

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт з дисципліни «Основи машинного навчання мовою Python»

для магістрів спеціальності 123 Комп'ютерна інженерія
денної форми навчання

Частина II

2024

Методичні вказівки до лабораторних робіт з дисципліни «Основи машинного навчання мовою Python» для магістрів спеціальності 123 Комп'ютерна інженерія, денної форми навчання. Частина II / Укл.: М.Б. Ільяшенко, Н.А. Куликовська – Запоріжжя: НУ «Запорізька політехніка», 2024. – 42 с.

Укладач: М.Б. Ільяшенко, к.т.н., доцент
Н.А. Куликовська, старший викладач

Рецензент: М.Ю. Тягунова, к.т.н., доцент

Відповідальний
за випуск: М.Б. Ільяшенко, к.т.н., доцент

Затверджено:
На засіданні кафедри
«Комп'ютерні системи та мережі»
Протокол № 7
від 22 березня 2024р.

Рекомендовано до видання
НМК факультету КНТ
Протокол № 8
від 27 березня 2024 р.

ЗМІСТ

1	ЛАБОРАТОРНА РОБОТА №3 МОДЕЛЬ РЕГРЕСІЇ	4
1.1	Теоретичні відомості	4
1.2	Інструменти та методи.....	5
1.2.1.	Очищення даних	5
1.2.2	Перегляд відсутніх та аномальних даних.....	7
1.2.3	Розвідувальний аналіз даних (РАД)	9
1.2.4	Конструювання та вибір ознак.....	17
1.2.5	Вибір базового рівня	21
1.3	Модель лінійної регресії.....	23
1.4	Хід роботи.....	28
1.5	Зміст звіту	29
1.6	Контрольні питання	29
2	ЛАБОРАТОРНА РОБОТА №4 КЛАСИФІКАТОР ДЕРЕВА РІШЕНЬ	30
2.1	Теоретичні відомості.....	30
2.2	Класифікатор дерева рішень	31
2.3	Хід роботи.....	34
2.4	Зміст звіту	35
2.5	Контрольні питання	35
3	ЛАБОРАТОРНА РОБОТА №5 МОДЕЛЬ БУСТІНГА	36
3.1	Теоретичні відомості.....	36
3.2	Модель градієнтного бустінга.....	36
3.3	Хід роботи	39
3.4	Зміст звіту.....	40
3.5	Контрольні питання.....	40
	ЛІТЕРАТУРА.....	42

1 ЛАБОРАТОРНА РОБОТА №3

МОДЕЛЬ РЕГРЕСІЇ

Мета роботи: необхідно провести EDA (exploratory data analysis) та побудову регресії моделі, що може зменшити вартість варіації залежно від параметрів, існуючих в dataset. Це необхідна для оптимізації параметрів моделі і забезпечує всі metrics як характеристики регресивних моделей.

1.1 Теоретичні відомості

Розглянемо проєкт застосування машинного навчання із використанням реальних даних.

Послідовність етапів:

1. Очищення та форматування даних.
2. Розвідувальний аналіз даних.
3. Конструювання та вибір ознак.
4. Побудова моделі.
5. Гіперпараметричне налаштування моделі.
6. Оцінка моделі на тестовому наборі даних.
7. Інтерпретування результатів роботи моделі.
8. Висновки та робота з документами.

Мета: використовувати наявні дані для побудови моделі, яка прогнозує кількість балів Energy Star Score для конкретної будівлі та інтерпретувати результати для пошуку факторів, що впливають на підсумковий бал.

Дані вже включають присвоєні бали Energy Star Score, тому далі завдання являє собою машинне навчання з керованою регресією:

- керована (Supervised): відомі ознаки та ціль, і завдання – навчити модель, яка зможе зіставити перше з другим;
- регресія (Regression): бал Energy Star Score – це безперервна змінна.

Модель має бути точною — для прогнозування значення Energy Star Score близько до справжнього, — і інтерпретована — для розуміння її прогнозів. Знаючи цільові дані, можна використовувати їх при прийнятті рішень у міру поглиблення даних і створення моделі.

1.2 Інструменти та методи

1.2.1. Очищення даних

У реальних даних мало порядку, тому перш ніж приступити до аналізу, їх потрібно очистити і привести до прийнятного формату. Очищення даних обов'язкова процедура при вирішенні більшості завдань аналізу даних.

Спочатку треба завантажити дані у вигляді кадру даних (dataframe) Pandas та вивчити їх:

```
import pandas as pd
import numpy as np

# Read in data into a dataframe
data =
pd.read_csv('data/Energy_and_Water_Data_Disclosure_for_Local_
Law_84_2017__Data_for_Calendar_Year_2016_.csv')

# Display top of dataframe
data.head()
```

3rd Largest Property Use Type - Gross Floor Area (ft²)	Year Built	Number of Buildings - Self- reported	Occupancy	Metered Areas (Energy)	Metered Areas (Water)	ENERGY STAR Score	Site EUI (kBtu/ft²)	Weather Normalized Site EUI (kBtu/ft²)	Weather Normalized Site Electricity Intensity (kWh/ft²)	Weather Normalized Site Natural Gas Intensity (therms/ft²)
Not Available	1963	2	100	Whole Building	Not Available	Not Available	305.6	303.1	37.8	Not Available
Not Available	1969	12	100	Whole Building	Whole Building	55	229.8	228.8	24.8	2.4
Not Available	1924	1	100	Not Available	Not Available	Not Available	Not Available	Not Available	Not Available	Not Available

Рисунок 1.1 - Вигляд реальних даних

Це фрагмент таблиці із 60 колонок.

Ми не вивчатимемо всі колонки, але точно розберемося з Energy Star Score, яка описується так:

Ранжування по перцентілі від 1 до 100, яка розраховується на основі звітів про енергоспоживання, що самостійно заповнюються власниками будівель за рік. Energy Star Score – це відносний показник, який використовується для порівняння енергоефективності будівель.

Другий етап — відсутні значення, позначені як Not Available. Це рядкове значення в Python, яке означає, що навіть рядки з числами будуть зберігатися як типи даних object, тому що якщо в колонці є якась рядкова, Pandas конвертує її в колонку, що повністю складається з

рядкових. Типи даних колонок можна дізнатися за допомогою методу `dataframe.info()`:

```
# See the column data types and non-missing values
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11746 entries, 0 to 11745
Data columns (total 60 columns):
Order                                     11746 non-null int64
Property Id                             11746 non-null int64
Property Name                           11746 non-null object
Parent Property Id                      11746 non-null object
Parent Property Name                    11746 non-null object
BBL - 10 digits                         11735 non-null object
NYC Borough, Block and Lot (BBL) self-reported 11746 non-null object
NYC Building Identification Number (BIN) 11746 non-null object
Address 1 (self-reported)               11746 non-null object
Address 2                               11746 non-null object
Postal Code                             11746 non-null object
Street Number                           11622 non-null object
Street Name                             11624 non-null object
Borough                                 11628 non-null object
DOF Gross Floor Area                    11628 non-null float64
Primary Property Type - Self Selected    11746 non-null object
List of All Property Use Types at Property 11746 non-null object
Largest Property Use Type                11746 non-null object
Largest Property Use Type - Gross Floor Area (ft²) 11746 non-null object
2nd Largest Property Use Type            11746 non-null object
2nd Largest Property Use - Gross Floor Area (ft²) 11746 non-null object
3rd Largest Property Use Type            11746 non-null object
3rd Largest Property Use Type - Gross Floor Area (ft²) 11746 non-null object
```

Рисунок 1.2 - Невизначені дані

Деякі колонки, які явно містять числа (наприклад, `ft²`), збережені як об'єкти. Не можна застосовувати числовий аналіз до рядкових значень, так що конвертуємо їх у числові типи даних (особливо `float`)!

