

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
Факультет комп'ютерно інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

Методичні вказівки до виконання лабораторних робіт з
дисципліни

«Машинне навчання»

для студентів освітньо-професійної програми «Штучний інтелект»
спеціальності 122 «Комп'ютерні науки» першого (бакалаврського)
рівня вищої освіти

м. Тернопіль – ЗУНУ

2024 р.

Методичні вказівки до виконання лабораторних робіт з дисципліни «Машинне навчання» для студентів освітньо-професійної програми «Штучний інтелект» спеціальності 122 «Комп'ютерні науки» першого (бакалаврського) рівня вищої освіти / Укладачі: Ліп'яніна-Гончаренко Х.В. та Комар М.П. – Тернопіль, ЗУНУ, 2024, – с.80

Укладачі:

- *Ліп'яніна-Гончаренко Х.В., доцент, к.т.н.*
- *Комар М.П., професор, д.т.н.*

Рецензенти:

- Денис А. С. – Керівник ТОВ “СААСДЖЕТ”
- Піцун О.Й. – к.т.н., доцент кафедри комп'ютерної інженерії Західноукраїнського національного університету

Рекомендовано на засіданні кафедри
(Протокол № 1 від «27» серпня 2024р.)

Зміст

Вступ	4
Лабораторна робота №1. Кластерний аналіз.....	5
Лабораторна робота №2. Регуляризаційні лінійні регресійні моделі	17
Лабораторна робота №3. Методи опорних векторів (SVM)	21
Лабораторна робота №4. Дерева рішень	26
Лабораторна робота №5. Ансамблеве навчання	34
Лабораторна робота №6. Добування ознак із зображень	40
Лабораторна робота №7. Обробка природної мови	50
Лабораторна робота №8. Масштабування процесу машинного навчання.....	59
Додаток А. Перелік варіантів для лабораторних робіт 1-5	74
Додаток Б. Перелік варіантів для лабораторної роботи 6	76
Додаток В. Перелік варіантів для лабораторної роботи 7	78
Література	80

Вступ

Методичні вказівки до лабораторних робіт з машинного навчання охоплюють основні методи та підходи, які використовуються для аналізу даних, побудови моделей та створення прогнозів. Сучасні технології машинного навчання активно застосовуються у різних галузях — від обробки великих обсягів даних до оптимізації бізнес-процесів і розробки штучного інтелекту. У цьому курсі студенти ознайомляться з ключовими алгоритмами та методами, що застосовуються для розв’язання різноманітних задач.

Метою цих лабораторних робіт є надання практичних навичок щодо використання різних методів машинного навчання на реальних наборах даних. Кожна лабораторна робота присвячена окремому методу або техніці, починаючи з основних алгоритмів кластеризації та лінійних регресійних моделей і закінчуючи складними техніками, як ансамблеве навчання та масштабування моделей за допомогою хмарних технологій.

Лабораторні роботи організовані так, щоб студенти мали можливість застосувати теоретичні знання на практиці. Завдання включають обробку даних, розробку моделей, оцінку їхньої ефективності та візуалізацію результатів. У ході виконання робіт студенти навчаються критично аналізувати якість моделей, підвищувати їхню точність і використовувати сучасні платформи для масштабування процесів машинного навчання.

Система оцінювання та вимоги.

Всі виконані практичні завдання повинні бути сформовані у звіт та представлений викладачу. Оцінювання звітів буде формуватись відносно проведеної роботи, а саме:

- До 60 балів - звіт сформований відносно наявного зразку програмної реалізації;
- Від 60 балів до 90 балів – сформуєте короткий аналітичний звіт;
- Вище 90 балів - сформуєте розширений аналітичний звіт.

Структура аналітичного звіту:

Розширений звіт	Короткий звіт
<ul style="list-style-type: none">•Постановка проблеми.•Аналіз останніх досліджень і публікацій.•Мета дослідження•Методи дослідження.•Результати дослідження.•Висновок.•Список використаної літератури.	<ul style="list-style-type: none">•Постановка проблеми.•Мета дослідження•Результати дослідження.•Висновок.

Лабораторна робота №1.

Кластерний аналіз

Теоретична частина

Кластеризація (clustering) — це метод машинного навчання, який передбачає розподіл набору даних на групи, що називаються кластерами. Мета кластеризації полягає в тому, щоб дані всередині одного кластера були схожі між собою, а дані з різних кластерів — відрізнялися. Це допомагає знаходити приховані структури в даних, коли інформація не має явних міток класів, як у задачах класифікації.

Алгоритми кластеризації:

1. Кластеризація K-середніх (K-means)

Цей алгоритм є одним із найпростіших і найпоширеніших методів кластеризації. Його метою є розбиття даних на K кластерів, при цьому кожна точка даних належить до найближчого кластера, визначеного відстанню до його центру.

