regression: $y = a + ßx$

- y:response variable(target)
- x:explanatory variable(feature)
- a:intercept
- $ß$:slope

ini merupakan formula linear regression nya yaitu $y = a + ßx$**,** dimana **y** ini merupakan **response variabel atau target**, sedangkan **X** ini merepresentasikan **explanatory variabel atau feature** nya, lalu berikutnya disini kita juga akan mengenal dua parameter penting yaitu **alpha(a)** dan **beta(ß), alpha** ini dikenal juga dengan istilah **intercept** dan **beta** nya dikenal dengan istilah **slope**, dan keduanya merupakan nilai parameter yang perlu kita pahami.

Intercept merupakan titik pada sumbu **y,** dimana garis linear yang terbentuk menabrak, dengan kata lain intercept adalah nilai pada sumbu **y** ketika sumbu **x** nya bernilai 0.

Lalu berikutnya parameter kedua yang perlu kita pahami adalah **slope,** nilai slope akan berpengaruh pada tingkat kemiringan dari garis linier yang terbentuk,dimana nilai **slope** 0 akan menghasilkan garis horizontal, pada scikit-learn Kita juga bisa mendapatkan nilai **intercept** dan **slope** dari model linear regression yang sudah kita training sebelumnya.

berikut akan kita demokan caranya

```
print(f'Intercept: {model.intercept_}')
print(f'Slope: {model.coef_}')
```

Untuk mendapatkan nilai **intercept**, kita bisa pakai object **model** kita lalu kita panggilke **intercept_ ,**sedangkan untuk nilai **slope,** bisa kita peroleh dengan cara memanggil objek **model** nya lalu kita panggil ke **coef_** .

Kita coba eksekusi kodenya

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

```
import matplotlib.pyplot as plt
import pandas as pd

pizza = {'diameter': [6, 8, 10, 14, 18],
         'harga': [7, 9, 13, 17.5, 18]}

pizza_df = pd.DataFrame(pizza)

X = np.array(pizza_df['diameter'])
y = np.array(pizza_df['harga'])

X = X.reshape(-1, 1)


model = LinearRegression()
model.fit(X, y)

print(f'Intercept: {model.intercept_}')
print(f'Slope: {model.coef_}')
```

dan ini hasilnya

```
Intercept: 1.965517241379315
Slope: [0.9762931]
```

Bisa kita lihat di sini nilai **intercept** nya adalah **1.965517241379315,** dan nilai **slope** nya adalah **0.9762931.**

◆ Mencari Nilai Slope

Kita akan mempelajari lebih detail seputar proses kalkulasi dibalik nilai slope yang diperoleh, nilai slope pada linear regression bisa diperoleh dengan memanfaatkan formula sebagai berikut

$ß = cov(x, y) / var(x)$

Jadi nilai beta atau nilai slope tadi diperoleh dengan cara melakukan proses pembagian antara nilai **covariance** dari **x** dan **y**,dibagi dengan nilai **variance** dari **x** nya, di mana **x** ini merepresentasikan sekumpulan nilai features sedangkan **y** nya, merepresentasikan sekumpulan nilai target.

```
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
pizza = {'diameter': [6, 8, 10, 14, 18],
         'harga': [7, 9, 13, 17.5, 18]}

pizza_df = pd.DataFrame(pizza)

X = np.array(pizza_df['diameter'])
y = np.array(pizza_df['harga'])

X = X.reshape(-1, 1)


model = LinearRegression()
model.fit(X, y)

print(f'X:\n{X}\n')
print(f'X flatten: {X.flatten()}\n')
print(f'y: {y}')
```

Pertama-tama kita akan lihat dulu **X** & **y** kita denga eksekusi kode di atas.

```
X:
[[ 6]
 [ 8]
 [10]
 [14]
 [18]]

X flatten: [ 6  8 10 14 18]

y: [ 7.   9.  13.  17.5 18. ]
```

Kalau kita lihat disini nilai **X** setelah kita melakukan proses reshaping, Jadi pada dasarnya nilai **X** kita ini berbentuk 2 dimensi dengan jumlah baris 5 dan jumlah kolomnya 1, untuk melakukan proses kalkulasi **covariance** dan **variance** ada baiknya kita **flatten** kan dulu, atau kita kembalikan ke bentuk asalnya.

Utuk mengembalikan ke bentuk asal, kita bisa gunakan method **flatten,** bisa kita lihat efek setelah kita memanggil **X.flatten,** dari yang tadinya dua dimensi kembali menjadi satu dimensi, flatten berarti diratakan kembali.

Disini juga kita akan merevisi kembali nilai **y** kita,kalau kita lihat disini nilainya sudah satu dimensi, berarti tidak perlu Kita flatten kan.

Variance

Berikutnya kita akan menghitung nilai **variance**.

```
variance_x = np.var(X.flatten(), ddof=1)
print(f'variance: {variance_x}')
```

proses perhitungan nilai **variance** juga sebenarnya cukup mudah, di sini Kita tinggal panggil saja **np.var,** ini akan membutuhkan dua parameter, parameter yang pertama adalah nilai dari variabel **X** yang sudah di flatten kan, lalu disini kita juga membutuhkan parameter kedua yaitu **ddof,** yang kita set sebagai 1.

Bagi kalian yang masih bingung dengan istilah **ddof** ini ,ada baiknya kalian untuk mengacu kembali ke pembelajaran statistika, karena istilah ini bisa kalian temui di statistika, termasuk juga sebenarnya nilai variance, karena variance maupun covariance sebenarnya bisa kita pelajari di topik pembelajaran statistika.

Disini hasil perhitunga nvariance nya kita tampung ke dalam variabel **variance_x,** yang kemudian ditampilkan ke layar.

kita coba eksekusi kode nya

```
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

pizza = {'diameter': [6, 8, 10, 14, 18],
         'harga': [7, 9, 13, 17.5, 18]}

pizza_df = pd.DataFrame(pizza)

X = np.array(pizza_df['diameter'])
y = np.array(pizza_df['harga'])

X = X.reshape(-1, 1)


model = LinearRegression()
model.fit(X, y)
variance_x = np.var(X.flatten(), ddof=1)
print(f'variance: {variance_x}')
```

ini dia hasilnya

```
variance: 23.2
```

nilai variance nya adalah **23.2,**setelah variance nya ketemu, berikutnya kita akan coba kalkulasi nilai covariance nya.

Covariance

Di sini untuk menghitung covariance juga cukup mudah

```
np.cov(X.flatten(), y)
covariance_xy = np.cov(X.transpose(), y)[0][1]
print(f'covariance: {covariance_xy}')
```

kita cukup panggil saja **np.cov** ,lalu membutuhkan dua parameter,parameter pertama adalah nilai **X,** yang sudah di flatten kan, lalu parameter keduanya adalah **y**.

Disini pemanggilan fungsi **np.cov** ini akan menghasilkan yang namanya covarians matriks.

```
Array([23.2, 22.65],
      [22.65, 24.3])
```

Disini matriksnya terdiri dari 2 kolom dan 2 baris ,tetapi sebenarnya nilai covariance yang mau kita gunakan adalah yang berada dalam diagonal **22.65 ,**kalau kita lihat kedua nilai ini sama persis, yang mau kita ambil adalah nilai ini nya.

Berarti disini untuk mengambil nilai covariance nya, setelah kita melakukan proses pemanggilan fungsi **np.cov(X.flatten(), y),** maka nilainya disini perlu kita ambil yaitu, indeks ke 0, lalu dimensi keduanya indeks ke 1, karena kita mau ambil yang **22.65** saja. lalu hasil dari konferensi ini kita tampung ke dalam variabel **covariance_xy.**

Kita coba eksekusi kodenya

```
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
pizza = {'diameter': [6, 8, 10, 14, 18],
         'harga': [7, 9, 13, 17.5, 18]}

pizza_df = pd.DataFrame(pizza)

X = np.array(pizza_df['diameter'])
y = np.array(pizza_df['harga'])

X = X.reshape(-1, 1)


model = LinearRegression()
model.fit(X, y)

np.cov(X.flatten(), y)
covariance_xy = np.cov(X.flatten(), y)[0][1]
print(f'covariance: {covariance_xy}')
```

hasilnya ini merupakan nilai covariance **X** dan **y** nya.

```
Covariance: 22.65
```

Slope

Setelah nilai variance dan nilai covariance nya kita temukan, berikutnya kita bisa menggunakan formula $ß = cov(x, y) / var(x)$, dimana nilai **slope** itu bisa kita hasilkan atau kita peroleh dengan cara melakukan pembagian nilai **covariance** dengan nilai dari **variance** nya.

```
slope = covariance_xy / variance_x
print(f'slope: {slope}')
```

di sini **slope** = **covariance_xv** / **variance_x** dan kita tampung nilainya ke dalam variabel **slope** untuk selanjutnya kita tampilkan ke layar.

kita coba eksekusi kode nya.

```
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

pizza = {'diameter': [6, 8, 10, 14, 18],
         'harga': [7, 9, 13, 17.5, 18]}
```

```
pizza_df = pd.DataFrame(pizza)

X = np.array(pizza_df['diameter'])
y = np.array(pizza_df['harga'])

X = X.reshape(-1, 1)


model = LinearRegression()
model.fit(X, y)

variance_x = np.var(X.flatten(), ddof=1)
np.cov(X.flatten(), y)
covariance_xy = np.cov(X.flatten(), y)[0][1]
slope = covariance_xy / variance_x
print(f'slope: {slope}')
```

hasilnya bisa nampak

```
slope: 0.9762931034482758
```

disini nilai slope nya adalah **0.9762931034482758**, dan kalau kita kembali ke atas, nilainya sama persis yaitu **0.9762931034482758**.

◆ Mencari Nilai Intercept

Setelah kita mempelajari proses kalkulasi dibalik nilai slope, berikutnya kita akan pelajari proses kalkulasi dibalik nilai intercept.

Nilai intercept Pada linear regression bisa diperoleh dengan memanfaatkan formula berikut

$a = \bar{y} - ßx$

Nilai Alfa atau nilai intercept ini diperoleh dengan cara menselisihkan nilai rata rata dari **y** atau nilai rata-rata dari targetnya, di selisihlan dengan hasil perkalian antara slope dengan nilai rata-rata dari **x.**

Kita akan coba kalkulasi kan dengan menggunakan script

```
intercept = np.mean(y) - slope*np.mean(X_
print(f'Intercept: {intercept}')
```

Untuk mencari nilai rata-rata dari **y**, kita bisa panggil **np.mean(y),** lalu kita selisihkan yang kita kurangkan dengan Perkalian antara nilai slope yang sebelumnya kita hitung, lalu kita coba kalikan dengan **np.mean** dari **X,** dan nilai nya ini kita tampung ke dalam variable intercept untuk selanjutnya kita cetak.

kita coba eksekusi

```
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

pizza = {'diameter': [6, 8, 10, 14, 18],
         'harga': [7, 9, 13, 17.5, 18]}

pizza_df = pd.DataFrame(pizza)

X = np.array(pizza_df['diameter'])
y = np.array(pizza_df['harga'])

X = X.reshape(-1, 1)


model = LinearRegression()
model.fit(X, y)

variance_x = np.var(X.flatten(), ddof=1)
np.cov(X.flatten(), y)
covariance_xy = np.cov(X.flatten(), y)[0][1]
slope = covariance_xy / variance_x
intercept = np.mean(y) - slope * np.mean(X)
print(f'Intercept: {intercept}')
```

hasilnya bisa nampak di sini

```
Intercept: 1.9655172413793114
```

Nilai intercept adalah **1.9655172413793114,** dan kalau kita bandingkan dengan hasil sebelumnya, nilainya sama yaitu **1.9655172413793114.**

Bedanya di atas kita tidak melakukan proses perhitungan secara manual, kita tinggal mengambil saja parameter yang ada dalam objek model kita, kita sedangkan kalau di sini kita lakukan proses secara manual, karena kita mau memahami proses perhitungan dibalik nilai slope dan nilai interceptnya.

◆ Prediksi Harga Pizza

Setelah memahami proses kalkulasi nilai slope dan intercept, berikutnya kita akan gunakan model linier regression yang telah kita training sebelumnya untuk melakukan prediksi harga pizza berdasarkan ukuran diameternya.

Untuk kasus Kali ini kita akan memprediksi harga pizza berdasarkan tiga nilai diameter.

Berikut akan kita demokan prosesnya.

```
diameter_pizza = np.array([12, 20, 23]).reshape(-1, 1)
diameter_pizza
```

Disini kita sudah menyiapkan tiga nilai diameter yaitu **12**, **20**, **23** sebagai feature yang mau kita prediksi, disini ketiga nilai ini akan kit bundle sebagai suatu numpy array, makanya di sini kita panngil **np.array,** karena disini kita hanya memiliki satu nilai features, yaitu diameter saja maka arahnya perlu kita reshape untuk menjadi array 2 dimensi, makanya di sini kita Panggil **.reshape(-1, 1)** ,lalu nilai yang terbentuk ini kita tampung ke dalam variabel diameter **diameter_pizza.**

Coba kita eksekusi kode nya

```
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

pizza = {'diameter': [6, 8, 10, 14, 18],
         'harga': [7, 9, 13, 17.5, 18]}

pizza_df = pd.DataFrame(pizza)

X = np.array(pizza_df['diameter'])
y = np.array(pizza_df['harga'])

X = X.reshape(-1, 1)


model = LinearRegression()
model.fit(X, y)

diameter_pizza = np.array([12, 20, 23]).reshape(-1, 1)
print(diameter_pizza)
```

dai ini hasilnya

```
array([[12]
 [20]
 [23]])
```

ini merupakan nilai features dari diameter pizza yang mau kita prediksi harganya, disini array sudah array 2 dimensi yang terdiri dari 3 baris dan satu kolom, artinya sudah sesuai dengan ketentuan pada saat scikit-learn.

Berikutnya kita sudah siap untuk melakukan prediksi harga pizza berdasarkan ketiga nilai diameter ini

```
prediksi_harga = model.predict(diameter_pizza)
prediksi_harga
```

di sini kita panggil saja **model.predict** dan kita msukkan **diameter_pizza** nya.lalu hasil prediksi nya akan kita tampung ke dalam variabel **prediksi_harga** dan selanjutnya kita coba tampilkan ke layar.

Kita coba eksekusi kode nya

```
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

pizza = {'diameter': [6, 8, 10, 14, 18],
         'harga': [7, 9, 13, 17.5, 18]}

pizza_df = pd.DataFrame(pizza)

X = np.array(pizza_df['diameter'])
y = np.array(pizza_df['harga'])

X = X.reshape(-1, 1)


model = LinearRegression()
model.fit(X, y)

diameter_pizza = np.array([12, 20, 23]).reshape(-1, 1)
prediksi_harga = model.predict(diameter_pizza)
print(prediksi_harga)
```

dan ini dia hasilnya

```
array([13.68103448 21.49137931 24.42025862])
```

Bisa kita lihat di sini, ini merupakan ketiga harga pizza yang diprediksi dengan memanfaatkan model linear regression yang sudah kita training sebelumnya.Dan kalau kita mau melihat dengan lebih jelas, kita akan coba melakukan proses looping dan printing

```
for dmtr, hrg in zip(diameter_pizza, prediksi_harga):
        print(f'Diameter: {dmtr} prediksi harga: {hrg}')
```

kita coba eksekusi kode nya

```
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

pizza = {'diameter': [6, 8, 10, 14, 18],
         'harga': [7, 9, 13, 17.5, 18]}

pizza_df = pd.DataFrame(pizza)

X = np.array(pizza_df['diameter'])
y = np.array(pizza_df['harga'])

X = X.reshape(-1, 1)


model = LinearRegression()
model.fit(X, y)

diameter_pizza = np.array([12, 20, 23]).reshape(-1, 1)
prediksi_harga = model.predict(diameter_pizza)
for dmtr, hrg in zip(diameter_pizza, prediksi_harga):
      print(f'Diameter: {dmtr} prediksi harga: {hrg}')
```

dan ini hasilnya

```
Diameter: [12] prediksi harga: 13.681034482758621
Diameter: [20] prediksi harga: 21.491379310344826
Diameter: [23] prediksi harga: 24.42025862068965
```

Bisa kita lihat untuk pizza dengan diameter **12** ini kita prediksi memiliki harga **13.681034482758621,** sedangkan pizza dengan ukuran diameter **20** kita prediksi memiliki harga **21.491379310344826,** dan terakhir pizza dengan ukuran **23** kita prediksi memiliki harga **24.42025862068965.**

◆ Evaluasi Simple Linear Regression Model

Kita akan mempelajari Salah satu teknik yang dapat kita gunakan untuk melakukan evaluasi performa dari linear regression model, untuk keperluan ini kita akan bentuk kembali training set yang pernah kita gunakan sebelumnya,selain itu kali ini kita juga akan sertakan testing setnya.

Training & Testing Dataset

```
X_train = np.array([6, 8, 10, 14, 18]).reshape(-1, 1)
y_train = np.array([7, 9, 13, 17.5, 18])

X_test = np.array([8, 9, 11, 16, 12]).reshape(-1, 1)
y_test = np.array([11, 8.5, 15, 18, 11])
```

Disini kita bentuk dua set data, yang pertama adalah train set, yang kedua adalah testing set, untuk train set nya juga lengkap ya di sini ada **X_train** dan **y_train** ,nilai ini sebenarnya sama persis dengan nilai yang sebelumnya kita gunakan, tetapi disini kita langsung bentuk dengan menggunakan numpy array.

Karena disini hanya terdiri dari 1 feature, sedangkan scikit-learn membutuhkan feature dalam bentuk matriks atau array 2 dimensi, maka kita perlu reshape dengan menggunakan .**reshape(-1, 1),** berikutnya nilainya kita masukkan ke dalam variabel **X_train** maupun **y_train.**

Demikian juga untuk testing set, testing set ini kita bentuk juga dalam format numpy array yang akan kita tampung ke dalam variabel X**_test** dan **y_test.**

Di sini Kita sudah punya **X_train, y_train,** X**_test** dan **y_test.** **X** merepresentasikan data features, **y** nya merepresentasikan data target.

Training simple Linear Regression Model

Selanjutnya disini kita akan bentuk kembali objek modelnya untuk kita training.

```
model = LinearRegression()
model.fit(X_train, y_train)
```

Di sini kita Panggil **LinearRegression,** kita bentuk kembali object dari class linear regression ini lalu kita tampung ke dalam variabel **model,** dan berikutnya objek modelnya kita training dengan memanggil method **fit,** dan menyertakan **X_train, y_train** sebagai parameternya.

Evaluasi Linear Regression Model dengan Coefficient of Determination atau R-squared(R^2)

Selanjutnya kita akan berfokus pada proses evaluasi dari modelnya,teknik evaluasi yang akan kita pelajari kali ini adalah Coefficient of Determination atau biasa dikenal sebagai R-squared(R^2).

Bagi kalian yang butuh dan ingin mempelajari lebih lanjut kita juga menyatakan referensi dari halaman Wikipedia, jadi bagi kalian yang ingin mempelajari lebih lanjut mengenai coefficient of determination kalian bisa pelajari di sini.

Refrensi : https://en.wikipedia.org/wiki/Coefisient_of_determination

Berikut akan kita demokan mekanisme untuk menggunakan teknik evaluasi ini dengan memanfaatkan scikit-learn.

```
from sklearn.metrics import r2_score

y_pred = model.predict(X_test)
r_squared = r2_score(y_test, y_pred)

print(f'R-squared: {r_squared}')
```

Disini pertama-tama kita akan coba impor dulu metrics evaluasinya Dengan cara memanggil **from sklearn.metrics import** r2-score, yang kita Import adalah r2-score atau squared scores nya,

selanjutnya di sini kita akan melakukan proses prediksi nilai target dengan memanfaatkan nilai features pada testing set kita, makanya di sini kita Panggil **model.predict(X_test),** dan nilai prediksinya kita akan lambung ke dalam variabel **y_pred,** berarti di sini kita memiliki nilai **y** hasil prediksi, dan juga nilai **y** yang real,dari dataset testing kita, yang ditampung dalam variabel **y_test.**Kedua nilai ini mau kita ukur atau mau kita bandingkan, caranya di sini Kita tinggal panggil saja fungsin **r2_score(y_test, y_pred)** ,dan hasil pengukuran mereknya ini akan kita tampung ke dalam variabel **r_squared** untuk selanjutnya kita tampilkan ke layar.

Kita eksekusi kode nya

```
import numpy as np
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression

X_train = np.array([6, 8, 10, 14, 18]).reshape(-1, 1)
y_train = np.array([7, 9, 13, 17.5, 18])

X_test = np.array([8, 9, 11, 16, 12]).reshape(-1, 1)
y_test = np.array([11, 8.5, 15, 18, 11])

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
r_squared = r2_score(y_test, y_pred)

print(f'R-squared: {r_squared}')
```

dan ini hasil nya

```
R-squared: 0.6620052929422553
```

Bisa kita lihat di sini ya nilai R-squared nya adalah **0.6620052929422553** , nilai R-squared ini semakin ia mendekati 1 itu semakin baik ,semakin dia menjauhi 1 atau mendekati 0, itu sebenarnya mungkin buruk, lalu ketika kita memiliki model yang kualitasnya memang benar-benar buruk sekali maka kita akan mendapati nilai R-square dengan nilai negatif.

R-quared ini hanya merupakan salah satu teknik evaluasi model machine learning yang bisa kita gunakan, pada beberapa sesi pembelajaran selanjutnya kita juga akan mempelajari sejumlah teknik evaluasi model lainnya.

◆ Mencari Nilai R-squared(R^2)

Nilai R-squared yang diusung oleh scikit-learn ini seringkali disalah tafsirkan sebagai nilai R-square pada pearson correlation coefficient, padahal nilai R-squared di sini mengacu pada nilai Coeficient of determination.

Oleh karenanya disini kita akan coba pelajari bagaimana scikit-learn melakukan kalkulasi nilai R-squared ini.

Kurang lebih seperti ini formulanya

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

$$SS_{res} = \Sigma_i^n =_1 (y_i - f(x_i))^2$$

$$SS_{tot} = \Sigma_i^n (y_i - \bar{y})^2$$

R-squared ini diperoleh dengan cara 1 di selisihkan dengan pembagian antara nilai SS_{res} ,SS_{res} ini merepresentasikan Sum Square residual, ini dibagi dengan SS_{tot} ,SS_{tot} ini merepresentasikan sum square total.

Sum Square resudial ini pada dasarnya adalah proses perhitungan selisih antara setiap nilai **y** pada testing set dengan nilai prediksinya, lalu nilai selisihnya ini kita pangkatkan 2 dan kita jumlahkan secara keseluruhan, ini akan membentuk Sum Square residua,Dengan kata lain sebenarnya di sini kita menghitung nilai error atau akumulasi nilai errornya. lalu berikutnya SS_{tot} atau sum square total, ini pada dasarnya adalah proses perhitungan selisih nilai untuk setiap nilai **y** pada testing set dengan nilai rata-ratanya, makanya di sini y_i diselesaikan dengan $\bar{y}$ atau y Min, dan nilai selisih ini kita pangkatkan 2, lalu kita akan akumulasikan dengan keseluruhan nilai yang ada, dan nilainya ini dikenal dengan istilah sum square total.

Disini kita juga sudah menyiapkan script yang mensimulasikan proses perhitungan sum square residual, kalau kita perhatikan di sini ya kita banyak menggunakan proses yang namanya list comprehension.

```
ss_res = sum([(y_i - model.predict(x_i.reshape(-1, 1))[0])**2
      for x_i, y_i in zip(X_test, y_test)])
print(f'ss_res: {ss_res}')
```

kita coba eksekusi kodenya

```
import numpy as np
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression

X_train = np.array([6, 8, 10, 14, 18]).reshape(-1, 1)
y_train = np.array([7, 9, 13, 17.5, 18])

X_test = np.array([8, 9, 11, 16, 12]).reshape(-1, 1)
y_test = np.array([11, 8.5, 15, 18, 11])

model = LinearRegression()
model.fit(X_train, y_train)

ss_res = sum([(y_i - model.predict(x_i.reshape(-1, 1))[0])**2
      for x_i, y_i in zip(X_test, y_test)])
print(f'ss_res: {ss_res}')
```

dan ini hasilnya

```
ss_res: 19.1980993608799
```

Ini merupakan nilai sum square residual nya, berikutnya ya di sini kita juga akan melakukan proses kalkulasi untuk nilai sum square total nya.

Kita coba eksekusi kode nya

```
import numpy as np
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression

X_train = np.array([6, 8, 10, 14, 18]).reshape(-1, 1)
y_train = np.array([7, 9, 13, 17.5, 18])

X_test = np.array([8, 9, 11, 16, 12]).reshape(-1, 1)
y_test = np.array([11, 8.5, 15, 18, 11])

model = LinearRegression()
model.fit(X_train, y_train)

mean_y = np.mean(y_test)
ss_tot = sum([(y_i - mean_y)**2 for y_i in y_test])
```

```
print(f'ss_tot: {ss_tot}')
```

dan ini hasilnya

```
ss_tot: 56.8
```

Ini merupakan nilai dari sum square totalnya

Setelah kita mendapatkan nilai sum square residual dan nilai sum square total, maka selanjutnya kita bisa melakukan proses perhitungan untuk R-square.

```
r_squared = 1 - (ss_res / ss_tot)
print(f'R-squared: {r_squared}')
```

Disini nilai R-suared bisa kita peroleh dengan cara, 1 di selisihkan dengan hasil pembagian antara sum squared residual dibagi dengan sum square total, yang nilainya kita tampung ke dalam variabel r_square dan berikutnya kita akan coba tampilkan ke layar.

Kita eksekusi kode nya

```
import numpy as np
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression

X_train = np.array([6, 8, 10, 14, 18]).reshape(-1, 1)
y_train = np.array([7, 9, 13, 17.5, 18])

X_test = np.array([8, 9, 11, 16, 12]).reshape(-1, 1)
y_test = np.array([11, 8.5, 15, 18, 11])

model = LinearRegression()
model.fit(X_train, y_train)

ss_res = sum([(y_i - model.predict(x_i.reshape(-1, 1))[0])**2
      for x_i, y_i in zip(X_test, y_test)])

mean_y = np.mean(y_test)
ss_tot = sum([(y_i - mean_y)**2 for y_i in y_test])

r_squared = 1 - (ss_res / ss_tot)
print(f'R-squared: {r_squared}')
```

dan ini hasilnya

```
R-squared: 0.6620052929422553
```

Hasilnya R-squared adalah **0.6620052929422553,** dan kalau kita bandingkan dengan hasil perhitungan menggunakan metrics nya scikit-learn, ini sama persis hasilnya adalah **0.6620052929422553,** bedanya disini menggunakan metrics dari scikit-learn sedangkan kalau di sini yang kita lakukan prosesnya secara manual.

◆ Bab 6 : Classification dengan KNN(K Nearest Neighbours)

Dalam ini kita akan mempelajari model machine learning lain yang juga umum ditemui di kelas machine learning yaitu K Nearst Neighbours atau biasa di singkat KNN.

KNN adalah model machine learning yang dapat digunakan untuk melakukan prediksi berdasarkan kedekatan karakteristik dengan sejumlah tetangga terdekat.

Prediksi yang dilakukan dapat diterapkan baik pada classification maupun regression tasks.

Dsini berarti KNN ini bisa digunakan baik untuk classification maupun untuk regression, dalam sesi pembelajaran kita kali ini, kita akan berfokus pada penerapan KNN untuk classification tasks, bagi kalian yang masih bingung dengan istilah classification tasks, kita sangat menyarankan untuk mempelajari kembali bab sebelumnya di seri belajar mesin learning ini,

Bagi kalian yang membutuhkan referensi lebih lanjut terkait KNN, Kalian juga bisa memanfaatkan halaman Wikipedia berikut ini.

Refrensi: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

Ini adalah halaman Wikipedia untuk KNN, Jadi bagi kalian yang tertarik, kalian bisa mempelajari lebih lanjut pada halaman Wikipedia ini.

◆ Sample Dataset

Pertama-tama kita akan persiapkan terlebih dahulu dataset yang akan kita pergunakan dalam sesi pembelajaran ini, kita akan membentuk dataset yang berisi daftar berat dan tinggi badan sejumlah partisipan, beserta gender atau jenis kelaminnya, dataset ini akan kita tampung kedalam format Pandas dataframe.

berikut akan kita demokan caranya

```
import pandas as pd

sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }

sensus_df = pd.DataFrame(sensus)
print(sensus_df)
```

Disini pertama-tama kita akan import dulu modul pandas nya dengan **import pandas as pd** lalu,berikutnya disini kita juga siapkan suatu variabel yang digunakan untuk menampung Suatu data dictionary, dimana dictionary ini akan menampung 4 buah keys yaitu, tinggi, berat, dan jk, jk ini nanti akan berasosiasi dengan daftar jenis kelamin, sedangkan tinggi dan berat ini akan berasosiasi dengan nilai tinggi badan dan juga berat badan, lalu selanjutnya dictionary ini kita tampung ke dalam variabel dengan nama **sensus**,dictionary sensus ini kemudian akan kita bentuk menjadi suatu pandas DataFrame,oleh karenanya di sini kita Panggil **pd.DataFrame(sensus)**, lalu objek DataFrame nya kita tampung ke dalam variabel **sensus_df** untuk berikutnya kita tampilkan kalayar.

dan ini hasilnya

	tinggi	berat	jk
0	158	64	pria
1	170	86	pria
2	183	84	pria
3	191	80	pria
4	155	49	wanita
5	163	59	wanita
6	180	67	wanita
7	158	54	wanita
8	178	67	wanita

Kalau kita lihat dataset kita kali ini, terdiri dari tiga buah kolom yaitu tinggi Berat,dan jk, dalam sesi pembelajaran kita kali ini, kita akan membentuk model mesin learning sederhana yang dapat digunakan untuk memprediksi gender atau jenis kelamin seseorang berdasarkan data tinggi dan berat badannya.

Dengan kata lain, disini data tinggi dan berat badan akan berperan sebagai features sedangkan jenis kelamin akan berperan sebagai target.

◆ Visualisasi Data

Disini kita akan coba visualisasikan dataset sensus yang telah kita bentuk sebelumnya dalam scatterplot, dimana sumbu x akan berasosiasi dengan tinggi badan sedangkan sumbu y akan berasosiasi dengan berat badan, di sini kita juga akan menggunakan warna marker yang berbeda untuk merepresentasikan jenis kelamin pria dan wanita.

berikut akan kita demokan caranya

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
for jk, d in sensus_df.groupby('jk'):
      ax.scatter(d['tinggi'],d[‘berat'], label=jk)

plt.legend(loc='upper left')
plt.title('Sebaran Data Tinggi Badan, Berat Badan, dan Jenis Kelamin')
plt.xlabel('Tinggi Badan (cm)')
plt.ylabel('Berat Badan (kg)')
plt.grid(True)
plt.show()
```

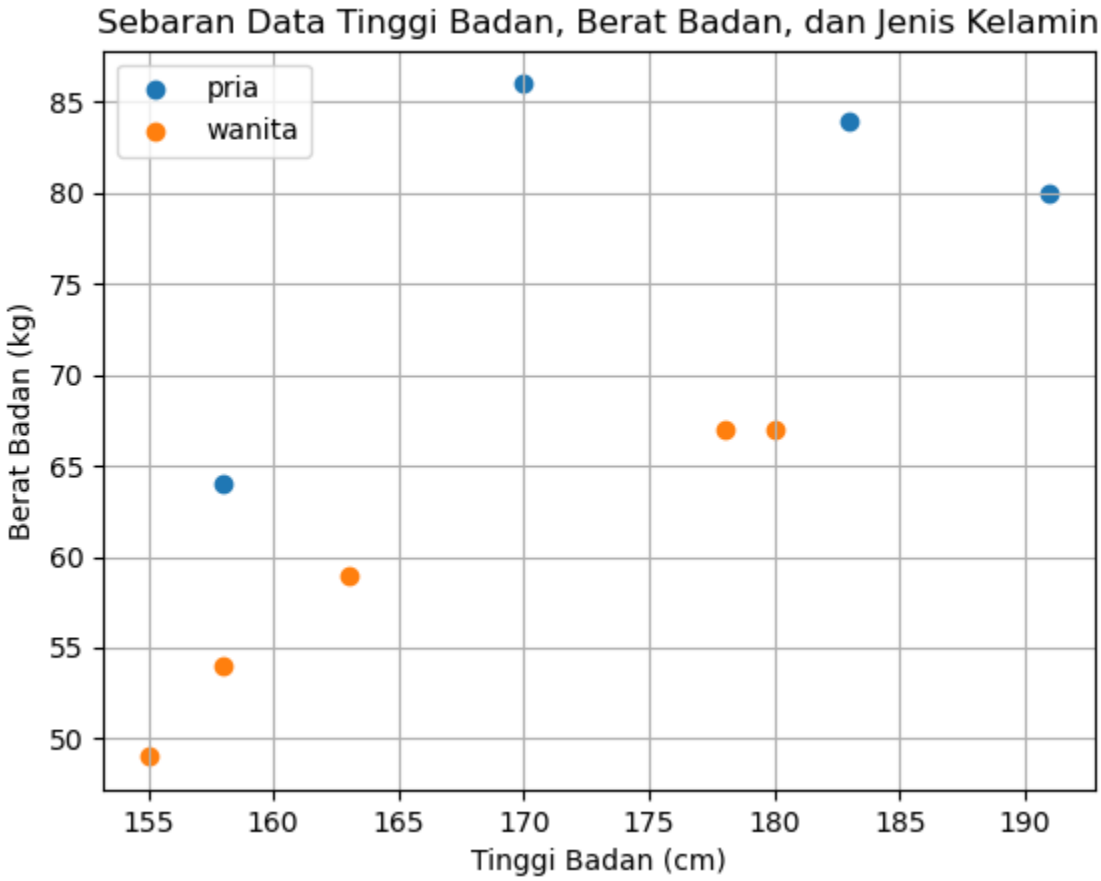
Disini pertama-tama kita import dulu modul penting kita **import matplotlib.pyplot as plt,** lalu berikutnya disini kita akan membentuk **subplots** dimana kita akan menangkap objek sigure dan aksesnya,lalu selanjutnya kita juga akan melakukan looping terhadap data sensus_df ini, dimana sensus_df nya kita grouping dulu berdasarkan **jk** atau berdasarkan jenis kelamin nya.Disini di tiap

literasinya kita akan menangkap 2 nilai, yaitu JK yang merepresentasikan jenis kelamin berdasarkan pengelompokannya, dan yang 2 adalah **d**,**d** ini akan berisi sekumpulan basis data dari kelompok jenis kelamin tertentu, lalu berikutnya karena kita ingin membentuk scatterplot maka kita panggil **ax.scatter.** Dimana sumbu **x** nya akan diasosiasikan dengan data baris dengan kolom tinggi sedangkan sumbu **y** nya akan diasosiasikan dengan data baris dengan kolom berat, lalu disini kita juga sertakan label dimana labelnya ini akan Kita sesuaikan dengan kelompok jenis kelaminnya.

kita coba eksekusi dulu kode nya

```
import pandas as pd
import matplotlib.pyplot as plt

sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }

sensus_df = pd.DataFrame(sensus)
fig, ax = plt.subplots()
for jk, d in sensus_df.groupby('jk'):
      ax.scatter(d['tinggi'],d['berat'], label=jk)

plt.legend(loc='upper left')
plt.title('Sebaran Data Tinggi Badan, Berat Badan, dan Jenis Kelamin')
plt.xlabel('Tinggi Badan (cm)')
plt.ylabel('Berat Badan (kg)')
plt.grid(True)
plt.show()
```

dan ini hasilnya



Disini kita juga sertakan beberapa keterangan tambahan diantaranya adalah **legend,** kalau kita lihat disini **plt.legend** ini akan berasosiasi dengan label yang ada di **label=jk,** kalau di sini kita tambahkan parameter **label**, bakal lainnya hanya akan menampilkan warna biru dan orange saja tanpa disertai label pria dan wanita, oleh karenanya disini sangatlah penting untuk menyertakan **label.**

Lalu berikutnya suatu kita memanggil **plt.legend,** kita juga menyatakan parameter **loc,** parameter **loc** ini digunakan untuk mengatur lokasi atau posisi dari legend nya, disini loc nya kita set sebagai **upper left,** berarti posisinya ada di sudut kiri atas, selanjutnya sebagai data tambahan disini kita juga menyertakan judul lalu label untuk sumbu **x** dan juga label untuk sumbu **y,** di sini kita ada **title**, ada **x label** dan ada **y label.**

Selain itu disini kita juga menyertakan upper left yang kita beri nilai **True** untuk memunculkan grid, dan terakhir ya di sini Kita juga memanggil **plt.show** untuk memunculkan hasil floating kita.

Kalau kita hitung ya jumlah markernya ada 9 marker, kesembilan masker ini akan berasosiasi dengan setiap baris dari dataset kita, di dalam pandas DataFrame kita ini kita memiliki 9 entris atau 9 rows,oleh karenanya setiap baris ini akan menghasilkan satu datapoint pada skater plotnya.

Pada kasus kita kali ini tujuan utamanya adalah, kalau kita menemukan suatu datapoint baru maka datapoint tersebut akan coba diprediksi apakah dia masuk kedalam kategori pria atau wanita.

◆ Classification dengan KNN

Setelah kita memahami konteks dataset dan juga permasalahan nya, kita akan coba menerapkan KNN untuk melakukan klasifikasi jenis kelamin berdasarkan data tinggi dan berat badan, sesuai dengan namanya model machine learning yang satu ini akan melakukan prediksi dalam kasus ini adalah prediksi gender atau prediksi jenis kelamin berdasarkan kemiripan karakteristik atau features dengan dataset yang kita miliki. KNN juga termasuk salah satu model machine learning dasar yang wajib dikuasai.

◆ Preprocessing Dataset

sebelum kita melangkah ke proses training model, kita perlu melakukan beberapa penyesuaian pada data sensus ini, dimana kita akan konversikan data tinggi badan dan berat badan ke dalam numpy array untuk selanjutnya kita tampung ke dalam variabel x_train sebagai sekumpulan features untuk training set, lalu untuk kolom jenis kelamin juga akan kita konversikan menjadi numpy array untuk selanjutnya kita tabung ke dalam variabel y_train sebagai sekumpulan nilai target untuk traing set.

berikut akan kita demokan prosesnya

```
import numpy as np

X_train = np.array(sensus_df[['tinggi', 'berat']])
y_train = np.array(sensus_df['jk'])

print(f'X_train:\n{X_train}\n')
print(f'y_train: {y_train}')
```

Disini kita akan membentuk dua buah numpy array, untuk numpy array yang pertama akan kita bentuk berdasarkan kolom tinggi dan kolom berat dari **sensus-df**, sedangkan numpay array yang kedua akan kita bentuk berdasarkan kolom jk dari sensus-df, numpy array yang pertama akan kita tampung ke dalam variabel **X_train**, sedangkan sampai ada yang kedua akan kita tampung ke dalam variabel **y_train.**

Kita coba eksekusi kodenya

```
import numpy as np
import pandas as pd
sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }

sensus_df = pd.DataFrame(sensus)

X_train = np.array(sensus_df[['tinggi', 'berat']])
y_train = np.array(sensus_df['jk'])

print(f'X_train:\n{X_train}\n')
print(f'y_train: {y_train}')
```

dan ini hasilnya

```
X_train:
[[158  64]
 [170  86]
 [183  84]
 [191  80]
 [155  49]
 [163  59]
 [180  67]
 [158  54]
 [178  67]]

y_train: ['pria' 'pria' 'pria' 'pria' 'wanita' 'wanita' 'wanita' 'wanita' 'wanita']
```

X_train ini berisi sekumpulan nilai features, sedangkan y_train ini akan berisi sekumpulan nilai target, kalau kita perhatikan di sini, sekumpulan nilai features untuk training set yang kita miliki sudah dalam format array 2 dimensi, tipe datanya pun sudah berupa data numerik, artinya sekumpulan nilai features sudah siap digunakan untuk proses training, tapi kalau kita perhatikan nilai targetnya disini tipe datanya adalah string, sebenarnya untuk kasus classification dengan KNN, target dengan tipe data string tetap bisa diproses dengan baik hanya saja kita akan menghadapi kendala sewaktu melakukan evaluasi model, oleh karenanya disini kita akan konversikan nilai string ini menjadi numerik, karena jenis kelamin ini hanya terdiri dari 2 nilai dalam kasus ini adalah pria dan wanita maka kita bisa memanfaatkan **LabelBinarizer** untuk melakukan konversi nilai string ini menjadi numerik biner.

berikut akan kita demokan prosesnya

```
from sklearn.preprocessing import LabelBinarizer

lb = LabelBinarizer()
y_train = lb.fit_transform(y_train)
print(f'y_train:\n{y_train}')
```

Disini kita bisa memanfaatkan kelas preprocessing yang disertakan oleh scikit-learn yaitu **LabelBinarizer,** oleh karenanya di sini kita import dulu **from sklearn.preprocessing import** LabelBinarizer, lalu berikutnya kita akan coba bentuk objek dari class **LabelBinarizer,** ini caranya juga cukup mudah di sini Kita tinggal panggil saja **LabelBinarizer()**, objek yang terbentuk ini akan kita tampung ke dalam variabel **lb,** lalu objek ini akan kita manfaatkan untuk melakukan transformasi data yang terdapat dalam variabel y_train, oleh karenanya di sini kita Panggil **lb.fit_transform(y_train),** dan nilainya akan kita tampung lagi ke dalam variabel **y_train** ,selanjutnya disini kita juga akan melakukan print out nilai dari variabel y_train nya.

Kita coba eksekusi kode nya

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelBinarizer
sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }
```

```
sensus_df = pd.DataFrame(sensus)

X_train = np.array(sensus_df[['tinggi', 'berat']])
y_train = np.array(sensus_df['jk'])

lb = LabelBinarizer()
y_train = lb.fit_transform(y_train)
print(f'y_train:\n{y_train}')
```

dan ini dia hasilnya

```
y_train:
[[0]
 [0]
 [0]
 [0]
 [1]
 [1]
 [1]
 [1]
 [1]]
```

Bisa nampak di sini, nilai nya hanya ada 2, yaitu 0 dan 1, untuk kasus kita kali ini nilai 0 ini akan merepresentasikan data pria, sedangkan nilai 1 ini akan merepresentasikan data Wanita, hanya saja kalau kita perhatikan di sini, setelah melalui proses transformasi data dengan **LabelBinarizer,** sekumpulan nilai target yang kita miliki ini, sekarang tersimpan dalam format array 1 dimensi, oleh karenanya di sini kita perlu kembalikan menjadi array 1 dimensi dengan memanfaatkan method flatten.

berikut akan kita demokan caranya

```
y_train = y_train.flatten()
print(f'y_train: {y_train}')
```

Di sini caranya cukup mudah, Kita tinggal panggil saja **y_train.flatten()**, lalu hasil transformasinya akan kita kampung lagi ke dalam variabel **y_train** untuk kemudian kita cetak ke layar.

Kita coba eksekusi kodenya

```
import numpy as np
```

```
import pandas as pd
from sklearn.preprocessing import LabelBinarizer
sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }

sensus_df = pd.DataFrame(sensus)

X_train = np.array(sensus_df[['tinggi', 'berat']])
y_train = np.array(sensus_df['jk'])

lb = LabelBinarizer()
y_train = lb.fit_transform(y_train)
y_train = y_train.flatten()
print(f'y_train: {y_train}')
```

dan ini hasil nya

```
y_train: [0 0 0 0 1 1 1 1 1]
```

Sekarang sekumpulan nilai features dan targetnya sudah siap digunakan untuk training model.

◆ Training KNN Classification Model

Setelah training set nya kita persiapkan, selanjutnya kita akan melangkah ke proses training model, kali ini model machine learning yang akan kita gunakan adalah KNN.

berikut akan kita demokan prosesnya

```
from sklearn.neighbors import KNeighborsClassifier

k = 3
model = KNeighborsClassifier(n_neighbors=k)
model.fit(X_train, y_train)
```

Disini pertama-tama kita akan import dulu estimeter class nya,**from sklearn.neighbors import** KneighborsClassifier, di sini kita pilih KneighborsClassifier karena kita akan menggunakan KNN ini untuk classification tasks,disini terdapat satu parameter yang harus kita spesifikasikan nilainya, yaitu parameter **k,** untuk kasus kita disini nilai parameter **k** nya kita set sebagai 3, nilai parameter **k** ini digunakan untuk menentukan jumlah tetangga terdekat yang akan dilibatkan untuk proses prediksi, dan untuk kasus kita kali ini adalah 3 tetangga terdekat

Setelah nilai **k** nya Kita tentukan berikutnya kita akan bentuk objek dari kelas KneighborsClassifier, ini akan membutuhkan satu parameter **n_neighbors** , yang kita beri nilai **k** ,lalu berikutnya setelah objek ini terbentuk, kita tampung ke dalam variabel **model** untuk selanjutnya kita training dengan memanfaatkan **X_train** dan **y_train** Yang sudah kita persiapkan sebelumnya, disini proses terjadinya kita lakukan dengan cara memanggil **model.fit(X_train, y_train).**

Kita coba eksekusi kodenya

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelBinarizer
from sklearn.neighbors import KNeighborsClassifier
sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }

sensus_df = pd.DataFrame(sensus)

X_train = np.array(sensus_df[['tinggi', 'berat']])
y_train = np.array(sensus_df['jk'])

lb = LabelBinarizer()
y_train = lb.fit_transform(y_train)
y_train = y_train.flatten()
k = 3
model = KNeighborsClassifier(n_neighbors=k)
model.fit(X_train, y_train)
print(model.fit)
```

dan ini hasilnya

```
KNeighborsClassifier(n_neighbors=3)
```

Setelah model kita training, maka status model ini adalah train model atau model yang sudah di training, disini proses trainingnya terbilang cepat karena ukuran dataset yang kita miliki juga terbilang sangat kecil.

◆ Prediksi Jenis Kelamin

Setelah modal KNN classifcation nya kita training, tahapan selanjutnya kita akan gunakan train model ini untuk melakukan prediksi jenis kelamin berdasarkan data tinggi dan berat, disini kita akan siapkan terlebih dahulu data baru yang akan kita gunakan untuk prediksi.

```
tinggi_badan = 155
berat_badan = 70
X_new = np.array([tinggi_badan, berat_badan]).reshape(1, -1)
X_new
```

Disini semisal saja kita memiliki data tinggi badan 155 dan data berat badan 70,yang akan kita prediksi jenis kelaminnya apakah wanita atau pria, disini nilai features nya harus kita bentuk atau bundle dulu menjadi suatu numpy array, oleh karenanya di sini kita panggil **np.array** lalu kita bundle tinggi badan, berat badan nya, karena disini hanya terdapat satu dataset atau satu instance saja, maka disini perlu kita reshape menjadi array 2 dimensi dengan cara **.reshape(1, -1),** lalu berikutnya are ini akan kita tampung ke dalam variabel **X_new** dan kita coba tampilkan ke layar.

kita coba eksekusi dulu kodenya

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelBinarizer
from sklearn.neighbors import KNeighborsClassifier
sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }

sensus_df = pd.DataFrame(sensus)
```

```
X_train = np.array(sensus_df[['tinggi', 'berat']])
y_train = np.array(sensus_df['jk'])

lb = LabelBinarizer()
y_train = lb.fit_transform(y_train)
y_train = y_train.flatten()
k = 3
model = KNeighborsClassifier(n_neighbors=k)
model.fit(X_train, y_train)
tinggi_badan = 155
berat_badan = 70
X_new = np.array([tinggi_badan, berat_badan]).reshape(1, -1)
print(X_new)
```

dan ini hasilnya

```
[[155  70]]
```

Bisa nampak di sini array adalah array 2 dimensi, dimana jumlah barisnya adalah 1 dan jumlah kolomnya adalah 2, disini kita juga perlu mengingatkan kalian kembali bahwa untuk scikit-learn, sekumpulan nilai features yang bisa digunakan itu harus berada dalam format numpy array dengan ukuran 2 dimensi,dimana dimensi yang mengatur baris, itu akan berkorelasi dengan jumlah data, sedenganken dimensi mengatur kolom, itu akan berkorelasi dengan jumlah features nya.

setelah data tinggi dan berat badannya siap,selanjutnya kita akan melakukan prediksi gender dengan memanfaatkan model KNN classifier yang sudah kita training sebelumnya.

```
y_new = model.predict(X_new)
y_new
```

Disini caranya juga cukup mudah, kita panggil saja **model.predict(X_new)**, hasil prediksinya akan kita tampung ke dalam variabel **y_new** untuk selanjutnya kita coba tampilkan ke layar.