Цей код спочатку замінює всі `Not Available` на `not a number` (`np.nan`), які можна інтерпретувати як числа, а потім конвертує вміст певних колонок в тип `float`:

```
# Replace all occurrences of Not Available with numpy
not a number
data = data.replace({'Not Available': np.nan})

# Iterate through the columns
for col in list(data.columns):
    # Select columns that should be numeric
    if ('ft²' in col or 'kBtu' in col or 'Metric Tons
CO2e' in col or 'kWh' in
        col or 'therms' in col or 'gal' in col or
'Score' in col):
        # Convert the data type to float
        data[col] = data[col].astype(float)
```

Коли значення у відповідних колонках стануть числами, можна починати досліджувати дані.

1.2.2 Перегляд відсутніх та аномальних даних

Поряд із некоректними типами даних одна з найчастіших проблем – відсутні значення. Вони можуть бути відсутні з різних причин, і перед навчанням моделі ці значення потрібно заповнити або видалити. Спочатку перевіряється скільки не вистачає значень у кожній колонці:

```
# Function to calculate missing values by column
def missing_values_table(df):
    # Total missing values
    mis_val = df.isnull().sum()

    # Percentage of missing values
    mis_val_percent = 100 * df.isnull().sum() /
len(df)

    # Make a table with the results
    mis_val_table = pd.concat([mis_val,
mis_val_percent], axis=1)

    # Rename the columns
    mis_val_table_ren_columns =
mis_val_table.rename(
        columns = {0 : 'Missing Values', 1 : '% of
Total Values'})

    # Sort the table by percentage of missing
descending
    mis_val_table_ren_columns =
mis_val_table_ren_columns[
        mis_val_table_ren_columns.iloc[:,1] !=
0].sort_values(
        '% of Total Values', ascending=False).round(1)

    # Print some summary information
    print ("Your selected dataframe has " +
str(df.shape[1]) + " columns.\n"
        "There are " +
str(mis_val_table_ren_columns.shape[0]) +
        " columns that have missing values.")

    # Return the dataframe with missing information
```

```
return mis_val_table_ren_columns
missing_values_table(data)
```

Your selected dataframe has 60 columns.
There are 46 columns that have missing values.

	Missing Values	% of Total Values
Fuel Oil #1 Use (kBtu)	11737	99.9
Diesel #2 Use (kBtu)	11730	99.9
Address 2	11539	98.2
Fuel Oil #6 & 8 Use (kBtu)	11152	94.9
District Steam Use (kBtu)	10810	92.0
Fuel Oil #4 Use (kBtu)	10425	88.8
3rd Largest Property Use Type - Gross Floor Area (ft²)	10262	87.4
3rd Largest Property Use Type	10262	87.4
Fuel Oil #2 Use (kBtu)	9165	78.0
2nd Largest Property Use Type	8005	68.2
2nd Largest Property Use - Gross Floor Area (ft²)	8005	68.2
Metered Areas (Water)	4609	39.2

Рисунок 1.3 - Відсутні дані

Далі треба видалити аномальні значення. Вони можуть виникати через друкарські помилки при введенні даних або через помилки в одиницях вимірювань, або це можуть бути коректні, але екстремальні значення. В даному випадку видаляються «зайві» значення, керуючись визначенням екстремальних аномалій :

- нижче за перший кuartиль – $3 * \text{інтерквартильний розмах}$;
- вище третього кuartилю + $3 * \text{інтерквартильний розмах}$.

```
# Get the columns with > 50% missing
missing_df = missing_values_table(data);
missing_columns = list(missing_df[missing_df['% of
Total Values'] > 50].index)
print('We will remove %d columns.' %
len(missing_columns))

# Drop the columns
data = data.drop(columns = list(missing_columns))

# For older versions of pandas
(https://github.com/pandas-dev/pandas/issues/19078)
# data = data.drop(list(missing_columns), axis = 1)
```

Після завершення процесу очищення даних та видалення аномалій у нас залишилося більше 11 000 будівель та 49 ознак.

1.2.3 Розвідувальний аналіз даних (РАД)

Мета - прогнозувати значення Energy Star Score Гістограма - простий, але ефективний спосіб візуалізації розподілу одиночної змінної і її можна легко побудувати за допомогою matplotlib.

```
import matplotlib.pyplot as plt
# Histogram of the Energy Star Score
plt.style.use('fivethirtyeight')
plt.hist(data['score'].dropna(), bins = 100, edgecolor = 'k');
plt.xlabel('Score'); plt.ylabel('Number of Buildings');
plt.title('Energy Star Score Distribution');
```

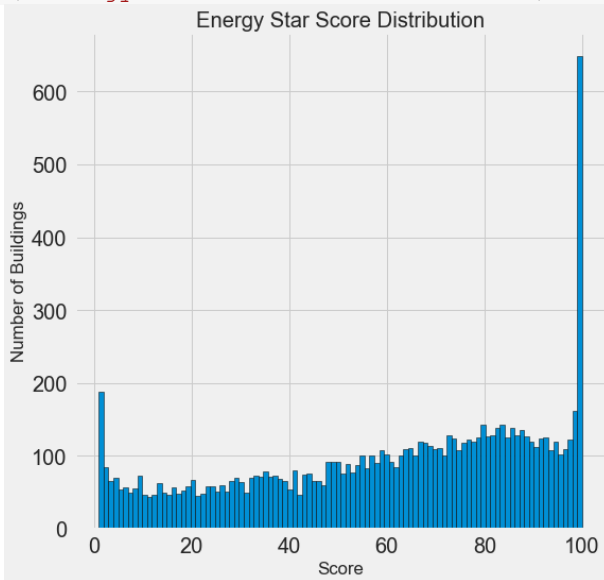


Рисунок 1.4 - Візуалізація

Бал Energy Star Score є процентилям, отже слід очікувати одноманітного розподілу, коли кожен бал присвоюється одному й тому кількості будинків. Однак вищий і нижчий результати одержало непропорційно велику кількість будівель (для Energy Star Score чим більше, тим краще).

Якщо знову подивитись на визначення цього балу, то побачимо, що він розраховується на основі звітів, що «самостійно заповнюються власниками будівель», що може пояснити надлишок дуже великих значень.

Головна частина РАД – пошук взаємозв'язків між ознаками та нашою метою. Змінні, що корелюють з нею, корисні для використання в моделі, тому що їх можна застосовувати для прогнозування.

Графік щільності можна вважати згладженою гістограмою, тому що він показує розподіл одиночної змінної. Можна розфарбувати окремі класи на графіці, щоб побачити, як категоріальна змінна змінює розподіл. Цей код буде графік щільності Energy Star Score, розфарбований залежно від типу будівлі (для списку будівель з більш ніж 100 вимірами):

```
# Create a list of buildings with more than 100
measurements
types = data.dropna(subset=['score'])
types = types['Largest Property Use
Type'].value_counts()
types = list(types[types.values > 100].index)

# Plot of distribution of scores for building
categories
figsize(12, 10)

# Plot each building
for b_type in types:
    # Select the building type
    subset = data[data['Largest Property Use Type'] ==
b_type]

    # Density plot of Energy Star Scores
    sns.kdeplot(subset['score'].dropna(),
                label = b_type, shade = False, alpha =
0.8);

# label the plot
plt.xlabel('Energy Star Score', size = 20);
plt.ylabel('Density', size = 20);
plt.title('Density Plot of Energy Star Scores by
Building Type', size = 28);
```

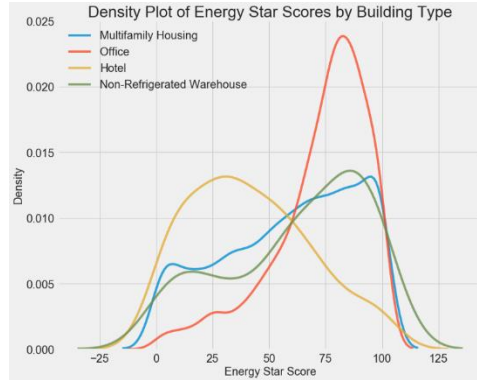


Рисунок 1.5- Залежності

Тип будівлі дуже впливає на кількість балів. Офісні будівлі зазвичай мають вищий бал, а готелі нижчий. Значить, потрібно включити тип будівлі в модель, тому що ця ознака впливає на нашу мету. Як категоріальна змінна треба виконати one-hot кодування типу будівлі.

