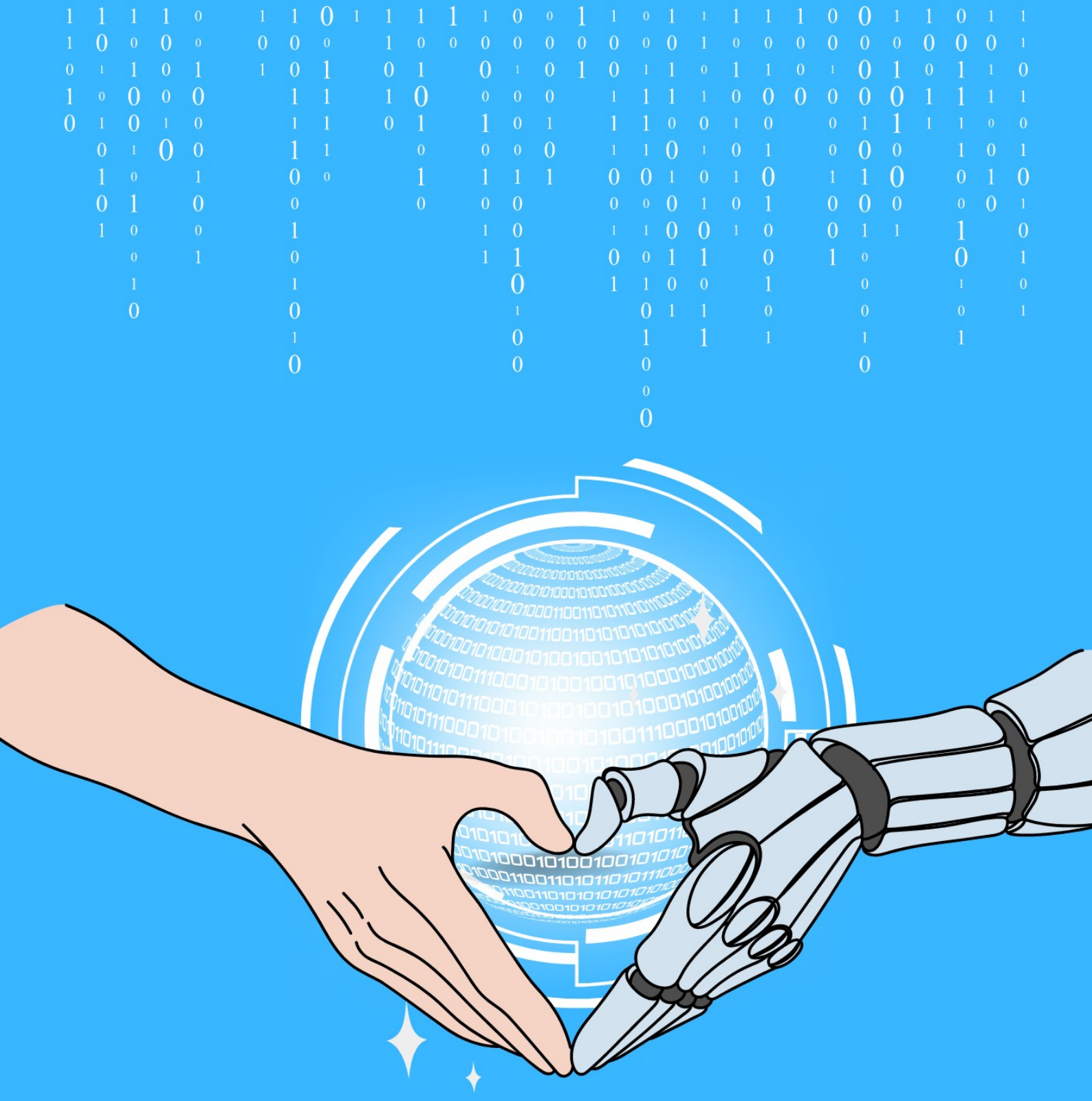


PEMBELAJARAN MESIN DAN
PEMBELAJARAN MENDALAM

Modul 5
Supervised Learning 2

Tim Pengampu
PMDPM

Semester Genap
2022-2023

Judul: Modul 4-Supervised Learning 2

Kontributor: Aloysius Gonzaga Pradnya Sidhawara, Blasius Yonas

Versi: Mei 2023

Daftar Isi

Konsep: Ensembles

Ensembles merupakan metode yang menggabungkan beberapa model pembelajaran mesin untuk membuat model yang lebih handal. Ada banyak model dalam literatur pembelajaran mesin yang termasuk dalam kategori ini, tetapi ada dua model ensemble yang telah terbukti efektif pada berbagai dataset untuk klasifikasi dan regresi yaitu Random Forest dan Gradient Boosted Decision Trees.

Konsep: Gradient Boosted Tree

Gradient Boosted Tree (GBT) yang merupakan salah satu model ensemble berbasis Decision Tree yang bekerja dengan cara membuat tree secara berurutan. Cara kerja GBT dimulai dengan membangun model Decision Tree yang lemah dengan satu label target (dependent variable) pada setiap langkahnya. Dalam setiap iterasi, GBT memperbaiki bobot kesalahan dari setiap prediksi dan mengoptimalkan model dengan menyesuaikan parameter Decision Tree pada setiap langkah. Akhirnya, GBT menggabungkan hasil prediksi dari semua Decision Tree untuk menghasilkan prediksi yang akurat. Kedalaman dari tree yang digunakan pada Gradient Boosted Tree relatif dangkal (antara 1 sampai 5) sehingga cepat dalam prediksi dan lebih ringan dalam penggunaan memori.

Gradient Boosted Tree memiliki atribut feature importances yang mirip dengan Random Forest, namun pada Gradient Boosted Tree tidak semua feature digunakan oleh tree. Kelebihan dari Gradient Boosted Tree adalah:

1. Mampu menangani data yang kompleks dan besar dengan berbagai jenis variabel
2. Mampu mengatasi overfitting dengan menggunakan teknik regularisasi (pengaturan pada learning rate)

Kekurangan dari Gradient Boosted Tree adalah:

1. Membutuhkan pengaturan parameter yang teliti dan membutuhkan waktu lama untuk dilatih

2. Cenderung sensitif terhadap nilai-nilai ekstrem atau outlier pada dataset.

Konsep: Random Forest

Random Forest adalah kumpulan Decision Tree dimana setiap pohon memiliki struktur yang sedikit berbeda dari yang lain. Random Forest bekerja dengan cara membuat sejumlah besar Decision Tree secara acak, dan menggabungkan hasil prediksi dari semua pohon keputusan tersebut.

Cara Kerja:

1. Random Forest membagi dataset menjadi beberapa subset data yang berbeda secara acak.
2. Decision Tree dibuat untuk setiap subset tersebut.
3. Selama pembuatan Decision Tree, pada setiap titik pembagian node, Random Forest memilih beberapa fitur secara acak dari seluruh fitur dataset.
4. Decision Tree terbentuk dengan menggunakan subset data dan fitur yang dipilih secara acak.
5. Voting atau penggabungan hasil prediksi dari semua Decision Tree dilakukan untuk menghasilkan prediksi final.