Kita coba eksekusi kodenya

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelBinarizer
```

```
from sklearn.neighbors import KNeighborsClassifier
sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }

sensus_df = pd.DataFrame(sensus)

X_train = np.array(sensus_df[['tinggi', 'berat']])
y_train = np.array(sensus_df['jk'])

lb = LabelBinarizer()
y_train = lb.fit_transform(y_train)
y_train = y_train.flatten()
k = 3
model = KNeighborsClassifier(n_neighbors=k)
model.fit(X_train, y_train)
tinggi_badan = 155
berat_badan = 70
X_new = np.array([tinggi_badan, berat_badan]).reshape(1, -1)
y_new = model.predict(X_new)
print(lb.inverse_transform(y_new))
```

dan ini haasilnya

```
['wanita']
```

Disini kita bisa lihat bahwa tinggi badan 155 dengan berat badan 70 ini diprediksi sebagai data **wanita,**oleh model yang sudah kita training sebelumnya.

◆ Visualisasi Nearest Neighbours

Untuk dapat memahami cara kerja KNN dengan lebih baik, kita akan coba visualisasikan posisi data baru tadi terhadap sekumpulan data tinggi dan berat badan yang kita gunakan sebelumnya untuk training model KNN.

berikut akan kita demokan prosesnya

```
fig, ax = plt.subplots()
for jk, d in sensus_df.groupby('jk'):
      ax.scatter(d['tinggi'],d['berat'], label=jk)

plt.scatter(tinggi_badan,
            berat_badan,
            marker='s',
            color='red',
            label='misterius')

plt.legend(loc='upper left')
plt.title('Sebaran Data Tinggi Badan, Berat Badan, dan Jenis Kelamin')
plt.xlabel('Tinggi Badan (cm)')
plt.ylabel('Berat Badan (kg)')
plt.grid(True)
plt.show()
```

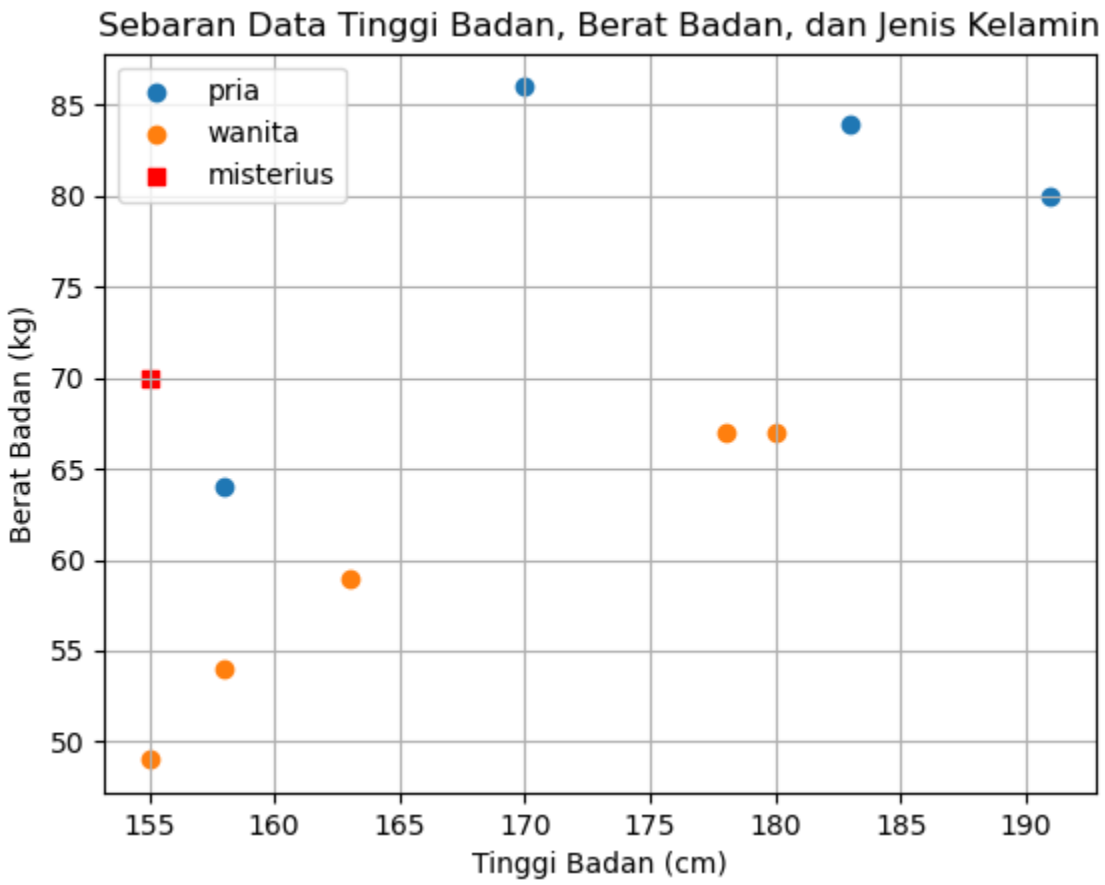
Caranya kurang lebih sama dengan apa yang pernah kita lakukan sebelumnya, hanya saja di sini kita akan menyertakan marker baru pada scatter plot kita, marker baru ini dihasilkan dengan memanggil **plt.scatter,** dimana nilai **x** nya itu berasal dari variabel tinggi_badan dan nilai **y** nya berasal dari berat_badan, sedangkan untuk marker nya kita beri nilai **s,** karena kita akan bentuk markernya sebagai Square atau kotak, lalu kita akan beri color atau warnanya merah dan labelnya kita beri label misterius.

kita coba eksekusi kode nya

```
import numpy as np
from sklearn.preprocessing import LabelBinarizer
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd
import matplotlib.pyplot as plt

sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }
sensus_df = pd.DataFrame(sensus)

X_train = np.array(sensus_df[['tinggi', 'berat']])
y_train = np.array(sensus_df['jk'])
```

```
lb = LabelBinarizer()
y_train = lb.fit_transform(y_train)
y_train = y_train.flatten()
k = 3
model = KNeighborsClassifier(n_neighbors=k)
model.fit(X_train, y_train)
tinggi_badan = 155
berat_badan = 70
X_new = np.array([tinggi_badan, berat_badan]).reshape(1, -1)
y_new = model.predict(X_new)

fig, ax = plt.subplots()
for jk, d in sensus_df.groupby('jk'):
      ax.scatter(d['tinggi'],d['berat'], label=jk)

plt.scatter(tinggi_badan,
            berat_badan,
            marker='s',
            color='red',
            label='misterius')

plt.legend(loc='upper left')
plt.title('Sebaran Data Tinggi Badan, Berat Badan, dan Jenis Kelamin')
plt.xlabel('Tinggi Badan (cm)')
plt.ylabel('Berat Badan (kg)')
plt.grid(True)
plt.show()
```

dan ini hasil nya



Kalau kita perhatikan secara visual, datapoint misterius ini kalau kita pilih 3 tetangga terdekatnya, maka 3 tetangga terdekatnya adalah, dua wanita dan satu adalah pria, oleh karenanya tidaklah mengherankan bila data misterius ini akan diprediksi sebagai wanita.

◆ Kalkulasi Distance (Euclidean Distance)

Setelah kita mempelajari cara kerja KNN secara umum atau secara general, berikutnya kita akan pelajari bagaimana KNN mengukur jarak atau kedekatan antara data baru yang diprediksi dengan Sekumpulan data lain pada dataset training, disini kita akan mengenal euclidean distance untuk mengukur Jarak antara satu datapoint dengan datapoint lainnya, untuk selanjutnya sejumlah **k** data poin dengan jarak terdekat inilah yang akan digunakan oleh KNN sebagai ini Kneighbours atau tetangga terdekatnya.

Bisa kita lihat di sini merupakan formula sederhana untuk euclidean distance, karena untuk kasus kita kali ini hanya terdapat 2 nilai features yaitu tinggi badan dan juga berat badan, maka formula dari euclidean distance nya bisa kita bentuk semacam ini

$$distance = \sqrt{(t_1 - t_2)^2 + (b_1 - b_2)^2}$$

t ini merepresentasikan tinggi badan,**b** ini merepresentasikan berat badan, t_1 berarti data tinggi badan dari datapoint yang pertama, t_2 ini merepresentasikan data tinggi badan dari datapoint yang ke-2,lalu b_1 ini merupakan data berat badan dari datapoint yang pertama dan b_2 ini merupakan berat badan dari datapoint yang kedua.

Formula ini dapat digunakan untuk mengukur jarak atau distance atau kedekatan antara data point pertama dengan data point kedua.Bagi kalian yang butuh mempelajari lebih lanjut mengenai euclidean ini, kalian bisa menggunakan halaman Wikipedia berikut ini

Refrensi: https://en.wikipedia.org/wiki/Euclidean_distance

Berikutnya kita akan coba demokan proses kalkulasi yang diterapkan oleh KNN dalam menentukan tetangga terdekat untuk suatu datapoint Nya.

```
misterius = np.array([tinggi_badan, berat_badan])
```

Disini datapoint misterius nya akan kita bentuk dulu ke dalam format numpy array dan numpy array yang terbentuk ini akan kita tampung ke dalam variabel misterius.

```
X_train
```

Lalu berikutnya kita juga akan Tampilkan sekumpulan nilai features dari training set kita.

Disini kita akan coba hitung jarak berdasarkan euclidean distance, antara datapoint misterius ini dengan setiap datapoint lain-lain pada dataset X_train ini.

Caranya cukup mudah di sini kita bisa memanfaatkan function euclidean yang sudah disertakan pada scipy.

```
from scipy.spatial.distance import euclidean
data_jarak = [euclidean(misterius, d) for d in X_train]
data_jarak
```

Oleh karenanya di sini kita Panggil **from scipy.spatial.distance import** euclidean, berikutnya kita tinggal pakai saja euclidean distance nya.Disini function euclidean ini digunakan untuk mengukur jarak atau kedekatan atau distance antara datapoint misterius dengan suatu datapoint lain di dalam variabel **X_train**,kembali di sini kita juga memanfaatkan yang namanya list comprehension, kumpulan nilai jarak ini lalu kita tampung ke dalam variabel **data_jarak** untuk kemudian kita cetak ke layar.

Pada tahapan berikutnya untuk mempermudah proses pengamatan kita akan bentuk kolom baru pada sensus_df ini dengan memanfaatkan data.

```
sensus_df['jarak'] = data_jarak
sensus_df.sort_values(['jarak'])
```

Oleh karenanya di sini Kita tinggal panggil saja **sensus_df['jarak'],** artinya disini kita akan membentuk kolom baru pada sensus_df dengan nama jarak, lalu datanya kita ambil dari variabel data jarak, untuk selanjutnya dataframe sensus_df ini akan kita urutkan nilainya berdasarkan kolom jarak.Pengurutan ini dilakukan secara ascending artinya dari nilai terkecil sampai dengan nilai ter besar

Kita coba eksekusi kode nya

```
import numpy as np
from sklearn.preprocessing import LabelBinarizer
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd
```

```
import matplotlib.pyplot as plt
from scipy.spatial.distance import euclidean

sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }
sensus_df = pd.DataFrame(sensus)

X_train = np.array(sensus_df[['tinggi', 'berat']])
y_train = np.array(sensus_df['jk'])

lb = LabelBinarizer()
y_train = lb.fit_transform(y_train)
y_train = y_train.flatten()
k = 3
model = KNeighborsClassifier(n_neighbors=k)
model.fit(X_train, y_train)
tinggi_badan = 155
berat_badan = 70
X_new = np.array([tinggi_badan, berat_badan]).reshape(1, -1)
y_new = model.predict(X_new)

misterius = np.array([tinggi_badan, berat_badan])
X_train

data_jarak = [euclidean(misterius, d) for d in X_train]
sensus_df['jarak'] = data_jarak
print(sensus_df.sort_values(['jarak']))
```

dan ini hasilnya

	tinggi	berat	jk	jarak
0	158	64	pria	6.708204
5	163	59	wanita	13.601471
7	158	54	wanita	16.278821
4	155	49	wanita	21.000000
1	170	86	pria	21.931712
8	178	67	wanita	23.194827
6	180	67	wanita	25.179357
2	183	84	pria	31.304952

3	191	80	pria	37.363083

Bisa kita lihat di sini datanya sudah terurut berdasarkan kolom jarak, karena untuk kasus kita kali ini nilai **k** nya adalah 3, oleh karenanya disini kita hanya akan memperhatikan 3 datapoint dengan jarak terdekat atau jarak terkecil dengan datapoint misterius kita.

Oleh karenanya bisa kita lihat di sini tiga baris pertama atau tiga neighbors terdekat yang akan kita perhatikan untuk melakukan proses prediksi untuk neighbors yang pertama jenis kelamin adalah pria, neighbors Kedua jenis kelamin adalah wanita dan neighbors ketiga jenis kelamin adalah wanita,berdasarkan data neighbors ini maka modal kene akan memprediksi datapoint misterius ini sebagai data wanita.

◆ Evaluasi KNN Classification Model

Setelah memahami Bagaimana KNN melakukan seleksi tetangga terdekat, selanjutnya kita akan mulai mempelajari beberapa metrics yang bisa kita gunakan untuk mengukur performa dari model machine-learning untuk kasus classification tasks,tapi sebelumnya kita akan siapkan terlebih dahulu testing setnya.

Testing set

```
X_test = np.array([[168, 65], [180, 96], [160, 52], [169, 67]])
y_test = lb.transform(np.array(['pria', 'pria', 'wanita', 'wanita'])).flatten()

print(f'X_test:\n{X_test}\n')
print(f'y_test:\n{y_test}')
```

Disini kita akan siapkan cassing setnya, dimana testing set yang kita persiapkan ini terdiri dari 4 datapoint, oleh karenanya disini kita akan siapkan numpy array yang terdiri dari 4 baris dan 2 kolom sebagai x-test,lalu berikutnya disini kita juga akan siapkan target yang terdiri dari 4 data, yaitu pria,pria,wanita,wanita.

Setiap elemen ini kan bertipe data string, dan nilai string ini perlu kita transformasikan menjadi nilai numerik biner dengan memanfaatkan **LabelBinarizer,** oleh karenanya di sini kita Panggil lb. transform ,kembali mengingatkan proses transformasi dari **LabelBinarizer** ini akan menghasilkan array 2 dimensi padahal yang kita harapkan untuk y_tes adalah array 1 dimensi, oleh karenanya di sini kita perlu memanggil method flatten.

Kita eksekusi kodenya

```
import numpy as np
from sklearn.preprocessing import LabelBinarizer
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd
import matplotlib.pyplot as plt
from scipy.spatial.distance import euclidean

sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }
sensus_df = pd.DataFrame(sensus)

X_train = np.array(sensus_df[['tinggi', 'berat']])
y_train = np.array(sensus_df['jk'])

lb = LabelBinarizer()
y_train = lb.fit_transform(y_train)
y_train = y_train.flatten()
k = 3
model = KNeighborsClassifier(n_neighbors=k)
model.fit(X_train, y_train)
tinggi_badan = 155
berat_badan = 70
X_new = np.array([tinggi_badan, berat_badan]).reshape(1, -1)
y_new = model.predict(X_new)

misterius = np.array([tinggi_badan, berat_badan])
X_train

data_jarak = [euclidean(misterius, d) for d in X_train]
sensus_df['jarak'] = data_jarak

X_test = np.array([[168, 65], [180, 96], [160, 52], [169, 67]])
y_test = lb.transform(np.array(['pria', 'pria', 'wanita', 'wanita'])).flatten()

print(f'X_test:\n{X_test}\n')
print(f'y_test:\n{y_test}')
```

dan ini hasilnya

```
X_test:
[[168  65]
 [180  96]
 [160  52]
 [169  67]]

y_test:
[0 0 1 1]
```

Kalau kita lihat formatnya disini sudah sesuai dengan apa yang kita butuhkan, dimana X_test ini terdiri dari sekumpulan nilai numerik yang di bundle atau yang dikemas sebagai array 2 dimensi, demikian juga nilai y_test nya yang terdiri dari sekumpulan nilai numerik biner dan di bundle dalam suatu numpy array 1 dimensi.

Prediksi terhadap testing set

Setelah testing set nya siap, selanjutnya kita akan melakukan prediksi terhadap testing set ini dengan memanfaatkan model KNN classifier yang sudah kita training sebelumnya.

```
y_pred = model.predict(X_test)
y_pred
```

Oleh karenanya di sini kita Panggil ya **model.predict,** yang kita prediksi adalah nilai X_test nya,hasil prediksinya kita akan masukkan ke dalam variabel y_pred.

kita coba eksekusi kodenya

```
import numpy as np
from sklearn.preprocessing import LabelBinarizer
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd
import matplotlib.pyplot as plt
from scipy.spatial.distance import euclidean

sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
```

```
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }
sensus_df = pd.DataFrame(sensus)

X_train = np.array(sensus_df[['tinggi', 'berat']])
y_train = np.array(sensus_df['jk'])

lb = LabelBinarizer()
y_train = lb.fit_transform(y_train)
y_train = y_train.flatten()
k = 3
model = KNeighborsClassifier(n_neighbors=k)
model.fit(X_train, y_train)
tinggi_badan = 155
berat_badan = 70
X_new = np.array([tinggi_badan, berat_badan]).reshape(1, -1)
y_new = model.predict(X_new)

misterius = np.array([tinggi_badan, berat_badan])
X_train

data_jarak = [euclidean(misterius, d) for d in X_train]
sensus_df['jarak'] = data_jarak
X_test = np.array([[168, 65], [180, 96], [160, 52], [169, 67]])
y_test = lb.transform(np.array(['pria', 'pria', 'wanita', 'wanita'])).flatten()

y_pred = model.predict(X_test)
print(y_pred)
```

dan ini hasilnya

```
[1 0 1 1]
```

Hasilnya bisa nampak di sini, nilai yang diharapkan adalah yang ada di dalam y_test, dimana **0 0 1 1,**sedangkan nilai prediksi dari model kita disini adalah **1 0 1 1,** yang kita tampung ke dalam variabel y_pred.

Untuk kasus evaluasi pada intinya adalah kita akan membandingkan nilai y_test ini dengan nilai y_pred nya.

1. Accuracy

Matrik evaluasi pertama yang akan kita pelajari adalah **accuracy** atau akurasi, akurasi adalah proporsi dari instance pada training tes yang diklasifikasikan secara benar atau yang berhasil diprediksi dengan tepat.

$$accuracy = \frac{tp + tn}{tp + tn + fp + fn}$$

Bisa kita lihat di sini ya formula untuk rekreasi adalah, tp atau true positif dijumlahkan dengan tn atau true negatif negatif, lalu dibagi dengan tp dijumlahkan dengan tn dijumlahkan dengan fn atau false positif dan dijumlahkan dengan fn atau false negatif.

Pada formula ini terdapat beberapa istilah dasar yang umum ditemui dalam classification tasks, istilah-istilah tersebut terdiri dari:

1. **True Positif** : merepresentasikan hasil prediksi atau klasifikasi yang benar,true positif berarti sesuatu yang bernilai positif telah dengan tepat diprediksi sebagai positif oleh model.

   Contoh kasus : true positif berarti model sudah dengan tepat memprediksi data pria sebagai pria dan data wanita sebagai wanita.

2. **True Negatif** : berarti sesuatu yang bernilai negatif telah dengan tepat diprediksi sebagai negatif oleh model.

   Contoh kasus : true negatif berarti model sudah dengan tepat memprediksi data wanita sebagai bukan pria dan data pria sebagai bukan wanita.

3. **False Positif** : Sesuatu yang bernilai negatif, telah keliru diprediksi sebagai positif oleh model.

   Contoh kasus : false positif berarti model sudah dengan keliru memprediksi data wanita sebagai pria dan data pria sebagai wanita.

4. **False Negatif** : Sesuatu yang bernilai positif,telah keliru diprediksi sebagai negatif oleh model.

   Contoh kasus : false negatif berarti model sudah dengan keliru memprediksi data pria sebagai bukan pria dan data wanita sebagai bukan wanita.

Bagi kalian yang butuh untuk mempelajari topik ini lebih lanjut,kita juga akan menyatakan referensi ke halaman Wikipedia.

Refrensi : https://en.wikipedia.org/wiki/Precision_and_recall

Selanjutnya kita akan demokan tahapan menggunakan matriks accuracy untuk melakukan evaluasi model machine learning pada scikit-learn.

```
from sklearn.metrics import accuracy_score
acc = accuracy_score(y_test,  y_pred)
print(f'Accuracy: {acc}')
```

Di sini untuk menghitung accuracy skor caranya cukup mudah,kita tinggal panggil saja **from sklearn.metrics import** accuracy_score,cara pemanfaatannya juga cukup sederhana Kita tinggal panggil saja **accuracy_score**, lalu kita akan menyertakan dua buah parameter,parameter pertama adalah **y_test** dan parameter kedua adalah **ypred**,**y_test** ini merupakan target dari testing set,sedangkan **y_pred** ini merupakan hasil prediksi untuk target dari testing set nya,berikutnya nilai akurasi ini kita tampung ke dalam variabel **acc** untuk berikutnya kita coba cetak layar.

Kita coba eksekusi kode nya

```
import numpy as np
from sklearn.preprocessing import LabelBinarizer
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd
import matplotlib.pyplot as plt
from scipy.spatial.distance import euclidean
from sklearn.metrics import accuracy_score
sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }
sensus_df = pd.DataFrame(sensus)

X_train = np.array(sensus_df[['tinggi', 'berat']])
y_train = np.array(sensus_df['jk'])

lb = LabelBinarizer()
y_train = lb.fit_transform(y_train)
y_train = y_train.flatten()
k = 3
model = KNeighborsClassifier(n_neighbors=k)
model.fit(X_train, y_train)
```

```
tinggi_badan = 155
berat_badan = 70
X_new = np.array([tinggi_badan, berat_badan]).reshape(1, -1)
y_new = model.predict(X_new)

misterius = np.array([tinggi_badan, berat_badan])
X_train

data_jarak = [euclidean(misterius, d) for d in X_train]
sensus_df['jarak'] = data_jarak
X_test = np.array([[168, 65], [180, 96], [160, 52], [169, 67]])
y_test = lb.transform(np.array(['pria', 'pria', 'wanita', 'wanita'])).flatten()
y_pred = model.predict(X_test)
acc = accuracy_score(y_test,  y_pred)
print(f'Accuracy: {acc}')
```

dan ini hasilnya

```
Accuracy: 0.75
```

Akurasinya adalah 0,75 atau dengan kata lain nilai akurasi adalah 75%.

2. Precission

metrics evaluasi ke-2 yang akan kita pelajari adalah precision, precision merupakan proporsi dari testing set yang diprediksi sebagai positif oleh model yang memang benar-benar positif, dan ini merupakan formulanya.

$$precission = \frac{tp}{tp + fp}$$

Bagi kalian memang butuh mempelajari precision dengan lebih lanjut, bisa menuju ke halaman Wikipedia terkait precision ini .

Refrensi : https://en.wikipedia.org/wiki/Precision_and_recall

Selanjutnya akan kita demo kan tahapan menggunakan metrics precission ini,untuk melakukan evaluasi model machine learning pada scikit-learn.

```
from sklearn.metrics import precision_score
prec = precision_score(y_test, y_pred)
print(f'Precission: {prec}')
```

kita cukup import dulu precision skornya kita impor dengan cara **from sklearn.metrics import** precision_score,cara menggunakannya juga sama seperti cara kita menggunakan evaluation matriks yang lain, kita cukup panggil saja precision_score, lalu kita sertakan nilai **y_test** dan nilai **y_pred** sebagai parameternya, lalu hasilnya akan kita tampung ke dalam variabel **prec**, atau precision untuk kemudian kita cetak ke layar.

Kita coba eksekusi kodenya

```
import numpy as np
from sklearn.preprocessing import LabelBinarizer
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd
import matplotlib.pyplot as plt
from scipy.spatial.distance import euclidean
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }
sensus_df = pd.DataFrame(sensus)

X_train = np.array(sensus_df[['tinggi', 'berat']])
y_train = np.array(sensus_df['jk'])

lb = LabelBinarizer()
y_train = lb.fit_transform(y_train)
y_train = y_train.flatten()
k = 3
model = KNeighborsClassifier(n_neighbors=k)
model.fit(X_train, y_train)
tinggi_badan = 155
berat_badan = 70
X_new = np.array([tinggi_badan, berat_badan]).reshape(1, -1)
```

```
y_new = model.predict(X_new)

misterius = np.array([tinggi_badan, berat_badan])
X_train

data_jarak = [euclidean(misterius, d) for d in X_train]
sensus_df['jarak'] = data_jarak
X_test = np.array([[168, 65], [180, 96], [160, 52], [169, 67]])
y_test = lb.transform(np.array(['pria', 'pria', 'wanita', 'wanita'])).flatten()
y_pred = model.predict(X_test)
acc = accuracy_score(y_test,  y_pred)
prec = precision_score(y_test, y_pred)
print(f'Precission: {prec}')
```

dan ini hasilnya

```
Precission: 0.6666666666666666
```

bisa nampak disini, nilai precision adalah **0.6666666666666666,**atau disini kita bisa bilang bahwa nilai precision nya adalah 66%.

3. Recall

metrics evaluasi ke-3 yang akan kita pelajari adalah recall,recall adalah proporsi dari data pada testing set kita yang memang benar-benar positif,yang diprediksi sebagai positif oleh model kita.disini nilai ekologis kita peroleh dengan formula sebagai berikut.

$$recall = \frac{tp}{tp + fn}$$

Bagi kalian ini mempelajari recall dengan lebih detail, kita juga menyertakan link untuk menuju halaman Wikipedia terkait precision dan recall ini.

Refrensi : https://en.wikipedia.org/wiki/Precision_and_recall

Selanjutnya disini kita akan demokan tahapan menggunakan metrics recall untuk melakukan evaluasi model machine- earning pada scikit-learn.

```
from sklearn.metrics import recall_score
rec = recall_score(y_test, y_pred)
print(f'Recall: {rec}')
```

Caranya juga cukup mudah, kita tinggal impor saja **from sklearn.metrics import** recall-score, lalu cara pemanfaatannya juga sama, Kita tinggal panggil saja recall_score,lalu kita akan menyatakan **y_test** dan **y-pred** sebagai parameter nya, nilai nya akan kita tampung ke dalam variabel **rec,** untuk selanjutnya kita saya tak layar.

Kita coba eksekusi kode nya

```
import numpy as np
from sklearn.preprocessing import LabelBinarizer
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd
import matplotlib.pyplot as plt
from scipy.spatial.distance import euclidean
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }
sensus_df = pd.DataFrame(sensus)

X_train = np.array(sensus_df[['tinggi', 'berat']])
y_train = np.array(sensus_df['jk'])

lb = LabelBinarizer()
y_train = lb.fit_transform(y_train)
y_train = y_train.flatten()
k = 3
model = KNeighborsClassifier(n_neighbors=k)
model.fit(X_train, y_train)
tinggi_badan = 155
berat_badan = 70
X_new = np.array([tinggi_badan, berat_badan]).reshape(1, -1)
y_new = model.predict(X_new)
```

```
misterius = np.array([tinggi_badan, berat_badan])
X_train

data_jarak = [euclidean(misterius, d) for d in X_train]
sensus_df['jarak'] = data_jarak
X_test = np.array([[168, 65], [180, 96], [160, 52], [169, 67]])
y_test = lb.transform(np.array(['pria', 'pria', 'wanita', 'wanita'])).flatten()
y_pred = model.predict(X_test)
acc = accuracy_score(y_test,  y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
print(f'Recall: {rec}')
```

dan ini hasilnya

```
Recall: 1.0
```

Nilai recall nya adalah **1.0,** ini merupakan nilai tertinggi atau dengan kata lain nilai recall adalah 100%.

4. F1 Score

Metrics evaluasi ke-4 yang akan kita pelajari adalah f1 sore, F1 Score ini merupakan harmonic mind, atau nilai rata-rata harmonik dari precision dan recall, ini merupakan formula untuk melakukan kalkulasi F1 Score.

$$F1 = 2X \frac{precission\ X\ recall}{precission + recall}$$

$$F1 = 2X \frac{precission \times recall}{precission + recall}$$

Bagi kalian yang ingin mempelajari lebih lanjut mengenai F1 Score ini,kalian juga bisa menuju ke halaman Wikipedia terkait precision dan recall.

Refrensi : https://en.wikipedia.org/wiki/Precision_and_recall

Selanjut nya kita akan demo kan tahapan untuk menggunakan metrics F1 Score pada scikit-learn.

```
from sklearn.metrics import f1_score
f1 = f1_score(y_test, y_pred)
print(f'F1-Score: {f1}')
```

Disini caranya cukup mudah, Kita tinggal panggil saja **from sklearn.metrics import** f1_score, untuk menggunakannya juga Sama ya Kita tinggal panggil saja **f1_score** ,lalu kita akan menyertakan **y_test** dan **y_pred** sebagai parameter nya,dan nilainya kita akan tampung ke dalam variabel **f1** untuk selanjutnya kita cetak layar.

Kita coba eksekusi kode nya

```
import numpy as np
from sklearn.preprocessing import LabelBinarizer
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd
import matplotlib.pyplot as plt
from scipy.spatial.distance import euclidean
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }
sensus_df = pd.DataFrame(sensus)

X_train = np.array(sensus_df[['tinggi', 'berat']])
y_train = np.array(sensus_df['jk'])

lb = LabelBinarizer()
y_train = lb.fit_transform(y_train)
y_train = y_train.flatten()
k = 3
model = KNeighborsClassifier(n_neighbors=k)
model.fit(X_train, y_train)
tinggi_badan = 155
berat_badan = 70
```

```
X_new = np.array([tinggi_badan, berat_badan]).reshape(1, -1)
y_new = model.predict(X_new)

misterius = np.array([tinggi_badan, berat_badan])
X_train

data_jarak = [euclidean(misterius, d) for d in X_train]
sensus_df['jarak'] = data_jarak
X_test = np.array([[168, 65], [180, 96], [160, 52], [169, 67]])
y_test = lb.transform(np.array(['pria', 'pria', 'wanita', 'wanita'])).flatten()
y_pred = model.predict(X_test)
acc = accuracy_score(y_test,  y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print(f'F1-Score: {f1}')
```

dan ini hasilnya

```
F1-Score: 0.8
```

hasil F1-Score nya adalah **0.8**.

Classification Report

Ke empat metrics evaluasi yang kita sampaikan sebelumnya juga bisa ditampilkan secara keseluruhan sebagai classification repott,sesuai dengan namanya keempat matriks tersebut memang digunakan untuk mengukur performa model pada classification tasks.

berikut akan kita demokan cara.

```
from sklearn.metrics import classification_report
cls_report = classification_report(y_test, y_pred)
print(f'Classification Report: {cls_report}')
```

Caranya sederhana Kita tinggal panggil saja **from sklearn.metrics import** classification_report, lalu pemanfaatannya juga sama dengan matriks yang lain, kita tinggal panggil saja **classification_report**

lalu kita akan menyatakan **y_test** dan **y-pred** nya sebagai parameter, nilai classification repott ini berikutnya akan kita tampung ke dalam variabel **cls_report,** untuk selanjutnya kita tampilkan ke layar.

Kita eksekusi kode nya

```
import numpy as np
from sklearn.preprocessing import LabelBinarizer
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd
import matplotlib.pyplot as plt
from scipy.spatial.distance import euclidean
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }
sensus_df = pd.DataFrame(sensus)

X_train = np.array(sensus_df[['tinggi', 'berat']])
y_train = np.array(sensus_df['jk'])

lb = LabelBinarizer()
y_train = lb.fit_transform(y_train)
y_train = y_train.flatten()
k = 3
model = KNeighborsClassifier(n_neighbors=k)
model.fit(X_train, y_train)
tinggi_badan = 155
berat_badan = 70
X_new = np.array([tinggi_badan, berat_badan]).reshape(1, -1)
y_new = model.predict(X_new)

misterius = np.array([tinggi_badan, berat_badan])
X_train

data_jarak = [euclidean(misterius, d) for d in X_train]
sensus_df['jarak'] = data_jarak
X_test = np.array([[168, 65], [180, 96], [160, 52], [169, 67]])
y_test = lb.transform(np.array(['pria', 'pria', 'wanita', 'wanita'])).flatten()
y_pred = model.predict(X_test)
```

```
acc = accuracy_score(y_test,  y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
cls_report = classification_report(y_test, y_pred)
print(f'Classification Report: {cls_report}')
```

dan ini hasilnya

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.50      0.67         2
           1       0.67      1.00      0.80         2

    accuracy                           0.75         4
   macro avg       0.83      0.75      0.73         4
weighted avg       0.83      0.75      0.73         4
```

Keempat metrics evaluasi yang kita pelajari ini ditujukan untuk binary classifier, dimana berapapun banyaknya jumlah class atau kelompok nilai yang terdapat pada target, tetap akan diasumsikan sebagai dua kelompok nilai, saja yaitu positif atau negatif.

Disini nilai biner 1 akan dipandang sebagai nilai positif secara default, sedangkan nilai biner 0 akan dipandang sebagai nilai negatif,ini juga alasan mengapa di awal sesi pembelajaran ini kita melakukan konversi nilai target dari string menjadi numerik biner.

Kalau kita perhatikan di sini perhitungan precision,recall dan f1-score yang kita lakukan sebelumnya bisa terbilang timpang, karena hanya berdasarkan kelompok nilai 1 saja.

Untuk kasus kita disini nilai 1 ini berasosiasi dengan jenis kelamin wanita, kalau kita perhatikan di sini,nilai precision,recall dan f1-score yang kita peroleh sebelumnya tadi itu hanya dari nilai 1, kita tidak memperhatikan class yang 0, kita hanya memperhatikan kelas yang 1 saja, atau hanya memperhatikan jenis kelamin wanita saja, makanya nilai precission nya tadi adalah **0.6666666666666666,** recall nya **1** dan f1-score skornya adalah **0,8.**

Jadi sebenarnya perhitungan yang kita lakukan tadi itu cukup timpang, kita hanya memperhatikan yang class atau yang kelompok nilai 1 saja, classification report yang ditampilkan disini mencakup nilai precision,recall dan f1-score untuk keseluruhan class, baik yang 0 maupun 1, oleh karenanya kita lebih menyarankan untuk menggunakan classification repot daripada melakukan pengukuran precision,recall dan f1-score secara terpisah,

Selain itu, classification repot juga menampilkan nilai rata-rata untuk precision,recall dan f1-score nya, kalau kita lihat di sini, ada **macro avg** ada **weighted avg** untuk precision,recall maupun f1-score nya.

0.83 ini merupakan nilai rata-rata precission nya, karena nilai precission disini ada dua,**1.00** dan **0.67,** kalau kita rata-rata kan nilainya adalah **0.83,**

0.75 adalah nilai rata-rata untuk recall nya, yang diperoleh dari **0.50** dengan nilai **1.00,** kalau di rata-rata nilainya adalah **0.75.**

0.73 merupakan nilai f1-score nya, yang diperoleh dari nilai rata-rata antara nilai **0.67** dan nilai **0.80.**

Sedangkan nilai akurasi ini sama sekali tidak terpengaruh oleh clas nya, Apakah dilihat dari kelas 0 maupun dari class 1, nilai akurasi akan tetap sama.

Disini sebagian dari antara kalian mungkin akan bertanya-tanya Mengapa nilai average(avg) nya ada dua, untuk saat ini kita akan coba abaikan dulu karena disini kita tidak menerapkan weighted sama sekali sehingga nilai average(avg) untuk macro maupun weighted nya akan sama.

5. Matthews Correlation Coefficient (MCC)

metrics evaluasi ke-5 yang akan kita pelajari adalah Matius correlation coefficient atau biasa disingkat sebagai MCC.

MCC merupakan alternatif untuk F1-score, untuk keperluan pengukuran performa dari binary classifier. nilai tertingginya adalah 1, kalau prediksinya dilakukan secara random atau secara acak dan ngawur,maka nilai skornya adalah 0, atau kalau model klasik file-nya benar-benar ngaco maka dia akan menghasilkan nilai -1, oleh karenanya rentang nilai dari MCC ini mulai dari -1 untuk kondisi terburuk sampai dengan 1 untuk kondisi terbaik nya.

Bagi kalian yang penasaran dengan proses kalkulasi MCC berikut adalah formula untuk melakukan proses kalkulasi MCC.

$$MCC = \frac{tp \times tn + fp \times fn}{\sqrt{(tp+fp) \times (tp+fn) \times (tn+fp) \times (tn+fn)}}$$

Bagi kalian yang ingin mempelajari MCC lebih lanjut kalian dapat memanfaatkan halaman Wikipedia berikut ini, ini merupakan halaman Wikipedia yang memberikan penjelasan lebih lanjut mengenai Matthews Correlation Coefficient (MCC).

Refrensi : https://en.wikipedia.org/wiki/Matthews_correlation_coefficient

Selanjutnya akan kita demokan tahapan menggunakan MCC untuk melakukan evaluasi model machine learning pada scikit-learn.

```
from sklearn.metrics import matthews_corrcoef
mcc = matthews_corrcoef(y_test, y_pred)
print(f'MCC: {mcc}')
```

Disini caranya cukup sederhana, kita cukup panggil saja **from sklearn.metrics import** matthews_corrcoef, untuk menggunakan nya kita tinggal pakai saja matthews_corrcoef, lalu kita tinggal sertakan **y_test** dan **y-pred** sebagai parameternya, kemudian kita tampung ke dalam variabel **mcc** untuk kemudian kita tampilkan ke layar.

Kita coba eksekusi kode nya

```
import numpy as np
from sklearn.preprocessing import LabelBinarizer
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd
import matplotlib.pyplot as plt
from scipy.spatial.distance import euclidean
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import matthews_corrcoef
sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }
sensus_df = pd.DataFrame(sensus)

X_train = np.array(sensus_df[['tinggi', 'berat']])
y_train = np.array(sensus_df['jk'])

lb = LabelBinarizer()
y_train = lb.fit_transform(y_train)
```

```
y_train = y_train.flatten()
k = 3
model = KNeighborsClassifier(n_neighbors=k)
model.fit(X_train, y_train)
tinggi_badan = 155
berat_badan = 70
X_new = np.array([tinggi_badan, berat_badan]).reshape(1, -1)
y_new = model.predict(X_new)

misterius = np.array([tinggi_badan, berat_badan])
X_train

data_jarak = [euclidean(misterius, d) for d in X_train]
sensus_df['jarak'] = data_jarak
X_test = np.array([[168, 65], [180, 96], [160, 52], [169, 67]])
y_test = lb.transform(np.array(['pria', 'pria', 'wanita', 'wanita'])).flatten()
y_pred = model.predict(X_test)
acc = accuracy_score(y_test,  y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
cls_report = classification_report(y_test, y_pred)
mcc = matthews_corrcoef(y_test, y_pred)
print(f'MCC: {mcc}')
```

dan ini hasil nya

```
MCC: 0.5773502691896258
```

ini merupakan nilai MCC nya adalah**0.5773502691896258.**

◆ Bab 7 : Regression dengan KNN (K Nearest Neighbours)

Dalam bab ini kita akan meninjau kembali model machine learning yang sama yang telah kita pelajari di sesi pembelajaran sebelumnya yaitu KNN, hanya saja kali ini kita akan terapkan pada regression tasks, kita coba review sekilas ya mengenai apa itu KNN.

KNN adalah model machine learning yang dapat digunakan untuk melakukan prediksi berdasarkan kedekatan karakteristik dengan sejumlah tetangga terdekat, prediksi dilakukan dapat diterapkan baik pada classification maupun regression yang tasks.

◆ Sample Dataset

Pertama-tama kita akan siapkan terlebih dahulu dataset yang akan kita gunakan dalam sesi pembelajaran ini

kali ini kita juga membentuk dataset yang sama dengan dataset yang kita gunakan dalam sesi sebelumnya, dataset ini berisi berat badan, tinggi badan, dan jenis kelamin, dari sejumlah partisipan yang kita tampung dalam format pandasframe.

Berikut akan kita demokan caranya

```
import pandas as pd

sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }

sensus_df = pd.DataFrame(sensus)
```

Disini pertama-tama kita akan impor dulu modul pandas, caranya di sini kita panggil saja **import pandas as pd,** lalu berikutnya kita akan menyiapkan suatu dictionary yang terdiri dari 3 buah keys yaitu tinggi,jk dan berat, keys tinggi ini akan berasosiasi sekumpulan nilai tinggi badan, jk ini akan berasosiasi dengan sekumpulan nilai jenis kelamin, dalam hal ini jenis kelaminnya terdiri dari 2 nilai string yaitu, pria dan wanita, lalu berikutnya keys berat akan berasosiasi dengan sekumpulan nilai berat badan.

Dictionary ini lalu akan kita tampung ke dalam variabel yang kita bernama **sensus**, untuk selanjutnya kita gunakan sebagai dasar pembentukan Pandas dataframe, oleh karenanya di sini kita panggil pd.DataFrame(sensus), objek dataframe yang terbentuk ini akan kita tampung ke dalam variabel **sensus-df.**

Kita coba eksekusi kode nya

dan ini hasil nya

	tinggi	jk	berat
0	158	pria	64
1	170	pria	86
2	183	pria	84
3	191	pria	80
4	155	wanita	49
5	163	wanita	59
6	180	wanita	67
7	158	wanita	54
8	178	wanita	67

Bisa nampak disini terdiri dari 3 buah kolom yaitu tinggi, jk dan berat, dan kalau kita lihat di sini jumlah barisnya ada 9 baris, yang dimulai dari index 0 sampai dengan indeks 8, kalau diperhatikan dataset nya juga sama persis dengan dataset yang kita gunakan dalam sesi pembelajaran sebelumnya.

Dalam sesi pembelajaran kita kali ini, kita akan membentuk suatu model machine learning sederhana yang dapat kita gunakan untuk memprediksi berat badan seseorang berdasarkan tinggi badan dan jenis kelaminnya, dengan kata lain disini tinggi badan dan jenis kelamin akan berperan sebagai features sedangkan berat badan akan berperan sebagai target.

◆ Regression dengan KNN

Setelah kita memahami konteks Dataset juga permasalahan nya, kita akan coba menerapkan KNN untuk melakukan estimasi nilai berat badan.

Sedikit berbeda dengan sesi pembelajaran sebelumnya, dimana kita menggunakan KNN untuk melakukan classification, dalam sesi ini kita akan menggunakan KNN untuk melakukan Regression atau estimasi nilai, dimana dalam kasus ini nilai akan kita prediksi adalah nilai berat badan.

Feature & Target

Sebelum kita melangkah ke proses training model, kita akan kelompokan terlebih dahulu sekumpulan nilai features dan nilai target dari dataset yang kita miliki,

untuk kasus kita kali ini data tinggi badan dan jenis kelamin akan kita konversikan ke dalam numpy array dan kita tampung ke variabel **X_train** sebagai sekumpulan features untuk training set, lalu untuk data berat badan juga akan kita konversikan menjadi munpy array untuk selanjutnya kita tabung ke dalam variabel **y_train** sebagai sekumpulan nilai target.

Berikut akan kita demokan prosesnya

```
import numpy as np

X_train = np.array(sensus_df[['tinggi', 'jk']])
y_train = np.array(sensus_df['berat'])

print(f'X_train:\n{X_train}\n')
print(f'y_train:\n{y_train}')
```

Disini pertama-tama kita akan impor dulu **import numpy as np,** berikutnya untuk nilai features, dalam hal ini adalah tinggi dan jenis kita akan kita bundle menjadi suatu nunmpy array, dan berikutnya kita tampung ke dalam variabel **X_train,** sedangkan untuk sekumpulan nilai targetnya, dalam hal ini adalah berat badan, akan kita konversikan juga menjadi suatu numpy array yang kemudian kita tampung ke dalam variabel **y_train,** lalu berikutnya disini baik **X_train** maupun **y_train** akan kita cetak layar.

Kita coba eksekusi kode nya

```
import pandas as pd
```

```
import numpy as np
sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }

sensus_df = pd.DataFrame(sensus)
X_train = np.array(sensus_df[['tinggi', 'jk']])
y_train = np.array(sensus_df['berat'])

print(f'X_train:\n{X_train}\n')
print(f'y_train:\n{y_train}')
```

dan ini hasilnya

```
X_train:
[[158 'pria']
 [170 'pria']
 [183 'pria']
 [191 'pria']
 [155 'wanita']
 [163 'wanita']
 [180 'wanita']
 [158 'wanita']
 [178 'wanita']]

y_train:
[64 86 84 80 49 59 67 54 67]
```

Disini kita memanfaatkan KNN untuk melakukan prediksi berat badan berdasarkan data tinggi badan dan jenis kelaminnya, karena disini yang diprediksi berupa nilai continues dan bukan kategori seperti yang kita temui dalam sesi pembelajaran sebelumnya, maka kasus ini termasuk dalam regression tasks.

Pada sesi pembelajaran sebelumnya, kita sudah memahami bahwa KNN akan melakukan prediksi berdasarkan sejumlah tetangga terdekat, dimana tetangga terdekat ini ditentukan berdasarkan kalkulasi jarak dengan memanfaatkan euclidean distance, disini tentunya kita perlu memastikan agar nilai feature nya bertipe data numerik supaya dapat dihitung jarak antar data poinnya.

Preprocess Dataset: Konversi Label menjadi Numerik Biner

kita akan demokan tahapan data preprocessing untuk mengkonversi nilai pria dan wanita menjadi numerik biner 0 dan 1.

```
X_train_transposed = np.transpose(X_train)

print(f'X_train:\n{X_train}\n')
print(f'X_train_transposed:\n{X_train_Transposed}')
```

Pertama-tama kita akan melakukan proses tranpose terlebih dahulu terhadap X_train, agar lebih jelas dengan apa yang dimaksud dengan transpose ini kita coba eksekusi kode nya.

```
import pandas as pd
import numpy as np
sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }

sensus_df = pd.DataFrame(sensus)
X_train = np.array(sensus_df[['tinggi', 'jk']])
y_train = np.array(sensus_df['berat'])

X_train_transposed = np.transpose(X_train)

print(f'X_train:\n{X_train}\n')
print(f'X_train_transposed:\n{X_train_transposed}')
```

Dan ini hasil nya

ini merupakan kondisi ekstrim sebelum proses transpose

```
X_train:
[[158 'pria']
```

```
 [170 'pria']
 [183 'pria']
 [191 'pria']
 [155 'wanita']
 [163 'wanita']
 [180 'wanita']
 [158 'wanita']
 [178 'wanita']]
```

dan ini merupakan hasil transposenya

```
X_train_transposed:
[[158 170 183 191 155 163 180 158 178]
 ['pria' 'pria' 'pria' 'pria' 'wanita' 'wanita' 'wanita' 'wanita'
  'wanita']]
```

Proses transpose ini pada dasarnya akan mengubah posisi baris menjadi kolom dan posisi kolom menjadi baris, kalau kita perhatikan di sini, posisi barisnya ini mengindikasikan instance sedangkan posisi kolom ini mengindikasikan feature, kolom pertama merupakan feature pertama yang merupakan tinggi badan, sedangkan kolom keduea merupakan feature kedua yaitu jenis kelamin.

Disini kita mau ubah posisi feature tidak berada dalam kolom tetapi kita mau posisi feature berada dalam baris untuk itu kita melakukan proses transpose, bisa kita lihat di sini **X_train** nya akan dikenakan proses transpose, dan hasil transposenya akan kita tampung di dalam variabel **X_train_transposed,** lalu berikutnya baik nilai **X_train** maupun **X_train_transposed** nya kita cecak layar.

Kalau kita lihat pada **X_train_transposed ,** di sini terdiri dari 2 baris saja, dan kalau kita lihat nilai dari baris pertamanya ini akan berkorelasi dengan feature tinggi badan, sedangkan nilai untuk baris keduanya ini merepresentasikan sekumpulan nilai untuk jenis kelamin.

Pada tahapan berikut nya kita akan menerapkan yang namanya LabelBinarizer, disini LabelBinarizer akan kita gunakan karena kita mau mengkonversikan nilai pria dan wanita ini menjadi nilai biner 0 dan 1 .

```
from sklearn.preprocessing import LabelBinarizer

lb = LabeBinarizer()
jk_binarised = lb.fit_transform(X_train_transposed[1])

print(f'jk: {X_train_transposed[1]}\n')
```

```
print(f'jk_binarised:\n{jk_binarised}')
```

Pertama-tama kita akan import dulu LabelBinarizer nya, **from sklearn.preprocessing import** LabelBinarizer, lalu berikutnya kita akan membentuk objek dari class LabelBinarizer ini,objek nya akan kita tampung ke dalam variabel **lb**, disini proses transformasi LabelBinarizer nya akan kita terapkan pada jenis kelamin, oleh karenanya di sini kita panggil lb.fit_transform(X_train_transposed[**1**]), yang kita terapkan pada X_train transpose indeks ke-1, karena indeks ke 0 nya adalah tinggi badan, sedangkan indeks 1 nya adalah jenis kelamin, lalu hasil transform nya kita tampung ke dalam variabel **jk_binarised** ,untuk selanjutnya kita ke layar.

Kita cobaa eksekusi kode nya

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelBinarizer

sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }

sensus_df = pd.DataFrame(sensus)
X_train = np.array(sensus_df[['tinggi', 'jk']])
y_train = np.array(sensus_df['berat'])

X_train_transposed = np.transpose(X_train)

lb = LabelBinarizer()
jk_binarised = lb.fit_transform(X_train_transposed[1])

print(f'jk: {X_train_transposed[1]}\n')
print(f'jk_binarised:\n{jk_binarised}')
```

dan ini hasil nya

ini merupakan jenis kelamin sebelum dikenakan proses transform nya, di sini nilainya hanya ada dua yaitu pria dan wanita.

```
jk: ['pria' 'pria' 'pria' 'pria' 'wanita' 'wanita' 'wanita' 'wanita' 'wanita']
```

dan ini merupakan hasil transformasinya,bisa kita lihat di sini, data prianya berubah menjadi 0 sedangkan data wanitanya akan berubah menjadi 1.

```
jk_binarised:
[[0]
 [0]
 [0]
 [0]
 [1]
 [1]
 [1]
 [1]
 [1]]
```

Tetapi disini kita juga punya permasalahan lain,kalau kita lihat di kondisi awal ini kan datanya terdiri dari 1 dimensi saja, sedangkan setelah proses transformasi datanya berubah menjadi 2 dimensi, oleh karenanya di sini kita perlu melakukan proses yang lain yaitu proses flatten, dan kita tahu dari sisi pembelajaran sebelumnya metode ini dapat digunakan untuk mengkonversikan multidimensional array menjadi single dimensional array, atau dengan kata lain, method flatten ini digunakan untuk mengkonversikan array multidimensi menjadi array dimensi tunggal.

Kita coba tambahkan dan eksekusi kode nya

```
jk_binarised = jk_binarised.flatten()
jk_binarised
```

dan ini hasil nya

```
[0 0 0 0 1 1 1 1 1]
```

Bisa nampak di sini sekarang bentuk numpy array nya sudah dalam 1 dimensi.

Tahapan berikutnya adalah nilai yang sudah kita konversikan ini akan kita tampung lagi ke dalam **X_train_transposed,** karena nilai ini masih berada dalam **jk_binarised.**

```
X_train_transposed[1] = jk_binarised
X_train = X_train_transposed.transpose()

print(f'X_train_transposed:\n{X_train_transposed}')
```

```
print(f'X_train:\n{X_train}')
```

Caranya berarti disini kita akan Panggil dulu **X_train_transposed** indeks ke 1,karena indeks ke 1 ini yang berkorelasi dengan jenis kelamin, dan nilainya akan kita gantikan dengan nilai **jk_binarised,**berikutnya **X_train_transposed** ini akan kita transpose balik agar yang tadinya baris kembali menjadi kolom, yang tadinya kolom akan menjadi baris, dan hasil transpose ini akan kita kembalikan ke dalam variabel **X_train,** dan hasilnya akan kita cetak ke layar.

Kita coba eksekusi kode nya

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelBinarizer

sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }

sensus_df = pd.DataFrame(sensus)
X_train = np.array(sensus_df[['tinggi', 'jk']])
y_train = np.array(sensus_df['berat'])

X_train_transposed = np.transpose(X_train)

lb = LabelBinarizer()
jk_binarised = lb.fit_transform(X_train_transposed[1])

jk_binarised = jk_binarised.flatten()
X_train_transposed[1] = jk_binarised
X_train = X_train_transposed.transpose()

print(f'X_train_transposed:\n{X_train_transposed}')
print(f'X_train:\n{X_train}')
```

dan ini hasilnya

ini merupakan nilai X_train_transposed nya.

```
X_train_transposed:
[[158 170 183 191 155 163 180 158 178]
 [0 0 0 0 1 1 1 1 1]]
```

Sedangkan ini merupakan nilai X_train finalnya.

```
X_train:
[[158 0]
 [170 0]
 [183 0]
 [191 0]
 [155 1]
 [163 1]
 [180 1]
 [158 1]
 [178 1]]
```

Prosesnya memang nampak panjang, tetapi pada intinya yang kita lakukan ini untuk mengubah nilai jenis kelamin yang tadinya pria dan wanita menjadi nilai numerik biner yaitu 0 dan 1, kebetulan untuk kasus kita kali ini, nilai 0 itu merepresentasikan jenis kelamin pria sedangkan nilai 1 nya merepresentasikan jenis kelamin wanita.

Kalau kita lihat sekarang ini sekumpulan nilai features dan targetnya sudah siap digunakan untuk training model.