Аналогічний графік можна використовувати для оцінки Energy Star Score районами міста:

```
# Create a list of boroughs with more than 100
observations
boroughs = data.dropna(subset=['score'])
boroughs = boroughs['Borough'].value_counts()
boroughs = list(boroughs[boroughs.values > 100].index)
# Plot of distribution of scores for boroughs
figsize(12, 10)

# Plot each borough distribution of scores
for borough in boroughs:
    # Select the building type
    subset = data[data['Borough'] == borough]

    # Density plot of Energy Star scores
    sns.kdeplot(subset['score'].dropna(),
                label = borough);

# label the plot
plt.xlabel('Energy Star Score', size = 20);
plt.ylabel('Density', size = 20);
```

```
plt.title('Density Plot of Energy Star Scores by  
Borough', size = 28);
```

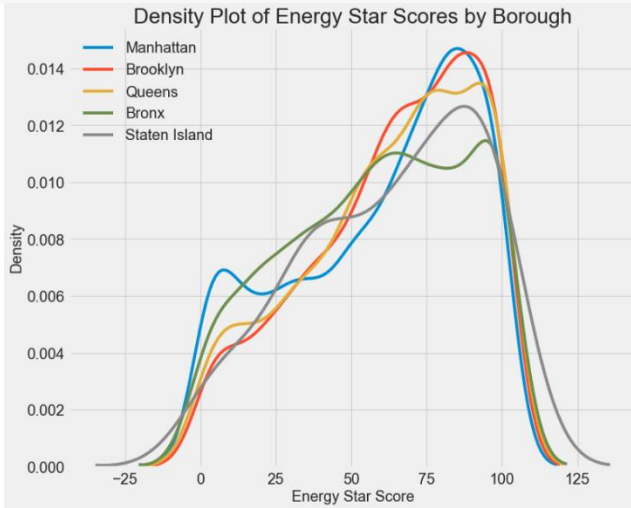


Рисунок 1.6 - Залежності

Район не так сильно впливає на бал, як тип будівлі. Тим не менш, включимо його в модель, тому що між районами існує невелика різниця.

Щоб порахувати взаємозв'язок між змінними, можна використовувати коефіцієнт кореляції Пірсона. Це міра інтенсивності та напрямки лінійної залежності між двома змінними. Значення +1 означає ідеально лінійну позитивну залежність, а -1 означає ідеально лінійну негативну залежність. Ось кілька прикладів значень коефіцієнта кореляції Пірсона:

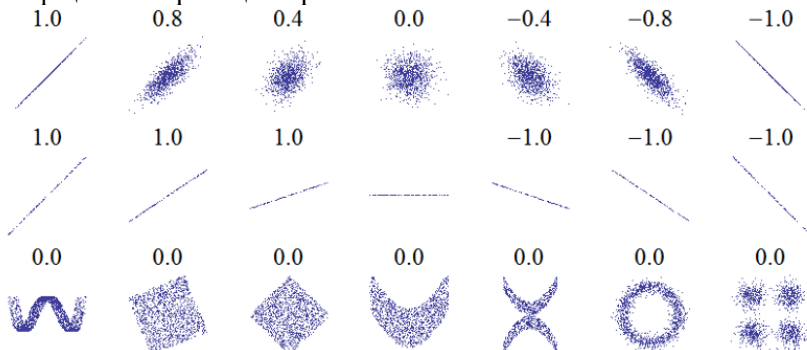


Рисунок 1.7 - Коефіцієнт Пірсона

Хоча цей коефіцієнт неспроможний відбивати нелінійні залежності, з нього можна розпочати оцінку взаємозв'язків змінних. У Pandas треба обчислити кореляції між будь-якими колонками у кадрі даних (dataframe):

```
# Find all correlations and sort
correlations_data = data.corr()['score'].sort_values()

# Print the most negative correlations
print(correlations_data.head(15), '\n')

# Print the most positive correlations
print(correlations_data.tail(15))
```

Найнегативніші кореляції з метою:

Site EUI (kBtu/ft ²)	-0.723864
Weather Normalized Site EUI (kBtu/ft ²)	-0.713993
Weather Normalized Source EUI (kBtu/ft ²)	-0.645542
Source EUI (kBtu/ft ²)	-0.641037
Weather Normalized Site Electricity Intensity (kWh/ft ²)	-0.358394

і найпозитивніші:

DOF Gross Floor Area	0.013001
Property GFA - Self-Reported (ft ²)	0.017360
Largest Property Use Type - Gross Floor Area (ft ²)	0.018330
Order	0.036827
Community Board	0.056612
Council District	0.061639

Є кілька сильних негативних кореляцій між ознаками і метою, причому найбільші з них відносяться до різних категорій EUI (способи розрахунку цих показників злегка різняться). EUI (Energy Use Intensity, інтенсивність використання енергії) - це кількість енергії, спожитої будівлею, поділена на квадратний фут площі. Ця питома величина використовується для оцінки енергоефективності, і чим вона менша, тим краще. Логіка підказує, що ці кореляції виправдані: якщо EUI збільшується, Energy Star Score має знижуватися.

Скористаємося діаграмами розсіювання для візуалізації взаємозв'язків між двома безперервними змінними. До кольорів точок можна додати додаткову інформацію, наприклад, категоріальну змінну. Нижче показано взаємозв'язок Energy Star Score та EUI, кольором позначені різні типи будівель:

```
figsize(12, 10)
```

```

# Extract the building types
features['Largest Property Use Type'] =
data.dropna(subset = ['score'])['Largest Property Use
Type']

# Limit to building types with more than 100
observations (from previous code)
features = features[features['Largest Property Use
Type'].isin(types)]

# Use seaborn to plot a scatterplot of Score vs Log
Source EUI
sns.lmplot('Site EUI (kBtu/ft²)', 'score',
           hue = 'Largest Property Use Type', data =
features,
           scatter_kws = {'alpha': 0.8, 's': 60},
fit_reg = False,
           size = 12, aspect = 1.2);

# Plot labeling
plt.xlabel("Site EUI", size = 28)
plt.ylabel('Energy Star Score', size = 28)
plt.title('Energy Star Score vs Site EUI', size = 36);

```

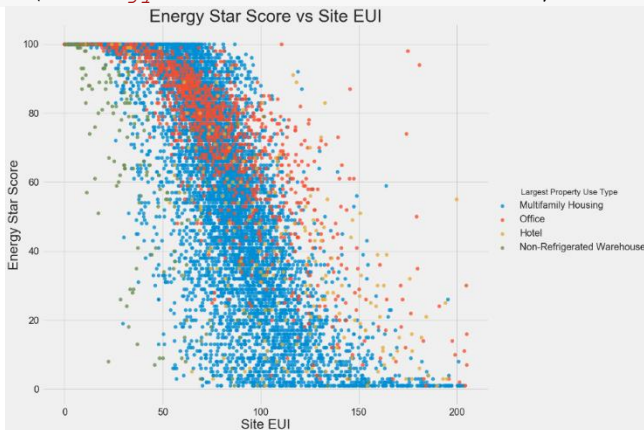


Рисунок 1.8 - Діаграма розсіювання

Цей графік дозволяє візуалізувати коефіцієнт кореляції $-0,7$. У міру зменшення EUI збільшується Energy Star Score, цей взаємозв'язок спостерігається у будівель різних типів.