Setiap pohon yang terbentuk mungkin melakukan prediksi yang relatif baik, tetapi kemungkinan besar akan terlalu cocok pada subset data saja. Setiap kali sebuah tree dalam Random Forest akan dibuat, dilakukan proses bootstrap sampel dari data yang kita miliki. Bootstraping pada Random Forest adalah teknik pengambilan sampel secara acak dengan penggantian (sampling with replacement) yang digunakan dalam proses pembentukan Decision Tree.

Proses bootstrap akan berjalan secara berulang (sesuai dengan jumlah sampel) dan mengambil sampel secara acak dengan kemungkinan memilih sampel yang sama. Decision Tree dibuat berdasarkan subset baru yang diperoleh dari bootstrap. Node dibuat berdasarkan pertanyaan yang terbaik yang melibatkan satu dari sekian feature pada subset. Pemilihan subset diulang secara terpisah pada setiap node sehingga setiap node pada tree dapat membuat keputusan menggunakan subset feature yang berbeda.

Kelebihan dari proses bootstraping pada Random Forest yaitu mampu mengurangi risiko overfitting dan menghasilkan model yang lebih handal. Karena setiap decision tree dibangun dengan subset yang berbeda-beda, sehingga mengurangi kemungkinan kesalahan. Namun, kekurangan dari teknik bootstraping pada Random Forest yaitu membutuhkan waktu yang lebih lama untuk melatih model, karena harus membangun banyak Decision Tree. Prediksi menggunakan Random Forest dilakukan dengan memprediksi data baru pada setiap tree di dalam Random Forest. Prediksi kasus regresi hasil berupa rerata nilai prediksi dari tiap – tiap tree. Prediksi kasus klasifikasi dengan melakukan “soft voting” hasil prediksi dari tiap – tiap tree.

Random Forest dapat menggunakan ratusan hingga ribuan tree untuk membentuk decision boundary yang lebih halus. Parameter yang perlu diatur pada Random Forest seperti n_estimators, max_features, dan dapat dilakukan pre-pruning dengan max_depth. Keunggulan dari Random Forest adalah:

1. Mampu bekerja dengan dataset yang tidak seimbang
2. Tidak membutuhkan normalisasi atau penskalaan data

Kekurangan dari Random Forest adalah:

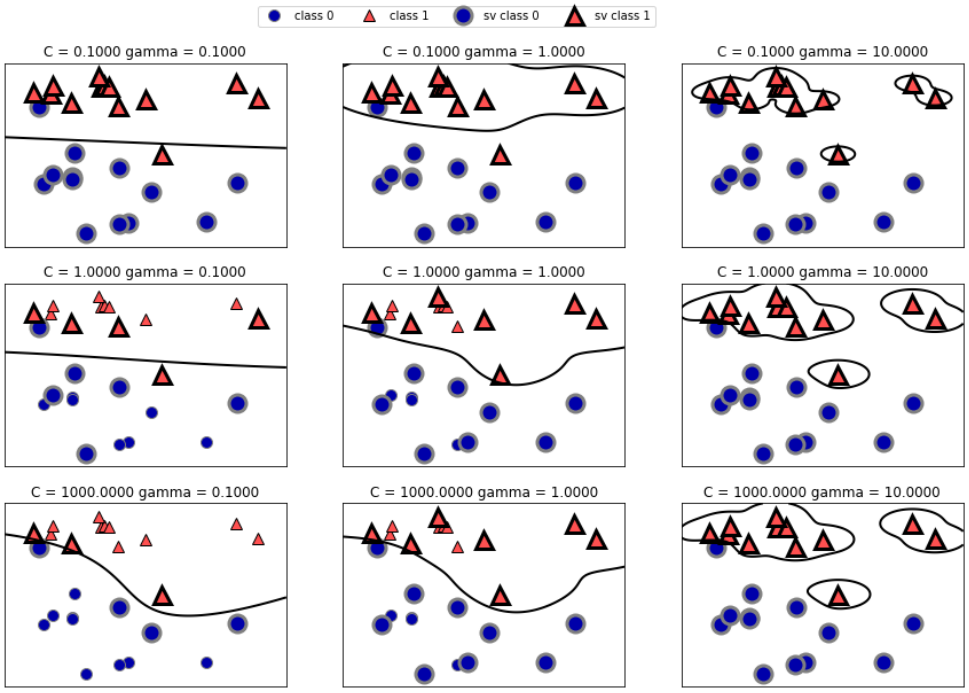
1. Tidak handal pada data dengan dimensionalitas tinggi atau data sparse seperti data teks
2. Membutuhkan lebih banyak memori dan lebih lambat untuk dilatih dibandingan model linear

Konsep: Support Vector Machine Kernel

SVM atau Support Vector Machine mempelajari bagaimana pentingnya masing-masing titik data pada training set untuk merepresentasikan decision boundary atau batasan keputusan antara dua kelas. Biasanya hanya sebagian kecil dari titik data pada training set yang dianggap penting untuk menentukan batasan keputusan, yaitu titik data yang terletak pada batas antara kelas. Data-data itu disebut support vector. Untuk memprediksi titik data yang baru, jarak antara titik data baru tersebut dengan masing-masing support vector diukur. Klasifikasi didasarkan pada jarak antara titik baru dengan

support vector dan tingkat kepentingan masing-masing support vector yang dipelajari saat training.

SVM Linear hanya dapat bekerja dengan data yang secara linear dapat dipisahkan. Hal ini berarti masing-masing kelas harus dapat dipisahkan oleh garis lurus di dalam ruang fitur. Dalam kasus dimana data tidak dapat dipisahkan secara linear, fungsi kernel dapat digunakan untuk mengubah data pada ruang fitur yang lebih tinggi dimana data dapat dipisahkan. Kernel adalah trik matematis yang memungkinkan kita membuat model pengklasifikasi mempelajari ruang dimensi yang lebih tinggi, tanpa benar-benar menghitung representasi baru yang mungkin sangat besar. Metode ini bekerja dengan secara langsung menghitung jarak (lebih tepatnya, produk skalar) dari titik data untuk representasi fitur yang diperluas, tanpa benar-benar menghitung ekspansinya. Salah satu cara untuk membuat model linier lebih fleksibel adalah menambahkan interaksi atau polinomial dari fitur input.



Gambar 1. Contoh manipulasi decision boundary SVM menggunakan parameter C dan Gamma

Parameter C adalah parameter yang mempengaruhi signifikansi dari tiap titik data. Dari Gambar 1 atas ke bawah, parameter C ditingkatkan dari nilai 0.1 ke 1000. C dengan nilai kecil berarti model yang dipakai sangat ketat. Dapat dilihat dari grafik paling atas

sebelah kiri yang memiliki decision boundary yang nyaris linier, dengan titik data yang salah diklasifikasikan tidak memiliki pengaruh apapun pada garis. Berbeda dengan yang memiliki nilai C yang besar, seperti grafik di paling bawah sebelah kanan, titik data memiliki pengaruh yang kuat terhadap model dan membuat decision boundary secara tepat mengklasifikasikan data.

Parameter Gamma mengatur radius dari Gaussian kernel, pada kasus ini dari nilai 0.1 hingga 10. Gamma dengan nilai kecil berarti Gaussian kernel memiliki radius yang besar dan banyak titik data yang dianggap berdekatan. Hal ini ditunjukkan dari decision boundary yang sangat halus di grafik-grafik sebelah kiri dan decision boundary makin fokus pada masing-masing titik di grafik-grafik sebelah kanan. Gamma dengan nilai yang kecil berarti model tidak kompleks dan decision boundary tidak bervariasi secara signifikan, sedangkan gamma dengan nilai yang besar memberi dampak model yang lebih kompleks.