Training KNN Regression Model

Setelah training set nya kita persiapkan, selanjutnya kita akan melangkah ke proses training model,untuk kasus ini model machine learning akan kita gunakan adalah KNN, berikut akan kita demokan prosesnya.

```
from sklearn.neighbors import KNeighborsRegressor

k = 3
model = KNeighborsRegressor(n_neighbors=k)
model.fit(X_train, y_train)
```

Pertama-tama kita akan import dulu estimator classnya **from sklearn.neighbors import** KneighborsRegressor, disini sedikit berbeda dengan sesi pembelajaran sebelumnya, dimana pada sesi pembelajaran sebelumnya yang kita Import adalah kneighborsClassifier, sedangkan disini yang kita

Import adalah KneighborsRegressor, lalu berikutnya sama disini kita juga akan menentukan nilai **k** nya, nilai **k** ini akan berkorelasi dengan jumlah atau banyaknya neighbours atau banyaknya tetangga yang akan digunakan untuk melakukan proses prediksi, untuk kasus kita kali ini, kita juga akan menggunakan nilai **k = 3,** tahapan berikutnya kita akan membentuk objek modelnya, disini kita akan panggil **KneighborsRegressor,** dan kita akan menyertakan nilai parameter nya diisi dengan nilai **k,** atau dengan kata lain disini nilai Kneighbors nya kita set sebagai 4, setelah objek modelnya terbentuk lalu berikutnya kita tinggal melakukan proses training model, di sini kita tinggal panggil saja **model.fit l**alu kita sertakan **X_train** dan **y_train** sebagai parameternya.

Sekedar informasi, model machine learning yang digunakan untuk regression tasks seringkali juga dikenal dengan istilah regressor, sedangkan model machine learning yang digunakan untuk classification tasks seringkali dikenal dengan istilah classifier.

Predkisi Berat Badan

Setelah KNN Regression nya kita training, tahapan selanjutnya kita akan gunakan train model tersebut untuk melakukan prediksi berat badan berdasarkan data tinggi badan dan jenis kelamin.

Disini kita akan siapkan terlebih dahulu data baru yang akan kita gunakan untuk prediksi

```
X_new = np.array([[155, 1]])
X_new
```

Di sini kita siapkan data baru dimana nilai tinggi badannya adalah 155 cm dan jenis kelamin adalah wanita,disini juga kita sesuaikan jenis kelamin wanitanya sudah kita encode menjadi nilai 1, lalu berikutnya pasangan nilai feature ini kita bundle menjadi suatu array 2 dimensi,untuk selanjutnya kita tampung ke dalam variabel **X_new**.

Setelah data tinggi badan dan jenis kelaminnya siap, selanjutnya kita akan melakukan prediksi berat badan dengan memanfaatkan model KNNRegression yang sudah kita training sebelumnya.

```
y_pred = model.predict(X_new)
y_pred
```

Disini Kita tinggal panggil saja **models.predict**, lalu kita masukkan nilai **X_new** sebagai parameter nya, hasil prediksinya lalu kita masukkan ke dalam variabel **y_pred.**

Kita coba eksekusi kode nya

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelBinarizer
from sklearn.neighbors import KNeighborsRegressor

sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }

sensus_df = pd.DataFrame(sensus)
X_train = np.array(sensus_df[['tinggi', 'jk']])
y_train = np.array(sensus_df['berat'])

X_train_transposed = np.transpose(X_train)

lb = LabelBinarizer()
jk_binarised = lb.fit_transform(X_train_transposed[1])

jk_binarised = jk_binarised.flatten()
X_train_transposed[1] = jk_binarised
X_train = X_train_transposed.transpose()
k = 3
model = KNeighborsRegressor(n_neighbors=k)
model.fit(X_train, y_train)
X_new = np.array([[155, 1]])
y_pred = model.predict(X_new)
print(y_pred)
```

dan ini hasil nya

```
[55.66666667]
```

Disini kita bisa bilang bahwa untuk data dengan tinggi badan 150 cm dan jenis kelamin wanita ini diprediksi oleh model kita memiliki berat badan **55.66666667** kg.

◆ Evaluasi KNN Regression Model

Kita akan mulai mempelajari beberapa matriks yang bisa kita gunakan untuk mengukur performa dari model machine learning untuk kasus regression tasks, tapi sebelumnya kita akan siapkan terlebih dahulu testing setnya.

Disini untuk testing nya kita akan siapkan 4 instance atau 4 datapoint.

```
X_test = np.array([[168, 0], [180, 0], [160, 1], [169, 1]])
y_test = np.array([65, 96, 52, 67])

print(f'X_test:\n{X_test}\n')
print(f'y_test:\n{y_test}')
```

Pertama-tama disini kita akan siapkan dulu sekumpulan nilai features untuk testing set nya, untuk kemudian kita kamu ke dalam variabel **X_test**, selain itu kita juga akan mempersiapkan sekumpulan nilai target waktu kita tampung ke dalam variabel **y_test.**

Kita coba eksekusi kode nya

```
import numpy as np

X_test = np.array([[168, 0], [180, 0], [160, 1], [169, 1]])
y_test = np.array([65, 96, 52, 67])

print(f'X_test:\n{X_test}\n')
print(f'y_test:\n{y_test}')
```

Dan ini dia hasilnya

Ini Merupakan sekumpulan nilai features untuk testing set

```
X_test:
[[168   0]
 [180   0]
 [160   1]
 [169   1]]
```

Sedangkan ini merupakan sekumpulan nilai target untuk testing set.

```
y_test:
[65 96 52 67]
```

Setelah testing setnya siap, selanjutnya kita akan melakukan prediksi terhadap testing set ini dengan memanfaatkan modal KNN Regression yang sudah kita training sebelumnya.

```
y_pred = model.predict(X_test)
y_pred
```

Disini kita akan menerapkan prediksi terhadap sekumpulan feature dari X_test, ini dengan memanfaatkan modal KNN yang sudah kita training sebelumnya.

Oleh karenanya di sini kita Panggil **model.predict(X_test)**, hasil prediksinya akan kita tampung dalam variabel y_pred.

kita coba eksekusi

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelBinarizer
from sklearn.neighbors import KNeighborsRegressor

sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }

sensus_df = pd.DataFrame(sensus)
X_train = np.array(sensus_df[['tinggi', 'jk']])
y_train = np.array(sensus_df['berat'])

X_train_transposed = np.transpose(X_train)

lb = LabelBinarizer()
jk_binarised = lb.fit_transform(X_train_transposed[1])

jk_binarised = jk_binarised.flatten()
X_train_transposed[1] = jk_binarised
X_train = X_train_transposed.transpose()
k = 3
model = KNeighborsRegressor(n_neighbors=k)
model.fit(X_train, y_train)
```

```
X_test = np.array([[168, 0], [180, 0], [160, 1], [169, 1]])
y_test = np.array([65, 96, 52, 67])

y_pred = model.predict(X_test)
print(y_pred)
```

dan ini dia hasilnya

```
[69.66666667 72.66666667 59.         70.66666667]
```

Disini kita bisa bilang bahwa untuk data dengan tinggi badan 168 dan jenis kelamin pria ini diprediksi memiliki berat badan **69.66666667,** sedangkan data yang diharapkan adalah 65, lalu berikutnya untuk data point yang paling akhir, di sini Kita bisa bilang bahwa untuk data dengan tinggi badan 169 dan jenis kelamin wanita ini diprediksi memiliki berat badan **70.66666667,** sedangkan nilai yang diharapkan adalah 67.

Coefficient of Determination atau R^2

metrics evaluasi pertama yang akan kita gunakan adalah coefficient of determmination,atau biasa dikenal sebagai R-square.

Selanjutnya kita akan demokan tahapan menggunakan matriks R-Square untuk melakukan evaluasi model regression pada scikit-learn.

```
from sklean.metrics import r2_score

r_squared = r2_score(y_test, y_pred)
print(f'R_Squared: {r_squared}')
```

Disini caranya sederhana, pertama-tama kita akan import dulu **from sklean.metrics import** r2_score ,berikutnya untuk menggunakan metrics ini Kita tinggal panggil saja **r2_score,** lalu kira sertakan **y_test**, dan **y_pred** sebagai parameternya, nilai evaluasi ini akan kita tampung ke dalam variabel **r_squared** untuk kemudian kita cetak layar.

Kita coba eksekusi kode nya

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelBinarizer
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import r2_score
sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }

sensus_df = pd.DataFrame(sensus)
X_train = np.array(sensus_df[['tinggi', 'jk']])
y_train = np.array(sensus_df['berat'])

X_train_transposed = np.transpose(X_train)

lb = LabelBinarizer()
jk_binarised = lb.fit_transform(X_train_transposed[1])

jk_binarised = jk_binarised.flatten()
X_train_transposed[1] = jk_binarised
X_train = X_train_transposed.transpose()
k = 3
model = KNeighborsRegressor(n_neighbors=k)
model.fit(X_train, y_train)


X_test = np.array([[168, 0], [180, 0], [160, 1], [169, 1]])
y_test = np.array([65, 96, 52, 67])

y_pred = model.predict(X_test)
r_squared = r2_score(y_test, y_pred)
print(f'R_Squared: {r_squared}')
```

dan ini hasil nya

```
R_Squared: 0.39200515796260493
```

Ini merupakan nilai r-square untuk model kita, seperti pernah dijelaskan dalam sisi pembelajaran sebelumnya, R-Square ini semakin dia mendekati 1 semakin baik, dan semakin ia mendekati 0 atau bahkan ketika nilainya negatif ini mengindikasikan modelnya kurang baik.

Mean Absolute Error (MAE) atau Mean Absolute Deviation (MAD)

metrics evaluasi ke-2 yang akan kita pelajari disini adalah **Mean Absolute Error** atau kadang juga dikenal dengan **Mean Absolute Deviation, Mean Absolute Error** kalau dalam bahasa Indonesia kita bisa bilang nilai rata-rata dari Absolute error dari prediksi, dan ini merupakan formula untuk MAE.

$$MAE = \frac{1}{n}\sum_{1}^{n} =_1 |y_i - \hat{y}_1|$$

MAE ini akan menghitung selisih atau error antara y_i dengan $\acute{y}_i$, y_i ini merepresentasikan setiap nilai target pada testing set, sedangkan $\acute{y}_i$ ini merupakan nilai prediksi yang dihasilkan oleh model kita.

Proses perhitungan selisih ini akan memungkinkan saja untuk menghasilkan nilai positif atau negatif ,kalau hasil prediksinya ternyata lebih kecil dari apa yang seharusnya maka nilainya positif, Tetapi kalau hasil prediksinya ternyata lebih besar dari nilai yang semestinya maka nilainya akan menjadi negatif, untuk menghindari nilai negatif maka disini kita menerapkan yang namanya Absolute function, absolute function ini akan menghilangkan nilai negatif, artinya kalau kita mendapati nilai -2 setelah dikenakan Absolute akan menjadi 2, atau dengan kata lain semua nilai negatif akan berubah menjadi positif dengan hadirnya absolute function ini,lalu berikutnya setiap selisih nilai ini akan diakumulasikan atau dijumlahkan untuk berikutnya kita bagi dengar **n,n** ini merepresentasikan jumlah data poinnya.

Selanjutnya kita akan demokan tahapan menggunakan Mind Absolute Error untuk melakukan evaluasi model regression pada scikit-learn..

```
from sklearn.metrics import mean_absolute_error
MAE = mean_absolute_error(y_test, y_pred)

print(f'MAE: {MAE}')
```

Pertama-tama kita juga akan import Absolute error nya, caranya di sini kita Panggil **from sklearn.metrics import** mean_absolute_error, untuk menggunakan mengabsolut error ini caranya juga mudah ya kita panggil saja **mean_absolute_error** lalu kita sertakan **y_test** dan **y_pred** sebagai parameter, nilai dari Min Absolute error ini akan kita tampung ke dalam variabel MAE, berikutnya kita cetak ke layar .

Kita coba eksekusi kode nya

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelBinarizer
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }

sensus_df = pd.DataFrame(sensus)
X_train = np.array(sensus_df[['tinggi', 'jk']])
y_train = np.array(sensus_df['berat'])

X_train_transposed = np.transpose(X_train)

lb = LabelBinarizer()
jk_binarised = lb.fit_transform(X_train_transposed[1])

jk_binarised = jk_binarised.flatten()
X_train_transposed[1] = jk_binarised
X_train = X_train_transposed.transpose()
k = 3
model = KNeighborsRegressor(n_neighbors=k)
model.fit(X_train, y_train)


X_test = np.array([[168, 0], [180, 0], [160, 1], [169, 1]])
y_test = np.array([65, 96, 52, 67])

y_pred = model.predict(X_test)
r_squared = r2_score(y_test, y_pred)
MAE = mean_absolute_error(y_test, y_pred)

print(f'MAE: {MAE}')
```

dan ini hasil nya

```
MAE: 9.666666666666668
```

Karena yang dihitung disini adalah nilai errornya, atau nilai selisih antara prediksi dengan nilai sebenarnya, oleh karenanya semakin kecil nilai MAE nya akan mengindikasikan kualitas model yang lebih baik.

Mean Squared Error (MSE) atau Mean Squared Deviation (MSD)

metrics evaluasi yang ke-3 yang akan kita pelajari adalah **Mean Squared Error** atau kadang juga dikenal sebagai **Mean Squared Deviation**, disini **Mean Squared Error** ini merupakan nilai rata-rata dari erro kuadrat untuk prediksi, dan ini merupakan formulanya.

$$MAE = \frac{1}{n}\sum_{1}^{n} = {}_{1}(y_i - \hat{y}_i)^2$$

kalau kita lihat di sini sama kita juga akan menghitung error,nantinya disini kita juga akan menghitung nilai selisih antara $y_i$dengan $\hat{y}$ nya, $y_i$ini merupakan nilai target dari testing set,sedangkan $\hat{y}$ ini merupakan nilai estimasi yang di hasilkan oleh model kita, kedua nilai ini kita selisihkan.

serupa dengan kasus sebelumnya, selisih nilainya ini bisa positif bisa negatif, kalau nilai prediksinya ternyata lebih besar dari nilai sebenarnya maka akan menghasilkan nilai negatif, sedangkan kalau nilai produksinya lebih kecil dari nilai sebenarnya atau nilai ekspektasinya maka akan menghasilkan nilai positif.

Untuk menghindari kemunculan nilai negatif pada MSE, selisih nilainya ini akan kita pangkatkan dua ,lalu berikutnya hasil pemangkatan ini akan kita akumulasikan untuk kemudian kita bagi dengan **n,n** ini merupakan jumlah data Point nya.

Selanjutnya kita akan demokan tahapan menggunakan mean Square error ini untuk melakukan evaluasi model regression pada scikit-learn.

```
from sklearn.metrics import mean_squared_error
MSE = mean_squared_error(y_test, y_pred)
print(f'MSE: {MSE}')
```

Pertama-tama kita akan import dulu caranya kita akan Panggil **from sklearn.metrics import** mean_squared_error, lalu untuk menggunakan nya, caranya kita tinggal panggil saja

mean_squared_error lalu kita sertakan **y_test** dan **y_pred** sebagai parameternya,nilai nya ini kita tampung ke dalam variabel **MSE** untuk selanjut nya kita cetak ke layar.

kita coba eksekusi kode nya

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelBinarizer
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error

sensus = {
        'tinggi': [158, 170, 183, 191, 155, 163, 180, 158, 178],
        'berat': [64, 86, 84, 80, 49, 59, 67, 54, 67],
        'jk': [
            'pria', 'pria', 'pria', 'pria', 'wanita', 'wanita', 'wanita', 'wanita',
'wanita'
            ]
        }

sensus_df = pd.DataFrame(sensus)
X_train = np.array(sensus_df[['tinggi', 'jk']])
y_train = np.array(sensus_df['berat'])

X_train_transposed = np.transpose(X_train)

lb = LabelBinarizer()
jk_binarised = lb.fit_transform(X_train_transposed[1])

jk_binarised = jk_binarised.flatten()
X_train_transposed[1] = jk_binarised
X_train = X_train_transposed.transpose()
k = 3
model = KNeighborsRegressor(n_neighbors=k)
model.fit(X_train, y_train)


X_test = np.array([[168, 0], [180, 0], [160, 1], [169, 1]])
y_test = np.array([65, 96, 52, 67])

y_pred = model.predict(X_test)
r_squared = r2_score(y_test, y_pred)
MAE = mean_absolute_error(y_test, y_pred)
MSE = mean_squared_error(y_test, y_pred)

print(f'MSE: {MSE}')
```

dan ini hasil nya

```
MSE: 157.16666666666663
```

Untuk MSE juga sama dengan MAE, karena yang dihitung adalah nilai error, maka tentunya makin kecil nilai MSE nya akan mengindikasikan model yang makin baik, dan perlu diingat juga disini nilai MSE akan selalu lebih besar bila dibandingkan dengan nilai MAE, karena setiap nilai errornya untuk MSE akan kita pangkatkan 2.

◆ Permasalahan Scaling pada Features

Selanjut nya kita akan Jelaskan permasalahan terkait scaling pada nilai features, dataset yang diangkat pada sesi pembelajaran ini memiliki dua features, yaitu tinggi badan jenis kelamin.

Nilai tinggi badan direpresentasikan dalam satuan cm sedangkan nilai jenis kelamin direpresentasikan dalam bilangan biner 0 dan 1, kalau kita cermati disini untuk nilai tinggi badan sebenarnya bisa juga direpresentasikan dengan satuan pengukuran lain,misal saja mm dan m.

Kali ini kita akan mempelajari apakah perbedaan satuan pengukuran ini akan berdampak pada konsistensi hasil kalkulasi euclidean distance nya, untuk memperjelas pemahaman disini kita akan perkecil dulu ukuran dataset menjadi dua instance saja, selanjutnya kita akan lakukan dua eksperimen, pada eksperimen pertama kita akan menggunakan mm sebagai satuan pengukuran untuk tinggi badan, sedangkan pada eksperimen kedua kita akan menggunakan m sebagai satuan pengukurannya.

kita akan demokan eksperimen pertama terlebih dahulu

```
from scipy.spatial.distance import euclidean

#tinggi dalam milimeter
X_train = np.array([[1700, 0], [1600, 1]])
X_new = np.array([[1640, 0]])

[euclidean(X_new[0], d) for d in X_train]
```

Pertama-tama karena kita butuh untuk menggunakan euclidean distance, maka kita akan impor dulu **from scipy.spatial.distance import** euclidean, berikutnya disini kita akan siapkan dulu dataset nya, di sini ada dua variabel **X_train** dan **X_new, X_train** ini akan berisi sekumpulan nilai features untuk training set, sedangkan **X_new** nya akan berisi sekumpulan nilai features untuk datapoint yang akan

kita prediksi, hanya saja di sini kalau kita perhatikan nilai features untuk tinggi badannya direpresentasikan dalam satuan milimeter, oleh karenanya disini untuk data point pertama pada training set tinggi badannya adalah 1700 dan 1600, sedangkan pada datapoint baru yang akan diprediksi, tinggi badannya adalah 1640.

Disini kalian Gak perlu kaget mengapa nilainya tiba-tiba menjadi besar, nilai nya menjadi besar karena memang satuan pengukuran nya berubah, yang awalnya dari cm menjadi mm.

Lalu pada tahapan berikutnya kita akan coba ukur jarak untuk datapoint yang baru ini terhadap kedua datapoint pada training set nya,disini proses pengukurannya kita akan memanfaatkan euclidean distance.

Kita coba eksekusi kode nya

```
import numpy as np
from scipy.spatial.distance import euclidean

#tinggi dalam milimeter
X_train = np.array([[1700, 0], [1600, 1]])
X_new = np.array([[1640, 0]])

print([euclidean(X_new[0], d) for d in X_train])
```

dan ini hasil nya

```
[60.0, 40.01249804748511]
```

yang **60.0** ini merupakan jarak antara datapoint yang baru ini dengan data point pertama pada training set, sedangkan **40.01249804748511,** ini merupakan jarak antara datapoint yang baru ini terhadap datapoint kedua dari training setnya, pada eksperimen pertama ini nampak jelas bahwa datapoint yang baru lebih dekat dengan data point kedua bila dibandingkan dengan data point pertama, bisa nampak di sini nilai jarak nya lebih pendek.

Selanjutnya kita akan demokan eksperimen kedua dengan memanfaatkan dataset yang sama, hanya saja kita akan menggunakan m sebagai satuan pengukuran untuk tinggi badannya.

```
X_train = np.array([[1.7, 0], [1.6, 0]])
X_new = np.array([[1.64, 0]])
[euclidean(X_new[0], d) for d in X_train]
```

Untuk eksperimen kedua dataset nya masih sama, yang berbeda adalah untuk eksperimen kedua tinggi badannya diukur dalam satuan meter, kalau kita lihat disini untuk training setnya ada 1,7 dan 1,6 untuk tinggi badannya, 1,7 nya berarti 1,7 m dan 1,6 disini merupakan 1,6 M, nilai 1,7 m ini setara dengan 1700 mm demikian juga dengan 1,6 M ini setara dengan 1600 mm. lalu untuk datapoint yang baru di sini tinggi badannya adalah 1,64 , nilai 1,64 m ini setara dengan nilai 1640 mm, disini kita bisa yakin bahwa kedua dataset ini merupakan dataset yang sebenarnya identik, hanya saja direpresentasikan dengan satuan pengukuran yang berbeda, lalu berikutnya disini kita juga akan coba ukur jarak antara datapoint baru ini dengan kedua nilai data point pada training setnya, disini kita juga akan memanfaatkan euclidean distance.

Kita coba eksekusi kode nya

```
import numpy as np
from scipy.spatial.distance import euclidean

X_train = np.array([[1.7, 0], [1.6, 0]])
X_new = np.array([[1.64, 0]])

print([euclidean(X_new[0], d) for d in X_train])
```

dan ini hasilnya

```
[0.06000000000000005, 0.039999999999999813]
```

0.06000000000000005 ini merupakan jarak antara datapoint yang baru ini dengan datapoint pertama pada training set, lalu **0.039999999999999813** ini merupakan jarak antara datapoint yang baru ini dengan data point kedua pada trainig set nya.

Pada eksperimen kedua ini bisa nampak bahwa datapoint yang baru lebih dekat dengan datapoint pertama bila dibandingkan dengan data point kedua,di sini kita menemui ketidakkonsistenan dalam pengukuran jarak,padahal kita menggunakan dataset yang sama persis, yang berbeda disini hanya pada satuan pengukuran nya saja.

◆ Menerapkan Standard Scaler (Standard Score atau Z-Score)

Salah satu metode yang bisa kita gunakan untuk mengatasi permasalahan di atas adalah dengan menerapkan standar scaler, sesuai dengan namanya standard scaler ini menerapkan standar score atau z-score yang umum ditemui dalam bidang statistika.

Dan ini merupakan formula untuk scaler score.

$$z = \frac{x - \dot{x}}{S}$$

Bisa kita lihat di sini **x** nya merepresentasikan nilai features, sedangkan **x bar** disini merepresentasikan rata-rata nilai features nya, lalu nilai **S** merepresentasikan nilai Standard deviation dari sekumpulan nilai features tersebut.

Bagi kalian yang memang masih awam dengan istilah standard score ini kita sangat menyarankan kalian mempelajari topik statistika deskriptif terlebih dahulu, pemahaman akan dasar statistika sangat membantu suatu kita mendalami bidang machine learning, karena memang ada banyak sekali ilmu statistika yang diterapkan dalam machine learning.

Selanjutnya kita akan menemukan tahapan untuk menerapkan standar scaler ini pada kedua eksperimen yang sebelumnya kita lakukan, kita akan mulai dari eksperimen pertama dulu dimana tinggi badan direpresentasikan dalam satuan milimeter.

```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
```

Karena kita ingin menerapkan standar scaler, maka pertama-tama kita perlu import dulu class nya, caranya cukup mudah di sini kita tinggal panggil saja **from sklearn.preprocessing import** StandardScaler,selanjutnya kita akan bentuk objek dari class ini dan disini kita akan tampung objek nya ke dalam variabel **SS**.

Pada tahapan berikutnya kita akan coba replikasi eksperimen yang pertama

```
#tinggi dalam milimeter
X_train = np.array([[1700, 0], [1600, 1]])
X_train_scaled = ss.fit_transform(X_train)
print(f'X_train_scaled:\n{X_train_scaled}\n')

X_new = np.array([[1640, 0]])
X_new_scaled = ss.fit_transform(X_new)
print(f'X_new_scaled:\n{X_new_scaled}\n')

jarak = [euclidean(X_new_scaled[0], d) for d in X_train_scaled]
print(f'jarak: {jarak}')
```

Disini nilai features untuk tinggi badannya direpresentasikan dalam satuan milimeter, yang berbeda dengan eksperimen sebelumnya adalah setelah nilai features yang dibentuk berikutnya kita akan mengenakan standard scaler ini pada nilai **X_train** nya, di sini kita Panggil **ss.fit_transform** yang kita kenakan pada **X_train**, selanjutnya hasil transformasinya kita tampung ke dalam variabel **X_train_scaled**,untuk selanjutnya kita cetak layar.

Demikian juga untuk **X_new** nya, ini juga akan kita kenakan proses transformasi, hanya saja di sini proses transformasinya langsung kita Panggil transform, kita enggak panggil lagi fit_transform, karena proses fitnya akan kita kenakan pada **X_train**, tetapi proses transform nya akan kita kenakan baik pada **X_train** maupun pada **X_new**, hasil transformasi pada **X_new** ini kita tampung ke dalam variabel **X_new_scaled** untuk berikutnya kita tampilkan ke layar.

Selanjutnya di sini kita akan hitung jaraknya dengan menggunakan eucledian distance, yang kita hitung jaraknya disini adalah nilai **X_new_scaled** dengan nilai **X_train_scaled** nya.

Kita coba eksekusi kode nya

```
import numpy as np

from scipy.spatial.distance import euclidean
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
#tinggi dalam milimeter
X_train = np.array([[1700, 0], [1600, 1]])
X_train_scaled = ss.fit_transform(X_train)
print(f'X_train_scaled:\n{X_train_scaled}\n')

X_new = np.array([[1640, 0]])
X_new_scaled = ss.fit_transform(X_new)
print(f'X_new_scaled:\n{X_new_scaled}\n')

jarak = [euclidean(X_new_scaled[0], d) for d in X_train_scaled]
print(f'jarak: {jarak}')
```

dan ini dia hasil nya

Ini merupakan nilai **X_train_scaled** atau nilai **X_train** yang sudah dikenakan standar scaler.

```
X_train_scaled:
[[ 1. -1.]
 [-1.  1.]]
```

Sedangkan ini merupakan nilai **X_new_scaled** nya

X_new_scaled:

```
[[0. 0.]]
```

Berikutnya ini merupakan euclidean distance untuk **X_train_scaled** dengan **X_new_scaled** nya.

```
jarak: [1.4142135623730951, 1.4142135623730951]
```

Bisa kita lihat di sini nilai-nilai yang ada di sini sudah tidak lagi menggunakan satuan cm ataupun mm ataupun m, satuan nya sudah menggunakan satuan standar score atau z-score.

1.4142135623730951 ini merupakan jarak antara **X_new_scaled** dengan **X_train_scaled,** untuk datapoint pertama, aedangkan **1.4142135623730951** merupakan jarak antara **X_new_scaled** dengan **X_train_scaled** untuk data point kedua.

Pada eksperimen pertama ini nampak jelas bahwa datapoint yang baru lebih dekat dengan data point pertama bila dibandingkan dengan data point kedua.

Selanjutnya kita akan demokan eksperimen kedua dengan memanfaatkan dataset yang sama, hanya saja kita akan menggunakan m sebagai satuan pengukuran untuk tinggi badan.

```
#satuan dalam meter
X_train = np.array([[1.7, 0], [1.6, 0]])
X_train_scaled = ss.fit_transform(X_train)
print(f'X_train_scaled:\n{X_train_scaled}\n')


X_new = np.array([[1.64, 0]])
X_new_scaled = ss.fit_transform(X_new)
print(f'X_new_scaled:\n{X_new_scaled}\n')

jarak = [euclidean(X_new_scaled[0], d) for d in X_train_scaled]
print(f'jarak: {jarak}')
```

Ini merupakan eksperimen yang kedua, di sini sekumpulan nilai features nya sama, yang berbeda adalah satuan pengukuran nya saja, disini dalam meter sedangkan yang sebelumnya dalam mm.

Tetapi disini sebelum kita hitung jaraknya dengan menggunakan euclidean distance, nilai-nilai features ini akan kita scaling dulu dengan memanfaatkan standar scaler, di sini untuk **X_train** nya kita Panggil dulu **ss.fit_transform(X_train)** dan hasil transformasinya kita tampung ke dalam variabel **X_train_scaled**,lalu berikutnya untuk **X_new**, di sini Kita tinggal panggil saja **ss.fit_transform(X_new),** lalu hasil transformasinya kita tampung ke dalam variabel **X_new_scaled**

untuk berikutnya kita akan dihitung jaraknya, dan perhitungan jarak nya di sini juga sama kita akan memanfaatkan euclidean distance.

Kita coba eksekusi kode nya

```
import numpy as np

from scipy.spatial.distance import euclidean
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
#satuan dalam meter
X_train = np.array([[1.7, 0], [1.6, 0]])
X_train_scaled = ss.fit_transform(X_train)
print(f'X_train_scaled:\n{X_train_scaled}\n')


X_new = np.array([[1.64, 0]])
X_new_scaled = ss.fit_transform(X_new)
print(f'X_new_scaled:\n{X_new_scaled}\n')

jarak = [euclidean(X_new_scaled[0], d) for d in X_train_scaled]
print(f'jarak: {jarak}')
```

dan ini hasil nya

Ini merupakan **X_train_scaled** nya,

```
X_train_scaled:
[[ 1.  0.]
 [-1.  0.]]
```

ini merupakan **X_new_scaled** nya,

```
X_new_scaled:
[[0. 0.]]
```

dan ini merupakan perhitungan jarak dengan menggunakan euclidean distance.

```
jarak: [1.0000000000000022, 0.9999999999999978]
```

1.0000000000000022 merupakan jarak antara **X_new_scaled** dengan data point pertama pada **X_train_scaled,** sedangkan **0.9999999999999978** merupakan jarak antara **X_new_scaled** dengan data point kedua pada **X_train_scaled.**

Kalau kita lihat di eksperimen kedua in, menghasilkan nilai jarak yang sama atau setidaknya sangat mendekati dengan nilai jarak pada eksperimen pertama, disini juga bisa nampak bahwa datapoint baru juga lebih dekat dengan datapoint pertama bila dibandingkan dengan data point kedua.

Pada kasus ini harapannya bisa membantu kalian dalam memahami bagaimana penerapan standar scaler bisa membantu menjamin konsistensi hasil kalkulasi euclidean distance, dan kita tahu dari sesi pembelajaran sebelumnya betapa pentingnya peranan euclidean distance ini pada model KNN.

◆ Menerapkan Features Scaling pada KNN

Selanjut nya kita akan ulangi lagi proses training dan evaluasi model KNN yang pernah kita lakukan sebelumnya, tapi kali ini kita akan menerapkan features scaling.

Untuk kasus ini features scaling yang akan kita terapkan adalah standard scaler, kita akan lihat Bagaimana features scaler ini berpotensi dalam meningkatkan performa dari model KNN.

Dataset

pertama-tama kita akan persiapkan dulu training set nya

```
# training set
X_train = np.array([[158, 0], [170, 0], [183, 0], [191, 0], [155, 1], [163, 1],[180, 1],
[158, 1], [170, 1]])
y_train = np.array([64, 86, 84, 80, 49, 59, 67, 54, 67])


# test set
X_test = np.array([[168, 0], [180, 0], [160, 1], [169, 1]])
y_test = np.array([65, 96, 52, 67])
```

Disini kita akan siapkan **X_train** untuk menampung sekumpulan nilai features untuk training set dan **y_train** untuk menampung sekumpulan nilai target untuk training setnya.

Selain itu untuk keperluan evaluasi kita juga akan persiapkan **X_test** dan **y_test**.

Features Scaling (Standard Scaler)

Kalau kita perhatikan di sini training set dan testing set yang kita gunakan ini sama persis dengan training set dan testing set yang kita gunakan sebelumnya, yang berbeda disini adalah setelah training dan testing setnya tersedia kita tidak langsung melakukan proses turning model tetapi pertama-tama disini kita akan scaling dulu features nya.

```
X_train_scaled = ss.fit_transform(X_train)
X_test_scaled = ss.fit_transform(X_test)

print(f'X_train_scaled:\n{X_train_scaled}\n')
print(f'X_test_scaled:\n{X_test_scaled}\n')
```

Disini kita akan Panggil dulu **ss.fit_transform** yang kita terapkan pada **X_train** lalu hasil scaling nya kita tampung ke dalam variabel **X_train_scaled,** demikian juga untuk **X_test** , **ss.fit_transform(X_test),** lalu hasil scaling nya kita tampung ke dalam variabel **X_test_scaled**.

Kita coba eksekusi kode nya

```
import numpy as np
from scipy.spatial.distance import euclidean
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()


# training set
X_train = np.array([[158, 0], [170, 0], [183, 0], [191, 0], [155, 1], [163, 1],[180, 1],
[158, 1], [170, 1]])
y_train = np.array([64, 86, 84, 80, 49, 59, 67, 54, 67])


# test set
X_test = np.array([[168, 0], [180, 0], [160, 1], [169, 1]])
y_test = np.array([65, 96, 52, 67])

X_train_scaled = ss.fit_transform(X_train)
X_test_scaled = ss.fit_transform(X_test)

print(f'X_train_scaled:\n{X_train_scaled}\n')
print(f'X_test_scaled:\n{X_test_scaled}\n')
```

dan ini hsil nya

ini merupakan X_train yang sudah di scalling.

```
X_train_scaled:
[[-0.9908706  -1.11803399]
 [ 0.01869567 -1.11803399]
 [ 1.11239246 -1.11803399]
 [ 1.78543664 -1.11803399]
 [-1.24326216  0.89442719]
 [-0.57021798  0.89442719]
 [ 0.86000089  0.89442719]
 [-0.9908706   0.89442719]
 [ 0.01869567  0.89442719]]
```

ini merupakan X_test yang sudah di scalling.

```
X_test_scaled:
[[-0.17557375 -1.        ]
 [ 1.50993422 -1.        ]
 [-1.29924573  1.        ]
 [-0.03511475  1.        ]]
```

Setelah nilai features nya kita scalling, tahapan berikutnya kita baru akan melakukan proses training model.

Training & Evaluasi Model

```
model.fit(x_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)

MAE = mean_absolute_error(y_test, y_pred)
MSE = mead_squared_error(y_test, y_pred)

print(f'MAE: {MAE}')
print(f'MSE: {MSE}')
```

Disini objek modelnya kita training dengan memanggil **model.fit** lalu kita sertakan **x_train_scaled** dan **y_train** sebagai parameternya, disini features yang dilewatkan adalah features yang sudah kita scalling, lalu setelah model nya kita training, train model ini atau model yang sudah kita training ini akan kita gunakan untuk melakukan prediksi, di sini kita Panggil **model.predict,** lalu kita akan sertakan **X_test_scaled,** dan hasilnya kita tampung ke dalam variabel **y_pred,** selanjutnya barulah kita akan hitung nilai errornya, disini nilai error yang akan kita hitung adalah baik MAE maupun MSE nya,

caranya cukup mudah untuk MAE Kita tinggal panggil saja **mean_absolute_error(y_test, y_pred)** ,sedangkan untuk MSE kita akan Panggil **mean_square_error(y_test, y_pred).**

Kita coba eksekusi kode nya

```
import numpy as np
from scipy.spatial.distance import euclidean
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error

ss = StandardScaler()


# training set
X_train = np.array([[158, 0], [170, 0], [183, 0], [191, 0], [155, 1], [163, 1],[180, 1],
[158, 1], [170, 1]])
y_train = np.array([64, 86, 84, 80, 49, 59, 67, 54, 67])


# test set
X_test = np.array([[168, 0], [180, 0], [160, 1], [169, 1]])
y_test = np.array([65, 96, 52, 67])

X_train_scaled = ss.fit_transform(X_train)
X_test_scaled = ss.fit_transform(X_test)

k = 3
model = KNeighborsRegressor(n_neighbors=k)

model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)

MAE = mean_absolute_error(y_test, y_pred)
MSE = mean_squared_error(y_test, y_pred)

print(f'MAE: {MAE}')
print(f'MSE: {MSE}')
```

dan ini hasil nya

```
MAE: 7.583333333333336
MSE: 85.13888888888893
```

Kita akan bandingkan nilainya dengan hasil evaluasi model sebelumnya,sebelum kita menerapkan features scalling, hasil MSE nya adalah **157.16666666666663,**lalu kalau kita bandingkan hasil MAE adalah **9.666666666666668** , kalau kita lihat disini hasilnya lebih baik.

Nilai MAE dan nilai MSE setelah kita menerapkan features scaling bisa nampak hasilnya lebih kecil atau dengan kata lain kita bisa menghasilkan model dengan kualitas atau performa yang lebih baik setelah kita menerapkan features scaling.

◆ Bab 8 : Multiple Linear Regression & Polynomial Regression

Kali ini kita akan mempelajari dua model regression, yaitu multiple linear regression dan juga polinomial regresssion, keduanya merupakan pengembangan dari model simple linear regression yang pernah kita pelajari sebelumnya.

Pertama-tama kita akan persiapkan terlebih dahulu dataset yang akan kita gunakan dalam sesi pembelajaran ini.

◆ Sample Dataset

Disini kita akan kembali memanfaatkan dataset pizza, hanya saja kali ini kita akan menyertakan kolom baru yaitu jumlah topping atau n_toping, dataset yang akan kita bentuk ini berisi daftar ukuran diameter, jumlah topping dan harga, dari sejumlah pizza yang akan kita tampung kedalam format pandas DataFrame,kali ini kita juga akan langsung persiapkan baik training dataset maupun testing dataset nya.

Berikut akan kita demokan caranya.

Training dataset

```
import pandas as pd

pizza = {'diameter': [6, 8, 10, 14, 18],
         'n_topping': [2, 1, 0, 2, 0],
               'harga': [7, 9, 13, 17.5, 18]}

train_pizza_df = pd.DataFrame(pizza)
print(train_pizza_df)
```

Pertama-tama ya kita akan import dulu modul pandas nya, karena kita ingin membentuk dataset dalam format Pandas dataframe di sini kita panggil saja **import pandas as pd,** berikutnya kita akan siapkan dataset kita.

Dataset kita kali ini awalnya dibentuk dari suatu dictionary yang terdiri dari tiga buah keys, yaitu diameter, n_toping dan harga, dimana setiap keys ini akan menampung list yang berisi sekumpulan bilangan, berikutnya dictionary ini akan kita tampung ke dalam variabel **pizza**, untuk selanjutnya dictionary ini akan kita gunakan sebagai base atau sebagai basis untuk membuat Pandas dataframe, makanya di sini kita Panggil **pd.DataFrame(pizza),** artinya disini kita akan membentuk suatu pandas DataFrame dari data-data yang tersimpan dalam variabel **pizza** ini,selanjutnya dataframe yang terbentuk ini akan kita tampung ke dalam **variabel train_pizza_df**, untuk selanjutnya kita coba tampilkan ke layar.

Dan jika kita jalankan maka hasil nya seperti ini

	diameter	n_topping	harga
0	6	2	7.0
1	8	1	9.0
2	10	0	13.0
3	14	2	17.5
4	18	0	18.0

Ini merupakan training dataset yang baru saja kita bentuk.

Testing Dataset

Selanjutnya kita juga akan menggunakan teknik yang sama untuk mempersiapkan testing dataset nya,

```
import pandas as pd

pizza = {'diameter': [8, 9, 11, 16, 12],
         'n_topping':[2, 0, 2, 2, 0],
               'harga': [11, 8.5, 15, 18, 11]}

train_pizza_df = pd.DataFrame(pizza)
test_pizza_df = pd.DataFrame(pizza)
print(test_pizza_df)
```

Disini tahapannya sama persis, hanya saja datanya berbeda, dan untuk testing dataset nya akan kita tampung ke dalam variabel **test_pizza_df**.

Dan ini hasil nya

	diameter	n_topping	harga
0	8	2	11.0
1	9	0	8.5
2	11	2	15.0
3	16	2	18.0
4	12	0	11.0

Ini merupakan testing dataset kita.

Kedua dataset ini sama-sama terdiri dari 3 buah kolom yaitu kolom diameter, n_topping dan harga.

Dalam sesi pembelajaran kali ini kita akan membentuk suatu model machine learning sederhana yang dapat digunakan untuk memprediksi harga pizza berdasarkan ukuran diameter dan jumlah topingnya, dengan kata lain disini data diameter dan jumlah topping akan berperan sebagai features, sedangkan harga pizza akan berperan sebagai target, karena disini yang akan diprediksi berupa nilai continuous dan bukannya kategori maka kasus ini termasuk dalam regression tasks.

◆ Preprocessing Dataset

Sebelum kita melangkah ke proses training model, kita akan kelompokkan terlebih dahulu sekumpulan nilai features dan nilai target dari dataset yang kita miliki.

Untuk kasus kita kali ini, diameter dan jumlah topping akan kita konversikan ke dalam numpy array dan kita tampung ke dalam variabel X_train sebagai sekumpulan features untuk training set, lalu untuk harga pizza juga akan kita konversikan menjadi numpy array untuk selanjutnya kita tampung ke dalam variabel y_train, sebagai sekumpulan nilai target untuk training set, proses serupa juga akan kita terapkan pada testng setnya.

berikut akan kita Demokrat prosesnya

```
import numpy as np

X_train = np.array(train_pizza_df[['diameter', 'n_topping']])
y_train = np.array(train_pizza_df['harga'])

print(f'X_train:\n{X_train}\n')
print(f'y_train:\n{y_train}')
```

Disini karena kita ingin menampung nilai dataset ke dalam numpy array, maka kita akan impor dulu module nya **import numpy as np,**selanjutnya kita akan pisahkan antara features dengan targetnya, untuk kasus kita kali ini kolom yang menjadi features adalah kolom diameter dan jumlah topping, nilai dari kedua kolom ini akan kita konversikan ke dalam numpy array, oleh karenanya di sini kita Panggil **np.array(train_pizza_df[['diameter', 'n_topping']]),** selanjutnya sekumpulan nilai features ini akan kita tampung dalam variabel **X_train,** ini merupakan features untuk training set kita.

Lalu berikutnya kita juga akan tampung nilai targetnya ke dalam numpy array, oleh karenanya di sini kita Panggil **np.array(train_pizza_df['harga']),** dimana kolom harga ini merupakan target prediction nya, nilai ini lalu kita tampung dalam variabel **y_train,** berikutnya kita akan coba cetak layar.

Kita coba eksekusi kode nya

```
import pandas as pd
import numpy as np

pizza = {'diameter': [6, 8, 10, 14, 18],
         'n_topping':[2, 1, 0, 2, 0],
               'harga': [7, 9, 13, 17.5, 18]}
train_pizza_df = pd.DataFrame(pizza)



X_train = np.array(train_pizza_df[['diameter', 'n_topping']])
y_train = np.array(train_pizza_df['harga'])

print(f'X_train:\n{X_train}\n')
print(f'y_train:\n{y_train}')
```

dan ini hasilnya

```
X_train:
[[ 6  2]
 [ 8  1]
```

```
 [10  0]
 [14  2]
 [18  0]]

y_train:
[ 7.   9.  13.  17.5 18. ]
```

Seperti biasa,**X_train** selalu berada dalam format array 2 dimensi,sedangkan **y_train** akan selalu berada dalam format array 1 dimensi.

Berikutnya kita juga akan lakukan hal serupa untuk testing setnya.

```
X_test = np.array(train_pizza_df[['diameter', 'n_topping']])
y_test = np.array(train_pizza_df['harga'])

print(f'X_test:\n{X_test}\n')
print(f'y_test:\n{y_test}')
```

kita coba eksekusi kode nya

```
import pandas as pd
import numpy as np

pizza = {'diameter': [8, 9, 11, 16, 12],
        'n_topping':[2, 0, 2, 2, 0],
             'harga': [11, 8.5, 15, 18, 11]}
test_pizza_df = pd.DataFrame(pizza)

X_test = np.array(test_pizza_df[['diameter', 'n_topping']])
y_test = np.array(test_pizza_df['harga'])

print(f'X_test:\n{X_test}\n')
print(f'y_test:\n{y_test}')
```

dan ini hasil nya

```
X_test:
[[ 8  2]
 [ 9  0]
 [11  2]
 [16  2]
 [12  0]]

y_test:
```

```
[11.   8.5 15.  18.  11. ]
```

dan ini merupakan testing set kita baik untuk features maupun targetnya.

◆ Multiple Linear Regression

Setelah kita memahami konteks dataset dan permasalahan nya, kita akan coba menerapkan multiple linear regression untuk kasus kita kali ini, multiple linear regression merupakan generalisasi dari simple linear regression yang memungkinkan untuk menggunakan beberapa explanatory variables.

Jadi sebenarnya perbedaan mendasar antara simple linear regression dan multiple linear regression terletak pada jumlah features atau explanatory variables yang digunakan, pada simple linear regression kita hanya menggunakan satu features saja untuk melakukan prediksi nilai,sedangkan pada multiple linear regression kita akan menggunakan lebih dari satu features untuk melakukan prediksi.

Berikut adalah formula dari multiple linear regression.

$$y = a + ß_1 x_1 + ß_2 x_2 + ... + ß_n x_n$$

Bisa kita lihat disini, kita memiliki lebih dari satu nilai **x** , ada x1,x2,x3 sampai dengan x_n, dengan kata lain jumlah **x** ini atau jumlah features nya ini bisa lebih dari satu, yang perlu diperhatikan disini adalah untuk setiap nilai **x** nya akan dikalikan dengan **beta(ß)**, jadi di sini kalau kita memiliki **x1** maka satunya akan dikalikan dengan **beta(ß)1** dan kalau kita juga memiliki **x2,** maka nilai **x2** nya akan dikalikan dengan **beta(ß)2.**

Selanjutnya kita akan demokan implementasi multiple linear regression dengan scikit-learn.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print(f'r_squared: {r2_score(y_test, y_pred)}')
```

Pertama-tama kita akan impor dulu modulnya **from sklearn.linear_model import** LinearRegression,lalu untuk matriks ya kita akan impor juga **from sklearn.metrics import** r2_score, lalu tahapan berikutnya kita akan membentuk objek modelnya, di sini Kita tinggal panggil saja **LinearRegression()**, dan setelah objek modelnya terbentuk kita akan tampung ke dalam variabel **model**, lalu tahapan berikutnya adalah kita akan melakukan proses training model kita.

Di sini proses trainingnya juga cukup sederhana ya kita cukup panggil saja **model.fit(X_train, y_train)**,selanjutnya setelah model nya kita training,model ini akan kita gunakan untuk melakukan prediksi, disini kita akan gunakan train model kita untuk melakukan prediksi terhadap features yang terdapat dalam testing set kita, oleh karenanya di sini kita Panggil **model.predict(X_test),**lalu hasil prediksinya yang kita tampung ke dalam variabel **y_pred** untuk selanjutnya hasil prediksi ini akan kita gunakan untuk mengukur performa dari model kita, untuk kasus kita kali ini kita akan gunakan **r2_score** sebagai pengukuran performanya di sini kita Panggil **r2_score(y_test, y_pred)**.

Kita cona eksekusi kode nya

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score


pizza = {'diameter': [6, 8, 10, 14, 18],
         'n_topping':[2, 1, 0, 2, 0],
               'harga': [7, 9, 13, 17.5, 18]}
train_pizza_df = pd.DataFrame(pizza)

pizza = {'diameter': [8, 9, 11, 16, 12],
         'n_topping':[2, 0, 2, 2, 0],
               'harga': [11, 8.5, 15, 18, 11]}
test_pizza_df = pd.DataFrame(pizza)

X_train = np.array(train_pizza_df[['diameter', 'n_topping']])
y_train = np.array(train_pizza_df['harga'])

X_test = np.array(test_pizza_df[['diameter', 'n_topping']])
y_test = np.array(test_pizza_df['harga'])

model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print(f'r_squared: {r2_score(y_test, y_pred)}')
```

dan ini hasil nya

r_squared: 0.7701677731318468

Ini merupakan nilai dari r_square nya, bisa nampak disini harus keluarnya adalah **0.7701677731318468.**

Disini kita juga bisa menggunakan metrics pengukuran performa yang lain yang pernah kita jelaskan sebelumnya untuk kasus regression, kalau kalian perhatikan disini bisa nampak juga bahwa ternyata pada scikit-learn sebenarnya tidak ada perbedaan antara implementasi simple linear regression dan multiple linear regression yang berbeda yaitu hanya pada jumlah features yang dilewatkan nya saja.

◆ Polymonial Regression

Selanjutnya kita akan mempelajari bentuk pengembangan lain dari simple linear regression yaitu polymonial regression, polymonial regression memodelkan hubungan antara independent variabel x dan dependent variable y sebagai derajat polymonial dalam X ini merupakan definisi dari polymonial regression.

Disini yang dimaksud dengan independent variable itu sebenarnya features dan yang dimaksud dengan independent variables ini adalah targetnya.

Preprocessing Dataset

Disini untuk menyederhanakan proses belajar nya, kita akan pangkas jumlah features nya menjadi 1 features saja.

Berikut akan kita demokan prosesnya.

```
X_train = np.array(train_pizza_df['diameter']).reshape(-1, 1)
y_train = np.array(train_pizza_df['harga'])
```

```
print(f'X-train:\n{X_train}\n')
print(f'y_train:\n{y_train}')
```

Disini features nya yang kita pangkas menjadi 1 features, dalam kasus ini yang kita jadikan features adalah diameter pizzanya,tetapi target prediksinya tetap sama yaitu harga pizza, dan disini berarti kita akan siapkan suatu numpy array **np.array(train_pizza_df['diameter']**, karena kita hanya mau mengambil kolom diamer saja.

Lalu berikutnya di sini kita perlu reshape, **reshape(-1, 1),** Kenapa perlu kita reshape, karena pada scikit-learn, untuk features nya itu harus berada dalam array 2 dimensi, sedangkan di sini kita hanya memiliki 1 kolom saja sebagai features nya, artinya features yang terdiri dari 1 variabel ini lalu kita transformasikan ke dalam array 2 dimensi, dan disini kita bisa memanfaatkan method **reshape(-1, 1)**,berikutnya array yang terbentuk akan kita tampung ke dalam variabel **X_train**, selanjutnya kita juga akan membentuk array yang 2 yang akan menampung nilai target, disini kita akan Panggil **np.array(train_pizza_df['harga'])**, karena kolom harga inilah yang akan kita gunakan sebagai target prediction nya, array yang terbentuk ini akan kita tampung ke dalam variabel **y_train,** selanjutnya kita cetak ke layar.

Kita eksekusi kode nya

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score


pizza = {'diameter': [6, 8, 10, 14, 18],
        'n_topping':[2, 1, 0, 2, 0],
             'harga': [7, 9, 13, 17.5, 18]}
train_pizza_df = pd.DataFrame(pizza)


X_train = np.array(train_pizza_df['diameter']).reshape(-1, 1)
y_train = np.array(train_pizza_df['harga'])


print(f'X-train:\n{X_train}\n')
print(f'y_train:\n{y_train}')
```

dan ini hasil nya

ini X_train nya

```
X-train:
[[ 6]
 [ 8]
 [10]
 [14]
 [18]]
```

dan ini y_train nya

```
y_train:
[ 7.   9.  13.  17.5 18. ]
```

Bisa nampak di sini walaupun X_train terdiri dari satu features saja, tetapi array yang terbentuk ini tetap dalam format array 2 dimensi, ini dimungkinkan karena kita memanggil method **reshape(-1, 1).**

Polymonial Regression: Quadratic

Kita akan pelajari implementasi Polymonial regression Quadratic pada scikit-learn dengan derajat Polymonial 2 atau biasa dikenal dengan istilah Quadratic, ih selain derajat polynomial 2, Kita juga bisa menggunakan derajat polymonial yang lainnya seperti 3,4 dan seterusnya, hanya saja yang paling umum ditemui adalah derajat polymonial 2 dan 3 atau biasa dikenal dengan quadratic dan Cubic.