Останній дослідницький графік називається Pairs Plot (парний графік). Це інструмент, що дозволяє побачити взаємозв'язки між різними парами змінних та розподіл одиночних змінних. Скористаємося бібліотекою Seaborn та функцією PairGrid для створення парного графіка з діаграмою розсіювання у верхньому трикутнику, з гистограмою по діагоналі, двовірною діаграмою щільності ядра та коефіцієнтів кореляції у нижньому трикутнику.

```
# Extract the columns to plot
plot_data = features[['score', 'Site EUI (kBtu/ft²)',
                        'Weather Normalized Source EUI
                        (kBtu/ft²)',
                        'log_Total GHG Emissions (Metric
                        Tons CO2e)']]

# Replace the inf with nan
plot_data = plot_data.replace({np.inf: np.nan, -np.inf:
np.nan})

# Rename columns
plot_data = plot_data.rename(columns = {'Site EUI
(kBtu/ft²)': 'Site EUI',
                                         'Weather
Normalized Source EUI (kBtu/ft²)': 'Weather Norm EUI',
                                         'log_Total GHG
Emissions (Metric Tons CO2e)': 'log GHG Emissions'})

# Drop na values
plot_data = plot_data.dropna()

# Function to calculate correlation coefficient between
two columns
def corr_func(x, y, **kwargs):
    r = np.corrcoef(x, y)[0][1]
    ax = plt.gca()
    ax.annotate("r = {:.2f}".format(r),
                xy=(.2, .8), xycoords=ax.transAxes,
                size = 20)

# Create the pairgrid object
grid = sns.PairGrid(data = plot_data, size = 3)
```

```
# Upper is a scatter plot
grid.map_upper(plt.scatter, color = 'red', alpha = 0.6)

# Diagonal is a histogram
grid.map_diag(plt.hist, color = 'red', edgecolor =
'black')

# Bottom is correlation and density plot
grid.map_lower(corr_func);
grid.map_lower(sns.kdeplot, cmap = plt.cm.Red)

# Title for entire plot
plt.suptitle('Pairs Plot of Energy Data', size = 36, y
= 1.02);
```

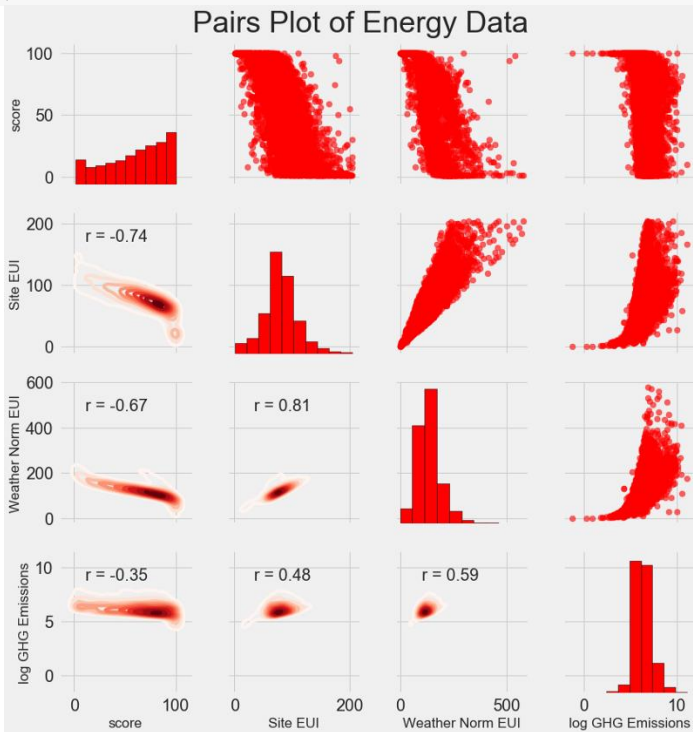


Рисунок 1.9 - Парні графіки

Щоб побачити взаємозв'язки змінних, треба подивитись на перетин рядів та колонок. Припустимо, потрібно подивитися кореляцію Weather Norm EUI і score тоді ми шукаємо ряд Weather Norm EUI і

колонку score, на перетині яких стоїть коефіцієнт кореляції -0,67. Ці графіки не тільки класно виглядають, а й допомагають вибрати змінні моделі.

1.2.4 Конструювання та вибір ознак

Конструювання та вибір ознак найчастіше приносить найбільшу віддачу з погляду часу, витраченого на машинне навчання. Спочатку дамо визначення:

Конструювання ознак: процес вилучення чи створення нових ознак із сирих даних. Щоб використовувати змінні моделі, можливо, їх доведеться перетворювати, скажімо, брати натуральний логарифм, або добувати квадратний корінь, або застосовувати one-hot кодування категоріальних змінних. Конструювання ознак можна як створення додаткових ознак із сирих даних.

Вибір ознак: процес вибору з даних найрелевантніших ознак, у ході якого ми видаляємо частину ознак, щоб допомогти моделі краще узагальнювати нові дані для отримання більш інтерпретованої моделі. Вибір ознак можна як видалення «зайвого», щоб залишилося лише найважливіше.

Модель машинного навчання може навчатися тільки на наданих нами даних, тому дуже важливо переконатися, що ми включили всю релевантну для нашого завдання інформацію. Якщо не надати моделі коректних даних, вона не зможе навчитися і не видаватиме точних прогнозів.

Для прикладу:

–застосуємо до категоріальних змінних (квартал та тип власності) one-hot кодування;

–додамо взяття натурального логарифму від усіх числових змінних.

One-hot кодування потрібне для того, щоб включити в модель категоріальні змінні. Алгоритм машинного навчання зможе зрозуміти тип «офіс», отже якщо будинок офісне, ми привласним йому ознака 1, і якщо офісне, то 0.

Додавання перетворених ознак допоможе моделі дізнатися про нелінійних взаємозв'язках всередині даних. В аналізі даних є нормальною практикою отримувати квадратне коріння, брати натуральні логарифми або ще якимось перетворювати ознаки, це

залежить від конкретного завдання або вашого знання найкращих методик. У цьому випадку ми додамо натуральний логарифм усіх числових ознак.

Цей код вибирає числові ознаки, обчислює їх логарифми, вибирає дві категоріальні ознаки, застосовує до них one-hot кодування і поєднує обидва множини в одну.

```
# Copy the original data
features = data.copy()

# Select the numeric columns
numeric_subset = data.select_dtypes('number')

# Create columns with log of numeric columns
for col in numeric_subset.columns:
    # Skip the Energy Star Score column
    if col == 'score':
        next
    else:
        numeric_subset['log_' + col] =
np.log(numeric_subset[col])

# Select the categorical columns
categorical_subset = data[['Borough', 'Largest Property
Use Type']]

# One hot encode
categorical_subset = pd.get_dummies(categorical_subset)

# Join the two dataframes using concat
# Make sure to use axis = 1 to perform a column bind
features = pd.concat([numeric_subset,
categorical_subset], axis = 1)
```

Тепер є більше 11 000 спостережень (будівель) із 110 колонками (ознаками). Не всі ознаки будуть корисними для прогнозування Energy Star Score, тому займемося вибором ознак і видалимо частину змінних.

Багато з наявних 110 ознак надмірні, тому що сильно корелюють один з одним. Наприклад, ось графік EUI та Weather Normalized Site EUI, у яких коефіцієнт кореляції дорівнює 0,997.

```

plot_data = data[['Weather Normalized Site EUI
(kBtu/ft²)', 'Site EUI (kBtu/ft²)']].dropna()

plt.plot(plot_data['Site EUI (kBtu/ft²)'],
plot_data['Weather Normalized Site EUI (kBtu/ft²)'],
'bo')
plt.xlabel('Site EUI'); plt.ylabel('Weather Norm EUI')
plt.title('Weather Norm EUI vs Site EUI, R = %0.4f' %
np.corrcoef(data[['Weather Normalized Site EUI
(kBtu/ft²)', 'Site EUI (kBtu/ft²)']].dropna(),
rowvar=False)[0][1]);

```

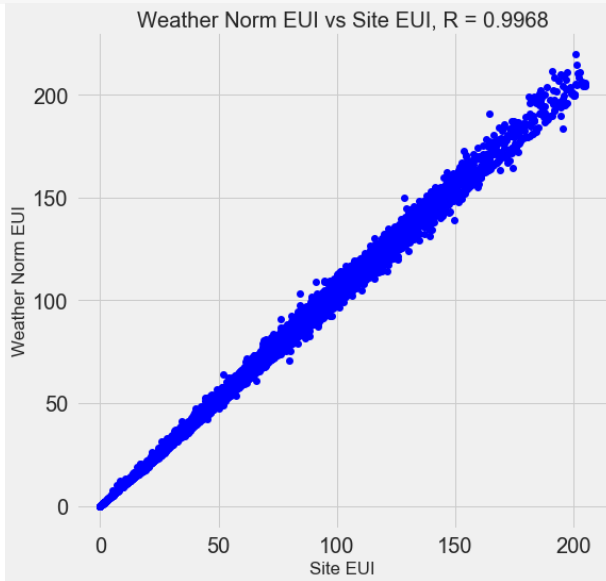


Рисунок 1.10 - Графік кореляції

Ознаки, які сильно корелюють одна з одною, називаються колінеарними. Видалення однієї змінної в таких парах ознак часто допомагає моделі узагальнювати та бути більш інтерпретованою. Зверніть увагу, що йдеться про кореляцію одних ознак з іншими, а не про кореляцію з метою, що тільки допомогло б нашій моделі!