SVM dapat membuat batasan keputusan yang kompleks, bahkan jika data hanya memiliki beberapa fitur. SVM bekerja dengan baik pada data berdimensi rendah dan berdimensi tinggi. Tetapi, performa SVM juga bergantung pada jumlah sampel. Menjalankan SVM pada data hingga 10.000 sampel mungkin masih dapat berfungsi dengan baik, tetapi bekerja dengan kumpulan data berukuran 100.000 atau lebih dapat menjadi tantangan dalam hal waktu proses dan penggunaan memori. Kelemahan lain dari SVM adalah memerlukan pra-pemrosesan data dan penyetelan parameter yang cermat.

Kelebihan dari SVM dengan fungsi kernel adalah:

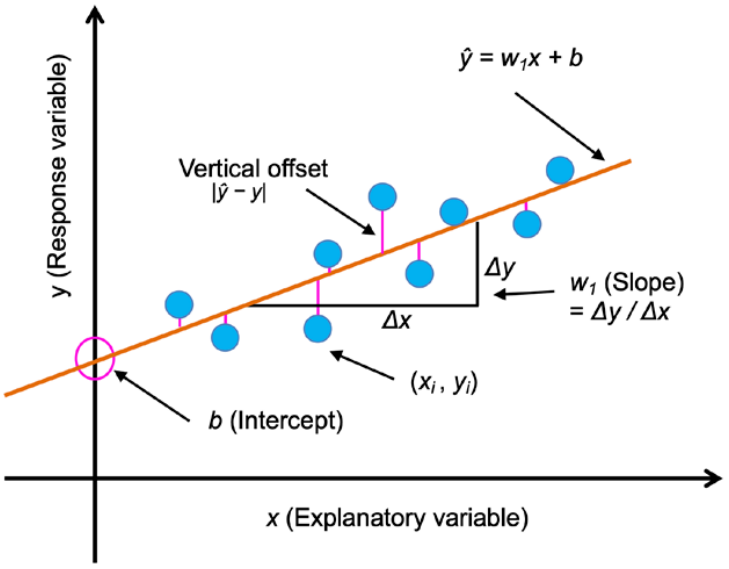
1. menangani data yang tidak dapat dipisahkan secara linear
2. mampu untuk menggeneralisasi dengan baik ke data yang baru.

Kekurangan dari SVM dengan fungsi kernel adalah:

1. membutuhkan sumber daya yang tinggi secara komputasi
2. sensitif terhadap pilihan penskalaan data dan fungsi kernel serta parameter-parameter yang digunakan.

Konsep: Regresi

Regresi dalam supervised learning adalah sebuah metode yang digunakan untuk memodelkan hubungan antara variabel independen (X) dan variabel dependen (y) dengan mencari persamaan matematis. Regresi memiliki tujuan untuk memprediksi nilai variabel dependen berdasarkan nilai variabel independen. Cara mudah untuk membedakan antara tugas klasifikasi dan regresi: apakah ada semacam kontinuitas dalam output. Contoh: memprediksi pendapatan tahunan seseorang dari pendidikan mereka, usia mereka, dan tempat tinggal mereka. Nilai yang diprediksi adalah jumlah, dan dapat berupa angka apa pun dalam rentang tertentu.



Gambar 2. Ilustrasi Regresi Linear [1]

Linear Regression atau Regresi Linear adalah salah satu metode regresi yang paling sederhana dan sering digunakan. Linear Regression bekerja dengan mencari garis lurus terbaik (linear) yang dapat menggambarkan hubungan antara variabel independen (X) dan dependen (y). Linear Regression menghitung parameter w dan b yang meminimalkan Mean Squared Error antara prediksi dan target regresi sebenarnya. Parameter w atau weights atau coefficients dapat diakses melalui atribut coef_ , sedangkan parameter b atau offset atau intercept dapat diakses melalui atribut intercept_.

Kelebihan dari linear regression adalah:

1. sederhana dan mudah dimengerti

2. dapat memberikan hasil yang akurat pada dataset yang linier
3. waktu komputasi yang relatif cepat.

Kekurangan dari metode linear regression adalah:

1. tidak dapat mengatasi hubungan non-linier
2. tidak dapat mengatasi multikolinearitas (situasi yang menunjukkan adanya korelasi atau hubungan kuat antara dua variabel bebas atau lebih dalam sebuah model regresi. Hal ini berpengaruh signifikan terhadap nilai prediksi oleh koefisien b).

Teknik regresi yang disebut Lasso dan Ridge dikembangkan untuk mengatasi masalah multikolinearitas dan overfitting dalam model regresi linier. Kedua teknik ini bekerja dengan memberikan penalty pada persamaan regresi. Menggunakan penalti L1, Lasso membuat model yang lebih sederhana dengan berfokus pada feature yang paling penting dan membuang feature minor. Model Lasso juga membatasi koefisien w agar mendekati nol. Konsekuensi dari regularisasi L1 adalah ketika menggunakan Lasso, beberapa koefisien memiliki nilai tepat nol. Hal ini menyebabkan beberapa fitur sepenuhnya diabaikan oleh model.

Sementara Ridge menggunakan penalti L2 dengan mengurangi koefisien regresi dan mengatasi multikolinearitas untuk menghasilkan model yang lebih stabil. Koefisien w pada Ridge tidak hanya digunakan memprediksi pada data training, tetapi juga untuk menyesuaikan batasan tambahan. Besarnya nilai koefisien dibuat menjadi sekecil mungkin; semua nilai w harus mendekati nol. Setiap feature harus memiliki efek sesedikit mungkin pada hasil (yang berarti memiliki kemiringan kecil jika digambarkan dalam bentuk grafik), namun harus tetap memprediksi dengan baik. Regularisasi secara eksplisit meminimalisir terjadinya overfit.

Kelebihan dari Lasso adalah dapat membuat model yang disederhanakan, memilih fitur yang paling penting, dan menghindari overfitting. Kekurangannya adalah sering mengabaikan feature yang tidak berdampak pada variabel dependen dan mungkin menghasilkan model yang tidak stabil. Ridge memiliki kelebihan dapat menghilangkan multikolinearitas dan membuat model yang lebih stabil. Kekurangannya adalah tidak

dapat mengabaikan feature yang tidak signifikan dan cenderung membangun model yang rumit.

Tugas

Studi kasus: Medical Insurance Dataset (https://www.kaggle.com/datasets/harshsingh2209/medical-insurance-payout)
Deskripsi: ACME Insurance Inc. menawarkan asuransi kesehatan yang terjangkau bagi ribuan pelanggan di seluruh Amerika Serikat. Sebagai ilmuwan data ditugaskan untuk membuat sistem otomatis untuk memperkirakan angsuran medis tahunan untuk pelanggan baru, menggunakan informasi seperti usia, jenis kelamin, BMI, jumlah anak-anak, kebiasaan merokok, dan wilayah tempat tinggal.
Informasi atribut:

- age: usia
- sex: jenis kelamin
- bmi: body mass index
- children: jumlah anak-anak
- smoker: perokok atau bukan
- region: daerah dimana pelanggan tinggal
- charges: variabel target, jumlah angsuran yang dibayarkan customer

Ketentuan tugas: Buatlah notebook untuk melakukan regresi terhada[dataset medical insurance. Terapkan pemrosesan dataset dari loading, outlier removal, pelatihan, hingga evaluasi model secara sederhana menggunakan code yang sudah dipelajari dalam latihan.