Berikut adalah formula dari Polymonial regression Quadratic

$$y = a + ß_1 x + ß_2 x^2$$

Kalau kita perhatikan di sini, nilai features nya hanya 1 yaitu **x**, hanya saja nilai features yang tunggal ini akan kita representasikan dalam format polymonial, dan untuk kasus kita kali ini kita akan menggunakan polynomial derajat 2, artinya kita akan memiliki x pangkat 1, dan juga x pangkat 2, untuk x pangkat 1 nya akan dikalikan dengan beta1 dan untuk x pangkat 2 nya kan kita kalikan dengan beta2.

Kalau kita menggunakan Cubic, berarti nanti akan ada tambahan satu lagi yaitu x pangkat 3 yang nilainya akan di kali kan dengan beta3.

Polynomial Features

Untuk menerapkan polynomial regression, pertama-tama kita perlu melakukan transformasi terhadap features dari dataset yang kita miliki.

Berikut akan kita demokan prosesnya

```
from sklearn.preprocessing import PolynomialFeatures

quadratic_feature = PolynomialFeatures(degree=2)
X_train_quadratic = quadratic_feature.fit_transform(X_train)

print(f'X_train_quadratic:\n{X_train_quadratic}\n')
```

Pertama-tama kita akan impor dulu module nya, di sini Kita tinggal panggil saja **from sklearn.preprocessing import** PolinomialFeatures, lalu berikutnya kita akan bentuk objek polynomial features nya, **PolynomialFeatures(degree=2),** karena untuk kasus kita kali ini kita ingin membentuk quadratic polynomial regression,maka degree nya kita beri nilai 2.

Objek yang terbentuk ini berikutnya akan kita tampung ke dalam variabel **quadratic_feature,** dimana objek ini akan kita gunakan untuk melakukan proses transformasi terhadap data set features yang kita miliki, oleh karenanya di sini kita Panggil **quadratic_feature.fit_transform(X_train)**, lalu hasil transformasinya akan kita tampung ke dalam variabel **X_train_quadratic, X_train_quadratic** ini akan berisi features dari training set kita yang sudah di transformasi kan ke dalam polynomial features, lalu berikutnya kita akan coba cetak layar hasilnya.

Kita coba eksekusi kode nya

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.preprocessing import PolynomialFeatures

pizza = {'diameter': [6, 8, 10, 14, 18],
         'n_topping':[2, 1, 0, 2, 0],
               'harga': [7, 9, 13, 17.5, 18]}
train_pizza_df = pd.DataFrame(pizza)


X_train = np.array(train_pizza_df['diameter']).reshape(-1, 1)
y_train = np.array(train_pizza_df['harga'])


quadratic_feature = PolynomialFeatures(degree=2)
X_train_quadratic = quadratic_feature.fit_transform(X_train)
```

```
print(f'X_train_quadratic:\n{X_train_quadratic}\n')
```

dan ini hasil nya

```
X_train_quadratic:
[[  1.   6.  36.]
 [  1.   8.  64.]
 [  1.  10. 100.]
 [  1.  14. 196.]
 [  1.  18. 324.]]
```

Bisa nampak di sini, nilai features yang tunggal tadi sekarang menjadi 3, untuk lebih jelasnya kita akan coba tinjau kembali nilai dari variabel X_train kita, nilainya ada 6, 8, 10, 14, 18, dan kita tahu nilai pada indeks pertamanya ini adalah 6, tapi kok bisa sekarang nilainya menjadi 1,6,3,6, prosesnya lebih kurang seperti ini, 1 ini diperoleh dari nilai x dipangkatkan 0, dan kita tahu nilai x pada indeks ke 0 nya adalah 6,6 dipangkatkan 0 adalah 1, lalu berikutnya 6 dipangkatkan 1 adalah 6, dan yang berikutnya 6 dipangkatkan 2 adalah 36.

Lalu berikutnya untuk nilai x pada indeks berikutnya adalah 8, nilai 8 ini akan kita pangkatkan 0, nilainya adalah 1, lalu 8 di pangkat kan 1 adalah 8, dan 8 pangkat 2 adalah 64, demikian seterusnya.

Training Model

Setelah kita memiliki polynomial features yang dibutuhkan, selanjutnya kita bisa mulai melakukan training model.

Berikut akan kita demokan prosesnya

```
model = LinearRegression()
model.fit(X_train_quadratic, y_train)
```

Untuk proses training model pertama-tama kita akan bentuk dulu objek modelnya, disini objek modelnya dibentuk dengan cara **LinearRegression()**, object model ini lalu kita tampung dalam variabel **model,** untuk selanjutnya object model ini akan kita training dengan memanfaatkan nilai features yang sudah kita transformasikan ke dalam polynomial, oleh karenanya di sini **X_train** nya kita Panggil **X_train_quadratic,** Sedangkan untuk **y_train** nya tetap di sini kita tidak melakukan proses transformasi,yang perlu digaris bawahi disini adalah proses transformasi polynomial nya itu dikenakan pada features dan tidak dikenakan pada target.

Bisa nampak juga di sini bahwa ternyata pada scikit-learn tidak ada perbedaan antara implementasi linear regression dan polynomial regression, yang berbeda hanyalah pada polynomial regression kita perlu melakukan transformasi features ke dalam polynomial features sebelum dilakukan proses training model.

Visualisasi Data

selanjutnya untuk memberikan gambaran yang lebih baik terkait quadratic polynomial regression ini, kita akan coba visualisasikan modelnya.

```
import matplotlib.pyplot as plt

X_vis = np.linspace(0, 25, 100).reshape(-1, 1)
X_vis_quadratic = quadratic_feature.transform(X_vis)
y_vis_quadratic = model.predict(X_vis_quadratic)

plt.scatter(X_train, y_train)
plt.plot(X_vis, y_vis_quadratic, '-r')

plt.title('Perbandingan Diameter dan Harga Pizza')
plt.xlabel('Diameter (inch)')
plt.ylabel('Harga(dollar)')
plt.xlim(0, 25)
plt.ylim(0, 25)
plt.grid(True)
plt.show()
```

Disini untuk memvisualisasikan modalnya pertama-tama kita akan import dulu modul nya **import matplotlib.pyplot as plt**,lalu kita akan bentuk dulu sederet bilangan mulai dari 0-25 sebanyak 100 datapoint, datapoint ini kemudian akan kita masukkan ke dalam variabel **X_vis**, selanjutnya nilai **X_vis** ini akan kita kenakan proses transformasi, kita gunakan **quadratic_feature.transform(X_vis),** disini kita tidak melakukan fit transform, tetapi kita hanya melakukan transform, karena kita akan menggunakan **quadratic_feature** yang kita bentuk sebelumnya, lalu hasil transformasinya kita masukkan ke dalam X_vis_quadratic, selanjutnya kita akan melakukan prediksi terhadap nilai **X_vis_quadratic** ini**,** hasil prediksi ini kita tampung ke dalam variabel **y_vis_quadratic.**

Disini yang akan kita visualisasikan itu ada dua hal, yang pertama adalah **X_train** dan **y_train** nya atau datapoint kita gunakan untuk melakukan proses training nya, untuk data training nya ini kita akan visualisasikan sebagai scatter plot, makanya di sini kita Panggil **plt.scatter(X_train, y_train),** dan ini akan menghasilkan lima titik, dan kelima titik Ini merepresentasikan kelima datapoint pada training set

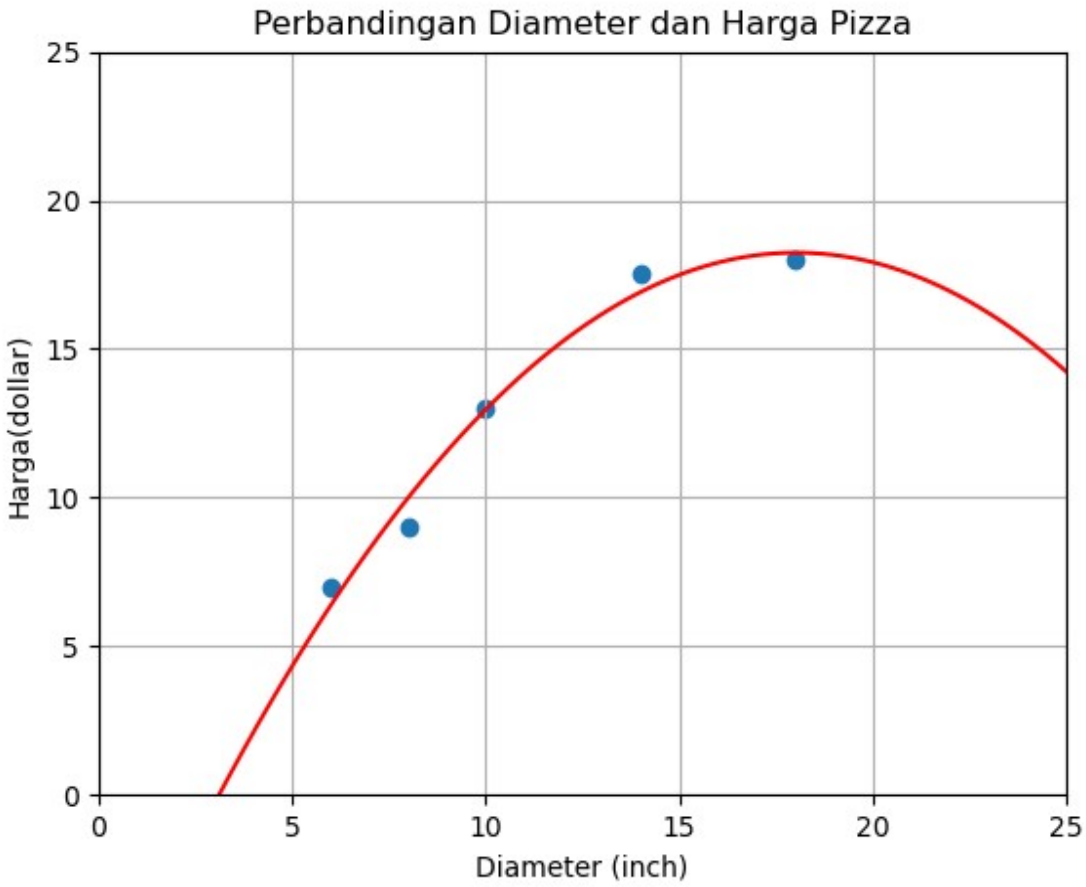
kita, lalu berikutnya kita akan floating hasil prediksi dari model kita, di sini kita floating **plt.plot(X_vis, y_vis_quadratic, '-r'),**dan kali ini kita akan floating sebagai line plot, atau floating garis dengan warna merah, bisa nampak Disini yang untuk pylinomial regression bentuknya tidak lagi garis lurus,lalu selanjutnya yang untuk keterangan tambahan disini kita juga cetak judul, Kita cetak label dan kita juga set ya batasan atau limit untuk sumbu-x maupun sumbu y nya, disini karena kita ingin menampilkan grid, maka plt.grid nya kita set sebagai True, yang terakhir di sini kita Panggil plt.show untuk memunculkan visualisasinya.

kita coba eksekusi kode nya

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt

pizza = {'diameter': [8, 9, 11, 16, 12],
        'n_topping':[2, 0, 2, 2, 0],
             'harga': [11, 8.5, 15, 18, 11]}

train_pizza_df = pd.DataFrame(pizza)
test_pizza_df = pd.DataFrame(pizza)

X_train = np.array(train_pizza_df['diameter']).reshape(-1, 1)
y_train = np.array(train_pizza_df['harga'])


quadratic_feature = PolynomialFeatures(degree=2)
X_train_quadratic = quadratic_feature.fit_transform(X_train)

model = LinearRegression()
model.fit(X_train_quadratic, y_train)
X_vis = np.linspace(0, 25, 100).reshape(-1, 1)
X_vis_quadratic = quadratic_feature.transform(X_vis)
y_vis_quadratic = model.predict(X_vis_quadratic)

plt.scatter(X_train, y_train)
plt.plot(X_vis, y_vis_quadratic, '-r')

plt.title('Perbandingan Diameter dan Harga Pizza')
plt.xlabel('Diameter (inch)')
plt.ylabel('Harga(dollar)')
plt.xlim(0, 25)
plt.ylim(0, 25)
```

```
plt.grid(True)
plt.show()
```

dan ini hasilnya



Disini untuk memperdalam pemahaman kita kita coba bandingkan antara model simple linear regression, quadratic polynomial regression dan juga Cubic polynomial regression secara visual.

Berikut akan kita demokan prosesnya.

```
# Training set
plt.scatter(X_train, y_train)

# Linear
model = LinearRegression()
model.fit(X_train, y_train)
X_vis = np.linspace(0, 25, 100).reshape(-1, 1)
y_vis = model.predict(X_vis)
plt.plot(X_vis, y_vis, '--r', label='linear')
```

```
# Quadratic
quadratic_feature = PolynomialFeatures(degree=2)
X_train_quadratic = quadratic_feature.fit_transform(X_train)
model = LinearRegression()
model.fit(X_train_quadratic, y_train)
X_vis_quadratic = quadratic_feature.transform(X_vis)
y_vis = model.predict(X_vis_quadratic)
plt.plot(X_vis, y_vis, '--g', label='quadratic')

# Cubic
cubic_feature = PolynomialFeatures(degree=3)
X_train_cubic = cubic_feature.fit_transform(X_train)
model = LinearRegression()
model.fit(X_train_cubic, y_train)
X_vis_cubic = cubic_feature.transform(X_vis)
y_vis = model.predict(X_vis_cubic)
plt.plot(X_vis, y_vis, '--y', label='cubic')

plt.title('Perbandingan Diameter dan Harga Pizza')
plt.xlabel('Diameter (inch)')
plt.ylabel('Harga(dollar)')
plt.legend()
plt.xlim(0, 25)
plt.ylim(0, 25)
plt.grid(True)
plt.show()
```

Pertama-tama di sini kita Panggil dulu **plt.scatter(X_train, y_train),** artinya disini kita akan melakukan floating training dataset nya, hasil plotting yaitu berupa kelima datapoint yang ada di visual nya atau kelima titik nya.

Lalu berikutnya disini kita akan membentuk model simple linear regression, pertama-tama kita bentuk dulu objek modelnya, **LinearRegression(),** kita tampung dalam variabel **model,**lalu model nya kita training dengan menggunakan **X_train** dan **y_train**, selanjutnya disini kita akan menyiapkan 100 datapoint dengan jangkauan mulai dari 0-25, lalu nilai datapoint ini sebelum ditampung ke dalam variabel **X_vish** akan kita **reshape** dengan nilai parameter -1,1, dan selanjutnya nilai yang ditampung dalam variabel **X_vish** ini akan kita gunakan sebagai features untuk melakukan prediksi, oleh karenanya di sini kita Panggil **model.predict(X_vis)**, dan hasil prediksinya kita tampung ke dalam variabel **X_vish** lalu berikutnya, hasil prediksinya ini akan kita tampilkan secara visual, kita Panggil **plt.plot(X_vis, y_vis, '--r', label='linear'),**kita floating sebagai garis putus-putus dengan warna merah dan kita sertai label **linear.**

Berikutnya kita akan bandingkan dengan yang quadratic, perbedaannya itu sebenarnya hanya pada penerapan polynomial features dengan degree atau derajat 2, di sini kita bentuk dulu obyeknya

PolynomialFeatures(degree=2),lalu kita set parameter degree nya dengan nilai 2, dan objek nya ini kita tampung ke dalam variabel **quadratic_feature**, selanjutnya kita akan gunakan objek **quadratic_feature** ini untuk melakukan fit transform terhadap nilai dari **X_train** nya,dan hasil transformasi quadratic polynomial nya akan kita tampung dalam variabel **X_train_quadratic**, lalu Pada tahapan berikutnya kita akan bentuk objek modelnya yang kita tampung ke dalam variabel **model** dan selanjutnya kita akan melakukan proses training terhadap model kita,hanya saja di sini modelnya akan kita training dengan menggunakan features yang sudah kita transformasikan ke dalam quadratic polynomial, oleh karenanya di sini kita Panggil **(X_train_quadratic, y_train)** ,lalu berikutnya nilai **X_vish** nya juga sama akan kita transformasikan menjadi quadratic polynomial untuk selanjutnya kita tampung ke dalam variabel **X_vis_quadratic,** dan nilai **X_vis_quadratic** ini akan kita gunakan untuk melakukan prediksi,selanjutnya hasil prediksinya juga lain kita floating, kali ini akan kita floating sebagai garis putus-putus berwarna hijau, oleh karenanya disini kita menggunakan **'--g'**, **'--g' i**ni merepresentasikan Green,lalu berikutnya kita juga akan sertakan suatu **label** dengan nilai **quadratic**.

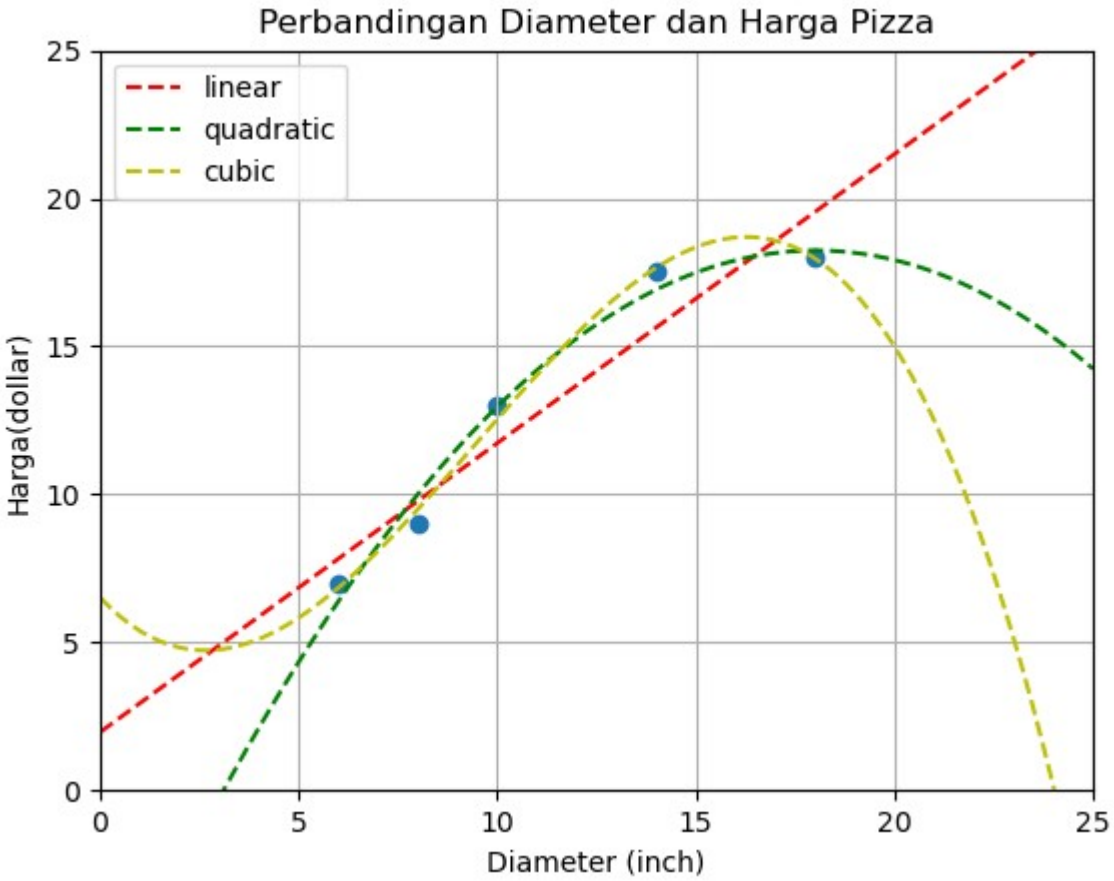
Berikutnya kita akan coba lihat Cubic,Cubic itu perbedaannya hanya sewaktu kita membentuk objek pylinomial features, kita sertakan degree nya adalah 3, sedangkan untuk proses berikutnya itu sama persis, di sini untuk cubic polynomial regression akan kita floating sebagai **'--y'**, artinya akan kita floating sebagai garis putus-putus berwarna yellow atau kuning dan kita akan sertakan **label qubic**.

kita eksekusi kode nya

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt

pizza = {'diameter': [6, 8, 10, 14, 18],
        'n_topping':[2, 1, 0, 2, 0],
             'harga': [7, 9, 13, 17.5, 18]}
train_pizza_df = pd.DataFrame(pizza)


X_train = np.array(train_pizza_df['diameter']).reshape(-1, 1)
y_train = np.array(train_pizza_df['harga'])

# Training set
plt.scatter(X_train, y_train)

# Linear
```

```
model = LinearRegression()
model.fit(X_train, y_train)
X_vis = np.linspace(0, 25, 100).reshape(-1, 1)
y_vis = model.predict(X_vis)
plt.plot(X_vis, y_vis, '--r', label='linear')

# Quadratic
quadratic_feature = PolynomialFeatures(degree=2)
X_train_quadratic = quadratic_feature.fit_transform(X_train)
model = LinearRegression()
model.fit(X_train_quadratic, y_train)
X_vis_quadratic = quadratic_feature.transform(X_vis)
y_vis = model.predict(X_vis_quadratic)
plt.plot(X_vis, y_vis, '--g', label='quadratic')

# Cubic
cubic_feature = PolynomialFeatures(degree=3)
X_train_cubic = cubic_feature.fit_transform(X_train)
model = LinearRegression()
model.fit(X_train_cubic, y_train)
X_vis_cubic = cubic_feature.transform(X_vis)
y_vis = model.predict(X_vis_cubic)
plt.plot(X_vis, y_vis, '--y', label='cubic')

plt.title('Perbandingan Diameter dan Harga Pizza')
plt.xlabel('Diameter (inch)')
plt.ylabel('Harga(dollar)')
plt.legend()
plt.xlim(0, 25)
plt.ylim(0, 25)
plt.grid(True)
plt.show()
```

dan hasilnya ini


Perbandingan Diameter dan Harga Pizza
linear
quadratic
cubic
Harga(dollar)
Diameter (inch)
0
5
10
15
20
25

Bab 9 : Categorical Encoding : Label Encoding & One Hot Encoding

Kali ini kita akan mempelajari dua macam categorical encoding yang umum ditemui dalam bidang machine learning, yaitu label encoding dan one hot encoding, materi yang akan kita pelajari ini membutuhkan pengetahuan terkait tipe data dan juga level of measurement yang umum ditemui dalam bidang statistika.

Apa itu Categorical Encoding?

Pertama-tama kita akan pahami terlebih dahulu apa itu categorical encoding dan mengapa kita membutuhkannya.

Secara umum Suatu data sedangkan terbentuk dari kombinasi nilai numerical dan categorical, hanya saja komputer atau mesin memiliki keterbatasan dimana pada dasarnya komputer hanya dapat memahami angka atau numerical, dan tidak dapat memahami teks atau categorical, oleh karenanya kita butuh untuk melakukan konversi nilai categorical ini menjadi nilai mumerical agar algoritma machine learning dapat memahaminya dengan baik.Proses konversi nilai categorical menjadi nilai numerical ini dikenal dengan istilah categorical encoding.

Dalam bidang machine-learning terdapat banyak jenis categorical encoding, dan 2 yang paling populer adalah label encoding dan one hot encoding,bagi kalian yang tertarik mempelajari lebih lanjut berikut kita juga sertakan halaman Wikipedia terkait sebagai referensi bagi kalian untuk memperdalam pemahaman.

Refrensi : https://en..wikipedia.org/wiki/One-hot

Di sini kita mempelajari label encoding terlebih dahulu.

1.Label Encoding

Pada label encoding setiap kategori pada suatu features akan di urutkan secara alfabet dan direpresentasikan dengan sebuah nilai integer.

Untuk lebih jelasnya berikut akan kita demokan prosesnya,disini pertama-tama kita akan siapkan sebuah database terlebih dahulu.

Dataset

```
import pandas as pd


df = pd.DataFrame({
       'country': ['India', 'US', 'Japan', 'US', 'Japan'],
       'age': [44, 34, 46, 35, 23],
       'salary': [7200, 6500, 9800, 4500, 3400]
})
print(df)
```

Di sini dataset nya akan kita bentuk dalam format Pandas dataframe,oleh karenanya pertama-tama di sini kita impor dulu **import pandas as pd,** selanjutnya dataframe ini akan kita bentuk dari suatu dictionary, disini terdapat tiga buah keys yang akan merepresentasikan 3 buah kolom, yaitu country, age, dan selery, dan setiap keys ini akan diberi value sekumpulan nilai yang berasosiasi dengan keys tersebut.

Selanjutnya dictionary ini akan digunakan sebagai basis untuk membentuk objek data frame, dan objek data frame nya akan kita tampung ke dalam variabel **df** untuk selanjutnya kita tampilkan ke layar.

dan ini hasil nya jika kode di eksekusi

	country	age	salary
0	India	44	7200
1	US	34	6500
2	Japan	46	9800
3	US	35	4500
4	Japan	23	3400

Bisa nampak di sini terdapat 3 buah kolom pada dataframe kita yaitu country, age dan salary, dari ketika kolom ini terdapat sebuah kolom dengan nilai categorical yaitu kolom country, dan kita akan terapkan label encoding pada kolam tersebut, berikut akan kita demokan prosesnya

Label Encoding pada Scikit Learn

Untuk menerapkan label encoding dengan scikit learn caranya cukup mudah.

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
df['country'] = label_encoder.fit_transform(df['country'])

print(df)
```

Pertama-tama kita akan import dulu modelnya, **from sklearn.preprocessing import** LabelEncoder, lalu selanjutnya kita akan bentuk objek dari class **LabelEncoder** ini, disini objek **LabelEncoder** nya akan kita tampung ke dalam variabel **label_encoder**, lalu selanjutnya **label_encoder** ini akan kita gunakan untuk melakukan fit transform terhadap kolom country dari dataframe kita, oleh karenanya di sini kita Panggil **label_encoder.fit_transform(df['country']),** dan hasil transformasinya akan kita tampung kembali ke dalam kolom country, oleh karenanya di sini kita panggil **df['country']** dan selanjutnya kita juga akan Tampilkan nilai **df** nya ke layar.

Kita coba eksekusi kode nya

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

df = pd.DataFrame({
      'country': ['India', 'US', 'Japan', 'US', 'Japan'],
      'age': [44, 34, 46, 35, 23],
      'salary': [7200, 6500, 9800, 4500, 3400]
})

label_encoder = LabelEncoder()
df['country'] = label_encoder.fit_transform(df['country'])

print(df)
```

dan ini hasil nya

	country	age	salary
0	0	44	7200
1	2	34	6500
2	1	46	9800
3	2	35	4500

```
4         1   23    3400
```

Ini adalah kondisi setelah dikenakan label encoding, sebelum label encoding nya diterapkan kolom country ini berisi data teks, sedangkan setelah label encoding nya kita terapkan,kolom country ini berisi data numerik.

Pertanyaan nya bagaimana proses dibaliknya, bagaimana nilai India ini bisa berubah menjadi 0,US ini menjadi 2, dan Japan yg menjadi 1, untuk memahami ini maka kita Panggil

```
print(label_encoder.classes_)
```

Dan ini hasil nya

```
[array(['India', 'Japan', 'US'], dtype=object)]
```

Ini merupakan kumpulan class atau kumpulan kategori yang menjadi acuan dari label encoder kita, bisa nampak di sini terdapat 3 buah kategori negara yaitu, India, Japan dan ,US,ketiga kategori atau label negara ini diurutkan secara alfabet, India,Japan lalu US.

Disini berarti India menempati posisi indeks ke-0,Japan menempati posisi indeks ke-1 dan US menempati posisi indeks ke-2, oleh karenanya di sini India akan direpresentasikan sebagai integer 0, lalu Japan akan direpresentasikan sebagai integer 1, dan US akan direpresentasikan sebagai integer 2.

Sebelum kita melangkah ke jenis encoding kedua, disini perlu di jelaskan terlebih dahulu salah satu kelemahan dasar dari label encoding ini, kalau kalian perhatikan disini kategori atau label negara akan diurutkan secara alfabet, oleh karenanya India direpresentasikan dengan integer 0, Japan direpresentasikan dengan integer 1, dan US direpresentasikan dengan integer 2.

Dengan representasi nilai semacam ini beberapa algoritma machine-learning bisa saja menarik hubungan bahwa India bernilai lebih kecil dari Japan, dan Japan bernilai lebih kecil dari US, kondisi semacam ini tentunya tidak kita harapkan,maka untuk mengantisipasi kondisi semacam ini maka kita akan mempelajari jenis kategori label encoding Yang kedua.

◆ 2. One Hot Encoding

Categorical encoding kedua yaitu one hot encoding, pada one hot encoding setiap kategori pada suatu features akan diurutkan secara alfabet dan direpresentasikan sebagai sekumpulan bits.

Uuntuk lebih jelasnya berikut akan kita demokan prosesnya.

Dataset

Pertama-tama kita akan siapkan sebuah dataset terlebih dahulu, untuk kasus kita kali ini kita juga akan menggunakan dataset yang sama persis dengan dataset yang kita gunakan sebelumnya.

```
df = pd.DataFrame({
       'country': ['India', 'US', 'Japan', 'US', 'Japan'],
       'age': [44, 34, 46, 35, 23],
       'salary': [7200, 6500, 9800, 4500, 3400]
})
df
```

	country	age	salary
0	India	44	7200
1	US	34	6500
2	Japan	46	9800
3	US	35	4500
4	Japan	23	3400

Selanjutnya kita akan terapkan one hot encoding pada kolom country, berikut akan kita demokan prosesnya.

One Hot Encoding pada Scikit Learn

Pertama-tama sebagai persiapan kita akan isolasi dulu sekumpulan nilai dari kolom country ini, dimana sekumpulan nilai ini akan kita konversikan menjadi sebuah array.

```
X = df['country'].values.reshape(-1, 1)
print(X)
```

Di sini kita panggil **df[‘country’].values.reshape(-1, 1),** lalu nilai array ini akan kita tampung ke dalam variabel **X** untuk kemudian kita tampilkan ke layar.

Sebagian dari kalian mungkin bertanya-tanya Kenapa kok kita butuh melakukan **reshape(-1, 1)**, jawabannya adalah karena sekumpulan nilai ini akan kita perlakukan sebagai nilai features, dimana nilai features dalam scikit learn itu diharapkan untuk ditampung dalam suatu array 2 dimensi, oleh karenanya di sini kita perlu reshape dengan menggunakan parameter **-1, 1.**

dan ini dia hasil nya

```
[['India']
 ['US']
 ['Japan']
 ['US']
 ['Japan']]
```

Bisa kita lihat di sini, ini merupakan array 2 dimensi yang berisi sekumpulan data negara atau country, lalu selanjutnya kita akan bersiap untuk menerapkan one hot encoding.

```
from sklearn.preprocessing import OneHotEncoder

onehot_encoder = OneHotEncoder()
X = onehot_encoder.fit_transform(X).toarray()
X
```

Untuk menerapkan one hot encoding dengan scikit learn, pertama-tama kita perlu impor dulu moduke nya, di sini kita Panggil **from sklearn.preprocessing import** OneHotEncoder, lalu selanjutnya kita akan bentuk objek dari class **OneHotEncoder** ini ,objek dari class ini akan kita tampung ke dalam variabel **onehot_encoder,** lalu selanjutnya objek ini akan kita gunakan untuk melakukan Fit transform terhadap **X,** dan kita tahu disini **X** ini berupa array yang menampung sekumpulan nilai features country, lalu berikutnya hasil transformasinya juga akan kita konversikan menjadi suatu array untuk kemudian kita tampung kembali ke dalam variabel **X** dan terakhir disini kita juga akan munculkan nilai **X** ke layar.kita coba eksekusi scriptnya disini.

```
[[1. 0. 0.]
 [0. 0. 1.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 1. 0.]]
```

Hasilnya ini merupakan hasil one hot encoding dari data yang ada atas setelah kita reshape.

Untuk memahami proses one hot encoding ini kita akan coba panggil

```
print(onehot_encoder_categories_)
```

ini dia hasilnya

```
[array(['India', 'Japan', 'US'], dtype=object)]
```

kita dihadapkan pada sebuah array yang menampung sekumpulan nilai country yang sudah diurutkan secara alfabetik, dan bisa nampak disini terdapat 3 kategori atau 3 label negara yaitu, India, Japan dan US,karena terdapat 3 negara, maka untuk tiap negaranya akan direpresentasikan dengan bits 3 digit.

Bisa kita lihat India merupakan negara pertama, Japan merupakan negara kedua, dan US merupakan negara ke-3,artinya disini India akan memiliki representasi bits 3 digit, di mana digit pertamanya akan bernilai 1, ini merupakan representasi untuk India,lalu berikutnya Japan, karena Japan disini merupakan negara kedua,maka Japan akan direpresentasikan dengan bits 3 digit di mana digit keduanya akan bernilai 1, ini merupakan representasi dari Japan, bisa kita lihat di sini digit keduanya bernilai 1, posisinya sama, ini merupakan representasi dari Japan,lalu berikutnya untuk US, karena US, untuk kasus disini merupakan negara dengan posisi ketiga, maka US direpresentasikan dengan bits 3 digit,dimana pada digit ketiganya akan bernilai 1, ini merupakan representasi dari US,bisa kita lihat di sini ya digit ketiganya bernilai 1.

Selanjutnya array hasil one hot encoding ini akan kita konversikan menjadi dataframe.

```
df_onehot = pd.DataFrame(X, columns=[str(i) for i in range(X.shape[1])])
df_onehot
```

Prosesnya kita panggil **pd.DataFrame**,ini artinya kita akan membentuk suatu objek dataframe, dimana obyeknya akan dibentuk dari sekumpulan nilai ditampung oleh **X,** selanjutnya kita juga menspesifikasikan kolom dari dataframe kita, selanjutnya objek data frame yang terbentuk ini akan kita tampung ke dalam variabel **df_onehot,** untuk selanjutnya kita tampilkan ke layar.

Dan ini hasilnya

hasilnya ini merupakan representasi harinya dan ini merupakan representasi data framenya dan keduanya merupakan representasi dari onehotencoder

◆ Label Encoding vs One Hot Encoding

Setelah mempelajari dua teknik categorical encoding yaitu Label Encoding dan One Hot Encoding, mungkin sebagian dari Anda bertanya, kapan sebaiknya menggunakan Label Encoding dan kapan menggunakan One Hot Encoding? Jawabannya tergantung pada karakteristik data dan kasus yang dihadapi.

Namun, berikut ini panduan umum yang dapat membantu Anda memutuskan teknik encoding mana yang lebih tepat:

Gunakan One Hot Encoding apabila:

- Nilai categorical bersifat nominal, artinya tidak ada urutan atau peringkat antar kategori. **Contohnya** jenis kelamin (laki-laki, perempuan), warna (merah, biru, hijau), atau kota kelahiran.
- Jumlah kategori tidak terlalu banyak, sehingga tidak menghasilkan terlalu banyak fitur baru setelah di-encode.

Gunakan Label Encoding apabila:

- Nilai categorical bersifat ordinal, artinya memiliki urutan atau peringkat antar kategori. **Contohnya** tingkat pendidikan (SD, SMP, SMA, S1), tingkat kepuasan (sangat tidak puas, tidak puas, netral, puas, sangat puas), atau peringkat dalam kompetisi.
- Jumlah kategori relatif banyak, sehingga One Hot Encoding akan menghasilkan terlalu banyak fitur baru yang dapat memperlambat komputasi dan mempersulit interpretasi model.

Nominal dan ordinal merupakan level pengukuran dalam statistika yang menggambarkan karakteristik data categorical. Perbedaan utamanya adalah data nominal tidak memiliki urutan atau peringkat, sedangkan data ordinal memiliki urutan atau peringkat.

One Hot Encoding lebih cocok untuk data nominal karena tidak mengasumsikan adanya urutan atau peringkat antar kategori. Dengan One Hot Encoding, setiap kategori direpresentasikan sebagai fitur biner terpisah, sehingga algoritma machine learning dapat memperlakukannya sebagai fitur yang independen.

Sedangkan Label Encoding lebih cocok untuk data ordinal karena mengasumsikan adanya urutan atau peringkat antar kategori. Dengan Label Encoding, setiap kategori diberikan nilai numerik yang mencerminkan urutannya, sehingga algoritma machine learning dapat menangkap pola dan tren dalam data berdasarkan urutan tersebut.

◆ Bab 10 : Text Processing : Bag of Words & Stop Word Filtering

kita akan mempelajari dua macam text Processing yang umum ditemui dalam bidang machine learning yaitu Bag of Words & Stop Word Filtering.

Dari sesi pembelajaran sebelumnya kita sudah mengetahui bahwa komputer atau mesin pada dasarnya tidak bisa memahami data text dengan baik, lalu Bagaimana bila kita dihadapkan pada sekumpulan data text dalam bentuk kalimat atau dokumen, dalam machine-learning terdapat bidang yang secara spesifik membahas dataset text, bidang ini dikenal sebagai Natural Language Processing,atau biasa disingkat sebagai NLP.

Apa yang kita pelajari disini bisa menjadi pengantar bagi kalian yang tertarik untuk mendalami NLP, dalam bidang mesin learning sendiri terdapat beberapa teknik yang umum digunakan untuk melakukan feature Extraction dari dataset text, kali ini kita akan mempelajari dua diantaranya yaitu Bag of Words & Stop Word Filtering.

◆ Bag of Word

Bag of Word menyederhanakan representasi text sebagai sekumpulan kata serta mengabaikan grammar dan posisi tiap kata pada kalimat.Teks akan dikonversi menjadi lowercase dan tanda baca akan diabaikan.

Bagi kalian yang tertarik untuk mempelajarinya lebih lanjut berikut adalah halaman Wikipedia yang bisa dijadikan referensi.

Refrensi : https://en.wikipedia.org/wiki/Bag-of-words_model

Dataset

Pertama-tama kita akan siapkan terlebih dahulu suatu dataset text, untuk kasus kita kali ini dataset ini berupa sekumpulan kalimat pendek datasets ini juga seringkali dikenal dengan istilah corpus.

```
corpus=[
        'Linux has been around since the Mid-1990s.',
        'Linux distributions include the Linux Kernel.',
        'Linux is one of the most prominent open-source software.'
        ]
print(corpus)
```

Untuk kasus kita kali ini corpus kita terdiri dari tiga kalimat pendek,ke-3 kalimat ini akan kita tampung ke dalam suatu list yang kemudian kita assign kedalam suatu variabel yang kita beri nama corpus, lalu berikutnya kita akan tampilkan isi dari variabel corpus nya.

Ini merupakan isi dari variabel corpus kita

```
[
'Linux has been around since the Mid-1990s.',
'Linux distributions include the Linux Kernel.',
'Linux is one of the most prominent open-source software.'
]
```

Bag of Words model dengan CountVectorizer

Selanjutnya kita akan memanfaatkan bag of word untuk melakukan features extraction dari dataset yang kita miliki, pada scikit learn, bag of words model dapat diterapkan dengan memanfaatkan modul **CountVectorizer.**

Berikut akan kita demokan prosesnya.

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
vectorized_X = vectorizer.fit_transform(corpus).todense()
vectorized_X
```

Pertama-tama kita akan impor dulu modul nya, **from sklearn.feature_extraction.text import** CountVectorizer, lalu selanjutnya kita akan membentuk objek dari class **CountVectorizer**, di sini kita Panggil **CountVectorizer(),** lalu objek yang terbentuk akan kita tampung ke dalam variabel **vectorizer,** selanjutnya kita akan menggunakan objek **vectorizer** ini untuk menerapkan metode **transform**

terhadap corpus dataset kita, lalu hasil yang terbentuk ini akan kita konversikan ke dalam suatu array, oleh karenanya di sini kita memanggil method **todense,**atau dengan kata lain metode ini akan mengkonversikan hasil transform dari objek **vektor,** ini menjadi suatu array 2 dimensi, object array 2 dimensi tersebut berikutnya akan kita tampung ke dalam variabel **vectorized_X,** untuk selanjutnya kita tampilkan ke layar.

Kita coba eksekusi kode nya

```
from sklearn.feature_extraction.text import CountVectorizer

corpus=[
        'Linux has been around since the Mid-1990s.',
        'Linux distributions include the Linux Kernel.',
        'Linux is one of the most prominent open-source software.'
        ]
vectorizer = CountVectorizer()
vectorized_X = vectorizer.fit_transform(corpus).todense()
print(vectorized_X)
```

dan ini hasil nya

```
[[1 1 1 0 1 0 0 0 1 1 0 0 0 0 0 1 0 0 1]
 [0 0 0 1 0 1 0 1 2 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 1 0 1 0 1 1 1 1 1 0 1 1 1]]
```

Bisa nampak disini hasilnya berupa array 2 dimensi, dan setiap baris ini akan merepresentasikan tiap kalimat yang berada dalam corpus kita.

Sebagian dari kalian mungkin bertanya-tanya Kenapa ada 1 dan 0 disini, dan apa maksudnya,untuk memperjelas pemahaman kita berikutnya kita akan Panggil method **vectorizer.get_feature_names_out().**

```
print(vectorizer.get_feature_names_out())
```

suatu kita memanggil **vectorizer.get_feature_names_out()**, maka akan dikembalikan sekumpulan kata yang berada dalam bag, atau keranjang, disini perlu kita ingatkan juga bahwa teknik yang kita gunakan disini adalah bag of words, atau kalau mau kita terjemahkan artinya adalah sekumpulan kata dalam keranjang atau dalam bag.

Dan ini Merupakan sekumpulan kata tersebut.

```
['1990s'
```

```
'around'
'been'
'distributions'
'has'
'include'
'is'
'kernel'
'linux'
'mid'
'most'
'of'
'one'
'open'
'prominent'
'since'
'software'
'source'
'the']
```

Kalau kita perhatikan di sini juga kumpulan katanya sudah tidak lagi diurutkan berdasarkan urutan kalimat, melainkan urutannya sekarang sudah diurutkan berdasarkan alfabetic counter.

Hal lain yang perlu diperhatikan disini adalah semua case nya menjadi lowercase, artinya sudah tidak ada lagi huruf besar atau uppercase nya,setiap kata yang ditampung dalam bag atau keranjang ini juga dikenal dengan istilah token.

Array atau sekumpulan bilangan 0 dan satu yang ada di atas merupakan representasi dari features yang ada di sini, atau dengan kata lain indeks ke-0 merepresentasikan 1990, lalu indeks pertama merepresentasikan around, dan seterus nya.

Kalau kita perhatikan di sini

```
[[1 1 1 0 1 0 0 0 1 1 0 0 0 0 0 1 0 0 1]
 [0 0 0 1 0 1 0 1 2 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 1 0 1 0 1 1 1 1 1 0 1 1 1]]
```

Deretan pertama ini merupakan representasi dari kalimat yang pertama, dan kalau kita perhatikan lebih lanjut berarti pada kalimat pertama terdapat kata 1990, makanya di sini nilainya 1, lalu pada kalimat pertama juga terdapat kata around, maka disini nilainya juga 1, pada kalimat pertama juga terdapat kata been, maka disini juga nilainya 1, tetapi pada kalimat pertama tidak terdapat kata distributions, oleh karenanya nilainya disini adalah 0, lalu pada kalimat pertama juga terdapat kata has, oleh karenanya di sini nilainya adalah 1.

Euclidean Distance untuk mengukur kedekatan/jarak antar dokumen (vektor)

Sebagai dari kalian mungkin bertanya-tanya bagaimana representasi text dalam format bag of words ini dapat membantu proses training dari suatu model atau algoritma machine learning, dan jawabannya adalah dengan representasi bag of words suatu algoritma mascine learning dapat dengan lebih mudah mengukur kedekatan atau kemiripan antara dokumen.

berikut akan kita demokan prosesnya.

```
from sklearn.metrics.pairwise import euclidean_distances

for i in range(len(vectorized_X)):
    for j in range(i,len(vectorized_X)):
        if i == j:
            continue
        jarak = euclidean_distances(vectorized_X[i].reshape(1, -1),
vectorized_X[j].reshape(1, -1))
        print(f'Jarak dokumen {i+1} dan {j+1}: {jarak}')
```

Untuk mengukur kedekatannya kita akan menggunakan euclidean distance, berarti di sini pertama-tama kita akan impor dulu modulnya **from sklearn.metrics.pairwise import** euclidean_distances, lalu berikutnya kita akan coba ukur jarak antar kalimatnya, kalimat pertama akan diukur kedekatannya dengan kalimat kedua, lalu Kalimat pertama juga akan diukur kedekatan atau jaraknya dengan kalimat ketiga, selanjutnya kita juga akan ukur kedekatan atau jarak antara kalimat kedua dengan kalimat ketiganya, proses semacam itu akan kita lakukan dengan memanfaatkan for loop statement seperti ini.

kita coba eksekusi kode nya

```
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import euclidean_distances

corpus=[
        'Linux has been around since the Mid-1990s.',
        'Linux distributions include the Linux Kernel.',
        'Linux is one of the most prominent open-source software.'
        ]
vectorizer = CountVectorizer()
vectorized_X = vectorizer.fit_transform(corpus).todense()
```

```
vectorized_X = np.array(vectorized_X)

for i in range(len(vectorized_X)):
    for j in range(i,len(vectorized_X)):
        if i == j:
            continue
        jarak = euclidean_distances(vectorized_X[i].reshape(1, -1),
vectorized_X[j].reshape(1, -1))
        print(f'Jarak dokumen {i+1} dan {j+1}: {jarak}')
```

dan ini hasil nya

```
Jarak dokumen 1 dan 2: [[3.16227766]]
Jarak dokumen 1 dan 3: [[3.74165739]]
Jarak dokumen 2 dan 3: [[3.46410162]]
```

Jarak antara dokumen pertama dengan dokumen kedua adalah **3.16227766 ,**lalu jarak antara dokumen pertama dengan dokumen ketiga adalah **3.74165739,** dan terakhir jarak antara dokumen kedua dengan dokumen ketiga adalah **3.46410162.**

Kalau kita lihat disini, nilai terkecilnya adalah **3.16227766,** dengan kata lain kita bisa simpulkan bahwa tingkat kemiripan antara dokumen pertama dengan dokumen kedua itu yang paling tinggi diantara ketiga dokumen ini.

◆ Stop Word Filtering

Selanjutnya kita akan mempelajari **Stop Word Filtering**,**Stop Word Filtering** menyederhanakan representasi text dengan mengabaikan beberapa kata seperti determiners, contohnya adalah the,a,an, auxiliary verbs seperti do,be,will, dan juga prepositions seperti on, in,at.

Stop word filtering juga cukup umum ditemui dalam bidang NLP atau natural language processing, bagi kalian yang tertarik untuk mempelajarinya lebih lanjut berikut adalah halaman Wikipedia yang bisa kita jadikan referensi.

Refrensi : https://en.wikipedia.org/wiki/Stop_word

Dataset

Untuk kasus kali ini kita akan menggunakan corpus atau dataset yang sama seperti yang kita gunakan sebelumnya.

```
corpus=[
        'Linux has been around since the Mid-1990s.',
        'Linux distributions include the Linux Kernel.',
        'Linux is one of the most prominent open-source software.'
        ]
print(corpus)
```

dan ini adalah corpus nya

```
[
'Linux has been around since the Mid-1990s.',
'Linux distributions include the Linux Kernel.',
'Linux is one of the most prominent open-source software.'
]
```

kalau kita perhatikan disini terdapat beberapa stopword, salah satunya adalah the,has,been,is,dan of,semuanya itu merupakan Stop word yang akan kita abaikan.