Існує ряд методів обчислення колінеарності ознак, і один із найпопулярніших - фактор збільшення дисперсії (variance inflation factor). Для пошуку та видалення колінеарних ознак скористаємось коефіцієнтом В-кореляції (thebcorrelation coefficient). Відкинемо одну пару ознак, якщо коефіцієнт кореляції між ними більше 0,6.

```
def remove_collinear_features(x, threshold):
    '''
    Objective:
        Remove collinear features in a dataframe with a
        correlation coefficient
        greater than the threshold. Removing collinear
        features can help a model
        to generalize and improves the interpretability
        of the model.

    Inputs:
        threshold: any features with correlations
        greater than this value are removed

    Output:
        dataframe that contains only the non-highly-
        collinear features
    '''

    # Dont want to remove correlations between Energy
    Star Score
    y = x['score']
    x = x.drop(columns = ['score'])

    # Calculate the correlation matrix
    corr_matrix = x.corr()
    iters = range(len(corr_matrix.columns) - 1)
    drop_cols = []

    # Iterate through the correlation matrix and
    compare correlations
    for i in iters:
        for j in range(i):
            item = corr_matrix.iloc[j:(j+1),
            (i+1):(i+2)]
            col = item.columns
            row = item.index
            val = abs(item.values)

            # If correlation exceeds the threshold
            if val >= threshold:
```

```

        # Print the correlated features and the
correlation value
        # print(col.values[0], "|",
row.values[0], "|", round(val[0][0], 2))
        drop_cols.append(col.values[0])

    # Drop one of each pair of correlated columns
    drops = set(drop_cols)
    x = x.drop(columns = drops)
    x = x.drop(columns = ['Weather Normalized Site EUI
(kBtu/ft2)',
                        'Water Use (All Water
Sources) (kgal)',
                        'log_Water Use (All Water
Sources) (kgal)',
                        'Largest Property Use Type -
Gross Floor Area (ft2)'])

    # Add the score back in to the data
    x['score'] = y

    return x
# Remove the collinear features above a specified
correlation coefficient
features = remove_collinear_features(features, 0.6);

# Remove any columns with all na values
features = features.dropna(axis=1, how = 'all')
features.shape

```

1.2.5 Вибір базового рівня

Для регресійних завдань як базового рівня треба використовувати медіанну значення мети на навчальному наборі всіх прикладів у тестовому наборі. Ці набори задають бар'єр відносно низький для будь-якої моделі.

Для метрики візьмемо середню абсолютну помилку (mae) у прогнозах.

Перш ніж обчислювати базовий рівень, потрібно розбити дані на навчальний та тестовий набори.

Навчальний набір ознак – те, що надається моделі разом із відповідями під час навчання. Модель має вивчити відповідність ознак мети.

Тестовий набір ознак використовується для оцінки навченої моделі. Коли вона обробляє тестовий набір, то не бачить правильних відповідей і має прогнозувати, спираючись лише на доступні ознаки. Відомі відповіді для тестових даних та можемо порівняти з ними результати прогнозування.

Для навчання використовуємо 70% даних, а для тестування – 30%:

```
# Split into 70% training and 30% testing set
X, X_test, y, y_test = train_test_split(features,
                                         targets,
                                         test_size =
0.3,
                                         random_state =
42)
```

Обчислення показника для вихідного базового рівня:

```
# Function to calculate mean absolute error
def mae(y_true, y_pred):
    return np.mean(abs(y_true - y_pred))

baseline_guess = np.median(y)

print('The baseline guess is a score of %0.2f' %
      baseline_guess)
print("Baseline Performance on the test set: MAE =
%0.4f" % mae(y_test, baseline_guess))
```

Загальна абсолютна помилка на тестовому наборі склала близько 25 пунктів. Оскільки ми оцінюємо в діапазоні від 1 до 100, то помилка становить 25% – досить низький бар'єр для моделі.

```
# Save the no scores, training, and testing data
no_score.to_csv('data/no_score.csv', index = False)
X.to_csv('data/training_features.csv', index = False)
X_test.to_csv('data/testing_features.csv', index =
False)
```

```
y.to_csv('data/training_labels.csv', index = False)
y_test.to_csv('data/testing_labels.csv', index = False)
```

1.3 Модель лінійної регресії

Підключаємо бібліотеки:

```
# Pandas and numpy for data manipulation
import pandas as pd
import numpy as np

# No warnings about setting value on copy of slice
pd.options.mode.chained_assignment = None
pd.set_option('display.max_columns', 60)

# Matplotlib for visualization
import matplotlib.pyplot as plt
%matplotlib inline

# Set default font size
plt.rcParams['font.size'] = 24

from IPython.core.pylabtools import figsize

# Seaborn for visualization
import seaborn as sns
sns.set(font_scale = 2)

# Imputing missing values and scaling values
from sklearn.preprocessing import Imputer, MinMaxScaler

# Machine Learning Models
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor

# Hyperparameter tuning
from sklearn.model_selection import RandomizedSearchCV,
GridSearchCV
```

Моделі машинного навчання не можуть працювати з відсутніми даними, тому нам потрібно заповнити їх .

Спочатку завантажуюмо дані:

```
import pandas as pd
import numpy as np
# Read in data into dataframes
train_features =
pd.read_csv('data/training_features.csv')
test_features =
pd.read_csv('data/testing_features.csv')
train_labels = pd.read_csv('data/training_labels.csv')
test_labels = pd.read_csv('data/testing_labels.csv')
Training Feature Size: (6622, 64)
Testing Feature Size: (2839, 64)
Training Labels Size: (6622, 1)
Testing Labels Size: (2839, 1)
```

Кожне NaN значення - це відсутній запис у даних. Заповнювати їх можна по-різному, скористаємось методом медіанного заповнення (median imputation), який замінює відсутні дані середні значення по відповідних колонках.

У наведеному нижче коді ми створимо Scikit-Learn - об'єкт Imputer з медіанною стратегією. Навчимо його на навчальних даних (за допомогою imputer.fit), і застосуємо для заповнення відсутніх значень у навчальному та тестовому наборах (за допомогою imputer.transform). Тобто записи, яких не вистачає в тестових даних, заповнюватимуться відповідним медіанним значенням з навчальних даних.

Робимо заповнення і не навчаємо модель даних як є, щоб уникнути проблеми з витокот тестових даних, коли інформація з тестового датасету переходить в навчальний.

```
# Create an imputer object with a median filling
strategy
imputer = Imputer(strategy='median')
# Train on the training features
imputer.fit(train_features)
# Transform both training data and testing data
X = imputer.transform(train_features)
X_test = imputer.transform(test_features)
Missing values in training features: 0
Missing values in testing features: 0
```

Тепер всі значення заповнені.

Масштабуванням називається загальний процес зміни діапазону ознаки. Це необхідний крок, тому що ознаки вимірюються в різних одиницях, а отже, покривають різні діапазони. Це сильно спотворює результати таких алгоритмів, як метод опорних векторів і метод найближчих сусідів, які враховують відстані між вимірюваннями. А масштабування дозволяє цього уникнути. І хоча методи на кшталт лінійної регресії та «випадкового лісу» не потребують масштабування ознак, краще не хтувати цим етапом при порівнянні кількох алгоритмів.

Масштабуватимемо за допомогою приведення кожної ознаки до діапазону від 0 до 1. Вибираємо всі значення ознаки, вибираємо мінімальне та ділимо його на різницю між максимальним та мінімальним (діапазон). Такий спосіб масштабування часто називають нормалізацією, а інший основний спосіб - стандартизацією.