Jawab pertanyaan berikut di dalam notebook menggunakan cell markdown:

1. Apakah ada pengaruh penskalaan data pada dataset terhadap performa model machine learning untuk regresi?
2. Model apa yang paling cocok digunakan untuk kasus tersebut?

Ketentuan modelling:

- Untuk NPM ganjil gunakan RandomForestRegressor dengan parameter setting random_state=2 digit terakhir NPM dan max_depth bebas. Latih model algoritme tersebut untuk masing-masing X_train dan ujikan pada X_test yang ***belum diubah*** dan ***sudah diubah*** menggunakan ***MinMaxScaler***. Bandingkan hasil MAE dan RMSE algoritme tersebut dengan algoritme Ridge dan Lasso yang sudah dipelajari di latihan (setting parameter Ridge dan Lasso sama dengan latihan)!
- Untuk NPM genap gunakan SupportVectorRegressor dengan parameter setting kernel='rbf', nilai C dan gamma bebas. Latih model algoritme tersebut untuk masing-masing X_train dan ujikan pada X_test yang **belum diubah**, dan **sudah diubah** menggunakan **StandardScaler**. Bandingkan hasil MAE dan RMSE algoritme tersebut dengan algoritme Ridge dan Lasso yang sudah dipelajari di latihan (setting parameter Ridge dan Lasso sama dengan latihan)!

Referensi

[1] A. C. Müller dan S. Guido, Introduction to machine learning with Python: a guide for data scientists., O'Reilly Media, Inc., 2016.

[2] S. Raschka, Y. H. Liu, V. Mirjalili dan D. Dzhulgakov, Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python, Packt Publishing Ltd, 2022.

Membaca Data & Preprocessing

Load dataset kedalam notebook dan ubah menjadi DataFrame. Selanjutnya, tampilkan 20 data pertama dari dataset yang sudah diload.

In [ ]:
```
import pandas as pd
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
df_cancer = pd.DataFrame(data=cancer.data,columns=cancer.feature_names,index=None)
df_cancer.head(20)
```

Out[ ]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	25.38	17.33
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	24.99	23.41
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	23.57	25.53
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	14.91	26.50
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	22.54	16.67
5	12.45	15.70	82.57	477.1	0.12780	0.17000	0.15780	0.08089	0.2087	0.07613	...	15.47	23.75
6	18.25	19.98	119.60	1040.0	0.09463	0.10900	0.11270	0.07400	0.1794	0.05742	...	22.88	27.66
7	13.71	20.83	90.20	577.9	0.11890	0.16450	0.09366	0.05985	0.2196	0.07451	...	17.06	28.14
8	13.00	21.82	87.50	519.8	0.12730	0.19320	0.18590	0.09353	0.2350	0.07389	...	15.49	30.73
9	12.46	24.04	83.97	475.9	0.11860	0.23960	0.22730	0.08543	0.2030	0.08243	...	15.09	40.68
10	16.02	23.24	102.70	797.8	0.08206	0.06669	0.03299	0.03323	0.1528	0.05697	...	19.19	33.88
11	15.78	17.89	103.60	781.0	0.09710	0.12920	0.09954	0.06606	0.1842	0.06082	...	20.42	27.28
12	19.17	24.80	132.40	1123.0	0.09740	0.24580	0.20650	0.11180	0.2397	0.07800	...	20.96	29.94
13	15.85	23.95	103.70	782.7	0.08401	0.10020	0.09938	0.05364	0.1847	0.05338	...	16.84	27.66
14	13.73	22.61	93.60	578.3	0.11310	0.22930	0.21280	0.08025	0.2069	0.07682	...	15.03	32.01
15	14.54	27.54	96.73	658.8	0.11390	0.15950	0.16390	0.07364	0.2303	0.07077	...	17.46	37.13
16	14.68	20.13	94.74	684.5	0.09867	0.07200	0.07395	0.05259	0.1586	0.05922	...	19.07	30.88
17	16.13	20.68	108.10	798.8	0.11700	0.20220	0.17220	0.10280	0.2164	0.07356	...	20.96	31.48
18	19.81	22.15	130.00	1260.0	0.09831	0.10270	0.14790	0.09498	0.1582	0.05395	...	27.32	30.88
19	13.54	14.36	87.46	566.3	0.09779	0.08129	0.06664	0.04781	0.1885	0.05766	...	15.11	19.26

20 rows × 30 columns

Lakukan train-test split dengan menggunakan rasio parameter test_size dan train size sebesar 30:70 serta random_state dengan nilai 42. Selanjutnya tampilkan bentuk dari train set dan test set.

In [ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df_cancer,
                                                    cancer.target,
                                                    test_size = 0.3,
                                                    random_state=42)

print(X_train.shape)
print(X_test.shape)
```

```
(398, 30)
(171, 30)
```

Buatlah obyek selector untuk seleksi fitur. Seleksi 30% fitur terbaik dari total fitur yang ada pada train set dengan menggunakan fungsi fit dan eliminasi fitur yang kurang penting dengan menggunakan fungsi transform Select Percentile.

Tampilkan bentuk dari X_train dan X_train setelah dilakukan seleksi fitur. Selanjutnya, tampilkan fitur yang dipilih oleh SelectPercentile dengan mengakses atribut mask

In [ ]:
```
from sklearn.feature_selection import SelectPercentile

selector = SelectPercentile(percentile=30)

selector.fit(X_train,y_train)

X_train_select = selector.transform(X_train)
X_test_select = selector.transform(X_test)

print(X_train_select.shape)
print(X_test_select.shape)

mask = selector.get_support()
feat_names=df_cancer.columns[mask]
print(feat_names)
```