Stop Word Filtering dengan CountVectorizer

Disini kita akan memanfaatkan stopword filtering untuk mengeluarkan stopword dari corpus yang kita miliki, pada scikit learn stop word filtering ini juga dapat diterapkan dengan memanfaatkan modul **CountVectorizer.**

Berikut akan kita demokan prosesnya

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(stop_words='english')
vectorized_X = vectorizer.fit_transform(corpus).todense()
vectorized_X
```

Disini kita akan impor dulu modul nya, **from sklearn.feature_extraction.text import** CountVectorizer, lalu selanjutnya kita akan bentuk dulu objek dari class **CountVectorizer**, yang berbeda kali ini adalah sewaktu kita membentuk objek **CountVectorizer,**kita juga menyatakan parameter stopword yang kita beri nilai **english,** karena kasusnya disini adalah untuk melakukan stopword filtering untuk bahasa Inggris, lalu selanjutnya objek **CountVectorizer** yang terbentuk akan kita tampung ke dalam variabel **vectorizer**, objek vectorizer ini kemudian akan kita Panggil untuk menerapkan Fit transform terhadap corpus yang kita miliki, dan hasil transform nya akan kita konversikan ke dalam array 2 dimensi, oleh karenanya di sini kita Panggil method **.todense(),**array 2 dimensi yang terbentuk akan kita tampung ke dalam variabel **vectorized_X** untuk kemudian kita tampilkan di layar.

Kita eksekusi kode nya

```
from sklearn.feature_extraction.text import CountVectorizer

corpus=[
        'Linux has been around since the Mid-1990s.',
        'Linux distributions include the Linux Kernel.',
        'Linux is one of the most prominent open-source software.'
        ]

vectorizer = CountVectorizer(stop_words='english')
vectorized_X = vectorizer.fit_transform(corpus).todense()
print(vectorized_X)
```

dan ini hasil nya

```
[[1 0 0 0 1 1 0 0 0 0]
 [0 1 1 1 2 0 0 0 0 0]
 [0 0 0 0 1 0 1 1 1 1]]
```

Bisa nampak di sini, kalau kita perhatikan di sini ukurannya jauh lebih kecil bila dibandingkan sebelumnya, untuk melihat representasinya dengan lebih detail maka kita bisa panggil **vectorizer.get_feature_names_out()**

```
print(vectorizer.get_feature_names_out())
```

dan ini hasil nya

```
['1990s'
'distributions'
'include'
'kernel'
```

```
'linux'
'mid'
'open'
'prominent'
'software'
'source']
```

Ini merupakan kumpulan kata atau token yang sudah kita filtering stopword nya, bisa nampak di sini kita sudah tidak lagi menyertakan stopword dalam kumpulan kata kita atau dalam token kita, hasil nya kita akan memperoleh representasi dari suatu kalimat yang lebih sederhana.

◆ Bab 11: Mengenal TF-IDF (Term Frequency – Inverse Document Frequency)

Kali ini kita akan mempelajari salah satu teknik yang umum ditemui untuk melakukan feature Extraction pada sekumpulan text atau dokumen yaitu, Term Frequency – Inverse Document Frequency atau Biasa disingkat sebagai TF-IDF.

Sebelumnya kita sudah mempelajari dua teknik dasar yang umum ditemui dalam text Processing yaitu bag 0f words dan stop word filtering, kali ini kita akan mempelajari suatu teknik untuk menghitung bobot suatu kata terhadap suatu dokumen dari sekumpulan dokumen.

Teknik ini dikenal sebagai Term Frequency – Inverse Document Frequency, atau Biasa disingkat TF-IDF,teknik ini juga cukup umum ditemui dalam bidang information retrieval,

TF-IDF merupakan salah satu metode statistik yang digunakan untuk mengukur seberapa penting suatu kata terhadap suatu dokumen tertentu dari sekumpulan dokumen atau corpus.

Bagi kalian tertarik untuk mempelajari lebih lanjut berikut adalah halaman Wikipedia yang bisa dijadikan referensi.

Refrensi :

- https://en.wikipedia.org/wiki/Tf%E2%80%93idf
- https://scikit-learn.org/stable/modules/feature_extraction.html#test-feature-extraction

TF-IDF ini pada dasarnya melibatkan Perkalian antara dua nilai, yaitu nilai TF atau Term Frequency dengan nilai IDF atau Inverse Document Frequency, disini kita akan pahami kedua bagian tersebut.

◆ Term Frequency (TF)

Terdapat beberapa opsi formula yang bisa digunakan untuk merepresentasikan term frekuensi, formula yang paling sederhana adalah dengan menghitung jumlah kemunculan suatu term atau suatu kata pada suatu dokumen,formula ini seringkali dikenal sebagai row count, hanya saja implementasi term frekuensi pada scikit learn tidak mengadopsi formula semacam ini.

Term frekuensi yang diimplementasikan pada scikit learn menerapkan formula sebagai berikut

- term frequency adjusted for document length

Di mana term frekuensi diekspresikan sebagai hasil pembagian antara jumlah kemunculan suatu term pada dokumen dengan total jumlah kata yang terkandung dalam dokumen tersebut.

$$\frac{f_{t,d} \Big/ \sum f_{t,d}}{t^1 \varepsilon d}$$

Ini merupakan ekspresinya secara matematis, ini merupakan formula dari term frekuensi yang juga diadopsi pada scikit learn.

◆ Inverse Document Frequency (IDF)

Sama halnya dengan term frekuensi, disini juga terdapat beberapa opsi formula yang bisa digunakan untuk merepresentasikan inverse decoment frequency.

Formula yang paling sederhana adalah dengan menghitung nilai log dari pembagian antara total jumlah dokumen dalam suatu corpus dengan jumlah dokumen yang mengandung term tertentu.

$$\log \frac{N}{n^t} = -\log \frac{n^t}{N}$$

Ini adalah ekspresinya secara matematis, hanya saja implementasi inverse document frekuensi pada scikit learn tidak tercover dalam halaman Wikipedia ini, di sini kita perlu pelajari documentation dari scikit learn.

$$idf\,(t) = \log \frac{1+n}{1+df\,(t)} + 1,$$

Ini merupakan formula yang yang pada scikit learn untuk merepresentasikan inverse decoment frequency.

n disini merepresentasikan total jumlah dokumen dalam suatu corpus, dan **df(t)** disini merepresentasikan jumlah dokumen yang mengandung term tertentu.

Selain itu, scikit learn juga menerapkan L2 normalization pada kalkulasi pembuatan DF-IDF.

Dataset

Pertama-tama kita akan siapkan terlebih dahulu suatu dataset text, untuk kasus kita kali ini data set berupa sekumpulan kalimat pendek, dataset text ini juga seringkali dikenal dengan istilah corpus.

corpus=[

```
        'the house had a tiny little mouse',
        'the cat saw the mouse',
        'the mouse ran away from the house'
        'the cat finally ate the mouse',
        'the end of the mouse story'
```

]

corpus

kita coba eksekusi kode nya

```
[
'the house had a tiny little mouse',
'the cat saw the mouse',
'the mouse ran away from the housethe cat finally are the mouse',
'the end of the mouse story'
]
```

Courpus yang kita miliki korpus ini terdiri dari lima buah kalimat pendek,kita akan gunakan corpus ini sebagai studi kasus.

◆ TF-IDF Weights dengan TfidfVectorizer

Selanjutnya kita akan menerapkan TF-IDF pada dataset corpus yang kita miliki pada scikit learn, TF-IDF dapat ditetapkan dengan memanfaatkan modul **TfidfVectorizer**.

Berikut akan kita demokan prosesnya

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(stop_words='english')
response = vectorizer.fit_transform(corpus)
print(response)
```

Pertama-tama kita akan import dulu module nya, **from sklearn.feature_extraction.text import** TfidfVectorizer, berikutnya kita akan bentuk objek dari class **TfidfVectorizer** ini, dan disini kita akan menyertakan parameter **stop_words** yang kita beri nilai **English,** untuk selanjutnya objek **TfidfVectorizer** ini tidak tampung ke dalam variabel **vectorizer,** tahapan berikutnya kita akan memanggil method fit transform dari objek **vectorizer** ini, dan akan kita terapkan pada corpus yang kita miliki, lalu hasil transform nya kita akan tampung ke dalam variabel **respons**, untuk selanjutnya kita cetak layar.

Kita coba eksekusi kode nya

```
from sklearn.feature_extraction.text import TfidfVectorizer

corpus=[
        'the house had a tiny little mouse',
        'the cat saw the mouse',
        'the mouse ran away from the house'
        'the cat finally ate the mouse',
        'the end of the mouse story'
        ]

vectorizer = TfidfVectorizer(stop_words='english')
response = vectorizer.fit_transform(corpus)
print(response)
```

dan ini hasil nya

```
  (0, 7)    0.2884767487500274
  (0, 6)    0.5528053199908667
  (0, 11)   0.5528053199908667
  (0, 4)    0.5528053199908667
```

```
  (1, 9)     0.7266414923403136
  (1, 1)     0.5728924988566925
  (1, 7)     0.3791916749655464
  (2, 3)     0.41845521457572643
  (2, 5)     0.41845521457572643
  (2, 0)     0.41845521457572643
  (2, 8)     0.41845521457572643
  (2, 1)     0.32991489760073706
  (2, 7)     0.436734580630646
  (3, 10)    0.6633846138519129
  (3, 2)     0.6633846138519129
  (3, 7)     0.34618161159873423
```

Mungkin kalian mulai mengalami kebingungan terkait bagaimana cara mengartikan setiap angka yang ada di sini, kita akan pelajari satu per satu.

Pertama adalah yang ada di sisi paling kiri dulu, angka-angka yang ada di sini ini merepresentasikan indeks dari corpus kita, di sini ada 0,1,2,dan 3, index ke-0 ini merepresentasikan Kalimat pertama dari corpus kita, index-1 berarti merepresentasikan kalimat kedua dari corpus, index-2 ini merepresentasikan kalimat ketiga dari corpus kita dan seterusnya.

Lalu berikutnya adalah angka pada kolom yang kedua, angka ini merepresentasikan indeks dari features name yang dihasilkan dari bag of words kita.

Untuk lebih jelasnya disini kita bisa panggil vectorizer.get_feature_names_out()

```
print(vectorizer.get_feature_names_out())
```

ini merupakan kumpulan token yang sudah dieliminir stpwords nya

```
['away'
'cat'
'end'
'finally'
'house'
'housethe'
'little'
'mouse'
'ran'
 'saw'
'story'
'tiny']
```

Token-token ini diurutkan berdasarkan urutan alfabetic, kalau kita perhatikan lagi di atas, untuk kalimat pada indeks ke-0, di sini ada 7,6,11 dan 5, artinya kalimat pada indeks ke-0 ini mengandung token indeks ke-7,berarti pada kalimat tersebut mengandung kata Mouse.

Selanjutnya juga mengandung token pada indeks ke-6, dan token pada indeks ke-6 adalah little, kalimat pertama juga mengandung kata little, berikutnya juga mengandung token pada indeks ke-5 dan token pada index ke-5 adalah house, pada kalimat yang pertama ini juga mengandung kata house, berikutnya disini juga mengandung token pada indeks ke-11, token pada indeks ke-11 adalah tiny, disini juga terdapat kata tiny pada kalimat pertama.

Jadi deret bilangan yang ini

(**0**,

```
(0,
(0,
(0,
(1,
(1,
(1,
(2,
(2,
(2,
(2,
(2,
(2,
(3,
(3,
(3,
```

Ini merepresentasikan indeks dari dokumen pada corpus kita.

sedangkan yang ada ini

7)

```
6)
11)
4)
9)
1)
7)
3)
5)
0)
```

```
8)
1)
7)
10)
2)
(3,
```

ini merepresentasikan indeks dari token yang terdapat dalam kalimat tersebut.

Berikutnya adalah sekumpulan angka yang ini

```
0.2884767487500274
0.5528053199908667
0.5528053199908667
0.5528053199908667
0.7266414923403136
0.5728924988566925
0.3791916749655464
0.41845521457572643
0.41845521457572643
0.41845521457572643
0.41845521457572643
0.32991489760073706
0.436734580630646
0.6633846138519129
0.6633846138519129
0.34618161159873423
```

sekumpulan angka yang ada di sini merepresentasikan bobot dari DF-IDF hasil kalkulasi yang dilakukan oleh **TfidfVectorizer.**

Untuk lebih jelasnya kita akan coba transformasikan hasil responsnya ke dalam bentuk array, disini caranya mudah kita cukup panggil saja **response.todense().**

```
print(response.todense())
```

dan ini hasil nya

```
[[0.         0.         0.         0.         0.55280532 0.
  0.55280532 0.28847675 0.         0.         0.         0.55280532]
 [0.         0.5728925  0.         0.         0.         0.
  0.37919167 0.         0.72664149 0.         0.        ]
 [0.41845521 0.3299149  0.         0.41845521 0.         0.41845521
  0.43673458 0.41845521 0.         0.         0.        ]
 [0.         0.         0.66338461 0.         0.         0.
0.34618161 0.         0.         0.66338461 0.        ]]
```

disini akan terbentuk array 2 dimensi, dimana setiap row nya ini akan merepresentasikan setiap kalimat atau setiap dokumen pada corpus.

kita agar tampilannya bisa lebih manis kita akan coba konversikan ke dalam format pandas dataframe.

kalau kita perhatikan di sini bentuk array nya adalah array 2 dimensi, dimana setiap barisnya ini merepresentasikan kalimatnya, hanya saja di sini dalam bentuk Pandas dataframe kita akan tampilkan kalimatnya dalam bentuk kolom, sedangkan barisnya akan digunakan untuk merepresentasikan setiap tokennya, oleh karenanya di sini kita butuhkan proses transpose terhadap array ini .

```
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd

corpus=[
        'the house had a tiny little mouse',
        'the cat saw the mouse',
        'the mouse ran away from the house'
        'the cat finally ate the mouse',
        'the end of the mouse story'
        ]

vectorizer = TfidfVectorizer(stop_words='english')
response = vectorizer.fit_transform(corpus)

df = pd.DataFrame(response.todense().T,
      index=vectorizer.get_feature_names_out(),
      columns=[f'D{i+1}' for i in range(len(corpus))])

print(df)
```

caranya Kita tinggal panggil saja **response.todense().T,** T disini artinya adalah kita ingin melakukan proses transpose.

kita coba eksekusi kode nya

dan ini hasil nya

	D1	D2	D3	D4
ate	0.000000	0.000000	0.386021	0.000000
away	0.000000	0.000000	0.418455	0.000000
cat	0.000000	0.572892	0.329915	0.000000
end	0.000000	0.000000	0.000000	0.663385
finally	0.000000	0.000000	0.418455	0.000000
house	0.552805	0.000000	0.000000	0.000000
housethe	0.000000	0.000000	0.418455	0.000000
little	0.552805	0.000000	0.000000	0.000000
mouse	0.288477	0.379192	0.436735	0.346182
ran	0.000000	0.000000	0.418455	0.000000
saw	0.000000	0.726641	0.000000	0.000000
story	0.000000	0.000000	0.000000	0.663385
tiny	0.552805	0.000000	0.000000	0.000000

Disini informasinya menjadi lebih jelas,D1 ini merupakan dokumen pertama,D2 berarti dokumen ke dua, dan seterusnya.

Baris pertama ini merepresentasikan token ate, baris ke dua merepresentasikan token away, baris ketiga ini merepresentasikan token cat,dan seterusnya.

Berikutnya cara membacanya seperti ini,untuk kata cat, ini terdapat dalam dokumen ke-2 dan juga terdapat dalam dokumen ke-4, hanya saja kata cat ini memiliki bobot yang lebih tinggi pada dokumen yang ke-2 bila dibandingkan pada dokumen yang ke-4, nilai pembobotan yang ada di sini merupakan nilai pembobotan yang sudah dinormalisasi dengan menggunakan L2 normalization, nilai terkecil nya adalah **0.0**,dan nilai terbesarnya adalah **1,0,** semakin tinggi bobot suatu kata terhadap suatu dokumen, mengindikasikan bahwa kata tersebut semakin layak untuk digunakan sebagai keyword atau kata pencarian terhadap dokumen tersebut.

◆ Bab 12 : Logistic Regression pada Binary Classification Task

Sebelumnya kita sudah mengenal salah satu model machine learning yang dapat diterapkan pada classification tasks, kali ini kita akan mempelajari model machine learning lain yang juga bisa kita terapkan pada classification tasks, khususnya pada binary calcification tasks,model machine learning yang akan kita pelajari kali ini adalah Logistic regression.

Disini sebagian dari kalian mungkin bertanya-tanya nama modelnya adalah Logistic regression, tetapi justru penerapannya pada classification tasks, dan bukannya pada regression tasks, kalian tidak perlu khawatir karena penjelasannya akan kita sampaikan dalam sesi pembelajaran ini.

Bagi kalian yang tertarik untuk mempelajari Logistic regression lebih lanjut berikut adalah halaman Wikipedia yang bisa dijadikan referensi.

refrensi : https://en.wikipedia.org/wiki/logistic_regression

◆ Formula Dasar

Berikutnya kita akan memahami formula dasar yang membentuk Logistic regression serta hubungannya dengan simpel dan multiple linier regression yang sudah kita pelajari sebelumnya, disini kalian gak perlu khawatir karena penjelasan matematis yang kita sampaikan akan tetap ramah bagi pemula.

kita akan mulai dengan mereview kembali formula dari simple linear regression dengan simple linear regression.

Simple Linear Regression

Dengan simple linear regression kita hanya dapat menyatakan satu feature saja untuk melakukan estimasi nilai,

Dan ini merupakan formula dari simple linear regression.

$$y = a + ßx$$

Dimana Alfa ini merupakan nilai intercept nya dan Beta merupakan nilai slope nya.

Formula semacam ini juga bisa dituliskan ulang dengan cara seperti ini

$$g(x) = a + ßx$$

Dimana y ini atau variabel ini bisa digantikan atau direpresentasikan dengan fungsi g yang menerima inputan x sebagai parameter, yang perlu diperhatikan disini adalah x-nya adalah x kecil yang merepresentasikan 1 buah features saja.

Multiple Linear Regression

Selanjutnya kita juga akan mereview kembali formula dasar dari multiple linear regression, berbeda dengan simple linear regression yang hanya menyertakan satu features saja, pada multiple linear regression kita dapat menyertakan lebih dari satu features untuk melakukan estimasi nilai.

Dan ini merupakan formula yang pernah kita pelajari sebelum nya.

$$y = a + ß_1 x_1 + ß_2 x_2 + ... + ß_n x_n$$

Kalau kalian perhatikan disini,faktor pembeda nya adalah, kalau pada simple linear regression nilai x ini berupa x tunggal atau feature tunggal, sedangkan pada multiple linear regression, x ini bisa banyak ,ada x1, x2, dan seterusnya, dimana x1 ini akan merepresentasikan features pertama, x2 merepresentasikan feature kedua dan seterusnya.

Penulisan formula semacam ini juga bisa diekspresikan dengan cara yang lain, yaitu sebagai berikut

$$g(X) = a + ßX$$

Yang perlu digarisbawahi disini adalah, X-nya menggunakan X besar, karena merepresentasikan sekumpulan features dan bukan hanya features tunggal.

Kalau kita perhatikan di sini kedua formulanya sangat mirip sekali, yang membedakan hanya di X nya saja, pada simple linear regression menggunakan notasi x kecil karena hanya merepresentasikan satu nilai features, sedangkan pada multiple linear regression menggunakan X besar karena merepresentasikan sekumpulan nilai features.

Logistic Regression

Kita akan melangkah pada formula dasar pembentuk Logistic regression,dan ini formulanya.

$$g(X) = sigmoid(a + ßX)$$

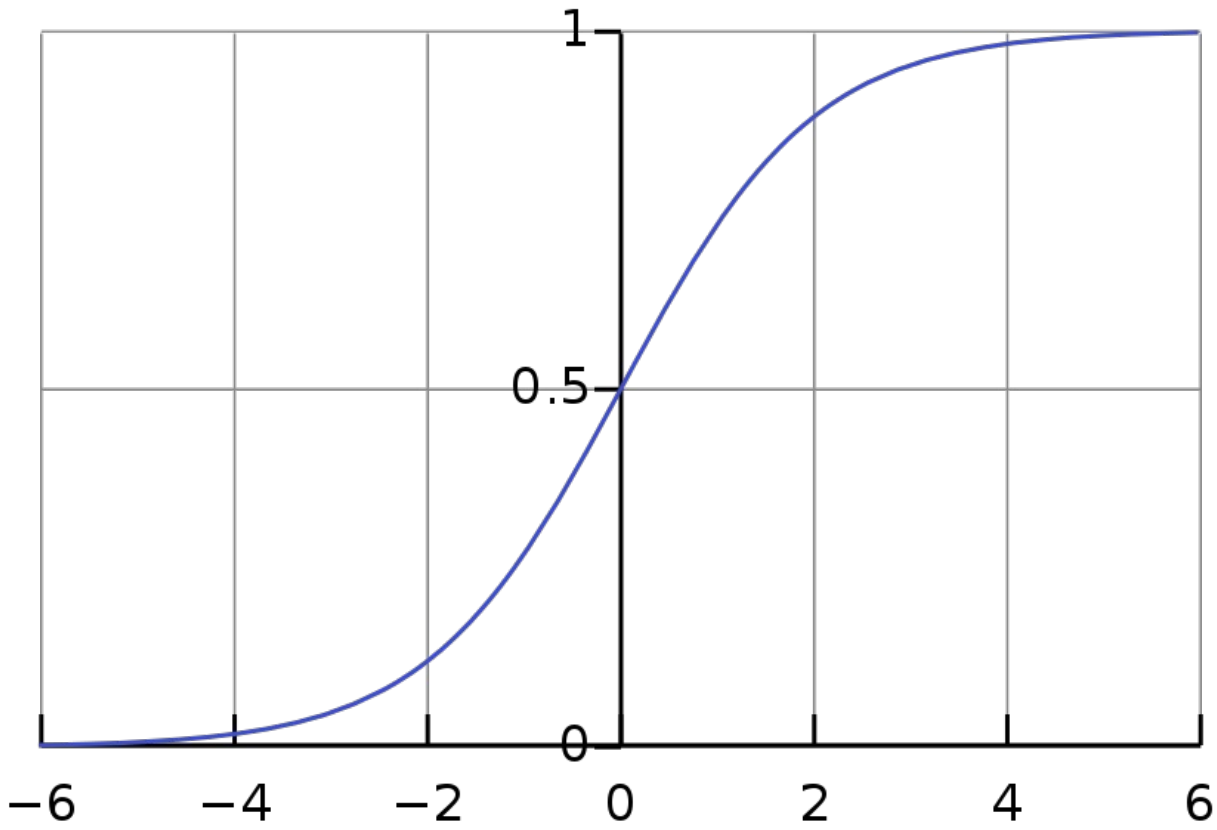
Fungsi sigmoid ini bisa di ekspresikan jika sebagai berikut

$$sigmoid(x) = \frac{1}{1 + \exp(-x)}$$

Fungsi sigmoid x ini sama dengan 1 dibagi 1 ditambah eksponen dari -x, nilai x ini akan digantikan dengan yang ada di **a+ßX.**

Disini bisa nampak dengan jelas bahwa formulanya mengadopsi persamaan linear dari linear regression yang pernah kita pelajari sebelumnya, hanya saja di sini persamaan linear tersebut akan dilewatkan ke dalam fungsi sigmoid, Alfa tambah beta X juga merupakan fungsi yang sama yang diterapkan pada multiple linear regression, yang membedakan disini adalah persamaan linear ini akan kita lewatkan kedalam fungsi sigmoid.

kalau kita visualisasikan maka akan membentuk kurva sigmoid sebagai berikut



Ini merupakan kurva sigmoid nya, kurva sigmoid ini akan berbentuk seperti huruf s, rentang nilai dari sumbu x pada ilustrasi di sini hanya sekedar contoh saja, karena rentang nilai ini akan sangat bergantung pada kasus yang sedang dihadapi,untuk kasus di sini rentang nilainya adalah -6 sampai 6, tetapi rentang nilai ini tidak harus -6 sampai 6, rentang nilainya bisa berapapun juga tergantung pada kasus ataupun dataset yang kita miliki.

Yang menarik dari kurva sigmoid ini adalah pada sumbu y nya, nilai sumbu y akan selalu berada pada rentang nilai 0 dan 1, dan rentang nilai ini akan berkorelasi dengan nilai binomial probability, bagi kalian yang masih awam dengan konsep binomial probability kita sangat menyarankan untuk mempelajarinya terlebih dahulu.

Sampai titik ini mestinya kalian sudah bisa memahami regression nya, karena pada dasarnya Logistic regression ini akan melakukan prediksi nilai yang berada dalam rentang 0 dan 1.

Pertanyaan berikutnya adalah, lalu bagaimana dengan penerapannya pada kasus binary classification task, sesuai dengan namanya dalam binary classification hanya terdapat 2 buah class saja, dan kedua kelas tersebut bisa kita encode sebagai 0 dan 1, dengan demikian ketika nilai probabilitasnya mendekati

0 maka akan diklasifikasikan sebagai class-0,sedangkan ketika nilai probability yang mendekati 1, maka akan diklasifikasikan sebagai class-1.

Selanjutnya kita akan pelajari tahapan untuk menerapkan Logistic regression model ini dengan scikit learn.

◆ Dataset SMS Spam Collection Data Set

Pertama-tama kita akan siapkan terlebih dahulu data set nya, untuk kasus kali ini, kita akan mengadopsi SMS Spam Collection dataset, ini merupakan open dataset yang ditawarkan oleh uci machine-learning.

Dan ini adalah link untuk mengakses dataset nya

Sumber data : https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection

Selanjutnya kita akan coba load dataset ini sebagai Pandas dataframe, berikut akan kita demokan proses.

```
import pandas as pd

df = pd.read_csv('./dataset/SMSSpamCollection',
                 sep='\t',
                 header=None,
                 names=['label', 'sms'])

print(df.head())
```

Pertama-tama kita akan impor dulu module nya, **import pandas as pd,** lalu berikutnya untuk kasus kita, kita sudah mendownload file SMS spam collection nya dari uci machine-learning repostory, dan saat ini posisi file-nya berada dalam direktori **dataset**,dengan nama file **SMSSpamcollection**,karena kita ingin meload dataset ini sebagai pandas dataframe, maka disini kita akan memanggil **pd.read_csv,**lalu kita akan arahkan ke posisi dari dataset nya, selanjutnya karena SMS dataset ini menggunakan separator nya berupa tab, maka disini kita spesifikasikan parameter sep nya adalah **\t**, berikutnya dataset juga tidak mengandung header maka kita set headernya **None**, dan selanjutnya terkait dengan penamaan kolomnya,dataset ini akan terdiri dari dua buah kolom, dan kolomnya akan kita beri nama **label** dan **sms**, lalu selanjutnya objek pandas dataframe yang terbentuk ini akan kita tampung ke dalam variabel **df,** untuk selanjutnya kita coba cetak layar lima baris pertama nya.

Dan ini hasil nya

	label	sms
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

Seperti bisa kita lihat di sini, dataset kita kali ini terdiri dari dua kolom, yaitu kolom label dan kolom sma, dimana kolom sms ini akan menampung sekumpulan text sms yang akan kita jadikan nilai features, sedangkan kolom label akan berisi klasifikasi dari setiap text sms nya.

Disini terdapat dua nilai label yaitu ham dan spam,text sms yang dikategorikan sebagai spam akan memiliki label spam, sedangkan text sms yang dikategorikan sebagai bukan spam akan memiliki label ham.

Dalam sesi pembelajaran ini kita akan memanfaatkan Logistic regression untuk melakukan prediksi apakah suatu text SMS termasuk dalam kategori spam atau ham, tasks semacam ini juga dikenal dengan istilah binary classification tasks, karena disini hanya terdapat dua class saja yaitu ham dan spam.

Selanjutnya disini kita juga akan coba cari tahu proporsi antara data ham dan data spam dalam dataset ini, caranya disini kita akan Panggil **df['label'].value_counts()**,karena kita mau hitung berapa banyak data ham nya dan berapa banyak data spam nya.

Kita coba eksekusi

```
print(df['label'].value_counts())
```

dan ini hasil nya

```
ham     4825
spam     747
```

Dari sini bisa kita ketahui bahwa jumlah data dengan label ham ternyata jauh lebih banyak bila dibandingkan dengan jumlah data berlabel spam, kondisi semacam ini juga dikenal dengan istilah inbalance dataset.

Dalam beberapa sesi selanjutnya kita akan membahas mekanisme untuk menangani kondisi semacam ini, hanya saja untuk sesi pembelajaran kali ini kita tidak akan melakukan penanganan khusus terhadap inbalance dataset ini.

Training & Testing Dataset

Selanjut nya kita akan bagi dataset nya ke dalam dua bagian yaitu training dan testing set, disini pertama-tama kita akan kelompokkan terlebih dahulu features dan targetnya, karena targetnya berupa label atau string yang terdiri dari dua nilai yaitu ham dan spam, maka disini kita akan memanfaatkan LabelBinarizer untuk melakukan konversi menjadi nilai biner 0 dan 1.

Berikut akan kita demokan prosesnya

```
from sklearn.preprocessing import LabelBinarizer

X = df['sms'].values
y = df['label'].values

lb = LabelBinarizer()
y = lb.fit_transform(y).ravel()
print(lb.classes_)
```

Pertama-tama kita impor dulu modulnya, **from sklearn.preprocessing import** LabelBinarizer, selanjutnya sekumpulan nilai dari kolom SMS akan kita tampung ke dalam variabel **X,** dan sekumpulan nilai dari kolom label akan kita tampung ke dalam variabel **y,** lalu berikutnya kita akan bentuk objek dari class **LabelBinarizer**, dan objek nya akan kita tampung ke dalam variabel lb, untuk selanjutnya kita akan memanggil fit transform dari objek **LabelBinarizer** ini dan akan kita kenakan terhadap nilai yang ditampung dalam variabel **y,** hal lain yang perlu kita perhatikan adalah nilai dari variabel **y** setelah dikenalkan fit transform dari **LabelBinarizer** akan menghasilkan multidimensional array, dan disini kita mau konversikan kembali menjadi array 1 dimensi, oleh karenanya kita bisa memanggil method **.ravel()**,dan hasil nya akan kita tampung ke dalam variabel **y,** lalu berikutnya kita juga mau mencoba mencari tahu proses **LabelBinarizer** nya, oleh karenanya di sini kita Panggil **lb.classes_.**

dan ini hasil nya kalau kita eksekusi kode nya

```
['ham' 'spam']
```

Bisa nampak disini class yang di handle oleh **LabelBinarizer** adalah ham dan spam,karena ham menempati posisi pertama maka ham ini akan di encode sebagai 0, sedangkan spam nya di encode sebagai 1, atau dengan kata lain class 0 ini merupakan class ham, sedangkan class 1 ini merupakan class spam.

Selanjutnya kita akan pecah dataset ini menjadi dua bagian dengan menggunakan modul train_test_split, berikut akan kita demokan prosesnya

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.25,
                                                    random_state=0)

print(X_train, '\n')
print(y_train)
```

Pertama-tama kita impor dulu modulnya **from sklearn.model_selection import** train_test_split, selanjutnya kita tinggal panggil saja **train_test_split**, lalu disertai dengan sekumpulan nilai features dan diikuti dengan sekumpulan nilai label nya, disini kita juga spesifikasikan **test_size**, yaitu **0,25** artinya porsi untuk testing dataset nya adalah 25%, sedangkan 70% nya akan digunakan sebagai training set, lalu kita juga set **random_state** sebagai 0, pemanggilan **train_test_split** seperti ini akan menghasilkan 4 kumpulan nilai, dan ke-4 kumpulan nilai ini akan kita tampung ke dalam variabel **X_train**, **X_test**, **y_train**, dan **y_test**,berikutnya kita akan coba tampilkan **X_train** dan **y_train** nya.

Kita coba eksekusi kode nya

```
import pandas as pd
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split

df = pd.read_csv('./dataset/SMSSpamCollection',
                 sep='\t',
                 header=None,
                 names=['label', 'sms'])


X = df['sms'].values
y = df['label'].values

lb = LabelBinarizer()
y = lb.fit_transform(y).ravel()
```

```
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.25,
                                                    random_state=0)
print(X_train, '\n')
print(y_train)
```

dan ini hasil nya

ini merupakan training set nya, X_train ini merupakan nilai features untuk training set nya.

```
['Its going good...no problem..but still need little experience to understand american
customer voice...'
 'U have a secret admirer. REVEAL who thinks U R So special. Call 09065174042. To opt out
Reply REVEAL STOP. 1.50 per msg recd. Cust care 07821230901'
 'Ok...' ...
 "For ur chance to win a £250 cash every wk TXT: ACTION to 80608. T's&C's
www.movietrivia.tv custcare 08712405022, 1x150p/wk"
 'R U &SAM P IN EACHOTHER. IF WE MEET WE CAN GO 2 MY HOUSE'
 'Mm feeling sleepy. today itself i shall get that dear']
```

Sedangkan y_train nya merupakan label untuk training set nya.

```
[0 1 0 ... 1 0 0]
```

Untuk features nya akan berisi data text SMS, Sedangkan untuk label akan berisi class yang bernilai 0 dan 1, kenapa 0 dan 1?, karena sebelumnya kita sudah menggunakan **LabelBinarizer**, dimana nilai 0 pada label ini akan berkorelasi dengan ham dan nilai 1 nya akan berkorelasi dengan spam .

◆ Feature Extraction dengan TF-IDF

Karena dataset yang kita miliki adalah dataset text, disini kita akan menerapkan TF-IDF, seperti telah kita pelajari sebelumnya, pada scikit learn TF-IDF dapat diterapkan dengan memanfaatkan modul **TfidfVectorizer**.

Berikut akan kita demokan prosesnya.

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words='english')

X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
print(X_train_tfidf)
```

Pertama-tama kita impor dulu module nya, **from sklearn.feature_extraction.text import** TfidfVectorizer, selanjutnya kita akan bentuk objek **TfidfVectorizer**, dengan menyatakan parameter **stop_words** yang kita beri nilai **english,** dan objek dari **TfidfVectorizer,** ini akan kita tampung ke dalam variabel **vectorizer,** lalu selanjutnya kita akan memanggil fungsi fit transform dari objek **vectorizer** ini, dan akan kita kenakan ke dalam variabel **X_train**, dan hasil transformasinya akan kita tampung ke dalam variabel **X_train_tfidf**, atau dengan kata lain, variabel **X_train_tfidf** ini akan menampung sekumpulan nilai **X_train** yang sudah ditransformasikan menjadi TF-IDF metrics, lalu berikutnya objek **vectorizer** yang sama juga akan kita gunakan untuk melakukan proses transformasi terhadap **X_test,** dan hasil transformasinya akan kita tampung ke dalam variabel **X_test_tfidf**, disini kita juga akan coba cetak ke layar nilai di tampung dalam variabel **X_train_tfidf** ini.

Kita coba eksekusi kode nya

```
import pandas as pd
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

df = pd.read_csv('./dataset/SMSSpamCollection',
                 sep='\t',
                 header=None,
                 names=['label', 'sms'])


X = df['sms'].values
y = df['label'].values

lb = LabelBinarizer()
y = lb.fit_transform(y).ravel()

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.25,
                                                    random_state=0)

vectorizer = TfidfVectorizer(stop_words='english')

X_train_tfidf = vectorizer.fit_transform(X_train)
```

```
X_test_tfidf = vectorizer.transform(X_test)
print(X_train_tfidf)
```

dan ini hasil nya

```
  (0, 6903)  0.3591386422223876
  (0, 2006)  0.2898082580285881
  (0, 900)   0.4114867709157148
  (0, 6739)  0.3546359942830148
  (0, 2554)  0.3825278811525034
  (0, 3926)  0.3126721340000456
  (0, 4453)  0.2297719954323795
  (0, 5123)  0.308974289326673
  (0, 3007)  0.21421364306658514
  (0, 2997)  0.23173982975834367
  (1, 36)    0.28902673040368515
  (1, 1548)  0.18167737976542422
  (1, 2003)  0.2711077935907125
  (1, 5301)  0.2711077935907125
  (1, 4358)  0.17341410292348694
  (1, 532)   0.20186022353306565
  (1, 6131)  0.16142609035094446
  (1, 5394)  0.16464655071448758
  (1, 4677)  0.24039776602646504
  (1, 216)   0.28902673040368515
  (1, 6013)  0.20089911182610476
  (1, 6472)  0.24039776602646504
  (1, 5441)  0.5009783758205715
  (1, 799)   0.25048918791028574
  (1, 5642)  0.24344998442301355
  :   :
  (4176, 343)       0.2811068572055718
  (4176, 107)       0.29968668460649284
  (4176, 2004)      0.25589560236817055
  (4176, 4350)      0.29968668460649284
  (4176, 637)       0.29968668460649284
  (4176, 7114)      0.4512018097459442
  (4176, 365)       0.2388005587702937
  (4176, 1612)      0.21138425595332702
  (4176, 779)       0.2811068572055718
  (4176, 7195)      0.17892283441772988
  (4176, 1569)      0.18895085073406012
  (4176, 7083)      0.19523751585154273
  (4176, 6684)      0.22114159453800114
  (4176, 6693)      0.16491299289150899
  (4176, 6792)      0.1407604617250961
  (4177, 2362)      0.6158854885899457
  (4177, 5565)      0.5506066649743346
```

```
  (4177, 4177)      0.3636187667918345
  (4177, 3319)      0.43046342221720785
  (4178, 5883)      0.548491137555895
  (4178, 4279)      0.4530624713751054
  (4178, 5720)      0.3963527249882828
  (4178, 6555)      0.2897850627168302
  (4178, 2641)      0.3993042639531407
  (4178, 2068)      0.3055766821331892
```

Ini merupakan nilai yang ditampung dalam variabel **X_train_tfidf**,bisa nampak di sini, nilai yang ditampung bukan lagi berupa text SMS melainkan sudah berupa vector TF-IDF.

◆ Binary Classicifation dengan Logistic Regression

Selanjutnya menerapkan Logistic regression untuk melakukan klasifikasi ham dan spam pada dataset SMS yang kita miliki.

berikut akan kita demokan prosesnya

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X_train_tfidf, y_train)
y_pred = model.predict(X_test_tfidf)

for pred, sms in zip(y_pred[:5], X_test[:5]):
      print(f'PRED: {pred} - SMS: {sms}\n')
```

disini pertama-tama kita akan impor dulu class dari model **LogisticRegression** nya, di sini kita Panggil f**from sklearn.linear_model import** LogisticRegression,selanjutnya kita akan bentuk objek dari class Logistic regression ini, dan objek yang terbentuk akan kita tampung setelah variabel **model**,tahapan selanjutnya object model ini akan kita training dengan memanggil method Fit dan menyertakan **X_train_tfidf** dan **y_train**, sebagai parameter untuk training, setelah modelnya ditraining maka model ini akan menjadi train model, dan train model ini akan kita gunakan untuk melakukan prediksi,dan sekumpulan nilai yang mau kita prediksi adalah label berdasarkan data-data features yang ditampung dalam **X_test_tfidf**, hasil prediksinya kita akan tampung ke dalam variabel **y_pred**, lalu berikutnya disini kita juga akan menampilkan suatu proses looping sederhana,dimana kita akan menampilkan lima hasil prediksi pertama.

Kita coba eksekusi kode nya

```
import pandas as pd
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression

df = pd.read_csv('./dataset/SMSSpamCollection',
                 sep='\t',
                 header=None,
                 names=['label', 'sms'])


X = df['sms'].values
y = df['label'].values

lb = LabelBinarizer()
y = lb.fit_transform(y).ravel()

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.25,
                                                    random_state=0)

vectorizer = TfidfVectorizer(stop_words='english')

X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

model = LogisticRegression()
model.fit(X_train_tfidf, y_train)
y_pred = model.predict(X_test_tfidf)

for pred, sms in zip(y_pred[:5], X_test[:5]):
      print(f'PRED: {pred} - SMS: {sms}\n')
```

dan ini hasil nya

```
PRED: 0 - SMS: Storming msg: Wen u lift d phne, u say "HELLO" Do u knw wt is d real
meaning of HELLO?? . . . It's d name of a girl..! . . . Yes.. And u knw who is dat girl??
"Margaret Hello" She is d girlfrnd f Grahmbell who invnted telphone... . . . . Moral:One
can 4get d name of a person, bt not his girlfrnd... G o o d n i g h t . . .@

PRED: 0 - SMS: <Forwarded from 448712404000>Please CALL 08712404000 immediately as there
is an urgent message waiting for you.

PRED: 0 - SMS: And also I've sorta blown him off a couple times recently so id rather not
text him out of the blue looking for weed
```

```
PRED: 0 - SMS: Sir Goodmorning, Once free call me.

PRED: 0 - SMS: All will come alive.better correct any good looking figure there itself..
```

Bisa nampak di sini, **SMS** ini merupakan text sms nya, lalu **PRED** ini merupakan hasil prediksinya.

◆ Evaluation Metrics pada Binary Classification

Selanjutnya kita akan mempelajari beberapa mertics evaluasi yang kita terapkan untuk kasus classification, atau lebih spesifiknya untuk kasus binary classification.

Di sini ada 6 mettrics yang akan kita pelajari yaitu:

- Confusion Matrix
- Accuracy
- Precission & Recall
- F1 Score
- ROC

Kalau kalian perhatikan beberapa diantara Matrix ini pernah kita pelajari sebelumnya, pada sesi pembelajaran terkait classification dengan KNN, hanya saja kali ini kita akan mereview kembali dan juga tambahkan beberapa Matrix yang memang belum sempat dibahas sebelumnya.

Terminologi Dasar

Terkait meatix evaluasi pada classification tasks, terdapat 4 terminologi dasar atau istilah dasar yang perlu dipahami terlebih dahulu.

Dua istilah pertama adalah

- True Positive(TP)
- True Negative(TN)

Keduanya merepresentasikan hasil prediksi atau klasifikasi yang benar, **true positive** berarti sesuatu yang bernilai positif telah dengan tepat diprediksi sebagai positif oleh model, sedangkan **true negatif** berarti sesuatu yang bernilai negatif telah dengan tepat diprediksi sebagai negatif oleh model.

Kalau kita mengacu pada contoh kasus di sini, **true positive** berarti model sudah dengan tepat memprediksi data spam sebagai spam, dan data ham sebagai ham, sedangkan **true negative** berarti modal sudah dengan tepat memprediksi data ham sebagai bukan spam, dan data spam sebagai bukan ham.

Dua stilah berikutnya adalah :

- False Positive(FP)
- False Negative(FN)

Keduanya merepresentasikan hasil prediksi atau klasifikasi yang salah, **False Positive** berarti sesuatu yang bernilai negatif telah keliru diprediksi sebagai positif oleh model, sedangkan **False Negative** berarti sesuatu yang bernilai positif telah keliru diprediksi sebagai negatif oleh model.

Kalau kita mengacu pada contoh kasus di sini, **False Positive** berarti modelsudah dengan keliru memprediksi data ham sebagai spam, dan data spam sebagai ham, sedengankan **False Negative** berarti suatu model sudah dengan keliru memprediksi data spam sebagai bukan spam dan data ham sebagai bukan ham.

1. Confusion Matrix

Kita akan mulai mempelajari Matrix evaluation pertama yaitu confusion matrix, confusion matrix seringkali juga dikenal sebagai error metric, confusion matrix berperan untuk menampilkan nilai true negatif,false positif, false negatif dan true positif.

Bagi kalian tertarik untuk mempelajari confusion matrix lebih lanjut berikut adalah halaman Wikipedia yang bisa dijadikan referensi

Refrensi : https://en.wikipedia.org/wiki/Confusion_matrix

Selanjutnya kita akan demokan tahapan pemanfaatan confusion matriks pada scikit learn.

```
from sklearn.metrics import confusion_matrix

matrix = confusion_matrix(y_test, y_pred)
matrix
```

Pertama-tama kita akan impor dulu modulnya **from sklearn.metrics import** confusion_matrix,berikutnya kita akan Panggil fungsi **confusion_matrix**, lalu kita akan sertakan dua parameter yaitu **y_test** dan **y_pred**, untuk selanjutnya kita tampung hasilnya ke dalam variabel **matrix**, lalu nilai dari variabel **matriks** ini akan coba kita tampilkan ke layar.

dan ini hasil nya

```
[[1207    1]
 [  48  137]]
```

Mungkin kalian sudah mulai bertanya-tanya ini maksudnya apa sih, atau bagaimana cara menginterpretasikan hasil ini, untuk dapat menginterpretasikan hasil ini dengan lebih enak kita akan coba konversikan dulu menjadi array 1 dimensi.

```
tn, fp, fn, tp = matrix.ravel()

print(f'TN {tn}')
print(f'FP {fp}')
print(f'FN {fn}')
print(f'TP {tp}')
```

Di sini kita Panggil **matrix.ravel()**, dan nilainya akan kita tabung ke dalam 4 variabel yaitu **tn**, **fp**, **fn**, dan **tp**.

Kita coba eksekusi kode nya

```
import pandas as pd
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix

df = pd.read_csv('./dataset/SMSSpamCollection',
                 sep='\t',
                 header=None,
                 names=['label', 'sms'])


X = df['sms'].values
y = df['label'].values
```

```
lb = LabelBinarizer()
y = lb.fit_transform(y).ravel()

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.25,
                                                    random_state=0)

vectorizer = TfidfVectorizer(stop_words='english')

X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

model = LogisticRegression()
model.fit(X_train_tfidf, y_train)
y_pred = model.predict(X_test_tfidf)
matrix = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = matrix.ravel()

print(f'TN {tn}')
print(f'FP {fp}')
print(f'FN {fn}')
print(f'TP {tp}')
```

dan ini hasil nya

```
TN 1207
FP 1
FN 48
TP 137
```

Yang perlu diperhatikan disini adalah urutannya, urutannya jangan sampai keliru.

Selain direpresentasikan dalam bentuk array semacam ini, confusion matrix juga dapat kita visualisasikan.

berikut akan kita demokan prosesnya

```
import matplotlib.pyplot as plt

plt.matshow(matrix)
plt.colorbar()

plt.title('Confusion Matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show
```

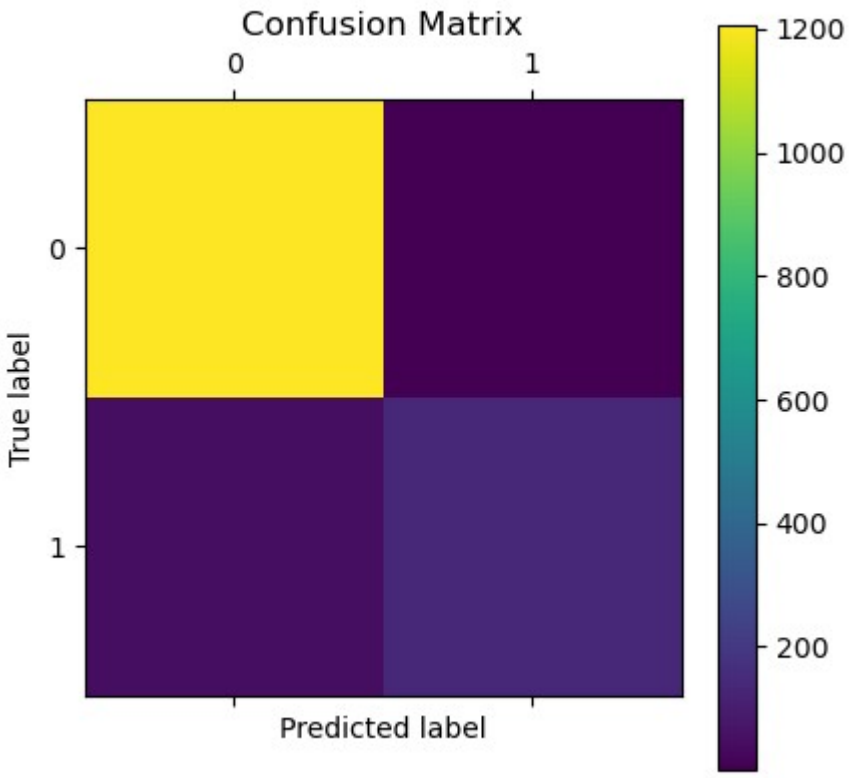
Di sini untuk memvisualisasikan confusion matrix pertama-tama kita akan import dulu module matplotlib nya, **import matplotlib.pyplot as plt,** lalu berikutnya kita akan panggil **plt.matshow**,dan kita sertakan variabel matrix sebagai parameternya.

Kita coba eksekusi kode nya

```
import pandas as pd
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

df = pd.read_csv('./dataset/SMSSpamCollection',
                 sep='\t',
                 header=None,
                 names=['label', 'sms'])


X = df['sms'].values
y = df['label'].values

lb = LabelBinarizer()
y = lb.fit_transform(y).ravel()

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.25,
                                                    random_state=0)

vectorizer = TfidfVectorizer(stop_words='english')

X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

model = LogisticRegression()
model.fit(X_train_tfidf, y_train)
y_pred = model.predict(X_test_tfidf)
matrix = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = matrix.ravel()
plt.matshow(matrix)
plt.colorbar()

plt.title('Confusion Matrix')
plt.ylabel('True label')
```

```
plt.xlabel('Predicted label')
plt.show()
```

dan ini hasil nya



Dsini Kita juga memanggil **plt.colorbar**,**plt.colorbar** ini untuk memunculkan colorbar yang berada di sebelah kanan, lalu pada floating nya juga kita sertakan judul atau title,kita beri juga y label dan X labelnya.

Bisa nampak di sini, ini merupakan representasi visual dari confusion matrix, di sisi sebelah kiri atas atau yang berwarna kunig,ini merepresentasikan true negatif,kemudian di sebelah nya ini merepresentasikan false positif, yang dibawah warna kuning ini merepresentasikan nilai false negatif,dan yang terakhir ini merepresentasikan nilai true positif.

Disini yang perlu diperhatikan adalah sumbu-x ini merepresentasikan predicted label, sedangkan sumbu-y ini merepresentasikan true label, dan disini juga dilengkapi dengan nilai 0 dan 1 baik pada sumbu-y maupun pada sumbu-x nya.

Nilai 0 pada sumbu-y ini merepresentasikan nilai 0 yang sebenarnya, atau nilai 0 yang memang diharapkan, lalu kita akan bandingkan dengan predicted label nya, di sini predicted label nya adalah 0, artinya kita mengharapkan nilai 0 dan hasil prediksinya adalah 0, oleh karenanya ini merupakan representasi dari area true negatif, karena nilai 0 ini akan berkorelasi dengan data bukan spam, faktanya adalah bukan spam dan diprediksi sebagai bukan spam, atau faktanya adalah ham dan diprediksi sebagai ham, dan yang warna kuning ini merepresentasikan nilai true negatif.

Baik true negatif maupun true positif ini merupakan hasil prediksi yang benar atau hasil prediksi yang tepat datanya, seharusnya ham diprediksi sebagai ham, dan data yang seharusnya spam diprediksi sebagai spam.

2. Accuracy

Matrix evaluasi ke-2 yang akan kita pelajari adalah Accuracy,accuracy mengukur porsi dari hasil prediksi yang tepat.

Dan ini merupakan formulanya

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{correct}{total}$$

Accuracy ini bisa dikalkulasi dengan cara menjumlahkan nilai TP dan TN nya, lalu dibagi dengan hasil penjumlahan TP,TN,FP dan FN,atau dengan kata lain disini kita mencoba melakukan kalkulasi pembagian dari semua hasil prediksi yang tepat dibagi dengan total jumlah prediksinya.

Nilai yang dihasilkan oleh matrix accuracy ini akan berada pada rentang nilai 0 dan 1.

Bagi kalian yang tertarik untuk mempelajari accuracy lebih lanjut berikut adalah halaman Wikipedia yang bisa dijadikan referensi.

Refrensi : https://en.wikipedia.org/wiki/Accuracy_and_precision

Selanjutnya kita akan demokan pemanfaatan matrix accuracy ini dengan scikit learn.