Цей процес реалізовується вручну, тому скористаємося об'єктом `MinMaxScaler` з `Scikit-Learn`. Код цього методу ідентичний коду для заповнення відсутніх значень, тільки замість вставки застосовується масштабування.

```
# Create the scaler object with a range of 0-1
scaler = MinMaxScaler(feature_range=(0, 1))

# Fit on the training data
scaler.fit(X)

# Transform both the training and testing data
X = scaler.transform(X)
X_test = scaler.transform(X_test)
# Convert y to one-dimensional array (vector)
y = np.array(train_labels).reshape((-1, ))
y_test = np.array(test_labels).reshape((-1, ))
```

Тепер у кожної ознаки мінімальне значення дорівнює 0, а максимальне 1. Заповнення відсутніх значень та масштабування ознак – ці два етапи потрібні майже у будь-якому процесі машинного навчання.

Приклад процесу створення, навчання (`.fit`) та тестування (`.predict`):

```
# Function to calculate mean absolute error
def mae(y_true, y_pred):
    return np.mean(abs(y_true - y_pred))
```

```
# Takes in a model, trains the model, and evaluates the
model on the test set
def fit_and_evaluate(model):

    # Train the model
    model.fit(X, y)

    # Make predictions and evaluate
    model_pred = model.predict(X_test)
    model_mae = mae(y_test, model_pred)

    # Return the performance metric
    return model_mae
lr = LinearRegression()
lr_mae = fit_and_evaluate(lr)

print('Linear Regression Performance on the test set:
MAE = %0.4f' % lr_mae)
```

Порівняння результатів алгоритму за замовчуванням та налаштованої моделі:

```
default_pred = default_model.predict(X_test)
final_pred = final_model.predict(X_test)

print('Default model performance on the test set: MAE =
%0.4f.' % mae(y_test, default_pred))
print('Final model performance on the test set: MAE =
%0.4f.' % mae(y_test, final_pred))
```

Щоб отримати уявлення про прогнози, треба побудувати графік розподілу істинних значень у тестовому наборі та прогнозованих значень у тестовому наборі.

```
figsize(8, 8)

# Density plot of the final predictions and the test
values
sns.kdeplot(final_pred, label = 'Predictions')
sns.kdeplot(y_test, label = 'Values')

# Label the plot
plt.xlabel('Energy Star Score'); plt.ylabel('Density');
```

```
plt.title('Test Values and Predictions');
```

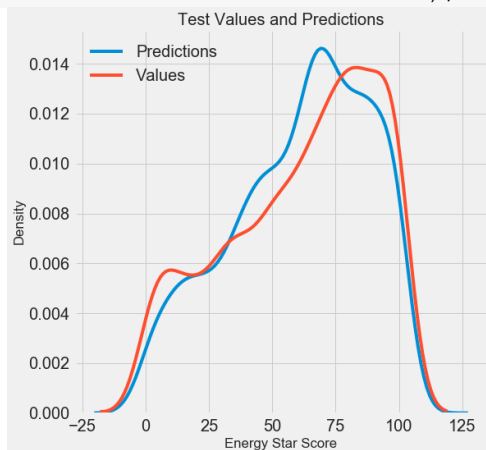


Рисунок 1.11 - Графік розподілу

Іншим діагностичним графіком є гістограма залишків. Найкращій результат, що залишки розподілені нормально, тобто модель помиляється однаково в обох напрямках (високому та низькому).

```
figsize = (6, 6)
```

```
# Calculate the residuals
residuals = final_pred - y_test

# Plot the residuals in a histogram
plt.hist(residuals, color = 'red', bins = 20,
         edgecolor = 'black')
plt.xlabel('Error'); plt.ylabel('Count')
plt.title('Distribution of Residuals');
```

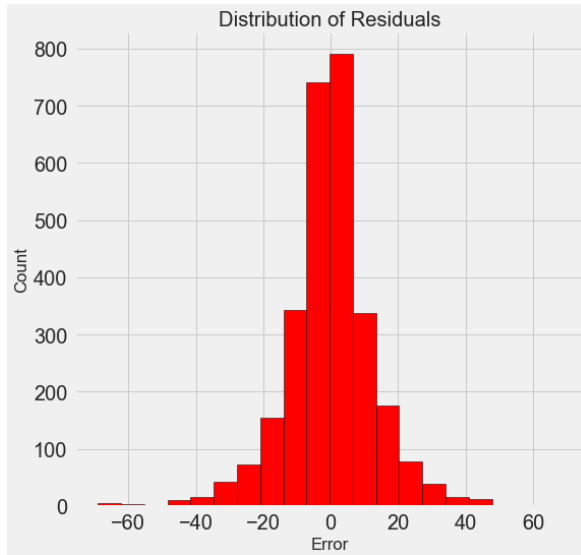


Рисунок 1.12 - Гістограма залишків

Залишки близькі до нормального розподілу з кількома помітними викидами на нижньому кінці. Вони вказують на помилки, коли оцінка моделі була набагато нижчою від справжнього значення.

1.4 Хід роботи

Завдання: Необхідно провести EDA і побудувати регресійну модель, яка зможе прогнозувати вартість лікування залежно від параметрів, що є в датасеті. Необхідно провести оптимізацію параметрів моделі та навести всі метрики характерні для моделей регресії.

Для виконання завдання необхідно використати дані про пацієнтів і вартість медичних послуг для них:

age - вік

sex - стать

bmi (body mass index) – індекс маси тіла

children - кількість дітей

smoker - курить чи ні

region - регіон проживання

charges - вартість лікування (значення, яке необхідно прогнозувати)

Результатом роботи є Jupyter Notebook із коментарями до виконаної роботи.

1.5 Зміст звіту

Звіт повинен мати такі складові:

- титульний лист;
- мета роботи;
- результат виконання завдань (програмний код, графіки, гістограми);
- відповіді на контрольні питання.

1.6 Контрольні питання

1. Що таке "one-hot кодування" і для чого воно застосовується?
2. Що таке "коефіцієнт кореляції Пірсона" і як він використовується для аналізу даних?
3. Що таке "парний графік" і для чого він використовується?
4. Що таке "колінеарність ознак" і чому її важливо усувати?
5. Що таке "вихідний базовий рівень" і як його обчислити?
6. Що таке "заповнення відсутніх значень" і чому це важливо для моделей машинного навчання?
7. Що таке "масштабування ознак" і чому воно корисне?
8. Які метрики використовуються для оцінки моделей регресії в цьому прикладі?

2 ЛАБОРАТОРНА РОБОТА №4

КЛАСИФІКАТОР ДЕРЕВА РІШЕНЬ

Мета роботи: провести аналіз та роботу щодо запобігання перенавченню моделей. Вивести всі метрики, які використовуються для моделей класифікації. Пояснити межі застосування кожного з рекомендованих до застосування в роботі варіантів DT.

2.1 Теоретичні відомості

Дерева рішень (DTs) — це непараметричний контрольований метод навчання, який використовується для класифікації та регресії. Мета полягає в тому, щоб створити модель, яка передбачає значення цільової змінної шляхом вивчення простих правил прийняття рішень, виведених із характеристик даних. Дерево можна розглядати як частково-постійне наближення.

Деякі переваги дерев рішень:

- прості для розуміння та інтерпретації. Дерева можна візуалізувати;

- вимагають невеликої підготовки даних. Інші методи часто вимагають нормалізації даних, необхідно створити фіктивні змінні та видалити порожні значення. Деякі комбінації дерева та алгоритму підтримують відсутні значення;

- вартість використання дерева (тобто прогнозування даних) є логарифмічною щодо кількості точок даних, які використовуються для навчання дерева;

- здатний обробляти як числові, так і категоріальні дані. Однак реалізація scikit-learn наразі не підтримує категоріальні змінні. Інші методи зазвичай спеціалізуються на аналізі наборів даних, які мають лише один тип змінної;

- здатні вирішувати проблеми з кількома виходами;

- використовує модель білого ящика. Якщо дана ситуація є спостережуваною в моделі, пояснення умови легко пояснити булевою логікою. Навпаки, у моделі чорного ящика (наприклад, у штучній нейронній мережі) результати може бути складніше інтерпретувати;

– можливість перевірити модель за допомогою статистичних тестів. Це дозволяє враховувати надійність моделі.