```
(398, 9)
(171, 9)
Index(['mean radius', 'mean perimeter', 'mean area', 'mean concavity',
       'mean concave points', 'worst radius', 'worst perimeter', 'worst area',
       'worst concave points'],
      dtype='object')
```

Tampilkan tingkat kepentingan masing-masing feature dengan mengakses feature_importances dari estimator.

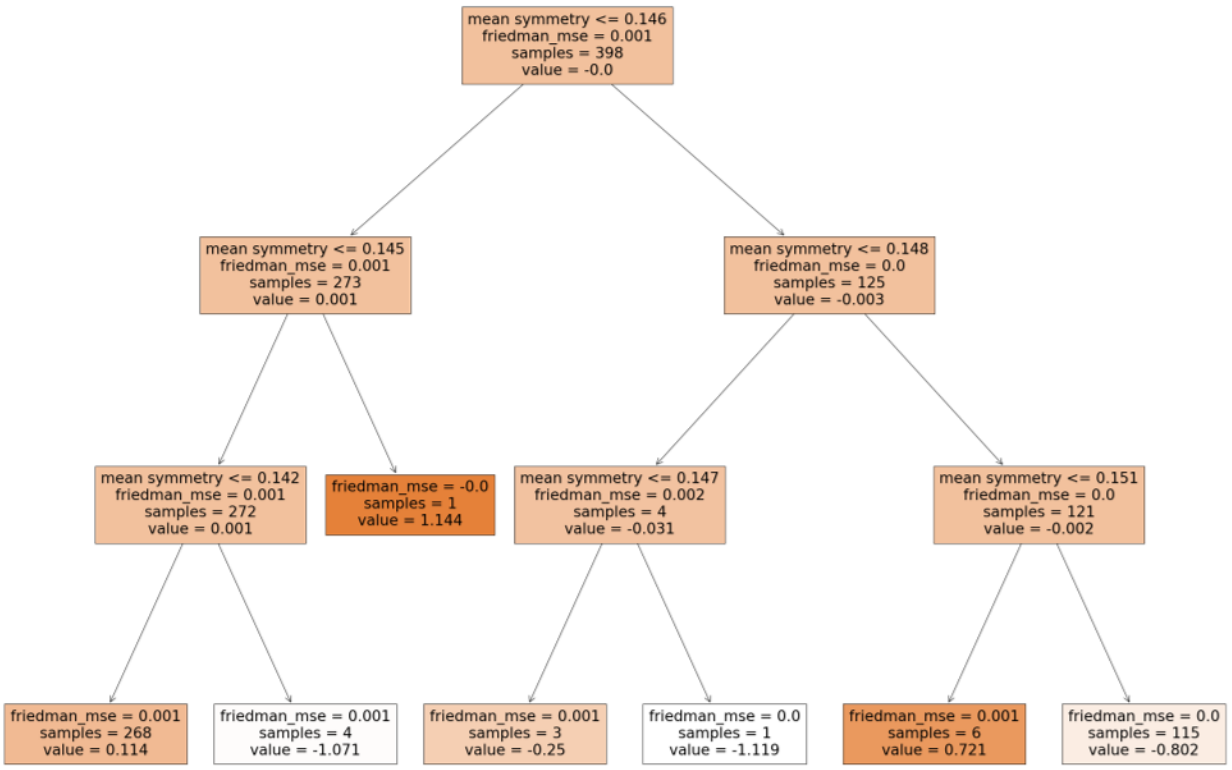
In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

def plot_feature_importances(classifier,dataset, feat_names):
  n_feat=dataset.shape[1]
  plt.barh(range(n_feat),classifier.feature_importances_,align='center')
  plt.yticks(np.arange(n_feat),feat_names)
  plt.xlabel("Feature importances")
  plt.ylabel("Features")
  plt.ylim(-1,n_feat)
  plt.show()
```

Gradient Boosting Classifier

Untuk dapat mengimplementasikan Gradient Boosting Classifier, langkah pertama adalah import semua library yang akan digunakan. Langkah kedua, buat objek GBC sebagai model dari Gradient Boosting Classifier dengan parameter n_estimators bernilai 100 dan random_state bernilai 42. Selanjutnya, latih model yang sudah dibuat sebelumnya dengan fungsi fit dan tampilkan akurasi dari hasil pengujian. Terakhir visualisasikan tree dari hasil pengujian.

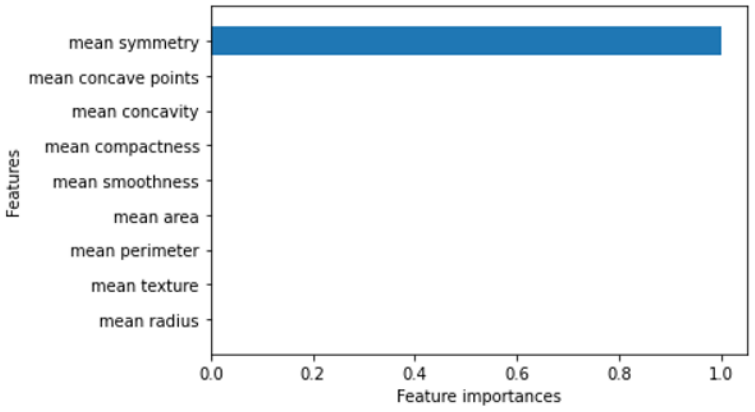
In [ ]:
```
from sklearn.tree import plot_tree
from sklearn.ensemble import GradientBoostingClassifier
import matplotlib.pyplot as plt

GBC = GradientBoostingClassifier(n_estimators=100,
                                 random_state=42).fit(X_train_select,y_train)

print("Akurasi model GBC:",format(GBC.score(X_test_select,y_test)))

plt.figure(figsize=(40,30))
dump = plot_tree(GBC.estimators_[99,0], filled=True,
                 feature_names = df_cancer.columns,
                 class_names = cancer.target_names)
```

```
Akurasi model GBC: 0.9649122807017544
```



Gunakan fungsi plot_feature_impotances untuk melihat diagram perbandingan kepentingan fitur

In [ ]:
```
plot_feature_importances(GBC.estimators_[99,0],
                         X_train_select,
                         df_cancer.columns)
```

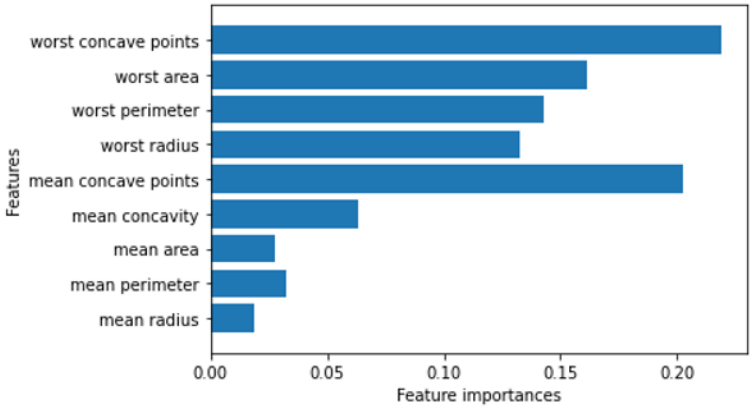


Random Forest

Untuk dapat mengimplementasikan Random Forest, langkah pertama adalah import semua library yang akan digunakan. Langkah kedua, buat objek RF sebagai model dari Random Forest dengan parameter n_estimators bernilai 100 dan random_state bernilai 42 dan latih model dengan fungsi fit. Selanjutnya, tampilkan akurasi dari hasil pelatihan dan pengujian. Terakhir tampilkan perbandingan kepentingan fitur dengan fungsi plot_feature_importances.

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier

RF=RandomForestClassifier(n_estimators=100,random_state=42)
RF.fit(X_train_select,y_train)

print("Akurasi train set:",RF.score(X_train_select,y_train))
print("Akurasi test set:",RF.score(X_test_select,y_test))

plot_feature_importances(RF,X_train_select,feat_names)
```

```
Akurasi train set: 1.0
Akurasi test set: 0.9532163742690059
```



Support Vector Machine Kernel

Import library SVC lalu buat sebuah objek SVC_rbf sebagai model SVM dengan parameter kernel 'rbf', C bernilai 10, dan gamma bernilai 0.1. Selanjutnya latih model tersebut dengan fungsi fit dan tampilkan nilai akurasi pelatihan dan pengujian model.

In [ ]:
```
from sklearn.svm import SVC

SVC_rbf = SVC(kernel='rbf',C=10,gamma=0.1)
SVC_rbf.fit(X_train_select,y_train)

print("Akurasi train set:",SVC_rbf.score(X_train_select,y_train))
print("Akurasi test set:",SVC_rbf.score(X_test_select,y_test))
```

```
Akurasi train set: 1.0
Akurasi test set: 0.631578947368421
```

Lakukan langkah yang sama seperti kode sebelumnya namun dengan nama objek SVC_poly dan pengaturan parameter kernel 'poly' serta C bernilai 1.

In [ ]:
```
from sklearn.svm import SVC