```
from sklearn.metrics import accuracy_score

accuracy_score(y_test, y_pred)
```

Pertama-tama kita akan panggil dulu mdule nya, **from sklearn.metrics import** accuracy_score,lalu cara menggunakannya juga cukup mudah, Kita tinggal panggil saja **accuracy_score**, dan kita akan sertakan nilai **y_test** dan **y_pred** sebagai parameternya.

Dan ini hasil nya

```
0.964824120603015
```

Ini merupakan nilai yang cukup tinggi untuk accuracy,karena nilai ini kalau direpresentasikan sebagai persentase maka nilainya adalah 96,48%, dan ini termasuk nilai accuracy yang sangat tinggi.

3. Precission & Recall

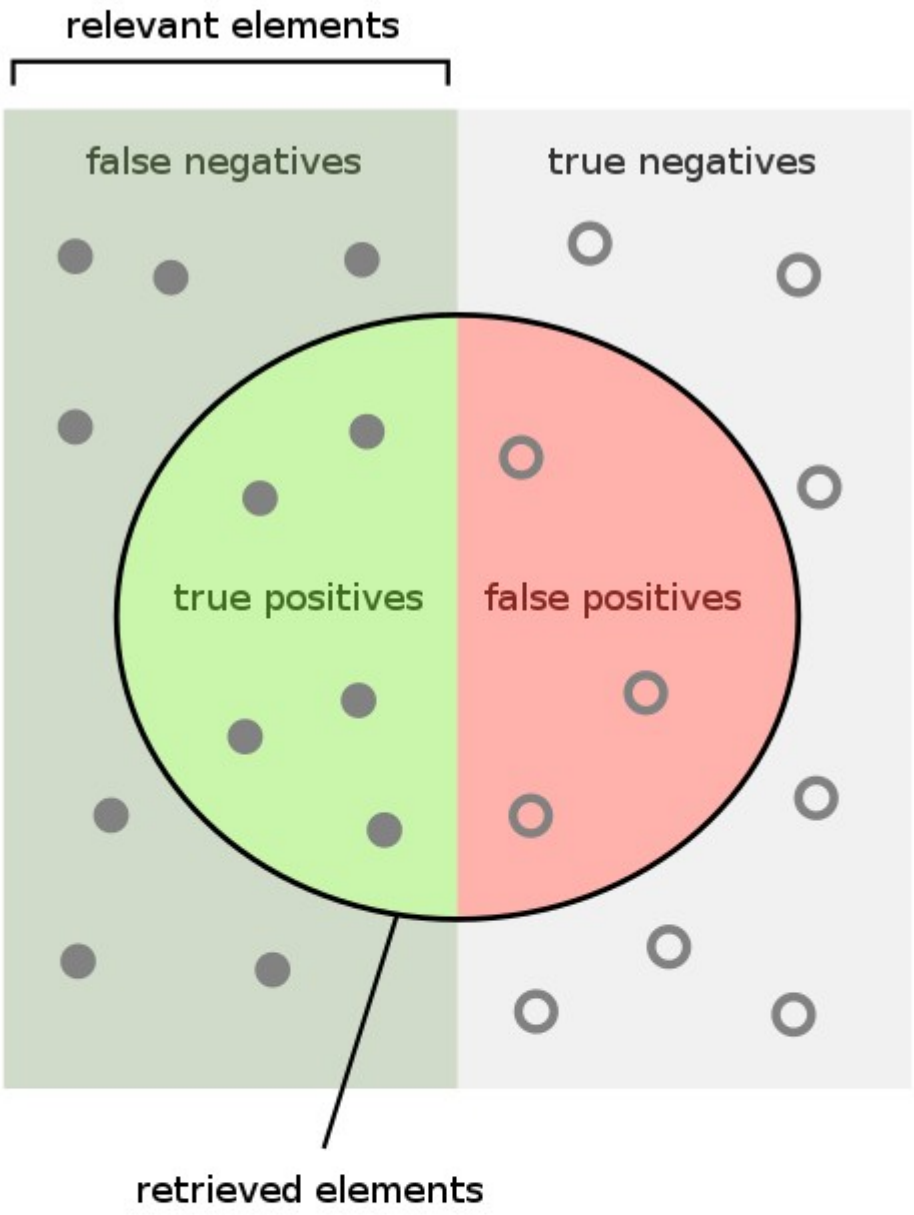
Selain menggunakan accuracy, performa dari suatu classifier umumnya juga diukur berdasarkan nilai Precision dan Recall.

Bagi kalian tertarik untuk mempelajari precision dan recall lebih lanjut berikut adalah halaman Wikipedia yang bisa dijadikan referensi

Refrensi : https://en.wikipedia.org/wiki/Precision_and_recall

Pada halaman Wikipedia ini terdapat ilustrasi yang cukup menarik, dan menurut kita layak untuk dibahas disini.

Kita akan coba berfokus pada figure atau ilustrasi yang ada di sini



How many retrieved items are relevant?

Precision = true positives / retrieved elements

How many relevant items are retrieved?

Recall = true positives / relevant elements

Kalau kita perhatikan di sini, ilustrasinya ini bisa kita bagi menjadi dua bagian, luar sebelah kiri Ini merepresentasikan area yang positif, dan ruas sebelah kanan ini merepresentasikan area ya negatif.

kalau untuk konteks spam pada dataset kita,area sebelah kiri Ini bisa dikategorikan sebagai area spam, dan area sebelah kanan ini bisa dikategorikan sebagai area ham, atau area bukan spam.

Berikutnya kita akan coba perhatikan area lingkaran, area lingkaran ini akan merepresentasikan keseluruhan data yang kita prediksi sebagai positif, atau kalau kita mengacu pada SMS spam dataset,berarti area lingkaran ini merupakan area yang berisi semua data yang diprediksi sebagai spam.

Dan bisa kita lihat di sini ternyata beberapa hasil prediksi kita juga salah, karena ada data-data yang seharusnya bukan spam dan kita prediksi sebagai spam, area inilah yang kita beri warna merah,kalau kita perhatikan dengan lebih detail disini, area hijau yang berada dalam lingkaran ini merupakan area true positif,kenapa true positif, Karena SMS nya merupakan SMS spam dan kita berhasil memprediksinya sebagai spam, oleh karenanya ini merupakan true positif.

Berbeda halnya dengan bagian lingkaran yang berwarna merah, data-data SMS yang ada di sini bukan merupakan data spam, tetapi kita prediksi sebagai spam, artinya ini merupakan hasil prediksi yang salah dan kesalahannya adalah false positif, karena data-data ini bukan merupakan spam, tapi kita prediksi sebagai positif suatu spam.

Selanjutnya kita akan perhatikan area di ruas sebelah kiri yang berada diluar lingkaran, ini akan merepresentasikan sekumpulan SMS yang seharusnya kita prediksi sebagai spam, tetapi classifier kita gagal memprediksinya sebagai spam, dengan kata lain berarti classifier kita melakukan suatu kesalahan dan kesalahannya adalah false, negatif karena disini classifier nya memandang bahwa data-data ini sebagai negatif spam,padahal seharusnya adalah positif spam.

Berikutnya kita perhatikan area di sebelah kanan, ini merupakan area yang memang bukan spam dan ketika dia berada di luar lingkaran artinya dia tidak diprediksi sebagai spam, atau dengan kata lain classifier kita setelah melakukan prediksi dengan bena,r dimana classifier kita tidak memprediksi data-data ini sebagai spam, karena classifier yang melakukan prediksi dengan benar maka disini kondisi adalah true atau lebih tepatnya adalah true negatif, karena classifier kita sudah dengan benar mengklasifikasikan data-data ini sebagai bukan spam.

Menurut kita ini merupakan ilustrasi yang sangat membantu untuk memahami apa yang dimaksud dengan true positif,false positif,false negatif dan true negatif,haharapannya ilustrasi ini juga bisa membantu memperdalam pemahaman kalian.

Berikutnya terkait dengan precision & recall, disini juga terdapat ilustrasi yang cukup membantu untuk memahami perbedaan antara precision & recall.

Bisa kita lihat di gambar ilustrasi,pada precision,yang dibandingkan adalah area true positif dibandingkan atau dibagi dengan keseluruhan hasil prediksi positive nya, sedangkan pada recall,yang dibandingkan atau yang dikalkulasikan adalah perbandingan antara nilai true positif dibandingkan dengan keseluruhan nilai positifnya.

Preccision of Positive Predictive Value (PPV)

Precision juga seringkali dikenal dengan istilah Preccision of Positive Predictive Value atau biasa di singkat PPV, dan ini merupakan formula dari Preccision.

$$Precission = \frac{TP}{TP + FP}$$

Bagi kalian yang tertarik untuk mempelajari precision dengan lebih detail, berikut adalah halaman Wikipedia yang bisa kita jadikan referensi.

Refrensi : https://en.wikipedia.org/wiki/Positive_and_negative_predictive_values

Selanjutnya kita akan demokan pemanfaatan Matrix precision ini dengan scikit learn

```
from sklearn.metrics import precision_score

precision_score(y_test, y_pred)
```

Pertama-tama kita akan impor dulu modulnya, **from sklearn.metrics import** precision_score, berikutnya kita akan panggil **precision_score,** dengan melewatkan **y_test** dan **y_pred** sebagai parameternya.

Dan ini hasil nya

```
0.9927536231884058
```

Sekedar informasi tambahan, nilai precision maupun recall itu juga memiliki rentang nilai yang serupa dengan accuracy,yaitu mulai dari 0 sampai dengan 1, dan kalau kita lihat di sini, nilai precision nya juga cukup tinggi.

Recall or True Positive Rate (RTP) or Sensitivity

Recallsendiri seringkali dikenal dengan istilah Recall or True Positive Rate, atau biasa disingkat sebagai RTP, dan untuk pengukuran recall juga dikenal sebagai Mesir of sensitivity, dari ini merupakan formula dari recall.

$$Recall = \frac{TP}{TP + FN}$$

Bagi kalian yang tertarik untuk mempelajari rekor dengan lebih detail berikut adalah halaman Wikipedia yang bisa kita jadikan referensi.

Refrensi : https://en.wikipedia.org/wiki/Sensitivity_and_specificity

Selanjutnya kita akan demokan pemanfaatan Matrix recall ini dengan scikit learn.

```
from sklearn.metrics import recall_score

recall_score(y_test, y_pred)
```

Pertama-tama kita akan impor dulu modulnya,**from sklearn.metrics import** recall_score,berikutnya kita Panggil fungsi **recall_score** dengan menyertakan **y_test** dan **y_pred** sebagai parameternya.

Dan ini hasil nya

```
0.7405405405405405
```

Recall score yang kita peroleh disini adalah **0.7405405405405405.**

asd

4. F1 Score

Matrix evaluasi ke-4 yang akan kita pelajari adalah F1 score,F1 score adalah harmonic mean dari precission dan recall.

Dan ini merupakan formula dari F1 score

$$F1\,Score = \frac{precission\,x\,recall}{precission + recall}$$

Bagi kalian tertarik untuk mempelajari nya lebih lanjut, berikut adalah halaman Wikipedia yang bisa kita jadikan referensi.

Refrensi : https://en.wikipedia.org/wiki/F-score

Selanjutnya kita akan demokan pemanfaatan F1-Score ini dengan scikit learn.

```
from sklearn.metrics import f1_score

f1_score(y_test, y_pred)
```

Pertama-tama kita akan Panggil dulu modulnya, **from sklearn.metrics import** f1_score,selanjutnya kita akan Panggil fungsi **f1_score**, dengan melewatkan **y_tes**t dan **y_pred** sebagai parameternya.

Dan ini hasil nya

```
0.8482972136222909
```

Bisa nampak disini hasil F1-Score nya adalah **0.8482972136222909.**

5. ROC : Receiver Operating Characeristic

Matrix evaluasi terakhir yang akan kita pelajari adalah ROC, ROC menawarkan visualisasi terhadap performa dari classifier dengan membandingkan nilai Recall atau TPR dan nilai Fallout atau FPR.

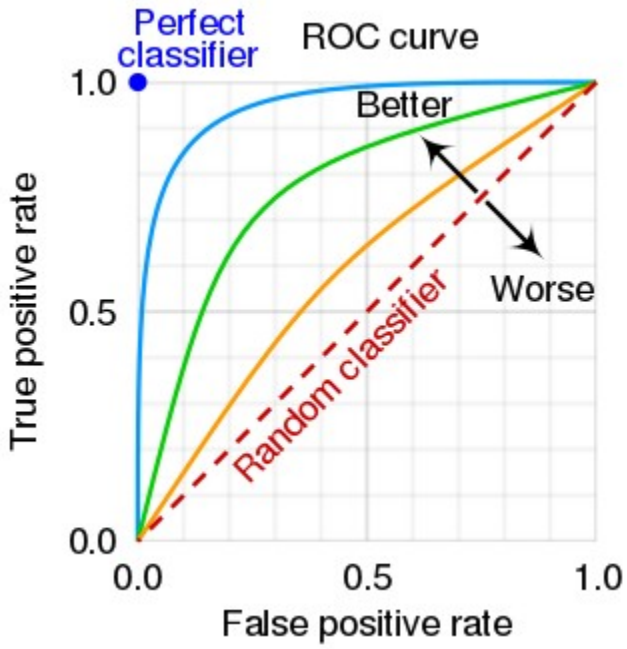
Nilai Fallout sendiri bisa kita peroleh dengan furmula berikut.

$$fallout = \frac{FP}{TN + FP}$$

Bagi kalian tertarik untuk mempelajari ROC lebih lanjut, berikut adalah halaman Wikipedia yang bisa kita jadikan referensi.

Refrensi : https://en.wikipedia.org/wiki/Receiver_operating_characteristic

Disini juga ada ilustrasi yang cukup menarik terkait ROC, ini merupakan kurva dari ROC.



kurva ROC ini pada dasarnya akan membandingkan dua buah parameter, yaitu Tue Positif Rate atau TPR, akan dibandingkan dengan False Positif Rate atau FBR, dimana rentang nilainya juga sama antara 0 sampai dengan 1.

Yang perlu diperhatikan disini adalah, pada umumnya curva ROC akan menampilkan garis linier diagonal, dan garis ini akan merepresentasikan random classifier.

Berdasarkan hasil visualisasi dari ROC, ketika hasil floating dari suatu model makin mengarah ke sisi kiri atas, maka performanya akan dinilai semakin baik, sedangkan ketika floating modelnya semakin mengarah ke sudut kanan bawah, maka model tersebut bisa dipandang sebagai model yang buruk.

Bisa nampak di sini terdapat tiga buah model yang di floating ke dalam kurva ROC ini, yang pertama dalam model dengan warna hijau, lalu model dengan warna orange, dan modal dengan warna biru, bisa

nampak di sini bahwa modal dengan warna biru merupakan model dengan performa terbaik, lalu disusul dengan model berwarna orange, dan yang paling buruk ke dalam modal dengan warna hija.

Suatu model akan mendapatkan nilai sempurna atau perfect classifier ketika posisinya berada di titik warna biru.

Selanjutnya kita akan demokan pemanfaatan ROC ini dengan menggunakan scikit learn.

```
from sklearn.metrics import roc_curve, auc

prob_estimates = model.predict_proba(X_test_tfidf)

fpr,  tpr, threshhold = roc_curve(y_test, prob_estimates[:, 1])
nilai_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, 'b', label=f'AUC={nilai_auc}')
plt.plot([0, 1], [0, 1], 'r--', label='Random Classifier')

plt.title('ROC: Receiver Operating Characteristic')
plt.xlabel('Fallout or False Positive Rate')
plt.ylabel('Recall or True Positive Rate')
plt.legend()
plt.show()
```

Pertama-tama kita akan impor dulu modulnya, **from sklearn.metrics import** roc_curve, dan disini kita juga akan impor **auc, auc** ini merupakan Area Under the Curve, lalu apa sih maksudnya Area Under the Curve, maksudnya semisal saja kita mau berfokus pada kurva yang warnanya biru ini, **auc** ini akan merepresentasikan wilayah atau area yang berada dibawah kurva berwarna biru ini,dan nilai **auc** nya akan merepresentasikan proporsi atau persentase dari area yang berada dibawah kurva ini, semakin kurva model yang mendekati titik yang warna biru,maka nilai **auc** nya akan semakin mendekati 1.

Pada tahapan berikutnya kita akan menghitung probability estimate nya, di sini kita panggil **model.predict_proba**,dan kita akan menyertakan **X_test_tfidf**, lalu berikutnya untuk membentuk ROC curve, kita bisa memanggil fungsi **roc_curve**, dengan menyertakan dua buah parameter, yaitu **y_test**, dan **prob_estimates**, disini **prob_estimates** nya akan menyertakan keseluruhan row, tetapi hanya akan menyertakan kolom yang kedua saja, oleh karenanya di sini indexnya kita beri nilai 1, dan hasil pemanggilan dari **roc_curve** ini akan mengembalikan 3 buah nilai, oleh karenanya disini kita akan ditangkap dengan menggunakan tiga buah variabel yaitu **fpr**, **tpr**, dan **threshhold.**

fpr ini berarti False Positif rate nya, **tpr** ini berarti True Posotive Rate nya, dan yang ketiga ini merupakan **threshhold** nya**.**

Selanjutnya berdasarkan nilai **fpr** dan **tpr** ini, kita bisa menghitung nilai **auc** nya, dengan cara memanggil fungsi **auc,** dan melewatkan **fpr** dan **tpr** nya sebagai parameter untuk selanjutnya nilai **auc** nya kita tampung ke dalam variabel **nilai_auc.**

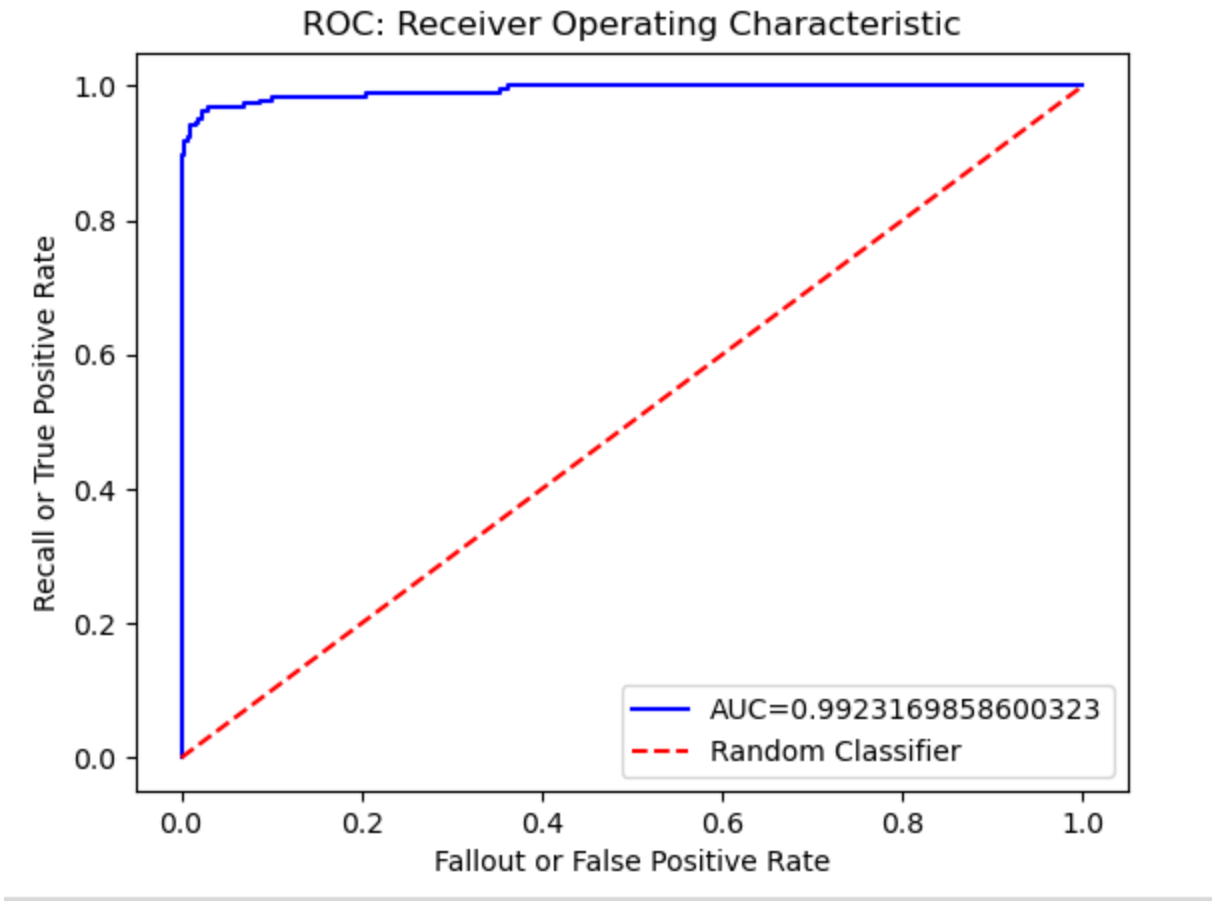
Dan itu terkait proses kalkulasinya,selanjutnya kita akan coba visualisasikan ROC curve nya.

Kita coba eksekusi kode nya

```
import pandas as pd
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

df = pd.read_csv('./dataset/SMSSpamCollection',
                 sep='\t',
                 header=None,
                 names=['label', 'sms'])


X = df['sms'].values
y = df['label'].values

lb = LabelBinarizer()
y = lb.fit_transform(y).ravel()

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.25,
                                                    random_state=0)

vectorizer = TfidfVectorizer(stop_words='english')

X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

model = LogisticRegression()
model.fit(X_train_tfidf, y_train)


prob_estimates = model.predict_proba(X_test_tfidf)

fpr,  tpr, threshhold = roc_curve(y_test, prob_estimates[:, 1])
nilai_auc = auc(fpr, tpr)
```

```
plt.plot(fpr, tpr, 'b', label=f'AUC={nilai_auc}')
plt.plot([0, 1], [0, 1], 'r--', label='Random Classifier')

plt.title('ROC: Receiver Operating Characteristic')
plt.xlabel('Fallout or False Positive Rate')
plt.ylabel('Recall or True Positive Rate')
plt.legend()
plt.show()
```

dan ini hasil nya



ini merupakan ROC curve nya, untuk menghasilkan visualisasi semacam ini yang perlu kita lakukan adalah pertama kita akan floating dulu garis yang berwarna biru ini,caranya kita panggil **plt.plot**, lalu kita menyatakan parameter **fpr**, dan **tpr** nya, lalu warnanya kita akan sebagai biru, oleh karenanya di

sini kita menyatakan string **b** atau **blue,** lalu untuk label nya kita beri nilai **AUC** disertai dengan **nilai_auc** nya.

Selanjutnya kita akan floating garis untuk random classifier nya, disini kita panggil saja **plt.plot** lalu kita Panggil **0,1** dan **0,1** baik untuk X maupun y nya, lalu untuk warna floating nya berwarna merah dan bentuknya berupa garis putus-putus, lalu untuk parameter **label** nya kita beri nilai **Random Classifier,** floating nya juga kita sertai dengan judul atau **plt.title**, kita serta juga dengan **xlabel** dan **ylabel**, dan juga kita tampilkan Legend nya.

Di sini kita mendapati nilai AUC nya adalah 0,9923169858600323 atau 99%, nilai AUC ini mengindikasikan bahwa luasan area yang berada dibawah kurva berwarna biru ini adalah 99% dari keseluruhan total area floating nya.

◆ Bab 13 : Naive Nayes Classification

Kali ini kita akan mempelajari model machine learning lain yang umum diterapkan dalam classification task yaitu Naive Bayes.

Naive Bayes ini termasuk model machine learning yang juga cukup umum diterapkan pada classification task,bagi kalian yang tertarik untuk mempelajari Naive Bayes lebih lanjut, berikut adalah halaman Wikipedia yang bisa dijadikan referensi.

Refrensi : https://en.wikipedia.org/wiki/Naive_bayes_classifier

◆ Bayes' Theorem

Bayes' Theorem menawarkan suatu formula untuk menghitung nilai probability dari suatu event dengan memanfaatkan pengetahuan sebelumnya dari kondisi terkait, atau seringkali dikenal dengan istilah conditional probability.

Bagi kalian masih awam dengan topik conditional probability kita sarankan untuk mempelajarinya terlebih dahulu,dan bagi kalian yang sudah mengenal conditional probability sebelumnya, tentunya kalian sudah tidak asing lagi dengan formula semacam ini

ini merupakan formula yang digunakan untuk mengekspresikan probability dari kemunculan event **A** bila diketahui event **B** muncul.

$$P(A|B)=\frac{P(B|A)\,X\,P(A)}{P(B)}$$

Proses kalkulasinya bisa kita lakukan dengan cara mencari tahu dulu probability dari kemunculan event **B** bila diketahui event **A** muncul, lalu dikalikan dengan nilai probability kemunculan event **A**, untuk selanjutnya kita bagi dengan nilai probability dari kemunculan event **B**.

Dan untuk konteks klasifikasi, formula ini juga bisa disesuaikan sebagai berikut:

$$P(y|X)=\frac{P(X|y)\,x\,P(y)}{P(X)}$$

Probability dari kemunculan nilai target label **y** bila diketahui kemunculan sekumpulan nilai features **X,** dan nilai ini bisa kita peroleh dengan cara mengkalkulasikan probability dari kemunculan sekumpulan nilai features **X** bila diketahui kemunculan nilai target label **y**, lalu nilainya kita kalikan dengan probability dari kemunculan nilai target label **y**, dan kita bagi dengan probability dari kemunculan sekumpulan nilai features**X.**

Kalau kita mau mengadopsi terminologi Naive Bayes, maka formulanya bisa direpresentasikan sebagai berikut:

$$Posterior=\frac{Likelihood\,x\,Prior}{Evidence}$$

◆ Pengenalan Naive Bayes classification

Disini kita sudah menyiapkan suatu studi kasus yang harapannya bisa membantu kita untuk memahami Naive Bayes dengan lebih mudah.

Studi kasus 1

Misalnya di sini terdapat sebuah warung yang menawarkan 3 menu, yaitu siomay, bakso, dan lumpia, warung ini juga hanya memiliki 2 pelanggan saja yaitu Asep, dan Joko, untuk setiap pelanggan sudah didata probabilitas pemesanan mereka untuk setiap menu yang ditawarkan.

Berikut adalah ilustrasinya.

Asep
+ siomay:0.1
+ bakso:0.8
+ lumpia:0.1

Joko
+ siomay:0.5
+ bakso:0.2
+ lumpia:0.3

Di sini probability untuk Asep memesan siomay adalah **0.1**, lalu untuk memesan bakso adalah **0.8**, probability untuk **Asep** memesan lumpia adalah **0.**

Lalu untuk Joko, probability Joko untuk memesan siomay adalah **0.5**, dan probability Joko untuk memesan bakso adalah **0,2**, sedangkan probability Joko untuk memesan lumpia **0.3**.

Untuk konteks ini, Asep dan Joko akan berperan sebagai target label atau class, disini kita akan menerapkan klasifikasi sederhana dengan Naive Bayes, untuk memprediksi siapa pelanggan yang memesan bila diketahui kombinasi pesanannya.

Misi:Lakukan prediksi siapa pelanggan yang melakukan pemesanan dengan diketahui pesanannya adalah **lumpia** dan **bakso.**

◆ Terminologi Dasar

Berikutnya kita akan mengenal beberapa terminologi dasar dalam Naive Bayes.

1. Prior Probability

Prior Probability adalah nilai probability dari kemunculan suatu nilai target label tertentu tanpa memperhatikan nilai features.

Dan ini merupakan notasi nya.

$$P(y)$$

Nilai ini tentunya akan bergantung pada dataset yang kita miliki, untuk kasus kita disini kita bisa asumsikan bahwa peluang untuk suatu pemesanan dilakukan oleh Asep atau Joko adalah seimbang, oleh karenanya Prior dari kasus ini bisa diekspresikan sebagai berikut

$P(Asep)=0.5$

$P(Joko)=0.5$

Probability pemesanan dilakukan oleh Asep adalah **0.5** atau 50%, dan probability pemesanan dilakukan oleh Joko juga bernilai **0.5** atau 50%.

Bagi kalian yang tertarik untuk mempelajari Prior lebih lanjut, berikut adalah halaman Wikipedia yang bisa dijadikan referensi

Refrensi : https://en.wikipedia.org/wiki/Prior_probability

2. Likelihood

Dalam statistika sebenarnya ada sedikit perbedaan antara probability dan likelihood, hanya saja untuk menyederhanakan penjelasan, disini kita akan asumsikan keduanya adalah sama.

Bagi kalian yang tertarik untuk mempelajari like ikut lebih lanjut berikut adalah halaman Wikipedia yang bisa dijadikan referensi

Refrensi : https://en.wikipedia.org/wiki/Likelihood_function

Likelihood untuk konteks Naive Bayes bisa didefinisikan sebagai probability kemunculan nilai features tertentu bila diketahui kemunculan nilai target label nya, atau biasa diekspresikan sebagai berikut.

$$P(X|y)$$

Karena disini kita memiliki 2 class, maka likelihood nya bisa diekspresikan sebagai berikut.

Nilai probability kemunculan pesanan lumpia dan bakso bila diketahui Asep adalah pemesan nya, bisa dikalkulasikan sebagai berikut.

- **Asep :**

$$P(lumpia, bakso|Asep) = (0.1 \times 0.8) \\ = 0.08$$

Karena pesanannya adalah lumpia dan bakso, dan kita tahu untuk Asep lumpia itu memiliki probability 0.1, sedangkan bakso memiliki probability 0.8, 0.1 dikalikan dengan 0.8 sehingga nilainya adalah 0.08.

Lalu nilai probability kemunculan pesanan lumpia dan bakso bila diketahui Joko adalah pemesannya bisa dikalkulasikan sebagai berikut.

- **Joko :**

$$P(lumpia, bakso|Joko) = (0.3 \times 0.2) \\ = 0.06$$

Karena pesanannya adalah lebih dan bakso, maka bisa kita lihat di sini untuk Joko, probability pemesanan lumpia adalah 0.3 sedangkan probability pemesanan bakso adalah 0.2, oleh karenanya 0.3 dan 0.2, 0.3 dikali dengan 0.2 hasilnya adalah 0.06.

3.Evidence atau Normalizer

Evidence atau seringkali juga dikenal sebagai Normalizer, dan ini merupakan notasi yang umum digunakan

$$P(X)$$

Evidence untuk konteks Naive Bayes bisa didefinisikan sebagai total akumulasi dari hasil perkalian antara nilai likelihood dengan Prior nya, sehingga bisa diekspresikan sebagai berikut.

$$Evidence = \sum (Likelihood \times Prior)$$

Untuk kasus kita disini,nilai evidence yang merupakan probability dari kemunculan pesanan lumpia dan bakso dan ini merupakan kalkulasinya.

$$P(lumpia, bakso) = (0.08 \times 0.5) + (0.06 \times 0.5)$$
$$= 0.07$$

Nilai 0.08 ini kita peroleh dari nilai Likelihood untuk sisi Asep, sedangkan 0.5 ini merupakan nilai Prior untuk sisi Asep, lalu Berikutnya 0.06 ini merupakan nilai Likelihood untuk sisi Joko,dan 0.5 ini merupakan prior untuk sisi Joko, oleh karenanya di sini 0.06 dikali 0.5 dan hasil kalkulasinya adalah 0.07.

4. Posterior Probability

Terminologi terakhir yang akan kita pelajari terkait Naive Bayes adalah posterior probability,ini merupakan nilai probability kemunculan suatu class atau target label dengan diketahui kemunculan sekumpulan nilai features nya atau bisa juga dinotasikan sebagai berikut

$P(y|X)$

Berikut merupakan formula untuk Posterior Probability.

$$Posterior = \frac{Likelihood \times Prior}{Evidence}$$

Bagi kalian yang tertarik untuk mempelajari posterior lebih lanjut, berikut adalah halaman Wikipedia yang bisa dijadikan referensi.

Refrensi : https://en.wikipedia.org/wiki/Posterior_probability

Disini karena kita memiliki dua class, maka posterior nya bisa diekspresikan sebagai berikut.

Pertama kita akan hitung nilai probability Asep sebagai pemesan bila diketahui pesanannya adalah lumpia dan bakso, dan proses kalkulasinya menjadi seperti ini.

- **Asep :**

$$P(Asep\ deivides\ lumpia, bakso) = \frac{0.08 \times 0.5}{0.07}$$
$$= 0.57$$

0.08 merupakan Likelihood untuk sisi Asep, 0.5 merupakan nilai Prior untuk sisi Asep, dan 0.07 merupakan nilai Evidence nya, dan hasilnya disini adalah 0.57.

Selanjutnya kita akan hitung nilai probabilitas Joko sebagai pemesan bila diketahui pesanannya adalah lumpia dan bakso dan ini merupakan proses kalkulasinya.

- **Joko :**

$$P(Joko\, deivides\, lumpia\, ,bakso) = \frac{0.06 \times 0.5}{0.07}$$
$$= 0.43$$

0.06 merupakan Likelihood untuk sisi Joko, dan 0.5 merupakan Prior untuk sisi Joko, sedangkan 0.07 merupakan nilai Evidence nya.

Untuk kasus kita disini, nilai Posterior untuk Asep lebih tinggi bila dibandingkan dengan nilai Posterior dari Joko, maka Naive Bayes akan mengklasifikasikan Asep sebagai pemesannya.

Studi Kasus 2

Untuk memperjelas pemahaman kita akan coba pelajari studi kasus kedua.

Disini kita juga diminta untuk melakukan klasifikasi pelanggan berdasarkan informasi pesanan yang diterima, hanya saja kali ini pesanannya adalah siomay dan bakso.

Asep
+ siomay:0.1
+ bakso:0.8
+ lumpia:0.1

Joko
+ siomay:0.5
+ bakso:0.2
+ lumpia:0.3

Misi: lakukan prediksi siapa pelanggan yang melakukan pemesanan dengan diketahui pesanannya adalah **siomay** dan **bakso.**

Disini kalian coba terapkan pemahaman yang sudah kita pelajari sebelumnya pada kasus ini,selanjutnya kita akan dan hasilnya.

Di sini kita asumsikan kalian sudah mencoba mengerjakan studi kasus ini dan kita akan mencocokkan hasilnya.

Pertama-tama kita akan hitung terlebih dahulu nilai Evidence atau Normalizer nya, untuk kasus kita disini nilai Evidence nya merupakan probability dari kemunculan pesanan siomay dan bakso, dan ini merupakan hasil kalkulasinya.

Evidence :

$P(X)$

$$P(siomay, bakso) = (0.1 \times 0.8 \times 0.5) + (0.5 \times 0.2 \times 0.5)$$
$$= 0.09$$

Disini perlu kita identifikasi bahwa 0.1 dikali 0.8 ini merupakan nilai Likelihood dari sisi Asep, sedangkan 0.5 dikali 0.2 ini merupakan nilai Likelihood dari sisi Joko, lalu 0.5 disini $(0.1 \times 0.8 \times 0.5)$,ini merupakan nilai Prior dari sisi Asep dan 0.5 disini $(0.5 \times 0.2 \times 0.5)$,ini merupakan nilai Prior dari sisi Joko, dan hasil kalkulasinya adalah 0,09.

Selanjutnya kita akan hitung nilai posterior nya. pertama-tama kita akan hitung nilai probability Asep sebagai pemesan bila diketahui pesanannya adalah siomay dan bakso,dan ini merupakan hasil kalkulasinya

- **Asep :**

$$P(Asep|siomay, bakso) = \frac{(0.1 \times 0.8) \times 0.5}{0.09}$$
$$= 0.444$$

Perkalian nilai 011 dengan 0.8 ini merupakan nilai Likelihood dari sisi Asep, sedangkan 0.5 ini merupakan nilai Prior dari sisi Asep, dan nilainya kita bagi dengan 0.09 yang merupakan nilai Evidence atau nilai Normalizer nya, dan disini kita memperoleh nilai 0,444.

Selanjutnya kita akan hitung nilai probability Joko sebagai pemesan bila diketahui pesanannya adalah siomay dan bakso dan ini merupakan hasil kalkulasinya.

- **Joko :**

$$P(Joko|siomay, bakso) = \frac{(0.05 \times 0.2) \times 0.5}{0.09}$$
$$= 0.555$$

Disini hasil perkalian 0.5 dengan 0.2 merupakan nilai Likelihood dari Joko, lalu nilai 0.5 ini merupakan nilai Prior dari sisi Joko, untuk selanjutnya hasil perkalian ini akan kita bagi dengan 0.09 yang merupakan nilai Evidence atau nilai Normalizernya,dan disini kita memperoleh nilai 0.555.

Untuk kasus kita kali ini, karena nilai posterior untuk Joko lebih tinggi dari Asep, maka Naive Bayes akan mengklasifikasikan Joko sebagai pemesannya.

◆ Mengapa disebut Naive?

Sampai di titik ini sebagian dari kalian mungkin bertanya-tanya mengapa model ini diberi nama Naive Bayes, kata Bayes berasal dari nama seorang ststistion yaitu Thomas Bayes, beliau yang menggagas suatu theorem yang akhirnya dikenal sebagai Bayes theorem.

Lalu pertanyaan berikutnya adalah mengapa model ini disebut Naive?, dan jawaban singkatnya adalah karena sewaktu kita mendefinisikan Likelihood nilai Probability kemunculan pemesanan lumpia dan bakso bila diketahui pemesannya adalah Asep, kita mengasumsikan bahwa nilai Probability kemunculan pemesanan lumpia bila diketahui pemesannya adalah Asep, itu conditional independen terhadap nilai Probability dari kemunculan nilai pesanan bakso bila diketahui pemesannya adalah Asep.

Demikian pula sebaliknya,sehingga dapat diformulasikan sebagai berikut.

$$P(lumpia, bakso|Asep) = P(lumpia|Asep) \times P(bakso|Asep)$$

Probability kemunculan pesanan lumpia dan bakso bila diketahui pemesannya adalah Asep bisa dikalkulasikan dengan cara,mengalikan nilai Probability kemunculan pemesanan lumpia oleh Asep, dikalikan dengan nilai Probability kemunculan pemesanan bakso oleh Asep.

Atau dengan kata lain, model Naive Bayes akan mengabaikan kemungkinan adanya korelasi antar nilai features, di sini Naive Bayes mengasumsikan bahwa pemesanan lumpia oleh Asep itu tidak dipengaruhi dan tidak ada hubungannya dengan pemesanan bakso oleh Asep, padahal dalam kasus sebenarnya bisa jadi pemesanan lumpia oleh Asep dipengaruhi oleh pemesanan bakso oleh Asep.

◆ Penerapan Naive Bayes dengan Scikit Learn

Pertama-tama kita akan siapkan terlebih dahulu Dataset nya, untuk kasus kali ini, kita akan mengadopsi Breast Cancer Wisconsin dataset.

Dataset Breast Cancer Wisconsin (Diagostic)

Dataset ini merupakan Open dataset yang di tawarkan oleh uci machine learning,dan ini merupakan Link yang bisa kita gunakan untuk mengakses dataset nya.

Refrensi : https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagostic)

Perlu kalian ketahui, dataset ini juga disertakan sebagai sample dataset pada scikit learn, disini kita akan coba demokan cara untuk mengakses deskripsi dari dataset ini.

```
from sklearn.datasets import load_breast_cancer

print(load_breast_cancer().DESCR)
```

Pertama-tama kita akan impor dulu modulnya, **from sklearn.datasets import** load_breast_cancer, berikutnya kita akan panggil **load_breast_cancer()**, dan disini kita akan coba akses description dari dataset nya dengan cara kita panggil .**DESCR,**yang perlu diperhatikan disini adalah DESCR, nya semuanya dalam huruf besar, lalu berikutnya nilai description ini akan kita cetak layar dengan memanfaatkan fungsi **print**.

dan ini merupakan hasil nya

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
```

```
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        worst/largest values) of these features were computed for each image,
        resulting in 30 features.  For instance, field 0 is Mean Radius, field
is Radius SE, field 20 is Worst Radius.

        - class:
                - WDBC-Malignant
                - WDBC-Benign

    :Summary Statistics:

    ===================================== ====== ======
                                           Min    Max
    ===================================== ====== ======
    radius (mean):                        6.981  28.11
    texture (mean):                       9.71   39.28
    perimeter (mean):                     43.79  188.5
    area (mean):                          143.5  2501.0
    smoothness (mean):                    0.053  0.163
    compactness (mean):                   0.019  0.345
    concavity (mean):                     0.0    0.427
    concave points (mean):                0.0    0.201
    symmetry (mean):                      0.106  0.304
    fractal dimension (mean):             0.05   0.097
    radius (standard error):              0.112  2.873
    texture (standard error):             0.36   4.885
    perimeter (standard error):           0.757  21.98
    area (standard error):                6.802  542.2
    smoothness (standard error):          0.002  0.031
    compactness (standard error):         0.002  0.135
    concavity (standard error):           0.0    0.396
    concave points (standard error):      0.0    0.053
    symmetry (standard error):            0.008  0.079
    fractal dimension (standard error):   0.001  0.03
    radius (worst):                       7.93   36.04
    texture (worst):                      12.02  49.54
    perimeter (worst):                    50.41  251.2
    area (worst):                         185.2  4254.0
```

```
    smoothness (worst):                    0.071  0.223
    compactness (worst):                   0.027  1.058
    concavity (worst):                     0.0    1.252
    concave points (worst):                0.0    0.291
    symmetry (worst):                      0.156  0.664
    fractal dimension (worst):             0.055  0.208
    ====================================== ====== ======

    :Missing Attribute Values: None

    :Class Distribution: 212 - Malignant, 357 - Benign

    :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

    :Donor: Nick Street

    :Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2

Features are computed from a digitized image of a fine needle
aspirate (FNA) of a breast mass.  They describe
characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using
Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree
Construction Via Linear Programming." Proceedings of the 4th
Midwest Artificial Intelligence and Cognitive Science Society,
pp. 97-101, 1992], a classification method which uses linear
programming to construct a decision tree.  Relevant features
were selected using an exhaustive search in the space of 1-4
features and 1-3 separating planes.

The actual linear program used to obtain the separating plane
in the 3-dimensional space is that described in:
[K. P. Bennett and O. L. Mangasarian: "Robust Linear
Programming Discrimination of Two Linearly Inseparable Sets",
Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/

.. topic:: References

   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction
     for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on
```

```
    Electronic Imaging: Science and Technology, volume 1905, pages 861-870,
    San Jose, CA, 1993.
  - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and
    prognosis via linear programming. Operations Research, 43(4), pages 570-577,
    July-August 1995.
  - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques
    to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994)
    163-171.
```

Ini merupakan deskripsi dari dataset kita, sini ada nama datasheetnya, nama dataset adalah **Breast cancer wisconsin (diagnostic) dataset**, lalu dataset ini terdiri dari **569 instances** atau 569 data, di sini juga ada keterangan nya **Number of Attributes: 30 numeric**, artinya disini terdapat 30 nilai numeric sebagai features nya atau sebagai kolomnya, dan **Attribute Information** ini merupakan keterangan terkait features nya,dan semua nilai features ini memiliki tipe data numerik dengan total jumlah features adalah 30 features.

Disini terkait class terdapat 2 **class** yaitu **WDBC-Malignant**,dan **WDBC-Benign**,**Malignant** ini mengindikasikan tumor ganas, sedangkan **Benign** ini mengindikasikan tumor Cina, disini juga ada keterangan **Missing Attribute Values: None**,artinya tidak terdapat missing value atau nilai yang kosong, lalu terkait dengan **class Distribution:,** atau proporsi class nya terdapat 212 data dan diklasifikasikan sebagai **Malignant** atau tumor ganas, dan terdapat 357 data dan diklasifikasikan sebagai **Benign** atau tumor Cina, dan **Creator,**merupakan keterangan dari creater atau pembuat dataset nya, lalu di sini dataset ini dibuat pada tahun 1995 pada bulan November.

Untuk mengakses dokumentasi dari fungsi **load_breast_cancer** di Jupyter Notebook, kita memiliki kebebasan dan kemudahan yang luar biasa. Sebagai contoh, jika kita ingin mengakses dokumentasi dari **load_breast_cancer**, caranya cukup mudah. Kita hanya perlu memanggil:

```
load_breast_cancer?
```

Ini akan menampilkan dokumentasi lengkap dari fungsi **load_breast_cancer** langsung di Jupyter Notebook.

Selain menggunakan Jupyter Notebook, terdapat beberapa metode lain untuk mengakses dokumentasi dari fungsi atau modul di Python, seperti menggunakan pydoc dan interpreter Python di terminal. Berikut adalah beberapa contohnya:

pydoc di terminal :

```
pydoc sklearn.datasets.load_breast_cancer
```

Ini akan menampilkan dokumentasi dari **load_breast_cancer** langsung di terminal Anda.

Python interpreter:

```
from sklearn.datasets import load_breast_cancer
help(load_breast_cancer)
```

Ini akan menampilkan dokumentasi dari **load_breast_cancer** di interpreter Python.

Dengan menggunakan salah satu metode di atas, Anda dapat dengan mudah mengakses dokumentasi untuk fungsi atau modul Python, baik di Jupyter Notebook, terminal, maupun interpreter Python.

Dan ini merupakan hasil nya

```
Help on function load_breast_cancer in sklearn.datasets:

sklearn.datasets.load_breast_cancer = load_breast_cancer(*, return_X_y=False,
as_frame=False)
    Load and return the breast cancer wisconsin dataset (classification).

    The breast cancer dataset is a classic and very easy binary classification
    dataset.

    =================   ==============
    Classes                          2
    Samples per class    212(M),357(B)
    Samples total                  569
    Dimensionality                  30
    Features            real, positive
    =================   ==============

    The copy of UCI ML Breast Cancer Wisconsin (Diagnostic) dataset is
    downloaded from:
    https://goo.gl/U2Uwz2

    Read more in the :ref:`User Guide <breast_cancer_dataset>`.
```

```
    Parameters
    ----------
    return_X_y : bool, default=False
        If True, returns ``(data, target)`` instead of a Bunch object.
        See below for more information about the `data` and `target` object.

        .. versionadded:: 0.18

    as_frame : bool, default=False
        If True, the data is a pandas DataFrame including columns with
        appropriate dtypes (numeric). The target is
        a pandas DataFrame or Series depending on the number of target columns.
        If `return_X_y` is True, then (`data`, `target`) will be pandas
        DataFrames or Series as described below.

        .. versionadded:: 0.23

    Returns
    -------
    data : :class:`~sklearn.utils.Bunch`
        Dictionary-like object, with the following attributes.

        data : {ndarray, dataframe} of shape (569, 30)
            The data matrix. If `as_frame=True`, `data` will be a pandas
            DataFrame.
        target : {ndarray, Series} of shape (569,)
            The classification target. If `as_frame=True`, `target` will be
            a pandas Series.
        feature_names : list
            The names of the dataset columns.
        target_names : list
            The names of target classes.
        frame : DataFrame of shape (569, 31)
            Only present when `as_frame=True`. DataFrame with `data` and
            `target`.

            .. versionadded:: 0.23
        DESCR : str
            The full description of the dataset.
        filename : str
            The path to the location of the data.

            .. versionadded:: 0.20

    (data, target) : tuple if ``return_X_y`` is True
        A tuple of two ndarrays by default. The first contains a 2D ndarray of
        shape (569, 30) with each row representing one sample and each column
        representing the features. The second ndarray of shape (569,) contains
```

```
        the target samples.  If `as_frame=True`, both arrays are pandas objects,
        i.e. `X` a dataframe and `y` a series.

        .. versionadded:: 0.18

    Examples
    --------
    Let's say you are interested in the samples 10, 50, and 85, and want to
    know their class name.

    >>> from sklearn.datasets import load_breast_cancer
    >>> data = load_breast_cancer()
    >>> data.target[[10, 50, 85]]
    array([0, 1, 0])
    >>> list(data.target_names)
    ['malignant', 'benign']
```

Ini merupakan documentation dari function load_breast_cancer,menurut kita ini merupakan fasilitas yang sangat membantu yang ditawarkan oleh Jupiter notebook bagi para penggunanya, jadi perlu diingat ketika kalian bekerja dengan Jupiter notebook dan kalian bingung dengan manfaat ataupun kegunaan dari suatu function tertentu, kalian cukup panggil saja nama function nya lalu diikuti dengan tanda tanya, dan jika script nya dieksekusi maka kalian akan mendapatkan akses ke documentation dari function tersebut.

Disini kita akan mencermati salah satu parameter yang bisa kita sertakan pada function **load_breast_cancer** ini, yaitu **return_X_y**, default value nya adalah **False**, dan kali ini kita akan set sebagai **True,** artinya function **load_breast_cancer,** ini akan secara otomatis membagi dan memisahkan antara features data dengan label data nya,tentunya layanan semacam ini akan sangat membantu dalam memudahkan pekerjaan kita.

Selanjutnya kita akan coba demokan cara untuk mengakses dataset ini.

```
from sklearn.datasets import load_breast_cancer

X, y = load_breast_cancer(return_X_y=True)
print(X.shape)
```

Disini **load_breast_cancer return** nya kita beri nilai **True,** di sini kita perlu menyiapkan 2 buah variabel untuk menangkap nilai kembalian dari function ini yaitu **X** dan **y,** dimana nilai **X** ini

digunakan untuk menampung nilai features, dan nilai **y** nya digunakan untuk menampung nilai label nya.

Dan ini hasilnya

```
(569, 30)
```

Bisa nampak disini jumlah barisnya ada 569,dan jumlah kolomnya atau jumlah features ada 30 features.