До недоліків дерев рішень належать:

– дерева рішень можуть бути нестабільними, оскільки невеликі варіації в даних можуть призвести до створення зовсім іншого дерева. Цю проблему можна пом'якшити, використовуючи дерева рішень у ансамблі;

– проблема вивчення оптимального дерева рішень, як відомо, є NP-повною за кількома аспектами оптимальності та навіть для простих понять. Отже, практичні алгоритми навчання дерева рішень базуються на евристичних алгоритмах, таких як жадібний алгоритм, де локально оптимальні рішення приймаються на кожному вузлі. Такі алгоритми не можуть гарантувати повернення глобально оптимального дерева рішень. Це можна пом'якшити, навчивши кілька дерев у груповому модулі, де функції та зразки відбираються випадковим чином із заміною;

– існують концепції, які важко вивчити, оскільки дерева рішень нелегко виражають їх, наприклад XOR, проблеми парності або мультиплексора.

2.2 Класифікатор дерева рішень

Підключаємо бібліотеки:

```
# Pandas and numpy for data manipulation
import pandas as pd
import numpy as np

# No warnings about setting value on copy of slice
pd.options.mode.chained_assignment = None
pd.set_option('display.max_columns', 60)

# Matplotlib for visualization
import matplotlib.pyplot as plt
%matplotlib inline

# Set default font size
plt.rcParams['font.size'] = 24

from IPython.core.pylabtools import figsize
```

```
# Seaborn for visualization
import seaborn as sns
sns.set(font_scale = 2)

# Imputing missing values and scaling values
from sklearn.preprocessing import Imputer, MinMaxScaler

# Machine Learning Models
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor

# Hyperparameter tuning
from sklearn.model_selection import RandomizedSearchCV,
GridSearchCV
```

На прикладі даних лабораторної роботи 3 треба виконати наступні дії. Хоча при очищенні даних відкинули стовпчики, в яких не вистачає більше половини значень, у нас ще немає чимало значень. Моделі машинного навчання не можуть працювати з відсутніми даними, тому нам потрібно заповнити їх .

Спочатку завантажуюмо дані:

```
import pandas as pd
import numpy as np
# Read in data into dataframes
train_features =
pd.read_csv('data/training_features.csv')
test_features =
pd.read_csv('data/testing_features.csv')
train_labels = pd.read_csv('data/training_labels.csv')
test_labels = pd.read_csv('data/testing_labels.csv')
Training Feature Size:  (6622, 64)
Testing Feature Size:   (2839, 64)
Training Labels Size:   (6622, 1)
Testing Labels Size:    (2839, 1)
```

Кожне NaNзначення - це відсутня запис у даних. Заповнювати їх можна по-різному , а ми скористаємося досить простим методом медіанного заповнення (median imputation), який замінює відсутні дані середні значення по відповідних колонках.

У наведеному нижче коді створюється Scikit-Learn -об'єкт `Imputer` медіанною стратегією. Потім навчаємо його за допомогою `imputer.fit`, і застосуємо для заповнення відсутніх значень у навчальному та тестовому наборах (за допомогою `imputer.transform`). Тобто записи, яких не вистачає в тестових даних, заповнюватимуться відповідним медіанним значенням з навчальних даних.

Робимо заповнення і не навчаємо модель даних як є, щоб уникнути проблеми з витокм тестових даних, коли інформація з тестового датасету переходить в навчальний.

```
# Create an imputer object with a median filling strategy
imputer = Imputer(strategy='median')
# Train on the training features
imputer.fit(train_features)
# Transform both training data and testing data
X = imputer.transform(train_features)
X_test = imputer.transform(test_features)
Missing values in training features:  0
Missing values in testing features:    0
```

Тепер всі значення заповнені.

```
# Create the scaler object with a range of 0-1
scaler = MinMaxScaler(feature_range=(0, 1))

# Fit on the training data
scaler.fit(X)

# Transform both the training and testing data
X = scaler.transform(X)
X_test = scaler.transform(X_test)
# Convert y to one-dimensional array (vector)
y = np.array(train_labels).reshape((-1, ))
y_test = np.array(test_labels).reshape((-1, ))
```

Тепер у кожної ознаки мінімальне значення дорівнює 0, а максимальне 1. Заповнення відсутніх значень та масштабування ознак – ці два етапи потрібні майже у будь-якому процесі машинного навчання.

Після всіх підготовчих робіт процес створення, навчання та прогону моделей відносно простий.

Далі наведено процес створення, навчання (`.fit`) та тестування (`.predict`):

```

# Function to calculate mean absolute error
def mae(y_true, y_pred):
    return np.mean(abs(y_true - y_pred))

# Takes in a model, trains the model, and evaluates the
# model on the test set
def fit_and_evaluate(model):

    # Train the model
    model.fit(X, y)

    # Make predictions and evaluate
    model_pred = model.predict(X_test)
    model_mae = mae(y_test, model_pred)

    # Return the performance metric
    return model_mae

random_forest = RandomForestRegressor(random_state=60)
random_forest_mae = fit_and_evaluate(random_forest)

print('Random Forest Regression Performance on the test
set: MAE = %0.4f' % random_forest_mae)

```

2.3 Хід роботи

Завдання: Необхідно провести EDA та побудувати класифікатор на базі DT. Спробувати застосувати:

- "Дрібне" дерево прийняття рішень
- "Глибоке" дерево прийняття рішень
- Ліс із "дрібних" дерев
- Ліс із "глибоких" дерев

Провести аналіз та роботу щодо запобігання перенавченню моделей. Вивести всі метрики, які використовуються для моделей класифікації. Пояснити межі застосування кожного з рекомендованих до застосування в роботі варіантів DT.

Для виконання завдання необхідно використати дані про біологічну відповідь молекул в залежності від їх хімічних властивостей.

Поля D1-D1776 – описують хімічні властивості молекул.

Поле target - містить два значення 0 або 1 - чи є молекула біологічно активною чи ні

Результатом роботи є Jupyter Notebook із коментарями.

2.4 Зміст звіту

Звіт повинен мати такі складові:

- титульний лист;
- мета роботи;
- результат виконання завдань (програмний код, графіки, гістограми);
- відповіді на контрольні питання.

2.5 Контрольні питання

1. Що таке дерева рішень (DTs) і для чого вони використовуються?
2. Які основні переваги використання дерев рішень згадані в тексті?
3. Які недоліки дерев рішень описані в тексті?
4. Що таке "перенавчення" в контексті дерев рішень?
5. Які методи запропоновані в тексті для запобігання перенавченню моделей?
6. Що мається на увазі під "дрібним" деревом прийняття рішень?
7. Що мається на увазі під "глибоким" деревом прийняття рішень?
8. Що таке "ліс з дрібних дерев" і як він може допомогти з перенавченням?
9. Які метрики використовуються для оцінки моделей класифікації?

3 ЛАБОРАТОРНА РОБОТА №5 МОДЕЛЬ БУСТІНГА

Мета роботи: Необхідно провести EDA і вирішити завдання бінарної класифікації, використовуючи один або два варіанти бустингу. Необхідно провести оптимізацію моделі та навести метрики необхідні для завдання бінарної класифікації.

3.1 Теоретичні відомості

Бустінг — це техніка ансамблевого моделювання, яка намагається побудувати сильний класифікатор із числа слабких класифікаторів. Це робиться шляхом побудови моделі з використанням слабких моделей у послідовності. По-перше, модель будується з навчальних даних. Потім будується друга модель, яка намагається виправити помилки, присутні в першій моделі. Ця процедура продовжується, і моделі додаються, доки не буде правильно передбачено повний набір навчальних даних або додано максимальну кількість моделей.