SVC_poly = SVC(kernel='poly',C=1)
SVC_poly.fit(X_train_select,y_train)

print("Akurasi train set", format(SVC_poly.score(X_train_select,y_train)))
print("Akurasi test set", format(SVC_poly.score(X_test_select,y_test)))
```

```
Akurasi train set 0.8969849246231156
Akurasi test set 0.9415204678362573
```

Import StandardScaler, buat sebuah model untuk StandardScaler, dan lakukan fit terhadap train set dan transformasikan bersama dengan test set. Latih kembali kedua model SVM sebelumnya dengan menggunakan train set yang sudah diubah skalanya dengan StandardScaler dan kemudian uji akurasi pada test set yang sudah diubah skalanya.

In [ ]:
```
from sklearn.preprocessing import StandardScaler

std = StandardScaler()
X_train_std = std.fit_transform(X_train_select)
X_test_std = std.transform(X_test_select)

SVC_rbf2 = SVC_rbf
SVC_rbf2.fit(X_train_std,y_train)

SVC_poly2=SVC_poly
SVC_poly2.fit(X_train_std,y_train)
print("Akurasi train set SVC RBF+Standard Scaler", format(SVC_rbf2.score(X_train_std,y_train)))
print("Akurasi test set SVC RBF+Standard Scaler", format(SVC_rbf2.score(X_test_std,y_test)))
print("Akurasi train set SVC Poly+Standard Scaler", format(SVC_poly2.score(X_train_std,y_train)))
print("Akurasi test set SVC Poly+Standard Scaler", format(SVC_poly2.score(X_test_std,y_test)))
```

```
Akurasi train set SVC RBF+Standard Scaler 0.9597989949748744
Akurasi test set SVC RBF+Standard Scaler 0.9649122807017544
Akurasi train set SVC Poly+Standard Scaler 0.8894472361809045
Akurasi test set SVC Poly+Standard Scaler 0.8947368421052632
```

Regresi

Gradient Boosting Regressor

Load dataset kedalam notebook dan ubah menjadi Data Frame. Pastikan dengan benar penulisan path tempat penyimpanan file dataset. Selanjutnya tampilkan 20 data pertama dari dataset yang sudah diload.

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')

path='/content/drive/MyDrive/Colab Notebooks/insurance.csv'
df_insurance = pd.read_csv(path)
df_insurance.head(20)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", f
orce_remount=True).
```

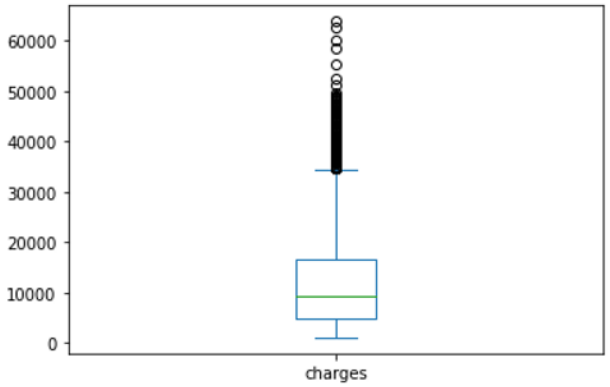
Out[ ]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
5	31	female	25.740	0	no	southeast	3756.62160
6	46	female	33.440	1	no	southeast	8240.58960
7	37	female	27.740	3	no	northwest	7281.50560
8	37	male	29.830	2	no	northeast	6406.41070
9	60	female	25.840	0	no	northwest	28923.13692
10	25	male	26.220	0	no	northeast	2721.32080
11	62	female	26.290	0	yes	southeast	27808.72510
12	23	male	34.400	0	no	southwest	1826.84300
13	56	female	39.820	0	no	southeast	11090.71780
14	27	male	42.130	0	yes	southeast	39611.75770
15	19	male	24.600	1	no	southwest	1837.23700
16	52	female	30.780	1	no	northeast	10797.33620
17	23	male	23.845	0	no	northeast	2395.17155
18	56	male	40.300	0	no	southwest	10602.38500
19	30	male	35.300	0	yes	southwest	36837.46700

Gunakan fungsi plot dengan parameter kind 'box' untuk melihat sebaran data dari fitur charges.

In [ ]:
```
df_insurance.charges.plot(kind='box')
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3d591ffa30>
```



Selanjutnya, sebuah function yang akan digunakan untuk membuang outlier dengan metode inter-quartile range. Selanjutnya tampilkan perbandingan jumlah baris dari Dataframe sebelum dan sesudah pembuangan outlier.

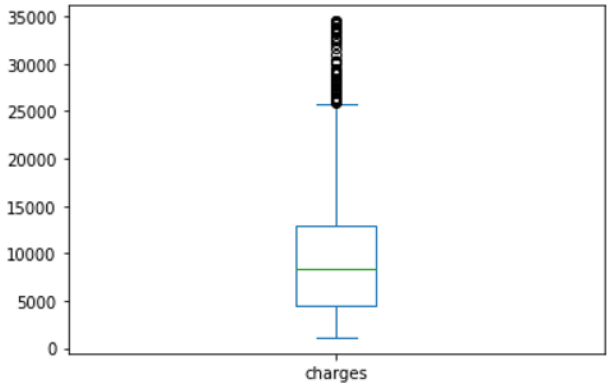
In [ ]:
```
from pandas.api.types import is_numeric_dtype

def remove_outlier(df_in):
  for col_name in list(df_in.columns):
    if is_numeric_dtype(df_in[col_name]):
      q1 = df_in[col_name].quantile(0.25)
      q3 = df_in[col_name].quantile(0.75)

      iqr = q3-q1
      batas_atas = q3+(iqr*1.5)
      batas_bawah = q1-(iqr*1.5)

      df_out = df_in.loc[(df_in[col_name]>=batas_bawah)& (df_in[col_name]<=batas_atas)]
  return df_out

df_insurance_clean = remove_outlier(df_insurance)
print("Jumlah baris DataFrame sebelum dibuang outlier",df_insurance.shape[0])
print("Jumlah baris DataFrame sesudah dibuang outlier",df_insurance_clean.shape[0])
```

```
Jumlah baris DataFrame sebelum dibuang outlier 1338
Jumlah baris DataFrame sesudah dibuang outlier 1199
```

Gunakan fungsi plot dengan parameter kind 'box' untuk melihat sebaran data dari fitur charges setelah outlier dibuang.

In [ ]:
```
df_insurance_clean.charges.plot(kind='box')
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3d58334b80>
```



Lakukan pemisahan antara data feature dan data target. Selanjutnya, lakukan train-test split dengan menggunakan rasio parameter test_size dan train size sebesar 25:75 serta random_state dengan nilai 42.

In [ ]:
```
from sklearn.model_selection import train_test_split

X_regress = df_insurance_clean.drop('charges',axis=1)
y_regress = df_insurance_clean.charges

X_train_ins, X_test_ins, y_train_ins, y_test_ins = train_test_split(X_regress,y_regress,\
                                                                    test_size=0.25,
                                                                    random_state=42)
```

Import OneHotEncoder untuk melakukan encoding dari beberapa feature kategorikal seperti sex, smoker, dan region. Buat sebuah obyek transformer untuk model OneHotEncoder lalu lakukan proses fit_transform untuk train set dan proses transform untuk test set. Selanjutnya buat 2 Dataframe yang akan digunakan sebagai penampung dari train set dan test set yang telah di encoding. Tampilkan 10 data teratas dari masing-masing Dataframe.