Training & Testing set

Selanjutnya kita akan bagi dataset ini ke dalam training dan testing set, berikut akan kita demokan prosesnya.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2,
                                                    random_state=0)

print(f'X_train shape {X_train.shape}')
print(f'X_test shape {X_test.shape}')
```

Disini kita akan impor dulu modulnya, **from sklearn.model_selection import** train_test_split,selanjutnya kita akan panggil **train_test_split(X, y, test_size=0.2, random_state=0),**dalam hal ini **test_size** nya adalah 0.2, artinya 20% dari data sekitar akan kita alokasikan untuk testing dataset, sedangkan 80% nya akan kita gunakan sebagai training dataset, lalu terkait **random_state** disini adalah **0**, nah fungsi **random_state** ini akan mengembalikan 4 buah kumpulan nilai yang perlu kita tangkap kedalam empat buah variabel yaitu, **X_train, X_test, y_train, y_test**, untuk selanjutnya disini kita juga akan coba tampilkan dimensi dari **X_train, X_test** nya.

Kita coba eksekusi kode nya

```
from sklearn.datasets import load_breast_cancer
```

```
from sklearn.model_selection import train_test_split


X, y = load_breast_cancer(return_X_y=True)


X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2,
                                                    random_state=0)

print(f'X_train shape {X_train.shape}')
print(f'X_test shape {X_test.shape}')
```

dan ini hasil nya

```
X_train shape (455, 30)
X_test shape (114, 30)
```

Bisa nampak disini, untuk **X_train** jumlah datanya ada 455,artinya ini merupakan 20%, dan untuk **X_test** jumlah datanya ada 114 dan ini merupakan 80% dari keseluruhan datast yang kita miliki.

◆ Naive Bayes dengan Scikit Learn

Berikutnya kita akan menerapkan Naive Bayes untuk melakukan klasifikasi data pada dataset Breast Cancer yang kita miliki, berikut akan kita demokan prosesnya.

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

model = GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

Pertama-tama disini kita akan impor dulu modulnya, **from sklearn.naive_bayes import** GaussianNB,NB ini merepresentasikan Naive Bayes, kita juga import **from sklearn.metrics import** accuracy_score, berikutnya kita akan bentuk dulu objek dari model GaussianNB ini lalu objeknya kita

tampung dalam variabel **model** untuk selanjutnya model ini akan kita training dengan menggunakan **metode.fit**, dimana kita menyertakan **X_train** dan **y_train** untuk mentraining model kita, setelah model kita training maka train model ini akan kita gunakan untuk melakukan prediksi terhadap nilai dari **X_test**, dan hasil prediksinya akan kita tampung ke dalam variabel **y_pred**, untuk selanjutnya kita akan ukur nilai accurary nya dengan memanggil fungsi **accuracy_score**, dan kita akan lewatkan parameter **y_test,** dan **y_pred.**

Kita coba eksekusi kode nya

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

X, y = load_breast_cancer(return_X_y=True)


X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2,
                                                    random_state=0)

model = GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

dan ini hasil nya

```
Accuracy: 0.9298245614035088
```

Nilai Accuracy nya adalah **0.9298245614035088**,nilai accuracy ini sebenarnya juga bisa kita peroleh dengan cara yang lebih sederhana yaitu dengan memanggil:

```
model.score(X_test, y_test)
```

Dan kalau kita eksekusi kodenya maka hasil nyaakan sama,pemanggilan **model.score** ini setara pemanggilan dua baris fungsi **y_pred** = **model.predict(X_test)** dan **accuracy** = **accuracy_score(y_test, y_pred)**.

◆ Bab 14 : Support Vector Machine Classification (SVM)

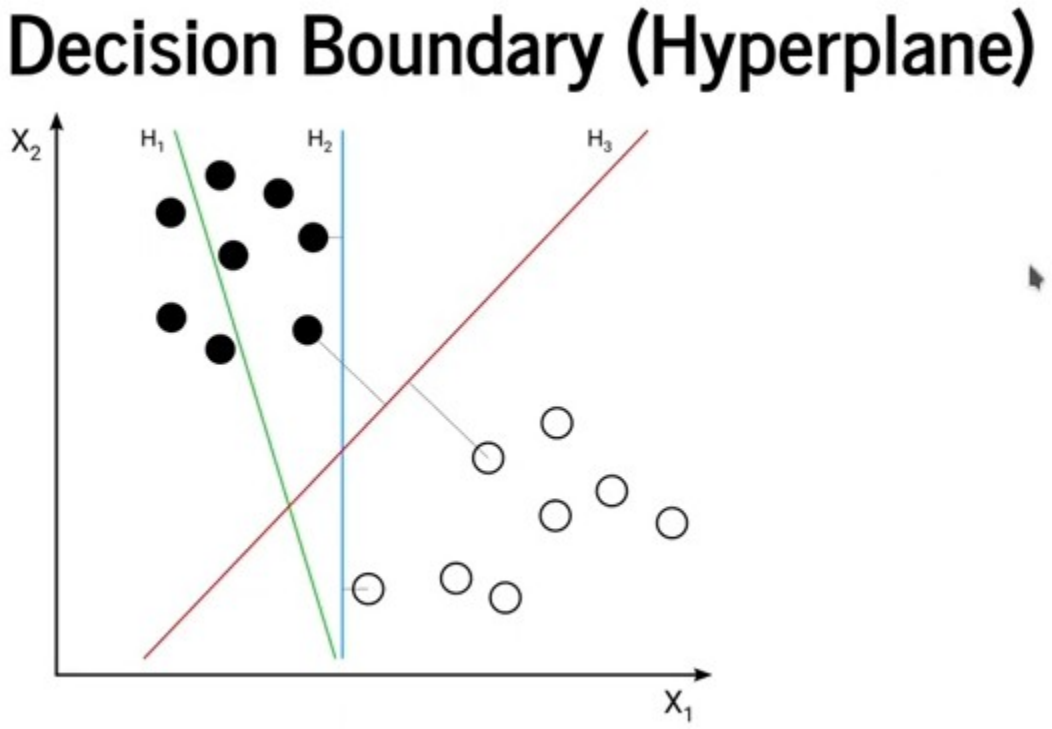
Support Vector Machine Classification (SVM), ini termasuk model machine learning yang juga cukup umum diterapkan pada classification task, bagi kalian yang tertarik untuk mempelajari SVM lebih lanjut berikut adalah free ebook yang bisa dijadikan referensi

Refrensi : https://www.svm-tutorial.com

Menurut kita ini merupakan ebook terbaik yang membahas SVM yang pernah kita temui Jadi, bagi kalian yang tertarik untuk mendalami SVM lebih lanjut kita sangat menyarankan untuk menggunakan ini sebagai referensi.

◆ Konsep Dasar

Pertama-tama disini kita akan menjelaskan beberapa konsep dasar yang harapannya dapat membantu kita untuk memahami mekanisme kerja dari support Vector machine,untuk itu kita akan cermati contoh kasus berikut ini.



Misalnya kita disini dihadapkan pada sebuah kasus klasifikasi dimana terdapat 2 buah class, sebut saja class hitam dan putih,pada kasus ini juga terdapat 2 buah seature yaitu x1 dan x2,di sini kita diminta untuk menarik suatu garis lurus atau garis linier yang dapat memisahkan kedua class ini, dalam classification task pemisah atau pembatas antar class ini juga seringkali dikenal dengan istilah di Decision Boundary.

Semisal saja di sini kita dihadapkan pada tiga pilihan garis linier yaitu H1, H2,dan H3, warna hijau nya H1, yang warna biru ini H2, yang warna merah ini adalah H3, kalau menurut kalian dari ketiga garis linear ini manakah yang paling baik untuk digunakan sebagai decision boundary atau pemisah antar dua class yang kita miliki.

Dini kita asumsikan kalian sudah memilih salah satu dari ketiga garis linier ini sebagai jawaban,
kita cermati satu-persatu dari ketiga pilihan garis linier ini.

Pertama-tama kita akan cermati garis H1,kalau kita lihat disini nampak jelas bahwa garis H1 ini tidak bisa digunakan sebagai garis pemisah antar dua class yang kita miliki, berarti di sini pilihannya tinggal dua yaitu garis H2 dan garis H3, dan kalau kita lihat disini kedua garis ini dapat memisahkan kedua class yang kita miliki dengan sempurna.

Yang menjadi pertanyaan berikutnya adalah manakah dari kedua garis ini yang paling baik dalam memisahkan kedua kelas kita dan jawabannya adalah H3, jadi antara H3 dengan H2 yang paling baik adalah H3,alasannya karena H3 ini memiliki Margin yang lebih besar bila dibandingkan dengan H2.

1. Decision Boundary(Hyperplane)

Hyperplane ini merupakan terminologi yang umum digunakan dalam SVM untuk merepresentasikan decision boundary, kita akan kembali memanfaatkan studi kasus diatas,seperti nampak di sini karena dalam kasus ini kita dihadapkan pada dua features, maka kita memiliki floating dua dimensi,dan ketika datanya di floating maka kita akan memperoleh floating dua dimensi, nah oleh karenanya di decision boundaru yang kita peroleh berupa suatu garis, dan dalam kasus ini berupa garis lurus atau garis linear.

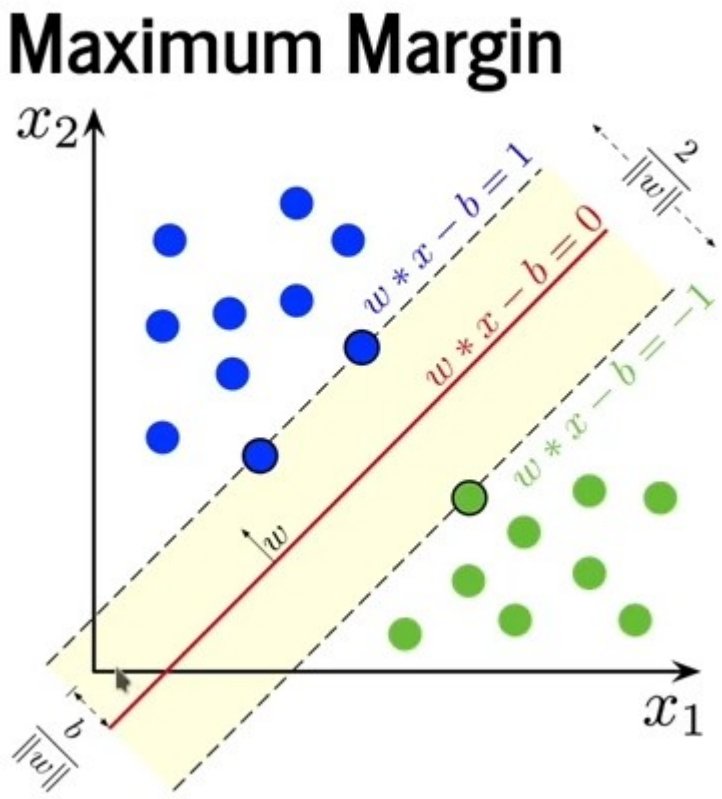
Tetapi ketika kita hanya memiliki satu features saja maka decision boundary atau pemisah antar class nya akan berupa titik atau nilai threshold, lalu ketika terdapat tiga features maka decision boundary nya berupa sebuah bidang datar atau juga biasa dikenal dengan istilah plane.

Ketika terdapat empat atau lebih features maka pemisah antar class nya berupa bidang multidimensi atau biasa dikenal dengan istilah hyperplane,pada SVM untuk menyederhanakan istilah maka setiap decision boundary yang dihasilkan umumnya akan disebut sebagai hyperplane.

Pada intinya SVM ini mencari decision boundary yang dapat memisahkan antar class dengan baik.

2. Maximum Margin

Pertama-tama kita akan pahami terlebih dahulu apa itu margin,margin ditentukan berdasarkan jarak terdekat antara decision boundary dengan anggota dari class yang ingin dipisahkan, untuk bisa memahaminya dengan lebih baik kita akan memanfaatkan contoh kasus berikut ini.



Pada kasus kali ini terdapat dua buah class yaitu, class biru dan class hijau, disini juga terdapat dua buah features yaitu x1 dan x2.

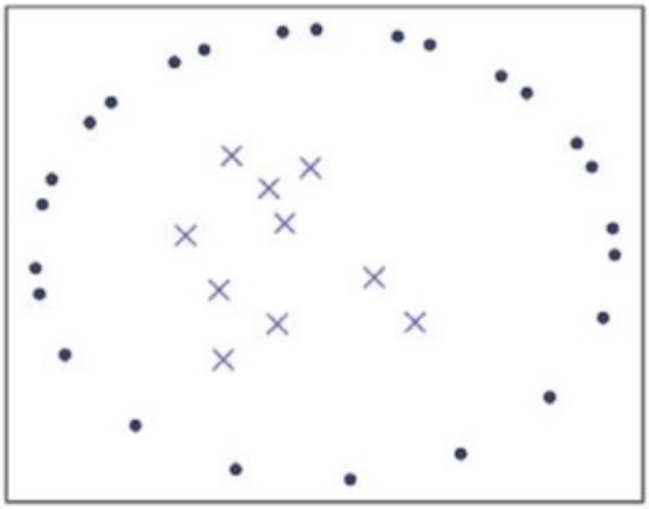
Semisal saja di sini kita memiliki sebuah decision boundary berupa garis linear berwarna merah yang digunakan untuk memisahkan kedua class yang kita miliki, garis linier merah ini merupakan decision boundary yang memisahkan class biru dengan class hijau ,dan area diarsir dengan warna kuning inilah yang dinamakan dengan margin.

Margin sendiri diperoleh berdasarkan jarak terdekat antara decision boundary dengan anggota dari class yang ingin dipisahkan, dan setiap anggota class yang berperan untuk menentukan margin dikenal sebagai Support Vector.

Untuk kasus kita disini terdapat tiga poin sebagai Support Vector, ketiga datapoint ini berperan sebagai Support Vector untuk kasus kita disini,dimana Support Vector ini merupakan anggota dari suatu class yang posisinya paling dekat dengan decision boundary, dalam menentukan decision boundary,SVM makan memilih berdasarkan margin terbesar atau juga dikenal dengan istilah Maximum margin.

3. Linearly Inseperable & Kernel Tricks

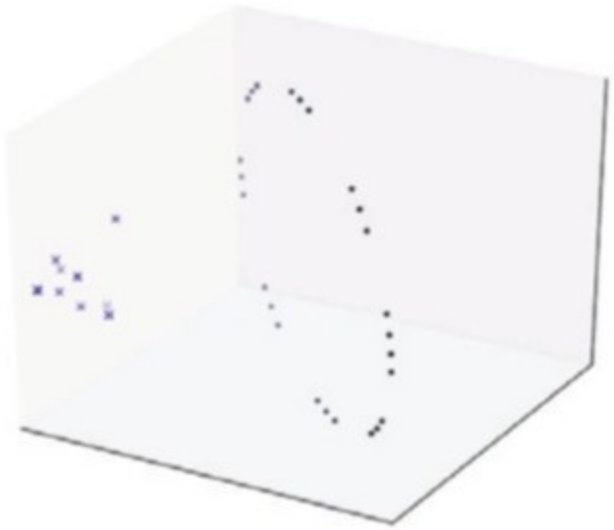
Pada kedua contoh kasus sebelumnya kita sudah mengenal pemanfaatan dari garis lurus atau garis linier sebagai decision boundary,hanya saja terdapat beberapa kasus dimana class yang ada tidak bisa dipisahkan dengan memanfaatkan garis linear, untuk lebih jelasnya kita akan pelajari contoh kasus berikut ini.



Pada contoh kasus kali ini juga terdapat dua buah class yaitu class titik, dan class X,disini juga terdapat dua buah features sehingga ketika dilakukan floating kita akan mendapatkan floating dua dimensi semacam ini.

Kalau kita perhatikan pada kasus kali ini,tidaklah memungkinkan bagi kita untuk menarik garis linear sebagai decision boundary,kondisi semacam ini juga dikenal dengan istilah Linearly Inseparable.

Untuk mengatasi kondisi semacam ini,SVM akan memproyeksikan data yang ada ke dimensi yang lebih tinggi,artinya bila data yang sebelumnya berada dalam dua dimensi maka SVM akan memproyeksikan nya ke tiga dimensi,seperti nampak pada gambar berikut



Ini merupakan hasil proyeksi tiga dimensi dari data sebelumnya yang berbentuk dua dimensi.

Bisa nampak disini setelah diproyeksikan ke dimensi yang lebih tinggi dimana untuk kasus ini adalah tiga dimensi, kedua class ini bisa dipisahkan dengan lebih mudah yaitu dengan menerapkan decision boundary berbentuk bidang datar,jadi disini kita bisa tempatkan suatu bidang datar yang akan berperan sebagai decision boundary yang memisahkan antara class X dengan class titik.

Upaya untuk memproyeksikan sekumpulan data ke dimensi yang lebih tinggi tentunya juga akan berimbas pada kenaikan beban komputasi, untuk menjawab kebutuhan semacam ini SVM menawarkan teknik yang sangat efisien yang dikenal dengan istilah Kernel Tricks,SVM sendiri menawarkan beberapa jenis Kernel seperti polynomial, sigmoid dan RBF.

Bagi kalian yang tertarik untuk mempelajari Kernel Tricks lebih lanjut, berikut adalah halaman quora orang yang bisa dijadikan referensi.

Refrensi : https://www.quora.com/What-is-the-kernel-trick

Pemanfaatan support Vector dan juga Kernel Tricks untuk membentuk decision boundary inilah yang akhirnya menjadikan model machine-learning ini diberi nama Support Vector Machine,dan apa yang baru saja kita jelaskan di sini merupakan beberapa konsep dasar yang memang perlu diketahui terkait SVM.

◆ Pemanfaatan SVM dengan Scikit Learn

Selanjutnya kita nama pelajari beberapa pengaplikasian SVM dengan memanfaatkan scikit learn.

Dataset: The MNIST database of handwritten digits

Pertama-tama kita akan siapkan terlebih dahulu dataset nya, untuk kasus kali ini kita akan mengadopsi handwritten dataset, ini merupakan open dataset yang ditawarkan oleh MNIST,dan ini adalah link untuk mengakses dataset nya.

Refrensi : https://yann.lecun.com/exdb/mnist

Perlu juga kalian ketahui,meskipun dataset ini tidak disertakan sebagai sampel dataset pada scikit learn, tetapi kita bisa mendownloadnya dengan memanfaatkan model fetch.openml, karena dataset ini juga tersedia di repository openml.

Disini kita akan demokan tahapan untuk mendownload serta mengakses dataset nya.

```
from sklearn.datasets import fetch_openml

X,y = fetch_openml(f'mnist_784', data_home='./dataset/mnist', return_X_y=True)
print(X.shape)
```

Pertama-tama kita akan impor dulu modul nya,**from sklearn.datasets import** fetch_openml, berikutnya kita tinggal panggil **fetch_openml**, dan disini kita akan sebutkan nama dataset adalah **mmist_784**, lalu kita juga akan spesifikasikan lokasi pada local machine kita untuk mendownload datset nya,untuk kasus kita kali ini kita akan tempatkan pada direktori **dataset/mnist**, berikutnya kita juga akan langsung pisahkan antara features dengan target labelnya, oleh karenanya disini kita sertakan parameter return_X_y, dan kita beri nilai **True,** disini kita juga akan siapkan dua variabel yaitu **X&y,** dimana variabel **X** akan menampung sekumpulan nilai features, dan variabel **y** nya digunakan untuk menampung sekumpulan nilai target labelnya, lalu berikutnya kita juga akan coba tampilkan dimensi data dari features nya.

Perlu di ingat bahwa sewaktu kalian mengeksekusi script ini pastikan kalian memiliki koneksi internet,karena ini akan melakukan proses download dataset melalui jaringan internet, lalu berapa lama waktu yang dibutuhkan ini akan sangat bergantung pada koneksi internet yang kalian miliki.

Atau kalian juga bisa mendownload nya langsung di website nya.

Dan ini hasil nya

```
(70000, 784)
```

Bisa nampak di sini dimensi dari variabel **X** nya adalah 70.000 untuk jumlah barisnya, dan 784 untuk jumlah kolomnya, dengan kata lain disini kita bisa bilang bahwa ukuran dataset kita kali ini lumayan besar bila dibandingkan dengan dataset lain yang pernah kita gunakan sebelumnya.

Disini terdapat 70.000 gambar berbeda yang berisi tulisan tangan manusia yang mencakup angka 0 sampai 9, dataset ini juga umum digunakan untuk melakukan perbandingan performa model machine-learning dalam mengenali angka atau bilangan dari tulisan tangan manusia.

Terdapat salah seorang tokoh besar dalam bidang computer vision yang terlibat dalam pengelolaan data set ini, namanya adalah yang Yann Lecun, Beliau memiliki kontribusi besar dalam bidang OCR atau Optical Character Recognition dan juga Handwritten Recognition.

Selanjutnya kita akan coba tampilkan 8 data pertama dari dataset ini, sekarang dataset yang kita miliki ini berupa data image atau data gambar maka kita akan gunakan matplotlib untuk menampilkan datanya, berikut kita akan coba jelaskan prosesnya.

```
import matplotlib.pyplot as plt
import matplotlib.cm as cm

pos = 1
for data in X_train[:8]:
    plt.subplot(1, 8, pos)
    plt.imshow(data.reshape((28, 28)),
               cmap=cm.Greys_r)
    plt.axis('off')
    pos += 1

plt.show()
```

Pertama-tama kita akan impor dulu modul nya, **import matplotlib.pyplot as plt** dan juga **import matplotlib.cm as cm,** cm ini merepresentasikan Color Map,selanjutnya kita akan coba lakukan looping terhadap delapan baris pertama dari dataset kita, oleh karenanya di sini kita panggil **for** data **in** X, dan kita melakukan slicing terhadap 8 data pertama atau 8 baris pertama dari variabel **X** untuk selanjutnya akan kita floating dengan memanfaatkan pyplot.

Di sini datanya juga akan di reshape ke-28 kali 28 karena untuk setiap gambarnya terdiri dari 28 kali 28 pixel.

Dan jika kalian mendownload dataset nya langsung dari website nya,kalian bisa menggunakan kode berikut

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm

def load_mnist_images(filename):
    with open(filename, 'rb') as f:
        data = np.frombuffer(f.read(), np.uint8, offset=16)
```

```
        data = data.reshape(-1, 28*28)
    return data

def load_mnist_labels(filename):
    with open(filename, 'rb') as f:
        data = np.frombuffer(f.read(), np.uint8, offset=8)
    return data

# Ubah path_to_data sesuai dengan lokasi file yang Anda unduh
X = load_mnist_images('./dataset/mnist/train-images-idx3-ubyte')
y = load_mnist_labels('./dataset/mnist/train-labels-idx1-ubyte')

# Lakukan visualisasi seperti sebelumnya
pos = 1
for data in X[:8]:
    plt.subplot(1, 8, pos)
    plt.imshow(data.reshape((28, 28)), cmap=cm.Greys_r)
    plt.axis('off')
    pos += 1
plt.show()
```

Dan ini hasil nya

Dan ini merupakan ke-8 data pertamanya, bisa nampak disini, ini merupakan tulisan tangan manusia atau human handwritten yang merepresentasikan bilangan atau angka.

Bagi kita manusia tentunya sangat mudah untuk memahami angka-angka yang ada di sini tetapi bagi komputer ini bukanlah hal yang mudah.

Selanjutnya kita juga akan tampilkan 8 label yang berkorelasi dengan 8 data pertama ini, kita cukup panggil saja

```
y[:8]
```

Dan ini merupakan label nya

```
array(['5', '0', '4', '1', '9', '2', '1', '3'])
```

Dengan kata lain gambar yang pertama ini akan diberi label 5 lalu gambarnya kedua ini akan diberi label 0, dan label ketiga ini akan diberi label 4 dan seterusnya.

Selanjutnya kita akan training model kita untuk mempelajari data ini,karena target label nya berupa angka atau bilangan maka jumlah class nya akan ada 10 yaitu mulai dari class 0 sampai 9,selanjutnya kita akan bagi dataset ini ke dalam training dan testing set, hanya saja kali ini kita tidak akan menggunakan keseluruhan dataset yang ada, ini dimaksudkan agar proses training model yang akan kita demokan tidak terlalu panjang, hanya saja kita mendorong kalian untuk menggunakan keseluruhan dataset sewaktu kalian mencobanya pada perangkat komputer yang kalian miliki, berikut akan kita demokan prosesnya.

Ini merupakan script jika kalian igin menggunakan seluruh data

```
X_train = X[:60000]
y_train = y[:60000]
X_test = X[:60000]
y_test = y[:60000]
```

Bisa nampak di sini ketika kalian mencoba, kalian akan menggunakan 60.000 data pertama sebagai training set, dan 1.000 data yang terakhir akan digunakan sebagai testing set nya, hanya saja untuk demo kita kali ini,kita hanya akan menggunakan 1,000 data pertama sebagai training set, dan 1000 data terakhir sebagai testing setnya.

```
X_train = X[:1000]
y_train = y[:1000]
X_test = X[:1000]
```

```
y_test = y[:1000]
```

◆ Classification dengan SVC (Support Machine Classification)

Setelah training and testing set nya terbentuk kita akan coba menerapkan Support Vector Machine untuk melakukan klasifikasi angka numeric berdasarkan data sheet tulisan tangan yang kita miliki.

Pertama-tama kita akan training modelnya terlebih dahulu, berikut akan kita demokan prosesnya.

```
from sklearn.svm import Svc

model = SVC(random_state=0)
model.fit(X_train, y_train)
```

Pertama-tama kita akan impor dulu modulnya, **from sklearn.svm import** Svc, berikutnya kita akan bentuk objek dari model **SVC** nya kita akan sertakan parameter yaitu **random_state** yang kita beri nilai **0**, object model yang terbentuk akan kita tampung ke dalam variabel **model** untuk selanjutnya object model ini akan kita training dengan memanggil method Fit dan menyertakan training set nya, di sini kita panggil **model.fit(X_train, y_train)**.

Setelah modelnya ditraining selanjutnya kita akan lakukan evaluasi performa dari model yang baru saja kita training tadi, disini kita akan memanfaatkan classification report.

```
from sklearn.metrics import classification_report

y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

Pertama-tama kita akan impor dulu classification repotnya, **from sklearn.metrics import** classification_report, berikutnya kita akan gunakan testing set nya untuk melakukan prediksi, **model.predict(X_test)**, dan hasil prediksinya akan kita tampung ke dalam variabel **y_pred** untuk selanjutnya hasil prediksi ini akan kita bandingkan dengan **y_test** nya, **classification_report(y_test, y_pred),** dan kita akan cetak hasil classification_report nya.

Kita coba eksekusi kode nya

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn.svm import SVC
from sklearn.metrics import classification_report

def load_mnist_images(filename):
    with open(filename, 'rb') as f:
        data = np.frombuffer(f.read(), np.uint8, offset=16)
        data = data.reshape(-1, 28*28)
    return data

def load_mnist_labels(filename):
    with open(filename, 'rb') as f:
        data = np.frombuffer(f.read(), np.uint8, offset=8)
    return data

# Ubah path_to_data sesuai dengan lokasi file yang Anda unduh
X = load_mnist_images('./dataset/mnist/train-images-idx3-ubyte')
y = load_mnist_labels('./dataset/mnist/train-labels-idx1-ubyte')

X_train = X[:1000]
y_train = y[:1000]
X_test = X[:1000]
y_test = y[:1000]

model = SVC(random_state=0)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

dan ini hasil nya

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.98        97
           1       0.99      0.98      0.99       116
           2       0.99      0.97      0.98        99
           3       1.00      0.98      0.99        93
           4       0.96      1.00      0.98       105
           5       0.96      0.98      0.97        92
           6       1.00      0.99      0.99        94
           7       0.96      1.00      0.98       117
           8       1.00      0.98      0.99        87
           9       1.00      0.94      0.97       100
```

```
    accuracy                           0.98      1000
   macro avg       0.98      0.98      0.98      1000
weighted avg       0.98      0.98      0.98      1000
```

Bisa nampak di sini terdapat 10 buah class, class 0 sampai dengan 9, lalu untuk setiap classnya terdapat nilai precision ,recall dan juga f1-score nya, lalu disini kita juga memiliki nilai precision,recall, f1-score dan juga accuracy secara keseluruhan.

◆ Hyperparameter Tuning dengan GridSearchCV

Sejauh ini,dalam pembelajaran machine learning yang di tawarkan di buku ini, kita hampir selalu menggunakan default parameter setiap kali melakukan training model, padahal untuk tiap model machine-learning terdapat sejumlah parameter yang bisa Kita sesuaikan, dalam konteks machine-learning parameter yang digunakan untuk mengatur proses training dari suatu model dikenal dengan istilah Hyperparameter,dan proses untuk mencari komposisi nilai optimum dari Hyperparameter ini dikenal dengan istilah Hyperparameter Tuning atau Hyperparameter Optimization.

Bagi kalian yang tertarik untuk mempelajari hyperparameter Tuning lebih lanjut, berikut adalah halaman Wikipedia yang bisa dijadikan referensi.

Refrensi : https://en.wikipedia.org/wiki/Hyperparameter_optimization

Disini kita akan melakukan Hyperparameters Tuning dengan memanfaatkan modul GridSearchCV, dan modul ini sudah disertakan oleh scikit learn, berikut akan kita demokan prosesnya.

```
from sklearn.model_selection import GridSearchCV

parameters = {
        'kernel': ['rbf', 'poly', 'sigmoid'],
        'C': [0.5, 1, 10, 100],
        'gamma': ['scale', 1, 0.1, 0.01, 0.001]
        }

grid_search = GridSearchCV(estimator=SVC(random_state=0),
                           param_grid=parameters,
```

```
                                n_jobs=6,
                                verbose=1,
                                scoring='accuracy')

print(grid_search.fit(X_train, y_train))
```

Pertama-tama akan impor dulu modulnya, **from sklearn.model_selection import** GridSearchCV, berikutnya disini kita akan spesifikasikan sekumpulan parameter beserta pilihan nilai yang akan kita kombinasikan, semisal saja di sini kita ingin melakukan parameter tuning untuk tiga parameter yaitu **kernel**,**C** dan **gamma**, dan ketiganya ini merupakan parameter yang bisa ditemui di dalam SVM.

Berikutnya untuk setiap parameter ini akan kita tentukan pilihan nilainya, untuk **kernel** di sini pilihannya adalah **rbf**,**poly** dan **sigmoid,** untuk nilai **C** nya disini kita akan tentukan **0.5**, **1**, **10** dan **100,**dan untuk **gamma** nya kita tentukan pilihannya mulai dari **scale 1**, **0.1**, **0.01** dan **0.001.**

Pada intinya disini kita ingin mencari tahu kombinasi nilai yang paling baik untuk parameter **kernel**,**C** dan g**amma** yang bisa kita terapkan terhadap objek Support Vector Classifier pada kasus kita, kita bisa saja mencobanya satu persatu dengan memanfaatkan looping hanya saja kode program yang dihasilkan tidak akan bersih, oleh karenanya disini kita coba memanfaatkan fasilitas yang sudah ditawarkan oleh scikit learn, dalam hal ini adalah GridSearchCV.

Cara menggunakan GridSearchCV pun cukup sederhana, pertama-tama kita akan bentuk objek dari GridSearchCV nya,lalu kita kanggil disini **GridSearchCV** lalu kita akan sertakan beberapa parameter,yaitu,estimator, parameter estimator ini akan kita beri nilai berupa objek dari model yang ingin kita terapkan, untuk kasus kita disini objek modelnya akan kita bentuk dari class SVC atau Support Vector Classifier yang kita beri nilai parameter **random_state** nya **0**, lalu berikutnya kita juga perlu spesifikasikan **param_grid**,nilai parameter ini akan kita asosiasikan dengan **parameters** yang baru saja kita spesifikasikan sebelumnya.

Selanjutnya hal lain yang perlu kita atur adalah **n_jobs** atau number of Jobs, disini number of Jobs nya kita set sebagai **6**, artinya kita mau menjalankan proses ini secara paralel pada 6 thread dari prosesor kita.

Disini Kalian juga perlu menyesuaikan dengan prosesor yang kalian miliki, untuk kasus kita di sini kita menggunakan processor Intel i7, dengan jumlah thread yang tersedia ada 12 thread, dan dari ke-12 thread yang kita miliki, kita akan menggunakan 6 untuk melakukan pemrosesan yang ada di sini.

Berikutnya kita juga set parameter **verbose** nya dengan nilai **1**, ini bertujuan agar ketika prosesnya berjalan kita juga mendapatkan feedback yang cukup informatif, lalu berikutnya kita juga perlu menentukan parameter **scoring,** untuk kasus kita di sini kita set parameter nya adalah **accuracy,** artinya nilai pembanding yang kita gunakan adalah **accuracy,** setelah objek **GridSearchCV** nya terbentuk, objek ini akan kita tampung ke dalam variabel **grid_search**, lalu penerapannya juga cukup mudah, disini kita tidak lagi memanggil model.fit melainkan method fit ini akan kita Panggil dari objek **grid_search** nya, **grid_search.fit(X_train, y_train)**.

Kita coba eksekusi kode nya

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV

def load_mnist_images(filename):
    with open(filename, 'rb') as f:
        data = np.frombuffer(f.read(), np.uint8, offset=16)
        data = data.reshape(-1, 28*28)
    return data

def load_mnist_labels(filename):
    with open(filename, 'rb') as f:
        data = np.frombuffer(f.read(), np.uint8, offset=8)
    return data

# Ubah path_to_data sesuai dengan lokasi file yang Anda unduh
X = load_mnist_images('./dataset/mnist/train-images-idx3-ubyte')
y = load_mnist_labels('./dataset/mnist/train-labels-idx1-ubyte')

X_train = X[:1000]
y_train = y[:1000]
X_test = X[:1000]
y_test = y[:1000]

model = SVC(random_state=0)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

parameters = {
        'kernel': ['rbf', 'poly', 'sigmoid'],
        'C': [0.5, 1, 10, 100],
        'gamma': ['scale', 1, 0.1, 0.01, 0.001]
        }

grid_search = GridSearchCV(estimator=SVC(random_state=0),
                           param_grid=parameters,
                           n_jobs=6,
                           verbose=1,
```

```
                           scoring='accuracy')

print(grid_search.fit(X_train, y_train))
```

Kalau kita menggunakan verbose=1, maka terdapat informasi yang cukup informatif selama scriptnya berjalan, lalu Berapa lama script ini akan berjalan?, tentunya akan sangat bergantung pada processor yang kalian miliki.

Dan ini hasil nya

```
Fitting 5 folds for each of 60 candidates, totalling 300 fits

[Paralel(n_jobs=6)]: Using backend LokyBackend with 6 concurrent workers.
[Paralel(n_jobs=6): Done 38 tasks              | elapsed:       33.2s
[Paralel(n_jobs=6): Done 38 tasks              | elapsed:       2.4min
[Paralel(n_jobs=6): Done 38 tasks              | elapsed:       3.2min finished

GridSearchCV(estimator=SVC(random_state=0), n_jobs=6,
             param_grid={'C': [0.5, 1, 10, 100],
                         'gamma': ['scale', 1, 0.1, 0.01, 0.001],
                         'kernel': ['rbf', 'poly', 'sigmoid']},
             scoring='accuracy', verbose=1)
```

Bisa nampak di sini jumlah Fitting nya ini akan dilakukan 5 folds atau 5 kali, dimana untuk setiap folds nya terdapat 60 candidates, artinya total terdapat 300 proses Fitting yang terjadi.

Untuk kasus kita disini proses ini berakhir dengan total waktu 3,2 menit, dan tentunya akan sangat bergantung pada kekuatan komputasi dari komputer yang kalian gunakan, hasil akhir dari proses training model dengan memanfaatkan GridSearchCV ini adalah suatu train model yang di training dengan menerapkan komposisi Hyperparameter paling optimum dari sejumlah opsi parameter yang kita spesifikasikan sebelumnya.

Disini kita juga bisa mendapatkan informasi terkait komposisi nilai parameter paling optimum dari hasil pencarian GridSearchCV ini.

berikut akan kita demokan prosesnya.

```
print(f'Best Score: {grid_search.best_score_}')

best_params = grid_search.best_estimator_.get_params()
```

```
print(f'Best Parameters:')
for param in parameters:
      print(f'\t{param}: {best_params[param]}')
```

Pertama-tama disini kita akan coba akses dulu score terbaik atau accuracy terbaiknya, caranya kita tinggal panggil saja **grid_search.best_score_**, baris ini baru untuk menampilkan nilai score terbaik nya, lalu pada tahapan berikutnya kita juga bisa mendapatkan nilai parameter terbaiknya, caranya kita tinggal akses saja **grid_search.best_estimator_.get_params,** lalu sekumpulan nilai parameter terbaik ini akan kita tampung adalah variabel **best_params**, untuk selanjutnya akan kita looping dan kita coba tampilkan ke layar.

Dan ini hasil nya

```
Best Score: 0.907
Best Parameters:
      kernel: rbf
      C: 10
      gamma: scale
```

Bisa nampak disini score accuracy terbaik dari proses kriteria adalah **0.907,** dan komposisi nilai parameter terbaik yang berhasil ditemukan adalah **kernel** nya adalah **rbf**, nilai C nya adalah **10**, dan nilai parameter **gamma** adalah **scale**.

◆ Predict & Evaluate

Selanjutnya kita akan lakukan evaluasi performa dari model yang baru saja kita training tadi, disini kita juga akan memanfaatkan classification report, untuk melakukan prediksi disini juga agak sedikit berbeda kita tidak menggunakan model.predict melainkan kita akan memanfaatkan objek GridSearch nya.

```
y_pred = grid_search.predict(X_test)
print(classification_report(y_test, y_pred))
```

Disini kita panggil **grid_search.predict**, dan kita akan sertakan nilai dari variabel **X_test**, lalu hasil prediksinya akan kita tampung ke dalam variabel **y_pred** untuk selanjutnya nilai **y_pred** ini akan kita bandingkan dengan **y_test** dengan menggunakan classification repot.

Dan ini merupakan classification report yang dihasilkan.

	precision	recall	f1-score	support
0	**0.97**	**1.00**	**0.98**	**97**
1	**0.99**	**0.98**	**0.99**	**116**
2	**0.99**	**0.97**	**0.98**	**99**
3	**1.00**	**0.98**	**0.99**	**93**
4	**0.96**	**1.00**	**0.98**	**105**
5	**0.96**	**0.98**	**0.97**	**92**
6	**1.00**	**0.99**	**0.99**	**94**
7	**0.96**	**1.00**	**0.98**	**117**
8	**1.00**	**0.98**	**0.99**	**87**
9	**1.00**	**0.94**	**0.97**	**100**
accuracy			**0.98**	**1000**
macro avg	**0.98**	**0.98**	**0.98**	**1000**
weighted avg	**0.98**	**0.98**	**0.98**	**1000**

◆ Bab 15 : Decision Tree Classification

Decision Tree ini juga termasuk model machine lainnya yang umum ditemui dan diterapkan pada classification task, bagi kalian yang tertarik untuk mempelajari decision tree lebih lanjut,berikut adalah halaman Wikipedia yang bisa dijadikan referensi.

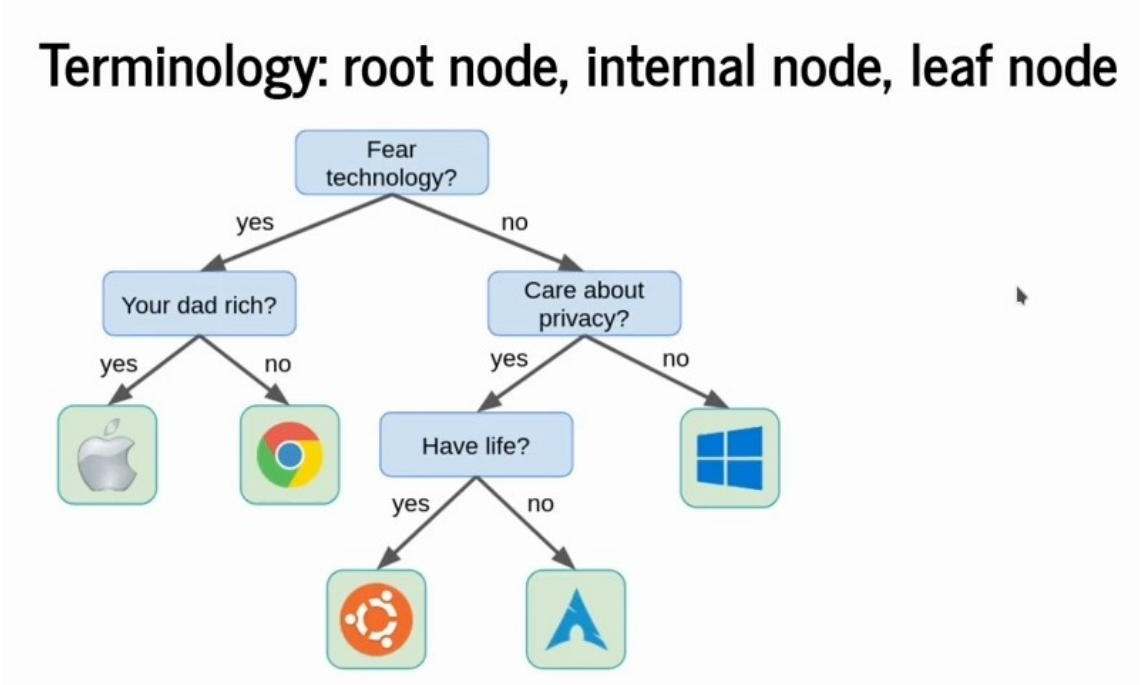
Refrensi : https://en.wikipedia.org/wiki/Decision_tree_learning

◆ Konsep Dasar

Pertama-tama disini kami akan menjelaskan beberapa konsep dasar yang harapannya dapat membantu kita untuk memahami mekanisme kerja dari Decision Tree.

Sesuai dengan namanya, Decision Tree atau beberapa orang menyebutnya sebagai pohon keputusan merupakan metode klasifikasi yang menerapkan struktur pohon.

berikut adalah salah satu contoh dari Decision Tree



Contoh dari Decision Tree ini kami adopsi dari meme yang sempat beredar di internet.

Pada contoh Decision Tree disini terdapat beberapa pertanyaan, pertanyaan pertama di sini **Fear technology?,** kalo jawabannya iya atau **yes** akan dilanjutkan pada pertanyaan susulan yaitu, **Your dad rich?,** dan kalau jawabannya adalah **yes** lagi, maka akan diarahkan ke pilihan **Apple,**sedangkan kalau jawaban untuk **Your dad rich?** adalah **no**, maka akan diarahkan ke **Chrome OS**.

Ketika pertanyaan **Fear technology?** nya dijawab dengan **no**, maka akan dihadapkan pada pertanyaan susulan kedua yaitu **Care about privacy?**, kalo jawabannya adalah **no** akan diarahkan ke **Windows,**tetapi kalau jawabannya adalah **yes** akan diarahkan ke pertanyaan ketiga yaitu **Have live?**, dan kalau jawabannya **yes** maka akan diarahkan ke **Ubuntu Linux,**sedangkan kalau jawabannya **no** akan diarahkan ke **Arch linux.**

Kembali kami tekankan ini merupakan meme atau jokes yang beredar di internet,gambar ini sengaja kami adopsi karena struktur semacam ini dikenal sebagai tree, atau lebih tepatnya adalah binary tree, karena tiap node nya akan memiliki dua cabang.

Sebagian dari kalian mungkin ada yang mulai bertanya-tanya, mengapa struktur semacam ini bisa disebut sebagai struktur pohon,padahal kalau dilihat bentuknya sangatlah jauh dari pohon,ini termasuk pertanyaan yang umum ditemui.

Struktur pohon ini memang tidak nampak jelas, karena ini lebih berupa struktur pohon terbalik, di sini setidaknya ada tiga komponen yang perlu diketahui,

1. **root node**

   Karena struktur pohon nya terbalik, maka root akan menempati posisi paling atas.

   Untuk kasus kita disini, pertanyaan **Fear technology** ini merupakan root node nya.

2. **leaf node**

   Yang perlu diketahui adalah sejumlah node yang posisinya di ujung bawah, karena struktur pohon yang terbalik komponen ini dikenal sebagai leaf node.

   Untuk kasus kita disini, leaf nya adalah **Apple, Chrome OS,Ubuntu Linux, Arch Linux dan Windows**, disini kita memiliki lima buah leaf, setiap leaf dari Decision Tree akan merepresentasikan prediksi class yang akan dihasilkan oleh struktur tree tersebut.

3. **internal node**

   Sejumlah node yang berada diantara root dan leaf ini biasa dikenal sebagai internal node, atau kadang disebut sebagai node.

   Untuk kasus kita disini notnya ada tiga,yaitu **Your dad rich?,Care about privacy?,** dan **Have live?.**

Dalam machine-learning terdapat beberapa Decision Tree algorithms atau algoritma Decision Tree, dan yang akan kita pelajari kali ini adalah Classification and Regression Tree, atau biasa disingkat sebagai CART, ini merupakan default algorithms yang diterapkan oleh scikit learn sebagai implementasi Decision Tree.

Selain CART terdapat beberapa algorithms lain yang cukup umum digunakan yaitu :

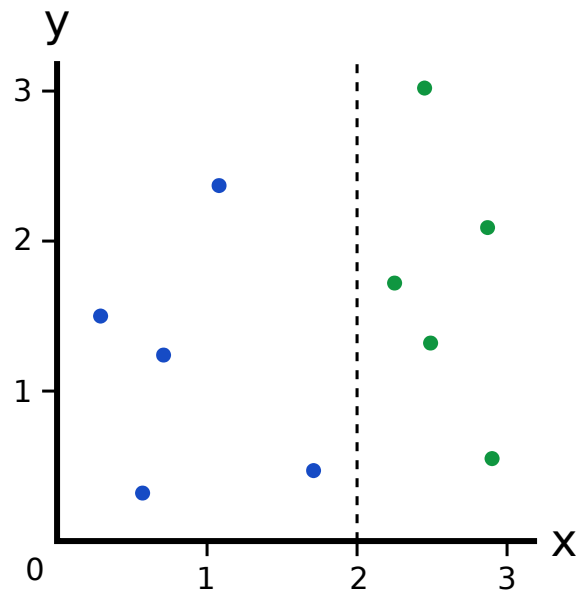
- ID3
- C4.5
- C5.0

◆ Gini Impurity

Selanjutnya kita akan mempelajari Impurity Measure, atau pengukuran ketidakmurnian, karena disini kita mengacu pada CART sebagai Decision Tree algorithms, maka kita perlu mempelajari Gini Impurity, ini merupakan Impurity Measure yang diterapkan pada algoritma CART.

Gini Impurity memiliki jangkauan nilai antara 0 dan , dimana nilai 0 mengindikasikan nilai murni yang sempurna,sedangkan nilai 1 mengindikasikan nilai paling impur atau paling tidak murni.

Untuk lebih jelasnya kita akan pergunakan contoh kasus berikut ini



Dalam kasus ini terdapat 10 datapoints yang terbagi dalam dua class, yaitu class biru dan class hijau, selanjutnya disini kita akan pisahkan data nya ke dalam dua bagian, sehingga dihasilkan dua buah ruas, sebut saja ruas kiri dan ruas kanan, kalau kita lihat disini garis putus-putus ini berperan sebagai Splitter atau pemisahnya,setelah datanya kita split maka akan terdapat dua ruas sebut saja ruas kiri dan ruas kanan.

Berikutnya kita akan lakukan pengukuran impurity pada kedua ruas ini, pertama-tama kita akan lakukan pengukuran pada ruas kiri terlebih dahulu.

Berikut adalah formula dari Gini Impurity.

Ruas Kiri:

$$G = 1 - \sum_{i}^{n} P_i^2$$
$$= 1 - P(biru)^2$$
$$= 1 - \left(\frac{4}{4}\right)^2 = 0$$

Jadi nilai Gini Impurity ini bisa kita peroleh dari 1 di selisihkan dengan total hasil penjumlahan probability kuadrat dari tiap class yang tersedia, untuk kasus kita disini ruas sebelah kirinya hanya menampung data class biru saja, artinya disini nilai probability yang kita hitung hanya nilai probability dati class biru saja, oleh karenanya proses kalkulasinya adalah 1 di selisihkan dengan nilai probability dari class biru lalu kita kuadratkan, nilai probability dari class birunya, bisa kita cari dengan cara berikut ini.

Kalau kita lihat di sini total jumlah datapoint nya ada 4, dan yang berwarna biru ada 4 juga, oleh karenanya di sini kita bisa hitung dengan cara 4 dibagi 4,4 yang pertamanya adalah jumlah yang berwarna biru, lalu 4 yang dibawah ini merupakan total keseluruhan data yang berada di ruas kiri,lalu nilainya kita pangkatkan 2,4 dibagi 4 ini kan 1,1 dikuadratkan adalah 1, artinya 1 di selisihkan dengan 1, dan hasilnya adalah 0.

Bisa kita lihat di sini nilai Giny Impurity nya adalah 0, yang mengindikasikan kemurniaan sempurna, dan ini enggak mengherankan karena untuk ruas sebelah kiri ini murni berisi data-data yang berada

dalam class berwarna biru, atau dengan kata lain disini kita bisa bilang bahwa ruas kiri ini sudah murni berisi class berwarna biru.

Ruas Kanan:

Selanjutnya kita lakukan pengukuran serupa pada ruas sebelah kanan, kalau kita perhatikan pada ruas sebelah kanan disini terdapat dua class, ada sebagian data yang berwarna biru, dan ada sebagian data yang berwarna hijau, oleh karenanya di sini proses kalkulasinya adalah sebagai berikut:

$$G = 1 - \sum_{i}^{n} P_i^2$$
$$= 1 - \left(P(biru)^2 + P(hijau)^2\right)$$
$$= 1 - \left(\left(\frac{1}{6}\right)^2 + \left(\frac{5}{6}\right)^2\right) = 0.278$$

1 diselisihkan dengan penjumlahan antara nilai probability biru kuadrat dengan nilai probability hijau kuadrat.

kita akan hitung dulu nilai probability birunya, kalau kita lihat di sini total jumlah datanya ada 6, yang berwarna biru itu ada 1, artinya nilai probability untuk data biru adalah 1/6 dan ini kita kuadratkan, karena nilai probability biru kuadrat, berikutnya nilai probability hijaunya adalah 5,dan kita bagi dengan total keseluruhan datanya ada 6,dan nilai probability hijaunya juga kita kuadratkan, dan kalau kita kalkulasikan disini hasilnya adalah 0.278.

Setelah kita mendapatkan Gini Impurity untuk kedua ruas, selanjutnya kita perlu mencari nilai rata-ratanya, berikut akan kami demokan proses kalkulasinya.

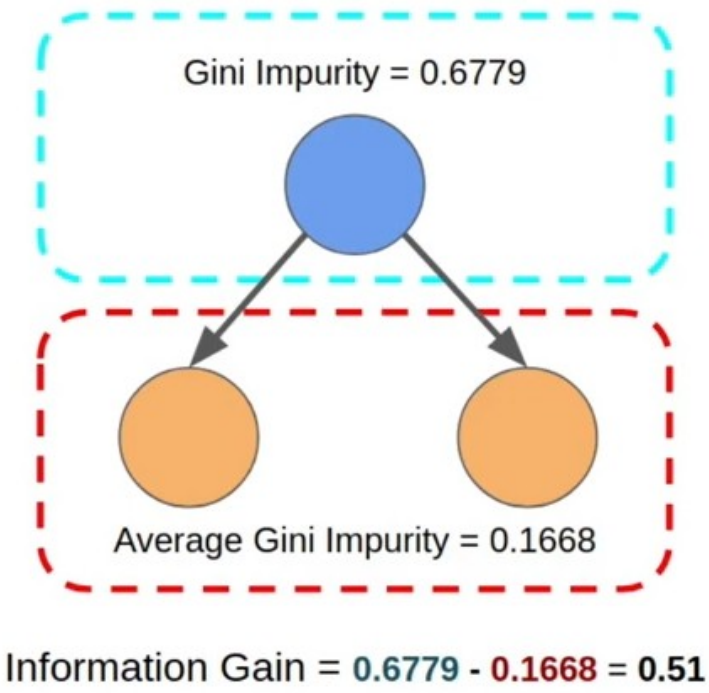
$$G = \frac{4}{4+6} \times 0 + \frac{6}{4+6} \times 0.278$$
$$= 0.1668$$

Disini kita lihat dulu untuk sisi sebelah kiri, total keseluruhan data yang ada adalah 10, karena dikiri 4 dan di kanan 6, dan untuk ruas sebelah kiri disini terdapat 4 data point, artinya nilai Gini Impurity dari sisi sebelah kiri ini harus kita kalikan dengan 4 per 10, di sini 4 per 4 tambah 6, karena 4 yang ada di sini merupakan jumlah data di sisi sebelah kiri, dan nilai 6 disini merupakan jumlah data di sisi sebelah kanannya, 4 per-10 dikali 0, dan nilainya adalah 0, nilai ini akan kita jumlahkan dengan sisi ruas sebelah kanannya, di ruas sebelah kanan ini terdapat 6 data poin dari total 10 datapoint, dan nilainya

kita kalikan dengan Gini Impurity dari ruas sebelah kanan yaitu 0.278, dan hasilnya adalah 0.1668, dan ini merupakan average Giny Impurity atau nilai rata-rata Gini Impurity nya.

◆ Information Giny

Information Gain ini juga merupakan topik bahasan yang sangat penting untuk dipahami sewaktu mempelajari Decision Tree, sebelumnya kita sudah mempelajari proses kalkulasi untuk mendapatkan nilai rata-rata Gini Impurity dari suatu proses spliting.



Ini merupakan bagian yang kita pelajari sebelumnya, dimana yang kita cari adalah nilai rata-rata dari Giny Impurity setelah dilakukan proses splitting,dan berdasarkan hasil kalkulasi kita sebelumnya, nilai Giny Impurity adalah 0.1668.

Tetapi kalau kita cermati,sebenarnya sebelum dilakukan proses spliting kita pun bisa menghitung nilai Giny Impurity dari node sebelumnya,semisal saja untuk kasus kita disini nilai Gini Impurity nya adalah

0.6779, ini hanya sekedar contoh aja, berarti disini merupakan nilai Gini Impurity sebelum proses spliting, dan untuk 0.1668 ini adalah nilai average Giny Impurity setelah proses splitting.

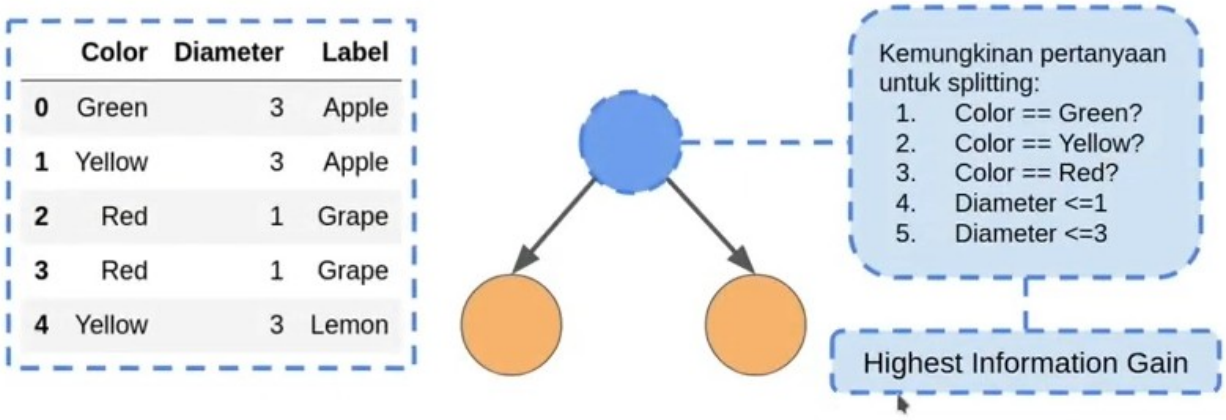
Dengan mengetahui kedua nilai ini, selanjutnya kita bisa kalkulasikan nilai Information Gain nya, Information Gain bisa kita peroleh dengan menyelesaikan nilai Gini Impurity sebelum proses splitting, dengan nilai rata-rata Giny Impurity setelah proses splitting, untuk kasus kita disini nilai Gini Impurity sebelum proses spliting adalah 0.6779,dan nilainya kita selesaikan dengan nilai rata-rata Gini Imp[urity setelah proses splitting yaitu 0.1668 dan hasilnya adalah 0.51.

Sampai di titik ini sebagian dari kalian mungkin ada yang mulai bertanya-tanya lalu apa peran dari Information Gini ini?,dan jawabannya ada pada topik bahasan selanjutnya.

◆ Membangun Decision Tree

Selanjutnya kita akan mempelajari proses pembentukan Decision Tree serta peranan dari Information Gain yang baru saja kita pelajari sebelumnya.

Untuk lebih jelasnya kita akan gunakan studi kasus berikut ini.



Di sini kita di hadapkan pada suatu dataset sederhana yang terdiri dari 5 baris dan 3 kolom, 2 kolom pertama yaitu kolom Color, dan kolom Diameter, akan berperan sebagai features, lalu kolom ke-3 akan berperan sebagai target label.

Dari dataset ini kita akan membentuk suatu Decision Tree, disini terdapat sejumlah pilihan pertanyaan yang bisa kita gunakan untuk melakukan splitting.

Berikut merupakan daftar kemungkinan pertanyaan yang bisa digunakan untuk melakukan spliting dari kondisi dataset semacam ini.

1. Apakah Color nya green?
2. Apakah Color nya yellow?
3. Apakah Color nya red?
4. Apakah nilai Diameter nya lebih kecil sama dengan 1 (<=1)?
5. Apakah nilai Diamer nya lebih kecil sama dengan 3 (<=3)?

Setiap pertanyaan ini berpotensi untuk kita gunakan sebagai splitter atau pembagi dataset nya menjadi dua bagian.

Lalu bagaimana cara kita memilih satu dari sekumpulan pilihan pertanyaan yang ada,dan jawabannya adalah berdasarkan nilai Information Gain tertinggi.

Pertama-tama kita akan kalkulasikan terlebih dahulu nilai Gini Impurity dari Sekumpulan data sebelum proses splitting.

berikut adalah proses kalkulasinya.

$$
\begin{aligned}
G &= 1 - \left(P(apple)^2 + P(grape)^2 + P(lemon)^2\right) \\
&= 1 - \left(\left(\frac{2}{5}\right)^2 + \left(\frac{2}{5}\right)^2 + \left(\frac{1}{5}\right)^2\right) \\
&= 0.63
\end{aligned}
$$

Disini acuannya dari tabel di atas,berarti nilai Gini Impurity nya bisa kita peroleh dengan cara, kita lihat dulu disini disini terdapat tiga class, class apel,grape dan lemon, berarti kita harus kalkulasikan nilai probability dari apple lalu di pangkatkan 2,berikutnya kita juga perlu hitung nilai probability dari grape yang kita pangkatkan 2, dan terakhir kita juga perlu hitung nilai probability dari lemon yang kita pangkat kan 2.

Nilai probability dari apple di sini apple muncul 2 kali berarti 2, lalu dibagi dengan total keseluruhan jumlah datanya yaitu 5 dan nilainya kita pangkatkan 2, berikutnya grape disini juga muncul 2 kali dengan total jumlah datanya 5,beratrti 2/5, berikutnya kita pangkatkan 2, dan terakhir, nilai probability untuk lemon muncul hanya sekali 1 per dengan total jumlah data 1, berarti 1/5 dan nilai probability nya kita pangkatkan 2, untuk selanjutnya hasil penjumlahan ini akan kita gunakan untuk menselisihkan dengan nilai 1, berarti 1 di selisihkan dengan keseluruhan total penjumlahan ada di sini,dan kita peroleh nilai 0.603.

Yang baru saja kita kalkulasikan disini adalah nilai Gini Impurity dari sekumpulan data sebelum proses spliting, itu berarti merupakan lingkaran yang berwarna biru.

Selanjutnya kita akan spliting data ini dengan menerapkan setiap kemungkinan pertanyaan yang ada satu persatu,lalu untuk setiap hasil spliting nya kita akan kalkulasikan nilai rata-rata Gini Impurity nya untuk kemudian kita hitung Information Gain nya.

Pertanyaan yang menghasilkan nilai Information Gain tertinggi lah yang akan kita pilih untuk melakukan proses spliting data,proses ini pun akan terus berlangsung secara iteratif sampai setiap node menghasilkan nilai Gini Impurity 0, atau ketika ke dalaman tree nya sudah mencapai batasan yang di tetapkan.

◆ Implementasi Decision Tree Classification Task

selanjutnya kita akan pelajari Implementasi Decision Tree on the Classification Task dengan scikit learn, pertama-tama kita akan siapkan terlebih dahulu dataset nya.

Dataset

Untuk kasus kali ini,kita akan mengadopsi iris dataset yang sudah disertakan oleh scikit learn sebagai sample dataset, dataset ini juga pernah kita gunakan pada beberapa sesi pembelajaran sebelumnya.

Bagi kalian yang butuh untuk mempelajari dataset ini lebih lanjut, berikut adalah halaman Wikipedia yang bisa dijadikan referensi

Refrensi : https://en.wikipedia.org/wiki/Iris_flower_data_set

Perlu di ingat,dataset ini berbicara mengenai dataset bunga iris dimana terdapat tiga spesies bunga iris yaitu iris setosa, iris virginica,dan iris versicolor,dan tujuan yang mau dicapai disini adalah untuk menghasilkan model klasifikasi yang bisa digunakan untuk memprediksi spesies bunga iris ini.

Selanjutnya kita akan coba load dataset nya, berikut akan kami demokan prosesnya.