1. Спочатку визначається кількість кластерів K.
2. Алгоритм вибирає K випадкових точок, які стають початковими центрами кластерів.
3. Для кожної точки даних обчислюється відстань до кожного центру кластера, і вона призначається до найближчого.
4. Після цього обчислюються нові центри кластерів (центроїди) як середнє значення всіх точок, що належать до кластера.
5. Процес повторюється до тих пір, поки центри кластерів і склад кластерів не стабілізуються.
6. Переваги K-середніх:
7. Легко реалізувати.
8. Підходить для великих наборів даних.
9. Недоліки:
10. Чутливість до початкових умов.
11. Працює лише з круглими кластерами (неможливо визначити складніші форми кластерів).

2. Ієрархічна кластеризація

Ієрархічна кластеризація будує ієрархію або дерево кластерів. Існує два основні підходи:

- **Агломеративна кластеризація:** спочатку всі точки даних вважаються окремими кластерами, а потім поступово об'єднуються в більші кластери.
- **Дивізивна кластеризація:** починається з одного великого кластера, який поступово розділяється на менші.

Результати ієрархічної кластеризації часто представлені у вигляді дендрограми, що показує послідовність об'єднання кластерів.

3. DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

DBSCAN базується на щільності точок даних і виявляє кластери будь-якої форми. Він розглядає кластери як області з високою щільністю точок, розділені областями з низькою щільністю. Ключовими параметрами алгоритму є:

- **eps:** радіус навколо кожної точки, який визначає сусідні точки.

- **min_samples:** мінімальна кількість точок, необхідна для того, щоб точка вважалася «ядровою».

Алгоритм добре працює з шумними даними та виявляє кластери неправильної форми.

Переваги DBSCAN:

- Виявляє кластери будь-якої форми.
- Стійкий до шуму в даних.

4. OPTICS (Ordering Points To Identify the Clustering Structure)

OPTICS є узагальненням DBSCAN і дозволяє варіювати параметр `eps` для різних областей даних. Алгоритм будує впорядковану структуру кластерів, що дозволяє виявляти кластери різної щільності. Це розширює можливості DBSCAN і забезпечує більшу гнучкість у виявленні кластерів.

Практична частина

Кластеризація k-середніх

Кластеризація k-середній - один з найпростіших і найбільш часто використовуваних алгоритмів кластеризації. Спочатку вибирається число кластерів `k`. Після вибору значення до алгоритм `k` -середня відбирає точки, які будуть представляти центри кластерів {cluster centers}. Потім для кожної точки даних обчислюється його евклідова відстань до кожного центру кластера. Кожна точка призначається до найближчого центру кластера. Алгоритм обчислює центроїди {centroids} - центри тяжкості кластерів. Кожен центр ваги - це вектор, елементи якого являють собою середні значення характеристик, обчислені по всіх точках кластера. Центр кластера зміщується в його центр ваги. Точки заново призначаються найближчого центру кластера. Етапи зміни центрів кластерів і перепризначення точок ітеративно повторюються до тих пір, поки кордони кластерів і розташування центроїдів не перестануть змінюватися, тобто на кожній ітерації в кожен кластер будуть потрапляти одні і ті ж точки даних.

Розпочнемо з завантаження бібліотеки та даних з пункту 6.1.

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
import pandas as pd

h = pd.read_csv('o.csv', sep=';')
```

Далі проведемо квантельну очистку даних та видаляємо не потрібні нам параметри.

```

Q1 = h.quantile(0.25)
Q3 = h.quantile(0.75)
IQR = Q3 - Q1

t = h[~((h < (Q1 - 1.5 * IQR)) |(h > (Q3 + 1.5 * IQR))).any(axis=1)]
t.shape
X = t.drop('Country', axis = 1)

```

Побудуємо діаграму розсіювання

```

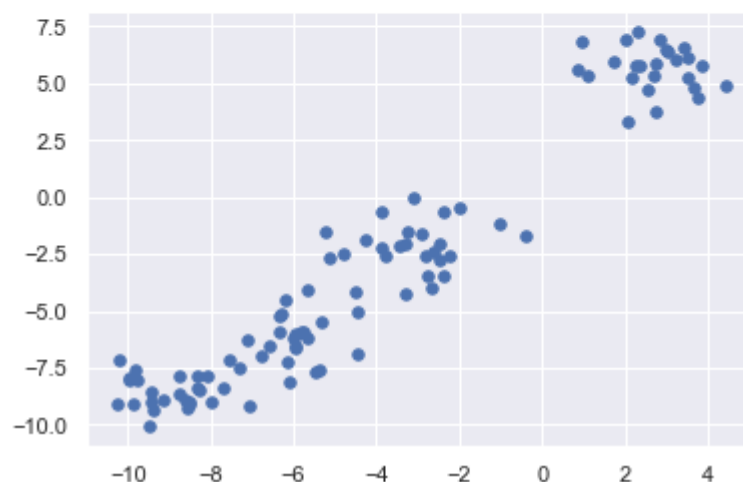
from sklearn.cluster import KMeans
from matplotlib import style
from sklearn.datasets.samples_generator import make_blobs

y=X['P']
X, y = make_blobs(n_samples = 100, centers = 4,
                  cluster_std = 1, n_features = 2)