In [ ]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer

cat_cols = ['sex','smoker','region']

transformer = make_column_transformer(
    (OneHotEncoder(), cat_cols),
    remainder='passthrough'
)

X_train_enc = transformer.fit_transform(X_train_ins)
X_test_enc = transformer.transform(X_test_ins)

df_train_enc=pd.DataFrame(X_train_enc,columns=transformer.get_feature_names_out())
df_test_enc=pd.DataFrame(X_test_enc,columns=transformer.get_feature_names_out())

df_train_enc.head(10)
df_test_enc.head(10)
```

Out[ ]:

	onehotencoder__sex_female	onehotencoder__sex_male	onehotencoder__smoker_no	onehotencoder__smoker_yes	onehotencoder__
0	0.0	1.0	1.0	0.0	
1	0.0	1.0	1.0	0.0	
2	1.0	0.0	0.0	1.0	
3	1.0	0.0	1.0	0.0	
4	0.0	1.0	0.0	1.0	
5	0.0	1.0	1.0	0.0	
6	1.0	0.0	1.0	0.0	
7	1.0	0.0	0.0	1.0	
8	1.0	0.0	1.0	0.0	
9	0.0	1.0	1.0	0.0	

Import Gradient Boosting Tree Regressor dan buat sebuah obyek sebagai model dari GradientBoostingRegressor dengan pengaturan parameter n_estimators bernilai 100 dan random_state bernilai 42. Latih model tersebut pada train set. Selanjutnya lakukan prediksi dari test set target menggunakan test set feature. Tampilkan nilai RMSE dari prediksi yang sudah dilakukan sebelumnya.

In [ ]:
```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error

GBR = GradientBoostingRegressor(n_estimators=100, random_state=42).fit(X_train_enc,y_train_ins)

df_results = pd.DataFrame(y_test_ins)
df_results['GBR prediction'] = GBR.predict(X_test_enc)


print("RMSE GBR:",format(mean_squared_error(y_test_ins,df_results['GBR prediction'], squared=False)))
df_results.head(20)
```

```
RMSE GBR: 4929.3249079869065
```

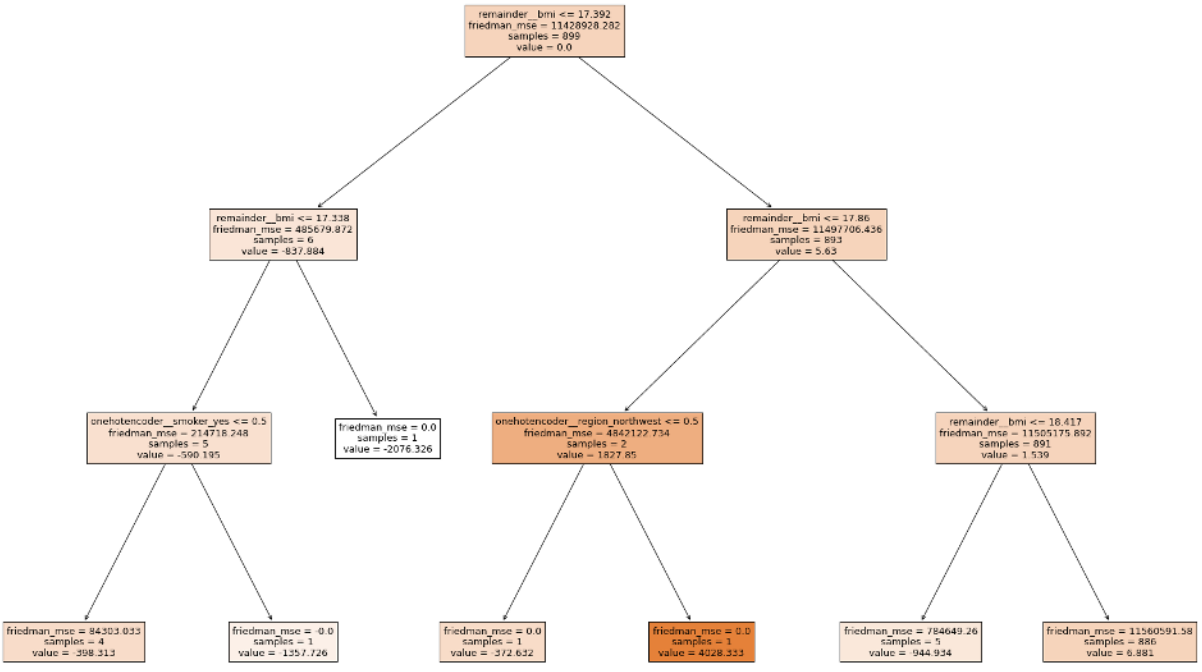
Out[ ]:

	charges	GBR prediction
1315	11272.33139	3794.739573
965	4746.34400	6130.585255
117	19107.77960	18050.997620
492	2196.47320	3950.396748
69	17663.14420	16767.746385

	charges	GBR prediction
1210	5373.36425	5646.071150
367	8017.06115	8234.765681
1085	19023.26000	18655.694723
462	15230.32405	16235.708065
1333	10600.54830	12302.421496
1132	20709.02034	11842.266453
54	8059.67910	8016.684954
60	8606.21740	10153.381735
944	12982.87470	13122.600887
273	9617.66245	11364.292649
192	2137.65360	3319.084154
584	1242.81600	982.553372
854	24106.91255	23731.693358
919	5245.22690	5595.953997
474	25382.29700	23231.051349

Lakukan visualisasi tree dimana nilai 99 merupakan indeks pohon yang ingin ditampilkan

In [ ]:
```
plt.figure(figsize=(30,20))
dump = plot_tree(GBR.estimators_[99,0],filled=True,feature_names=df_train_enc.columns)
```



Linear Regression

Buat model LinearRegression dan latih menggunakan train set. Tampilkan nilai koefisien yang diperoleh dengan menggunakan atribut coef *dan nilai intercept dari persamaan linier dengan menggunakan atribut intercept.*

In [ ]:
```
from sklearn.linear_model import LinearRegression

LR = LinearRegression().fit(X_train_enc,y_train_ins)
```

```
print("Koefisien/bobot:", LR.coef_)
print("Intercept/bias:", LR.intercept )
```