```
from sklearn.datasets import load_iris

X, y = load_iris(return_X_y=True)

print(f'Dimensi Features: {X.shape}')
print(f'CLass: {set(y)}')
```

Pertama-tama kita akan impor dulu modul nya, **from sklearn.datasets import** load_iris, berikutnya kita akan tanggil **load_iris** dan disini kita akan menyertakan parameter **return_X_y=True** ,konsekuensinya adalah kita harus menyiapkan dua buah variabel,kasus kita disini adalah variabel **X** ada **y**, dimana variabel **X** akan digunakan untuk menampung sekumpulan nilai features, dan variabel **y** akan digunakan untuk menampung sekumpulan nilai target labelnya, selanjutnya kita akan coba cetak ke layar dimensi dari features nya, di sini kita panggil **X.shape**, dan juga kita akan cetak ke layar class apa saja yang terdapat dalam target label nya.

Dan ini hasil nya

```
Dimensi Features: (150, 4)
CLass: {0, 1, 2}
```

Bisa nampak di sini dimensi data dari feature nya adalah **150**, **4,** artinya disini terdapat 150 baris dan 4 kolom, lalu terkait dengan class pada target labelnya disini terdapat tiga buah class yaitu class 0, 1 dan 2.

Selanjutnya kita akan bagi dataset ini ke dalam training dan testing set, berikut akan kami demokan prosesnya.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.3,
                                                    random_state=0)
```

Disini juga kita akan melakukan impor modul nya,**from sklearn.model_selection import** train_test_split, untuk selanjutnya kita akan panggil **train_test_split** nya dan kita sertakan **X**,dan **y** sebagai parameternya, lalu terkait parameter **test_size** kita set sebagai **0.3**,artinya disini testing set nya akan menempati 30% dari total keseluruhan dataset yang kita miliki,berikutnya untuk **random_state** kita beri nilai 0, pemanggilan fungsi **train_test_split** ini akan menghasilkan 4 Kumpulan data,dimana ke-4 data nya perlu kita tampung kedalam 4 buah variabel, untuk kasus disini kita tampung ke dalam **X_train**, **X_test**, **y_train** dan **y_test**.

Clasification dengan DecisionTreeClassifier

Berikutnya kita akan menerapkan Decision Tree untuk melakukan klasifikasi spesies iris, berikut akan kami demokan prosesnya.

```
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier(max_depth=4)
model.fit(X_train, y_train)
```

Pertama-tama disini kita akan impor dulu modul nya, **from sklearn.tree import** DecisionTreeClassifier, berikutnya kita akan bentuk objek dari class **DecisionTreeClassifier** dan disini kita akan menyertakan parameter yaitu parameter m**ax_depth**, dan kita set sebagai 4 , artinya DecisionTreeClassifier yang kita hasilkan itu memiliki kedalaman maksimum 4 layer, berikutnya objek dari **DecisionTreeClassifier** ini akan kita tampung ke dalam variabel **model** untuk berikutnya akan kita training dengan memanggil method Fit, proses training ini akan melibatkan **X_train** dan **y_train** sebagai training set nya.

Visualisasi Model

Yang menarik dari Decision Tree adalah kita bisa memvisualisasikan train model yang dihasilkan, berikut akan kami demokan prosesnya.

```
import matplotlib.pyplot as plt
from sklearn import tree

plt.rcParams['figure.dpi'] = 85
plt.subplots(figsize=(10, 10))
tree.plot_tree(model, fontsize=10)
plt.show()
```

Disini setidaknya ada dua buah modul yang perlu kita import, **import matplotlib.pyplot as plt** dan **from sklearn import** tree, selanjutnya pertama-tama disini kami akan atur **dpi** dari **figure** yang akan dihasilkan, untuk kasus kali ini kita set **dpi** nya adalah **85**, jika ini masih terlalu kecil,kalian bisa sesuaikan nilainya sesuai dengan kebutuhan kalian masing-masing, selanjutnya kita akan siapkan figure nya, di sini kita panggil plt.subplots(figsize=(**10**, **10**)), berikutnya untuk melakukan floating model Decision Tree kita bisa panggil **tree.plot_tree**, lalu kita panggil trainmodel nya,dan disini kita juga

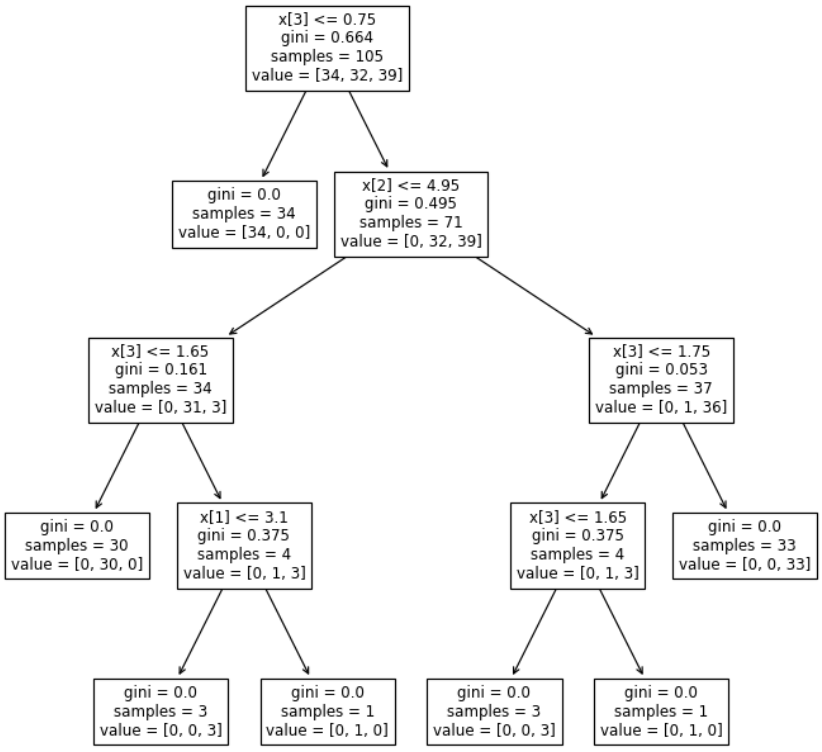
sertakan parameter lain yaitu **font_size** yang kita beri nilai **10** dan terakhir di sini kita tinggal tampilkan dengan cara plt.show().

Kita coba eksekusi kode nya

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
from sklearn import tree

X, y = load_iris(return_X_y=True)

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.3,
                                                    random_state=0)


model = DecisionTreeClassifier(max_depth=4)
model.fit(X_train, y_train)
plt.rcParams['figure.dpi'] = 85
plt.subplots(figsize=(10, 10))
tree.plot_tree(model, fontsize=10)
plt.show()

Press ENTER or type command to continue
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
from sklearn import tree

X, y = load_iris(return_X_y=True)

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.3,
                                                    random_state=0)


model = DecisionTreeClassifier(max_depth=4)
model.fit(X_train, y_train)
plt.rcParams['figure.dpi'] = 85
```

```
plt.subplots(figsize=(10, 10))
tree.plot_tree(model, fontsize=10)
plt.show()
```

dan ini hasil nya



Ini merupakan hasil visualisasi dari Decision Tree yang di hasilkan, disini untuk setiap node nya terdapat beberapa informasi, informasi yang paling atas di sini merupakan kriteria spliting nya, untuk root node nya bisa kita lihat splitting kriterianya adalah **X[3] <= 0.75**,artinya disini kita memanfaatkan features indeks ke-3, dan syaratnya adalah atau kriterianya adalah,apakah nilainya lebih kecil sama dengan 0.75.

Berikutnya disini juga terdapat **gini**, gini nya adalah **0.664**, gini disini merupakan Gini Impurity sebelum proses splitting, berikutnya terdapat juga keterangan **samples = 151,**artinya jumlah data sebelum proses spliting ada 105.

Selanjutnya terdapat **value,** dan nilai value nya ada 3,**32,34,** dan **39**,artinya untuk class indeks ke-0 itu terdapat **34** data, untuk class ini ke-1 itu terdapat **32** data, dan untuk class indeks ke-2 itu terdapat **39** data, dan ini merupakan kondisi sebelum dikenakan proses splitting,lalu setelah kita kenakan proses splitting dengan kriteria yang ada di sini,maka akan dihasilkan dua buah ruas.

Bisaa kita disini untuk ruas sebelah kiri, nilai Gini Impurity nya adalah 0, artinya disini sudah murni 100%, dimana jumlah sampel datanya ada 34 lalu untuk value nya disini 34.0.0,artinya keseluruhan data yang ada di sini sudah sepenuhnya merupakan data dari class indeks ke-0.

Berikutnya untuk ruas sebelah kanan, di sini masih perlu dilakukan proses splitting lebih lanjut karena nilai Gini Impurity nya adalah 0.495, dan jumlah sampel datanya ada 71, lalu distribusi value-nya atau distribusi datanya adalah 0,32, 39, artinya untuk class dengan indeks ke-1 disini terdapat 32 data, dan untuk class dengan indeks ke-2 disini terdapat 39 data.

Lalu terkait proses spliting selanjutnya ini akan memanfaatkan kriteria berikut, features indeks ke-2 akan dicek apakah lebih kecil sama dengan 4.95 dan seterusnya, dan bisa kita lihat di sini juga tingkat kedalaman maksimumnya adalah 4,atau dengan kata lain tingkat pertanyaannya itu maksimum ada 4 tingkat atau 4 layer.

◆ Evaluasi Model

Selanjutnya kita akan lakukan evaluasi performa dari model Decision Tree yang baru saja kita training tadi, disini kita akan memanfaatkan classification repot.

```
from sklearn.metrics import classification_report

y_pred = model.predict(X_test)

print(classification_report(y_test, y_pred))
```

Pertama-tamakita akan impor modulnya, **from sklearn.metrics import** classification_report, berikutnya kita akan melakukan prediksi **model.predict**, dengan memanfaatkan testing set kita, dan hasil prediksinya akan kita tampung ke dalam variabel **y_pred**, untuk selanjutnya akan kita evaluasi dengan menggunakan classification repott, disini parameter yang dilewatkan adalah **y_test** dan **y_pred**.

Kita eksekusi kode nya

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.metrics import classification_report

X, y = load_iris(return_X_y=True)

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.3,
                                                    random_state=0)


model = DecisionTreeClassifier(max_depth=4)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print(classification_report(y_test, y_pred))
```

dan ini hasil nya

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      0.94      0.97        18
           2       0.92      1.00      0.96        11

    accuracy                           0.98        45
   macro avg       0.97      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```

Bisa kita lihat disini terdapat kolom nilai precision,recall, dan f1-score,disini juga terdapat nilai total accuracy nya

◆ Bab 16 :Random Foresst Classification

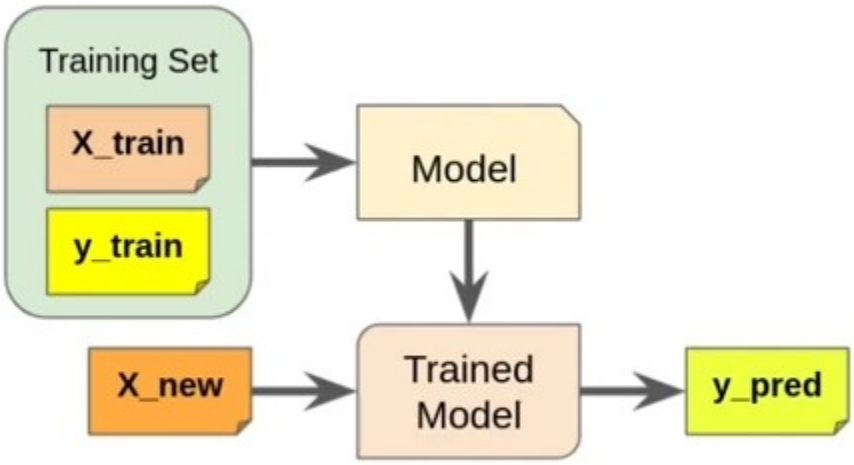
Random Forest ini termasuk model machine-learning yang umum ditemui dan merupakan pengembangan dari model Decision Tree yang sudah kita pelajari sebelumnya.

Bagi kalian yang tertarik untuk mempelajari Random Forest lebih lanjut berikut adalah halaman Wikipedia yang bisa dijadikan referensi.

Refrensi : https://en.wikipedia.org/wiki/Random_forest

◆ General ML Model Training

Sebelum kita membahas mekanisme kerja dari Random Forest, pertama-tama kita akan coba review terlebih dahulu general workflow dari proses training model dalam machine learning.



Proses training selalu diawali dengan Training set, dimana didalamnya terdiri dari sekumpulan teknik features atau biasa direpresentasikan sebagai X_train, dan juga sekumpulan target label yang biasa direpresentasikan sebagai y_t rain,training set ini kita gunakan untuk melakukan training model machine-learning, dan model yang sudah di training ini akan dikenal dengan istilah train model,selanjutnya train model ini akan kita gunakan untuk melakukan prediksi terhadap sekumpulan

nilai features yang baru,dalam ilustrasi disini sekumpulan nilai features baru direpresentasikan sebagai X_new,dan prediksi yang dihasilkan oleh train model ini akan direpresentasikan sebagai y_pred.

Ini merupakan general workflow yang sudah kita pelajari sejauh ini terkait proses training model machine-learning, kalau kita perhatikan disini kita hanya memanfaatkan satu model machine learning saja.

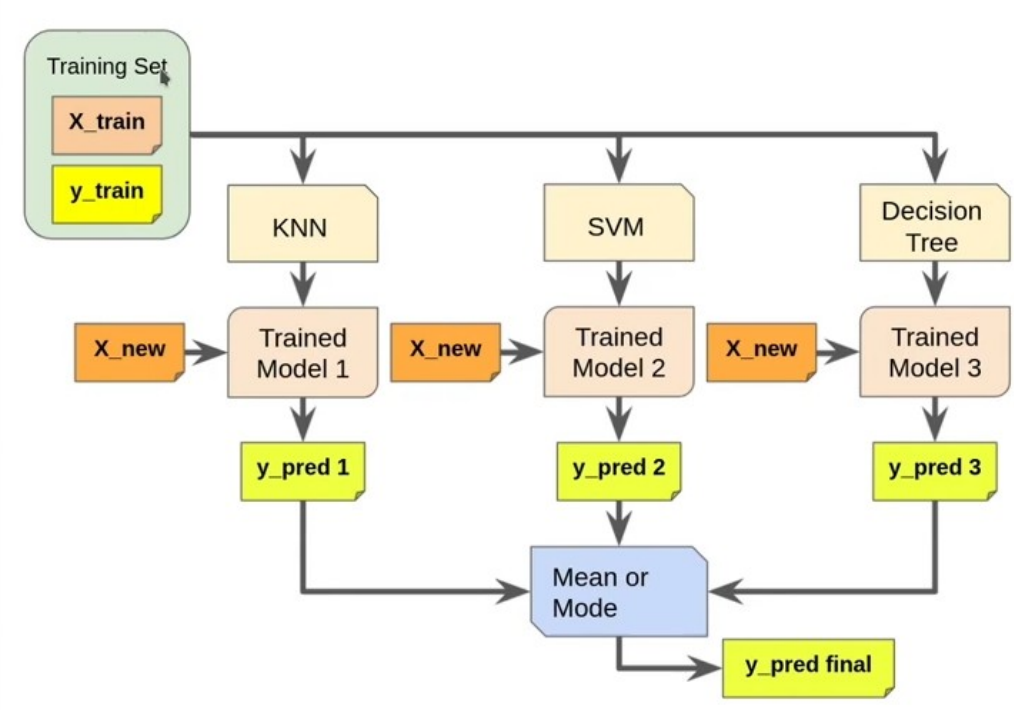
Berikutnya kita akan mempelajari suatu teknik dimana kita bisa menggabungkan beberapa model machine learning.

◆ Ensemble Learning: heterogeneous & homogeneous

Ensemble Learning ini merupakan suatu teknik yang dikenal dalam machine learning dimana kita menggabungkan beberapa model untuk melakukan prediksi.

Bagi kalian tertarik untuk mempelajari Ensemble Learning lebih lanjut, berikut adalah halaman Wikipedia yang bisa dijadikan referensi.

Refrensi : https://en.wikipedia.org/wiki/Ensemble_learning



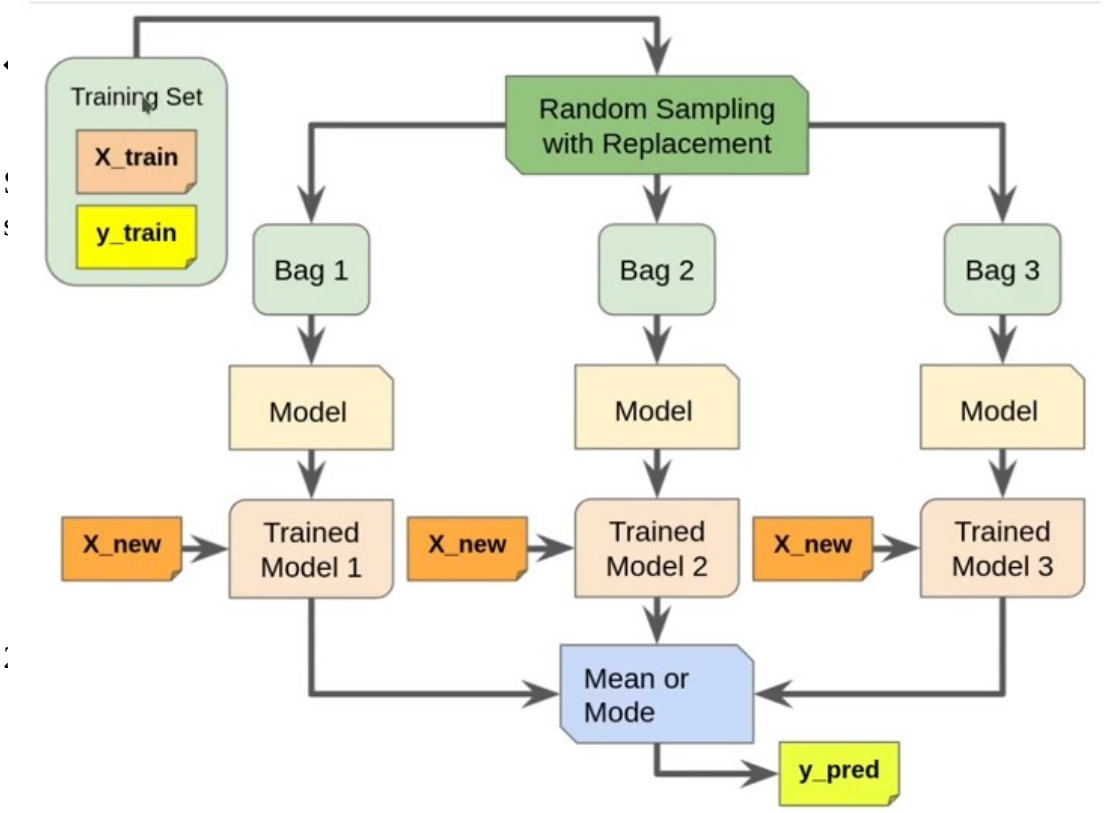
Ensemble learning juga diawali dari suatu training set,hanya saja kali ini training set yang tersedia akan kita gunakan untuk men training beberapa model, untuk kasus disini terdapat 3 model machine-learning yang akan kita training yaitu KNN, SVM dan DecisionTree, proses training ini menghasilkan 3 Train_Model, yaitu Trained_Model 1, Trained_Model 2 dan Trained_model 3.

Selanjutnya setiap Tained_Model ini akan kita gunakan untuk melakukan prediksi terhadap sekumpulan nilai features yang baru, dimana prediksi yang dihasilkan oleh setiap Tained_Model ini direpresentasikan sebagai y_pred 1,y_pred 2 dan y_pred 3, y_pred 1 ini merupakan hasil prediksi yang dihasilkan oleh Train_Model 1 terhadap X_New,lalu y_pred 2 nya merupakan hasil prediksi yang dihasilkan oleh Train_Model 2 terhadap X_new,demikian juga y_pred 3 ini merupakan hasil prediksi yang dihasilkan oleh Trained_Model 3 terhadap X_New.

Disini karena terdapat lebih dari 1 hasil prediksi, maka perlu disatukan dengan suatu mekanisme yang dikenal dengan istilah majority voting, pada kasus Regression Task, majority voting dilakukan dengan menggunakan nilai mean atau nilai rata-rata, sedangkan pada kasus Classification Task, majority voting dicapai dengan menerapkan Mode atau kemunculan terbanyak.

Salam ilustrasi ini hasil prediksi final direpresentasikan sebagai y_pred final,atau dengan kata lain disini y_pred 1,y_pred 2 dan y_pred 3, ini akan kita satukan dengan menggunakan majority voting sehingga dihasilkan y_pred final.

Karena disini kita memadukan sejumlah model machine-learning yang berbeda,maka seringkali juga dikenal sebagai heterogeneous Ensemble Learning,atau model Ensemble Learning yang heterogen,sedangkan untuk kasus dimana kita memadukan sejumlah model machine learning yang sejenis akan dikenal dengan istilah homogeneous Ensemble learning, atau model Ensemble Learning yang homogen.



Kalau kita perhatikan di sini, model machine-learning nya sejenis,jadi ketiga model menggunakan algoritma yang sama.

Proses ini juga diawali dengan training set, hanya saja di sini karena kita menerapkan sejumlah model yang sejenis, maka melakukan training terhadap setiap model menggunakan training set yang sama bisa dibilang sia-sia karena akan menghasilkan Trained_Model yang sama persis, disini kita akan berkenalan dengan suatu teknik yang dikenal dengan istilah bootstrap aggregating, atau biasa disingkat sebagai bagging.

Bagi kalian yang tertarik untuk mempelajari bagging lebih lanjut, berikut adalah halaman Wikipedia yang bisa dijadikan referensi.

Refrensi : https://en.wikipedia.org/wiki/Bootstrap_aggregating

Berikutnya kita akan bahas bagging lebih lanjut serta penerapannya pada homogeneous Ensemble Learning.

Mekanisme bagging pada dasarnya menerapkan proses Random Sampling with Replacement terhadap training set yang kita miliki, dan akan menghasilkan beberapa training set baru sejumlah model yang akan kita training.

Bagi kalian yang masih awam dengan konsep Random Sampling with Replacement kami sangat menyarankan untuk mempelajarinya terlebih dahulu.

Training set baru yang dihasilkan oleh proses bagging ini dikenal dengan istilah Bag,karena pada kasus ini kita akan mentraining 3 buah model, maka kita juga perlu persiapkan 3 buah Bag, dalam hal ini adalah Bag 1, Bag 2 dan Bag 3, ketiga Bag ini berisi sekumpulan sample data yang berbeda satu dengan lainnya karena ketiga Bag ini dihasilkan dari proses Random Sampling with Replacement yang diambil dari training set,selanjutnya setiap Bag ini akan kita gunakan untuk men training 3 model machine-learning yang sejenis.

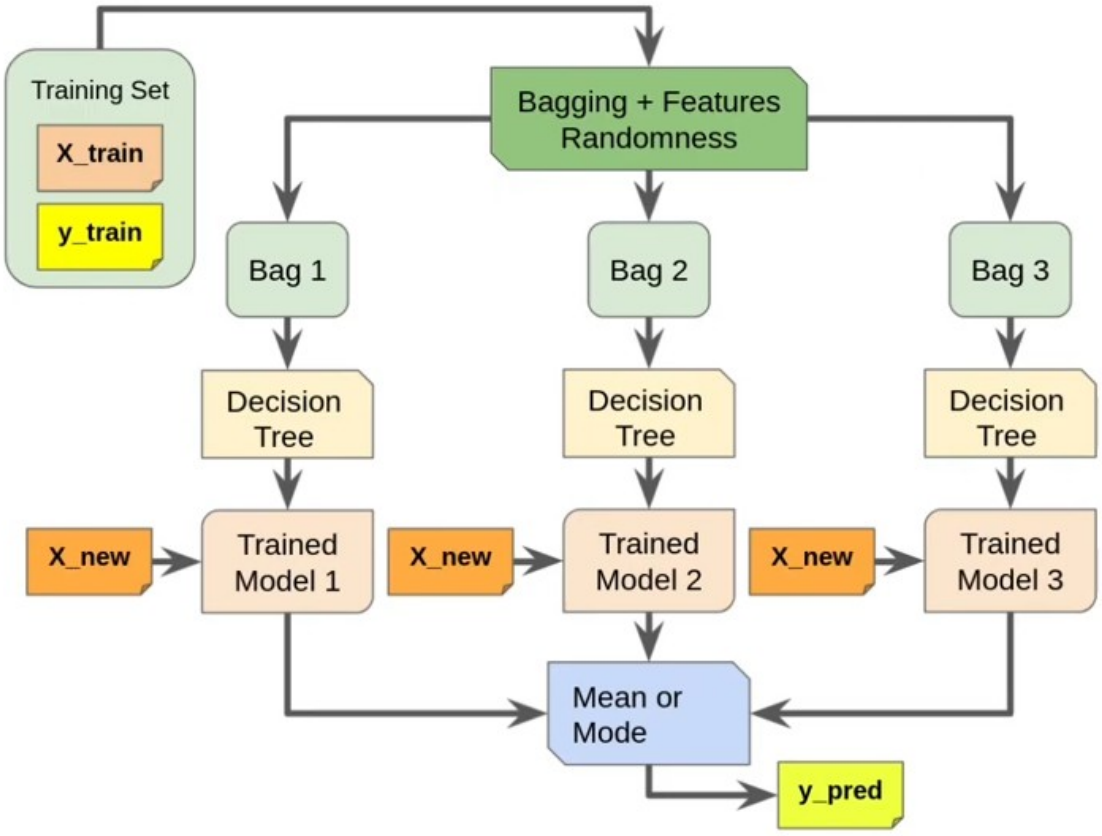
Disini Bag 1 digunakan untuk men training machine-learning,Bag 2 juga digunakan men training model machine-learning, demikian pula Bag 3, dan model machine learning yang kita training disini merupakan model machine-learning yang sejenis,lalu setelah melalui proses training maka akan dihasilkan Trained_Model, disini akan dihasilkan tiga Trained_Model yaitu Trained_Model 1,Trained_Model 2 dan Trained_Model 3, setiap Trained_Model ini berikutnya akan kita gunakan untuk melakukan prediksi terhadap sekumpulan nilai features yang baru,hasil prediksi yang dihasilkan oleh setiap Trained_Model selanjutnya akan kita satukan melalui proses majority voting untuk dihasilkan prediksi final.

Disini perlu di inngatkan lagi bahwa setiap Trained_Model ini akan menghasilkan hasil prediksi yang berbeda-beda, karena kita memiliki 3 buah Trained_Model berarti kita juga akan memiliki 3 buah hasil prediksi, dan ketiga hasil prediksi nya perlu kita satukan untuk menghasilkan hasil prediksi fina.

Sampai di titik ini sebagai dari kalian mungkin ada yang mulai bertanya-tanya,mengapa kita perlu mempelajari semua hal ini, dan jawabannya adalah karena Random Forest sebenarnya merupakan salah satu bentuk implementasi dari homogeneous Ensemble Learning.

◆ Random Forest

Random Forest meperupakan implementasi dari homogeneous Ensemble learning yang menerapkan DecisionTree.



Bisa nampak di sini model yang akan kita training ini model yang sejenis, dan untuk kasus disini adalah DecisionTree.

Disini mestinya kalian sudah bisa menangkap asal kata Random Forest atau hutan acak ini, karena pada dasarnya Random Forest ini terbentuk dari sekumpulan DecisionTree atau pohon keputusan.

Yang perlu ditekankan disini adalah selain bagging ,Random Forest juga menerapkan features Randomness, dimana untuk setiap Bag yang dihasilkan akan mengadopsi sejumlah features yang dipilih secara acak dari training set sumbernya, atau dengan kata lain disini setiap model DecisionTree akan di training dengan Bag yang berisi dataset yang beragam.

Tidak hanya dalam hal baris data saja melainkan juga beragam dalam hal features yang disertakan,berarti di sini Bag 1, Bag 2 dan Bab 3, selain mengusung baris data yang berbeda, mereka juga akan mengusung features yang berbeda yang mereka pilih secara acak dari training set sumbernya, karena setiap DecisionTree ini di training dengan menggunakan dataset yang berbeda,maka dihasilkan lah sejumlah Trained_Model yang beragam juga, dalam hal ini adalah Trained_Model 1, Trained_Model 2 dan Trained_Model 3.

Semua Trained_Model ini sebenarnya menggunakan model machine-learning yang sejenis yaitu DecisionTree, setiap Trained_Model ini selanjutnya akan kita gunakan untuk melakukan prediksi terhadap sekumpulan nilai features yang baru, prediksi yang dihasilkan oleh setiap Trained_Model akan disatukan melalui proses majority voting untuk menghasilkan nilai prediksi final, strategi semacam ini menjadikan Random Forest memiliki performa yang jauh lebih superior bila dibandingkan model DecisionTree yang biasa, karena setiap tree yang ada di training menggunakan subset dari keseluruhan training set yang tersedia.

◆ implementasi Random Forest dengan Scikit Learn

Selanjutnya kita akan mempelajari implementasi Random Forest ini dengan menggunakan scikit learn.

Pertama-tama disini kita akan siapkan terlebih dahulu dataset nya.

Dataset

Untuk kasus kali ini kita akan mengadopsi iris dataset yang sudah disertakan oleh scikit learn sebagai sample dataset, dataset iris sini juga pernah kita gunakan pada beberapa sesi pembelajaran sebelumnya.

Selanjutnya kita akan coba load dataset nya, berikut akan kami demokan prosesnya.

```
from sklearn.datasets import load_iris

X, y = load_iris(return_X_y=True)

print(f'Dimensi Features: {X.shape}')
print(f'CLass: {set(y)}')
```

Pertama-tama kita akan impor dulu modul nya, **from sklearn.datasets import** load_iris, berikutnya kita akan tanggil **load_iris** dan disini kita akan menyertakan parameter **return_X_y=True** ,konsekuensinya adalah kita harus menyiapkan dua buah variabel,kasus kita disini adalah variabel **X** ada **y**, dimana variabel **X** akan digunakan untuk menampung sekumpulan nilai features, dan variabel **y** akan digunakan untuk menampung sekumpulan nilai target labelnya, selanjutnya kita akan coba cetak ke layar dimensi dari features nya, di sini kita panggil **X.shape**, dan juga kita akan cetak ke layar class apa saja yang terdapat dalam target label nya.

Dan ini hasil nya

```
Dimensi Features: (150, 4)
CLass: {0, 1, 2}
```

Dan ini dimensi features atau dimensi dari variabel **X** nya adalah terdiri dari 150 baris dan 4 buah kolom,lalu terkait dengan target label nya disini terdapat tiga class atau tiga nilai class yang berbeda, yaitu class 0, 1 dan class 2, dimana setiap class ini merepresentasikan spesies dari iris dataset.

Selanjutnya kita akan bagi dataset ini ke dalam training dan testing set, berikut akan kami demokan prosesnya.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.3,
                                                    random_state=0)
```

Disini juga kita akan melakukan impor modul nya,**from sklearn.model_selection import** train_test_split, untuk selanjutnya kita akan panggil **train_test_split** nya dan kita sertakan **X**,dan **y** sebagai parameternya, lalu terkait parameter **test_size** kita set sebagai **0.3**,artinya disini testing set nya akan menempati 30% dari total keseluruhan dataset yang kita miliki,berikutnya untuk **random_state** kita beri nilai 0, bagi kalian yang bingung dengan istilah random_state, random_state ini pada dasarnya

adalah random set number, dan ini kita gunakan untuk menjamin agar eksperimen kita ini bisa kita replicated dan menghasilkan nilai yang konsisten,berikutnya pemanggilan fungsi **train_test_split** ini akan menghasilkan 4 Kumpulan data,dimana ke-4 data nya perlu kita tampung kedalam 4 buah variabel, untuk kasus disini kita tampung ke dalam **X_train**, **X_test**, **y_train** dan **y_test**.

Sekarang kita sudah memiliki features untuk training set dan juga features untuk testing set nya, kita juga memiliki target label untuk training set dan juga target label untuk testing setnya.

Classification dengan RandomForestClassifier

Berikutnya disini kita akan menerapkan Random Forest untuk melakukan klasifikasi spesies Iris, berikut akan kami demokan prosesnya.

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100,
                               random_state=0)

model.fit(X_train, y_train)
```

Mekanismenya cukup sederhana, pertama-tama kita akan impor dulu modulnya, **from sklearn.ensamble import** RandomForestClassifier karena kita ingin melakukan klasifikasi, tetapi perlu juga diingat di sini Random Forest juga bisa digunakan untuk regression, untuk kasus regression task,, modul yang perlu kalian import adalah RandomForestRegression, selanjutnya kita akan bentuk objek dari RandomForestClassifier, disini kita juga akan sertakan dua buah parameter,yaitu **n_estimators** yang kita beri nilai **100.**

Seperti kita pelajari sebelumnya, RandomForest itu terbentuk dari sejumlah DecissionTree, di sini Kita tentukan berapa banyak DecissionTree yang mau kita training, untuk kasus kita kali ini kita akan men training sejumlah 100 buah model DecissionTree,oleh karenanya di sini kita set nilai nya 100, lalu agar eksperimen kita ini bisa kita replicated dengan hasil yang konsisten,kita juga perlu set random_state atau renovasi tambahnya, untuk kasus kita disini kita set adalah **0**, objek yang terbentuk ini berikutnya akan kita tampung ke dalam variabel **model** untuk selanjutnya model ini akan kita trainig dengan memanggil method Fit dan kita sertakan **X_train** dan **y_train** nya.

Evaluasi Model

Selanjutnya kita lakukan evaluasi performa dari model Random Forest yang baru saja kita training tadi kita akan memanfaatkan Classification Report

```
from sklearn.metrics import classification_report

y_pred = model.predict(X_test)

print(classification_report(y_test, y_pred))
```

Pertama-tamakita akan impor modulnya, **from sklearn.metrics import** classification_report, berikutnya kita akan melakukan prediksi **model.predict**, dengan memanfaatkan testing set kita, dan hasil prediksinya akan kita tampung ke dalam variabel **y_pred**, untuk selanjutnya kita akan panggil classification repot, dan kita akan sertakan parameter **y_test** dan **y_pred** nya.

Kita eksekusi kode nya

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

X, y = load_iris(return_X_y=True)

#print(f'Dimensi Features: {X.shape}')
#print(f'CLass: {set(y)}')


X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.3,
                                                    random_state=0)

model = RandomForestClassifier(n_estimators=100,
                               random_state=0)

model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

Dan ini dia hasilnya

```
              precision    recall  f1-score   support

1.00      1.00      1.00        16
1.00      0.94      0.97        18
0.92      1.00      0.96        11

    accuracy                           0.98        45
   macro avg       0.97      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```

Ini merupakan classification report sebagai bentuk evaluasi terhadap model Random Forest yang baru saja kita training tadi.

◆ Penutup

Kami telah menempuh perjalanan panjang dalam mempelajari machine learning dengan Python dan library Scikit-Learn. Dari konsep dasar machine learning, teknik pra-pemrosesan data, hingga penerapan berbagai algoritma, kita telah menyelami berbagai aspek yang menjadikan bidang ini begitu menarik dan penuh potensi.

Belajar machine learning bukanlah perjalanan yang singkat atau mudah, tetapi setiap langkah yang Anda ambil membawa Anda lebih dekat untuk memahami dan menguasai kemampuan untuk membangun model yang dapat membuat prediksi akurat dan memberikan wawasan berharga dari data.

Di dunia yang semakin didorong oleh data ini, keahlian dalam machine learning tidak hanya membuka pintu ke berbagai peluang karir, tetapi juga memungkinkan Anda untuk berkontribusi dalam menyelesaikan masalah nyata yang kompleks.

Kami berharap ebook ini memberikan dasar yang kuat bagi Anda untuk terus belajar dan mengeksplorasi lebih dalam. Ingatlah bahwa praktik dan eksperimen terus menerus adalah kunci untuk menjadi mahir dalam machine learning. Jangan ragu untuk mencoba hal-hal baru, mengikuti perkembangan terbaru di bidang ini, dan bergabung dengan komunitas yang ada.

Terima kasih telah mengikuti materi dalam ebook ini. Semoga ilmu yang telah Anda peroleh dapat bermanfaat dan menjadi langkah awal dalam perjalanan Anda di dunia machine learning.

Selamat belajar dan sukses selalu!

◆ Sumber dan Referensi

- https://en.wikipedia.org/wiki/Iris_flower_data_set
- https://en.wikipedia.org/wiki/Least_absolute_deviations
- https://en.wikipedia.org/wiki/Least_squares
- https://en.wikipedia.org/wikiSimple_linear-regression
- https://en.wikipedia.org/wiki/Coefisient_of_determination
- https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
- https://en.wikipedia.org/wiki/Euclidean_distance
- https://en.wikipedia.org/wiki/Precision_and_recall
- https://en.wikipedia.org/wiki/Matthews_correlation_coefficient
- https://en.wikipedia.org/wiki/One-hot
- https://en.wikipedia.org/wiki/Bag-of-words_model
- https://en.wikipedia.org/wiki/Stop_word
- https://en.wikipedia.org/wiki/Tf%E2%80%93idf
- https://scikit-learn.org/stable/modules/feature_extraction.html#test-feature-extraction
- https://en.wikipedia.org/wiki/logistic_regression
- https://en.wikipedia.org/wiki/Confusion_matrix
- https://en.wikipedia.org/wiki/Accuracy_and_precision
- https://en.wikipedia.org/wiki/Positive_and_negative_predictive_values
- https://en.wikipedia.org/wiki/Sensitivity_and_specificity
- https://en.wikipedia.org/wiki/F-score
- https://en.wikipedia.org/wiki/Receiver_operating_characteristic
- https://en.wikipedia.org/wiki/Naive_bayes_classifier
- https://en.wikipedia.org/wiki/Prior_probability
- https://en.wikipedia.org/wiki/Likelihood_function
- https://en.wikipedia.org/wiki/Posterior_probability
- https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagostic)

- https://www.svm-tutorial.com
- https://www.quora.com/What-is-the-kernel-trick
- https://yann.lecun.com/exdb/mnist
- https://en.wikipedia.org/wiki/Hyperparameter_optimization
- https://en.wikipedia.org/wiki/Decision_tree_learning
- https://en.wikipedia.org/wiki/Random_forest
- https://en.wikipedia.org/wiki/Ensemble_learning
- https://en.wikipedia.org/wiki/Bootstrap_aggregating

◆ Daftar Istilah

- **Machine Learning**: Cabang dari kecerdasan buatan yang fokus pada pengembangan algoritma yang memungkinkan komputer belajar dari data.
- **Scikit-Learn**: Library Python yang menyediakan berbagai alat untuk machine learning dan data mining.
- **Model**: Representasi matematis yang dilatih untuk membuat prediksi atau keputusan berdasarkan data.
- **Supervised Learning**: Teknik machine learning di mana model dilatih menggunakan data berlabel.
- **Unsupervised Learning**: Teknik machine learning di mana model dilatih menggunakan data tanpa label.
- **Reinforcement Learning**: Teknik machine learning di mana model belajar melalui interaksi dengan lingkungan untuk mencapai tujuan tertentu.
- **Feature**: Atribut atau variabel input yang digunakan dalam model machine learning.
- **Label**: Output atau target yang digunakan dalam supervised learning.
- **Training Data**: Dataset yang digunakan untuk melatih model.
- **Test Data**: Dataset yang digunakan untuk mengevaluasi performa model setelah dilatih.
- **Validation Data**: Dataset yang digunakan untuk menyetel parameter model selama proses pelatihan.
- **Overfitting**: Situasi di mana model terlalu sesuai dengan data pelatihan sehingga performanya menurun pada data baru.
- **Underfitting**: Situasi di mana model tidak cukup belajar dari data pelatihan sehingga tidak dapat membuat prediksi yang akurat.
- **Cross-Validation**: Teknik untuk mengevaluasi model dengan membagi data menjadi beberapa subset dan melakukan pelatihan dan pengujian berulang kali.
- **Hyperparameter**: Parameter yang perlu disetel sebelum proses pelatihan model, seperti learning rate atau jumlah lapisan dalam jaringan saraf.
- **Algorithm**: Prosedur atau formula yang digunakan untuk memecahkan masalah dalam machine learning.
- **Regression**: Tugas machine learning yang bertujuan untuk memprediksi nilai kontinu.

- **Classification**: Tugas machine learning yang bertujuan untuk memprediksi kategori atau label diskrit.
- **Clustering**: Teknik unsupervised learning yang mengelompokkan data berdasarkan kesamaan fitur.
- **Dimensionality Reduction**: Teknik untuk mengurangi jumlah fitur dalam dataset sambil mempertahankan informasi yang penting.
- **Principal Component Analysis (PCA)**: Teknik untuk dimensionality reduction yang mengubah data ke dalam komponen utama.
- **Normalization**: Proses skala fitur sehingga nilainya berada dalam rentang tertentu, biasanya 0 hingga 1.
- **Standardization**: Proses mengubah fitur sehingga memiliki mean 0 dan standar deviasi 1.
- **Confusion Matrix**: Tabel yang digunakan untuk mengevaluasi performa model klasifikasi dengan membandingkan prediksi dan label sebenarnya.
- **Precision**: Proporsi prediksi positif yang benar-benar positif.
- **Recall**: Proporsi kasus positif yang terdeteksi dengan benar oleh model.
- **F1 Score**: Ukuran performa model yang menggabungkan precision dan recall dalam satu nilai harmonis.
- **ROC Curve**: Grafik yang menunjukkan trade-off antara true positive rate dan false positive rate untuk model klasifikasi pada berbagai threshold.
- **AUC (Area Under Curve)**: Ukuran performa model klasifikasi berdasarkan ROC curve, dengan nilai maksimum 1.
- **Gradient Descent**: Algoritma optimisasi yang digunakan untuk meminimalkan fungsi kerugian dengan iterasi.
- **Neural Network**: Model machine learning yang terinspirasi oleh struktur otak manusia, terdiri dari lapisan neuron yang saling terhubung.
- **Deep Learning**: Subset dari machine learning yang menggunakan jaringan saraf dengan banyak lapisan untuk mempelajari representasi data yang kompleks.
- **Support Vector Machine (SVM)**: Algoritma klasifikasi yang mencari hyperplane terbaik untuk memisahkan kelas-kelas dalam data.
- **Random Forest**: Algoritma ensemble yang menggabungkan banyak pohon keputusan untuk meningkatkan akurasi dan mengurangi overfitting. Random Forest bekerja dengan membuat beberapa pohon keputusan pada subset data yang berbeda dan menggabungkan prediksi dari masing-masing pohon.

- **K-Nearest Neighbors (KNN)**: Algoritma sederhana yang mengklasifikasikan titik data berdasarkan tetangga terdekatnya dalam ruang fitur. KNN bekerja dengan menghitung jarak antara titik data dan tetangganya serta memprediksi kelas mayoritas dari tetangga terdekat.
- **Naive Bayes**: Algoritma klasifikasi berbasis probabilistik yang mengasumsikan independensi antar fitur. Algoritma ini sangat efisien dan bekerja dengan baik untuk dataset besar dan kategori banyak.
- **Decision Tree**: Model prediktif yang menggunakan struktur pohon untuk membuat keputusan berdasarkan fitur input. Setiap node internal mewakili tes pada fitur, setiap cabang mewakili hasil tes, dan setiap daun mewakili label atau distribusi probabilitas.
- **Bagging (Bootstrap Aggregating)**: Teknik ensemble yang meningkatkan akurasi dan kestabilan model dengan melatih beberapa model pada subset data yang diambil secara acak dengan penggantian, lalu menggabungkan hasil prediksinya.
- **Boosting**: Teknik ensemble yang menggabungkan beberapa model lemah menjadi satu model kuat dengan memberi bobot lebih pada data yang salah klasifikasi dalam iterasi sebelumnya.
- **AdaBoost (Adaptive Boosting)**: Algoritma boosting yang menyesuaikan bobot data berdasarkan kesalahan model sebelumnya, fokus pada memperbaiki kesalahan untuk meningkatkan akurasi.
- **Gradient Boosting**: Teknik boosting yang menggabungkan beberapa model dengan memprediksi residu dari model sebelumnya dan menambahkan model baru untuk memperbaiki kesalahan.
- **XGBoost (Extreme Gradient Boosting)**: Implementasi dari gradient boosting yang dioptimalkan untuk efisiensi dan performa, sering digunakan dalam kompetisi machine learning karena kecepatan dan akurasinya.
- **Hyperparameter Tuning**: Proses mengoptimalkan hyperparameter model untuk meningkatkan performa. Teknik umum meliputi grid search dan random search.
- **Ensemble Learning**: Teknik yang menggabungkan prediksi dari beberapa model untuk meningkatkan akurasi dan kestabilan hasil prediksi.
- **Feature Engineering**: Proses membuat fitur baru atau mengubah fitur yang ada untuk meningkatkan performa model machine learning.
- **One-Hot Encoding**: Teknik untuk mengubah variabel kategori menjadi bentuk biner yang dapat digunakan oleh algoritma machine learning.
- **Label Encoding**: Teknik untuk mengubah variabel kategori menjadi nilai numerik.