Переваги бустингу

- покращена точність. Підвищення може підвищити точність моделі, об'єднавши кілька слабких моделей і усереднивши їх для регресії або голосуючи за них для класифікації, щоб підвищити точність остаточної моделі;

- стійкість до переобладнання – посилення може зменшити ризик переобладнання шляхом переважаювання неправильно класифікованих вхідних даних;

- покращена обробка незбалансованих даних – Boosting може впоратися з незбалансованими даними, зосереджуючись більше на точках даних, які неправильно класифіковані;

- краща інтерпретація – посилення може підвищити інтерпретабельність моделі шляхом розбиття процесу прийняття рішення на кілька процесів;

3.2 Модель градієнтного бустінга

Підключення бібліотек:

```

# Pandas and numpy for data manipulation
import pandas as pd
import numpy as np

# No warnings about setting value on copy of slice
pd.options.mode.chained_assignment = None
pd.set_option('display.max_columns', 60)

# Matplotlib for visualization
import matplotlib.pyplot as plt
%matplotlib inline

# Set default font size
plt.rcParams['font.size'] = 24

from IPython.core.pylabtools import figsize

# Seaborn for visualization
import seaborn as sns
sns.set(font_scale = 2)

# Imputing missing values and scaling values
from sklearn.preprocessing import Imputer, MinMaxScaler

# Machine Learning Models
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor

# Hyperparameter tuning
from sklearn.model_selection import RandomizedSearchCV,
GridSearchCV

```

На прикладі даних лабораторної роботи 3 треба виконати наступні дії. Моделі машинного навчання не можуть працювати з відсутніми даними, тому потрібно заповнити їх .

Завантаження даних:

```

import pandas as pd
import numpy as np

```

```
# Read in data into dataframes
train_features =
pd.read_csv('data/training_features.csv')
test_features =
pd.read_csv('data/testing_features.csv')
train_labels = pd.read_csv('data/training_labels.csv')
test_labels = pd.read_csv('data/testing_labels.csv')
Training Feature Size: (6622, 64)
Testing Feature Size: (2839, 64)
Training Labels Size: (6622, 1)
Testing Labels Size: (2839, 1)
```

Робимо заповнення і не навчаємо модель даних як є, щоб уникнути проблеми з витоком тестових даних, коли інформація з тестового датасету переходить в навчальний.

```
# Create an imputer object with a median filling
strategy
imputer = Imputer(strategy='median')
# Train on the training features
imputer.fit(train_features)
# Transform both training data and testing data
X = imputer.transform(train_features)
X_test = imputer.transform(test_features)
Missing values in training features: 0
Missing values in testing features: 0
```

Тепер всі значення заповнені.

```
# Create the scaler object with a range of 0-1
scaler = MinMaxScaler(feature_range=(0, 1))

# Fit on the training data
scaler.fit(X)

# Transform both the training and testing data
X = scaler.transform(X)
X_test = scaler.transform(X_test)
# Convert y to one-dimensional array (vector)
y = np.array(train_labels).reshape((-1, ))
y_test = np.array(test_labels).reshape((-1, ))
```

Тепер у кожної ознаки мінімальне значення дорівнює 0, а максимальне 1. Заповнення відсутніх значень та масштабування ознак – ці два етапи потрібні майже у будь-якому процесі машинного навчання.

Процес створення, навчання (`.fit`) та тестування (`.predict`) ми будемо за допомогою:

```
# Function to calculate mean absolute error
def mae(y_true, y_pred):
    return np.mean(abs(y_true - y_pred))

# Takes in a model, trains the model, and evaluates the
model on the test set
def fit_and_evaluate(model):

    # Train the model
    model.fit(X, y)

    # Make predictions and evaluate
    model_pred = model.predict(X_test)
    model_mae = mae(y_test, model_pred)

    # Return the performance metric
    return model_mae

gradient_boosted =
GradientBoostingRegressor(random_state=60)
gradient_boosted_mae =
fit_and_evaluate(gradient_boosted)

print('Gradient Boosted Regression Performance on the
test set: MAE = %0.4f' % gradient_boosted_mae)
```

3.3 Хід роботи

Для виконання завдання необхідно використати дані, які були зібрані системою в Гуз-Бей, Лабрадор. Ця система складається з фазованої решітки з 16 високочастотних антен із загальною потужністю, що передається, близько 6,4 кіловат. Мішенями були вільні електрони в іоносфері. "Хороші" відображення радарів - це ті, які свідчать про наявність будь-якої структури в іоносфері. "Погані" результати - це ті, які не повертаються; їх сигнали проходять через іоносферу.

Отримані сигнали оброблялися за допомогою автокореляційної функції, аргументами якої є час імпульсу та номер імпульсу. Для системи Goose Bay було 17 номерів імпульсів. Примірники у цій базі даних описуються двома атрибутами на номер імпульсу, що відповідають комплексним значенням, що повертається функцією, отриманою зі складного електромагнітного сигналу.

Колонки a01-a34 – результати вимірювань

class - один із двох класів "g" (good) або "b" (bad) які необхідно передбачити

Завдання. Необхідно провести EDA і вирішити завдання бінарної класифікації, використовуючи один або два варіанти бустингу. Необхідно провести оптимізацію моделі та навести метрики необхідні для завдання бінарної класифікації.

3.4 Зміст звіту

Звіт повинен мати такі складові:

- титульний лист;
- мета роботи;
- результат виконання завдань (програмний код, графіки, гістограми);
- відповіді на контрольні питання.

3.5 Контрольні питання

1. Що таке бустинг і яка його основна ідея?
2. Які основні переваги використання методів бустингу згадані в тексті?
3. Що таке градієнтний бустинг?
4. Для чого потрібно заповнювати відсутні значення в даних перед побудовою моделей машинного навчання?
5. Який метод заповнення відсутніх значень використовується в прикладі?
6. Що таке масштабування ознак і чому воно є важливим кроком?

7. Який метод масштабування ознак використовується в прикладі?
8. Що означають колонки a01-a34 в наданому датасеті?

JITEPATYPA

1. Machine Learning and Knowledge Extraction [Electronic resource] / ed. by A. Holzinger [et al.]. – Cham : Springer Nature Switzerland, 2023. – Mode of access: <https://doi.org/10.1007/978-3-031-40837-3>
2. Intelligent Systems and Machine Learning [Electronic resource] / ed. by S. Nandan Mohanty, V. Garcia Diaz, G. A. E. Satish Kumar. – Cham : Springer Nature Switzerland, 2023. – Mode of access: <https://doi.org/10.1007/978-3-031-35078-8>
3. Joshi A. V. Machine Learning and Artificial Intelligence [Electronic resource] / Ameet V. Joshi. – Cham : Springer International Publishing, 2023. – Mode of access: <https://doi.org/10.1007/978-3-031-12282-8>
4. Zollanvari A. Machine Learning with Python [Electronic resource] / Amin Zollanvari. – Cham : Springer International Publishing, 2023. – Mode of access: <https://doi.org/10.1007/978-3-031-33342-2>
5. Hu F. AI, Machine Learning and Deep Learning [Electronic resource] / Fei Hu, Xiali Hei. – Boca Raton : CRC Press, 2023. – Mode of access: <https://doi.org/10.1201/9781003187158>
6. Karpatne A. Knowledge-Guided Machine Learning [Electronic resource] / Anuj Karpatne, Ramakrishnan Kannan, Vipin Kumar. – Boca Raton : Chapman and Hall/CRC, 2022. – Mode of access: <https://doi.org/10.1201/9781003143376>
7. Aggarwal C. C. Machine Learning for Text [Electronic resource] / Charu C. Aggarwal. – Cham : Springer International Publishing, 2022. – Mode of access: <https://doi.org/10.1007/978-3-030-96623-2>
8. Valkenborg D. Mathematics for Machine Learning [Electronic resource] / Dirk Valkenborg, Jeroen Wynen. – Washington, DC, USA : American Chemical Society, 2022. – Mode of access: <https://doi.org/10.1021/acsinfocus.7e5034>
9. Indovina J. Machine Learning Systems: Learn How to Develop Machine-Learning Systems / Johnson Indovina. – [S. l.] : Independently Published, 2022.
10. Applied Machine Learning / Setareh Rafatirad [et al.]. – [S. l.] : Springer International Publishing AG, 2022.