```
Koefisien/bobot: [  270.29519256  -270.29519256 -7497.9055228   7497.9055228
   584.59072906   401.86373645  -251.60473032  -734.84973519
   245.76142841    72.2003622    376.52785173]
Intercept/bias: 3430.2814661695584
```

Jalankan prediksi terhadap test set. Muat hasil prediksi model ke dalam DataFrame. Tampilkan hasil prediksi dari LinearRegression dan bandingkan antara nilai hasil prediksi dengan nilai aslinya.

In [ ]:
```
LR_pred = LR.predict(X_test_enc)
df_results['LR prediction']=LR_pred
df_results.head(10)
```

Out[ ]:

	charges	GBR prediction	LR prediction
1315	11272.33139	3794.739573	3090.897297
965	4746.34400	6130.585255	5862.038677
117	19107.77960	18050.997620	20467.764847
492	2196.47320	3950.396748	3021.752660
69	17663.14420	16767.746385	20148.555302
1210	5373.36425	5646.071150	7517.069945
367	8017.06115	8234.765681	9483.496619
1085	19023.26000	18655.694723	23252.234041
462	15230.32405	16235.708065	15527.998928
1333	10600.54830	12302.421496	11717.644680

Kita dapat mengukur performa dari model regresi dengan menggunakan Mean Absolute Error, Mean Squared Error, dan Root Mean Squared Error. Mean Absolute Error menghitung rerata nilai absolute dari error (selisih antara nilai asli dengan nilai prediksi). Mean Squared Error menghitung rerata nilai kuadrat dari error. Root Squared Error menghitung akar pangkat dari rerata nilai kuadrat error. Semakin kecil nilai error maka garis regresi yang dibentuk oleh model semakin mendekati posisi kumpulan data aslinya.

In [ ]:
```
from sklearn import metrics
import numpy as np

print("Mean Absolute Error",metrics.mean_absolute_error(y_test_ins,LR_pred))
print("Mean Squared Error",metrics.mean_squared_error(y_test_ins,LR_pred))
print("Root Mean Squared Error",np.sqrt(metrics.mean_squared_error(y_test_ins,LR_pred)))
```

```
Mean Absolute Error 2706.741683352127
Mean Squared Error 26046639.666235358
Root Mean Squared Error 5103.590859996063
```

Ridge Regression

Buatlah model regresi Ridge dengan pengaturan parameter alpha=0.1. Parameter alpha digunakan sebagai regularisasi untuk membatasi efek dari feature pada hasil prediksi dan biasanya berlaku pada dataset dengan feature yang banyak. Hal ini juga dilakukan untuk meminimalisir overfitting. Selanjutnya, lakukan prediksi menggunakan model Ridge yang barusan dibuat, kemudian dimasukkan ke DataFrame yang sebelumnya kita gunakan untuk membandingkan hasil prediksi.

In [ ]:
```
from sklearn.linear_model import Ridge
RD = Ridge(alpha=0.1).fit(X_train_enc,y_train_ins)

RD_pred = RD.predict(X_test_enc)
df_results['Ridge prediction'] = RD_pred
df_results.head(10)
```

Out[ ]:

	charges	GBR prediction	LR prediction	Ridge prediction
1315	11272.33139	3794.739573	3090.897297	3092.116741
965	4746.34400	6130.585255	5862.038677	5863.740196
117	19107.77960	18050.997620	20467.764847	20459.818468

	charges	GBR prediction	LR prediction	Ridge prediction
492	2196.47320	3950.396748	3021.752660	3023.060648
69	17663.14420	16767.746385	20148.555302	20141.443845
1210	5373.36425	5646.071150	7517.069945	7518.070557
367	8017.06115	8234.765681	9483.496619	9484.832233
1085	19023.26000	18655.694723	23252.234041	23245.066902
462	15230.32405	16235.708065	15527.998928	15527.104926
1333	10600.54830	12302.421496	11717.644680	11718.446518

Lakukan evaluasi performa regresi menggunakan Mean Absolute Error, Mean Squared Error, dan Root Mean Squared Error

In [ ]:
```
print("Mean Absolute Error",metrics.mean_absolute_error(y_test_ins,RD_pred))
print("Mean Squared Error",metrics.mean_squared_error(y_test_ins,RD_pred))
print("Root Mean Squared Error",np.sqrt(metrics.mean_squared_error(y_test_ins,RD_pred)))
```

```
Mean Absolute Error 2706.4028871055702
Mean Squared Error 26044138.987415195
Root Mean Squared Error 5103.345862021816
```

Lasso Regression

Buatlah model regresi Lasso dengan pengaturan parameter alpha=0.0001 dan max_iter=1000. Parameter maximum iteration digunakan untuk membatasi iterasi yang dilakukan oleh model supaya memperoleh hasil yang optimal. Selanjutnya, lakukan prediksi menggunakan model Lasso yang barusan dibuat, kemudian dimasukkan ke DataFrame yang sebelumnya kita gunakan untuk membandingkan hasil prediksi.

In [ ]:
```
from sklearn.linear_model import Lasso
LS = Lasso(alpha=0.01,max_iter=1000).fit(X_train_enc,y_train_ins)

LS_pred = LS.predict(X_test_enc)
df_results['Lasso prediction']=LS_pred
df_results.head(20)
```

Out[ ]:

	charges	GBR prediction	LR prediction	Ridge prediction	Lasso prediction
1315	11272.33139	3794.739573	3090.897297	3092.116741	3090.934885
965	4746.34400	6130.585255	5862.038677	5863.740196	5862.117698
117	19107.77960	18050.997620	20467.764847	20459.818468	20467.627105
492	2196.47320	3950.396748	3021.752660	3023.060648	3021.760322
69	17663.14420	16767.746385	20148.555302	20141.443845	20148.452866
1210	5373.36425	5646.071150	7517.069945	7518.070557	7517.091552
367	8017.06115	8234.765681	9483.496619	9484.832233	9483.477286
1085	19023.26000	18655.694723	23252.234041	23245.066902	23252.141843
462	15230.32405	16235.708065	15527.998928	15527.104926	15527.961024
1333	10600.54830	12302.421496	11717.644680	11718.446518	11717.648939
1132	20709.02034	11842.266453	13163.303489	13162.554960	13163.319904
54	8059.67910	8016.684954	9636.003955	9636.870373	9635.971443
60	8606.21740	10153.381735	9919.398366	9920.429641	9919.418215
944	12982.87470	13122.600887	13530.645044	13530.696014	13530.644066
273	9617.66245	11364.292649	10893.531696	10894.500431	10893.564386
192	2137.65360	3319.084154	3412.949054	3415.245835	3412.979871
584	1242.81600	982.553372	1091.245653	1093.966647	1091.346076
854	24106.91255	23731.693358	26676.584094	26668.100417	26676.452024
919	5245.22690	5595.953997	7399.218642	7399.954872	7399.183000
474	25382.29700	23231.051349	26135.971842	26128.129928	26135.922036

Lakukan evaluasi model Lasso menggunakan ketiga metrik regresi.

In [ ]:
```
print("Mean Absolute Error",metrics.mean_absolute_error(y_test_ins,LS_pred))
print("Mean Squared Error",metrics.mean_squared_error(y_test_ins,LS_pred))
print("Root Mean Squared Error",np.sqrt(metrics.mean_squared_error(y_test_ins,LS_pred)))
```

```
Mean Absolute Error 2706.734393241719
Mean Squared Error 26046584.11520318
Root Mean Squared Error 5103.5854176454395
```

Lakukan visualisasi line chart untuk melihat perbandingan hasil prediksi yang dilakukan oleh Linear Regression, Ridge Regression, dan Lasso Regression.

In [ ]:
```
plt.figure(figsize=(20,5))
data_len = range(len(y_test_ins))
plt.scatter(data_len, df_results.charges, color='blue', label='actual')
plt.plot(data_len,df_results['GBR prediction'], color='red', label='GBR prediction')
plt.plot(data_len,df_results['LR prediction'], color='green', linestyle='--', label='LR prediction')
plt.plot(data_len,df_results['Ridge prediction'], color='black', linestyle='-.', label='Ridge prediction')
plt.plot(data_len,df_results['Lasso prediction'], color='yellow', linestyle=':', label='Lasso prediction')
plt.legend()
plt.show()
```