plt.scatter(X[:, 0], X[:, 1], s = 30, color = 'b')

```

<matplotlib.collections.PathCollection at 0x1dcdec43b48>



Побудуємо кластеризацію за методом k-середніх. Центри кластерів представлені у вигляді більших кругів, в той час як точки даних відображаються у вигляді кіл. Кольори вказують на приналежність до кластеру. Вказано, що шукаємо 2 кластери, тому алгоритм був ініціалізовано за допомогою випадкового вибору двох точок даних в якості центрів кластерів. Потім запускається ітераційний алгоритм. По-перше, кожна точка даних призначається найближчого центру кластера. Потім центри кластерів переносяться в центри важкості кластерів. Потім процес повторюється ще два рази. Після третьої ітерації приналежність точок кластерним центрам не змінюється, тому алгоритм зупиняється.

Отримавши нові точки даних, алгоритм k-середніх буде привласнювати кожному точку даних найближчого центру кластера.

Застосувати алгоритм k -середня, скориставшись бібліотекою scikit-learn, досить просто. Тут застосовуємо його до наших даних, які використовували для побудови попередніх графіків. Ми створюємо екземпляр класу KMeans і задаємо кількість

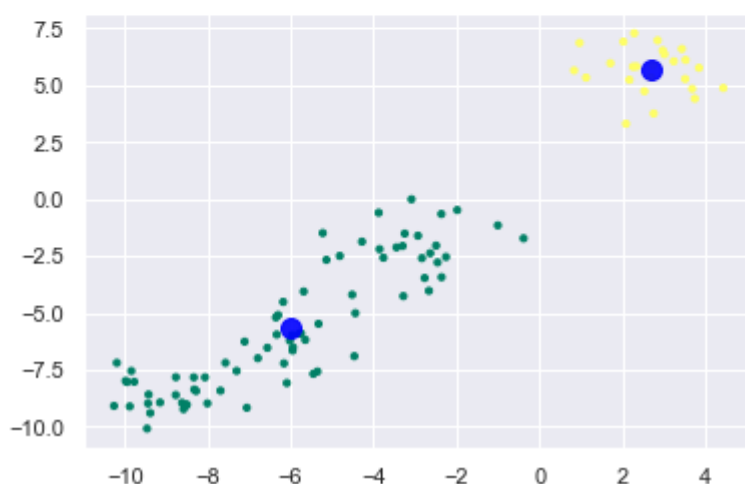
виділених кластеров. Потім ми викликаємо метод `fit` і передаємо йому в якості аргументу дані. Під час роботи алгоритму кожній точці навчальних даних `X` присвоюється мітка кластера. Ви можете знайти ці мітки в атрибуті `kmeans.labels`.

```
kmeans = KMeans(n_clusters = 2)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
```

```
kmeans.cluster_centers_.shape
```

```
(2, 2)
```

```
plt.scatter(X[:, 0], X[:, 1], c = y_kmeans, s = 10, cmap = 'summer')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c = 'blue', s = 100, alpha = 0.9);
plt.show()
```



```
print("Належність до кластерів:\n{}".format(kmeans.labels_))
```

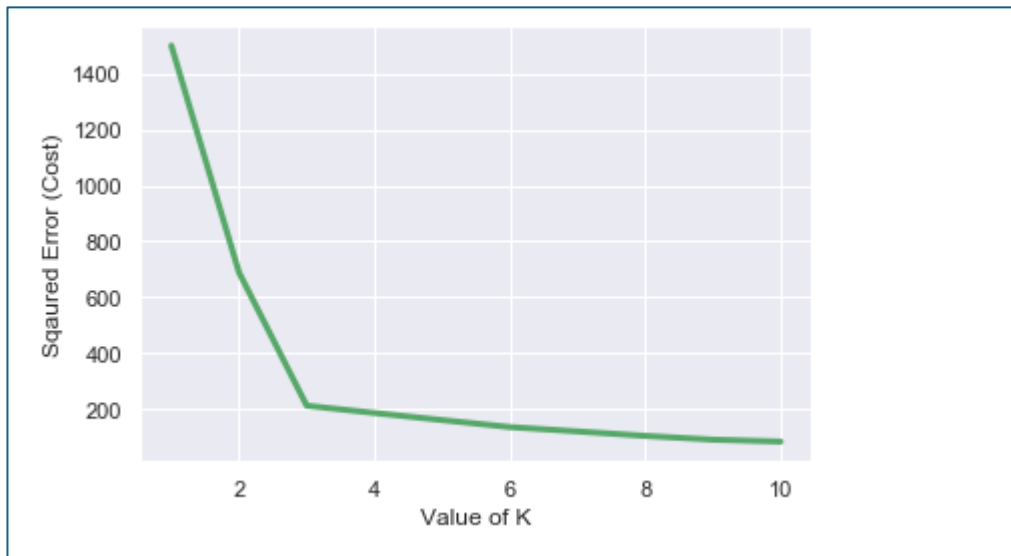
Належність до кластерів:

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

У цьому випадку оптимальне значення для `k` було б 3. (Спостерігається за розсіяними точками).

```
cost = []
for i in range(1, 11):
    KM = KMeans(n_clusters = i, max_iter = 500)
    KM.fit(X)
    # Обчислює помилку у квадраті для кластеризованих точок
    cost.append(KM.inertia_)

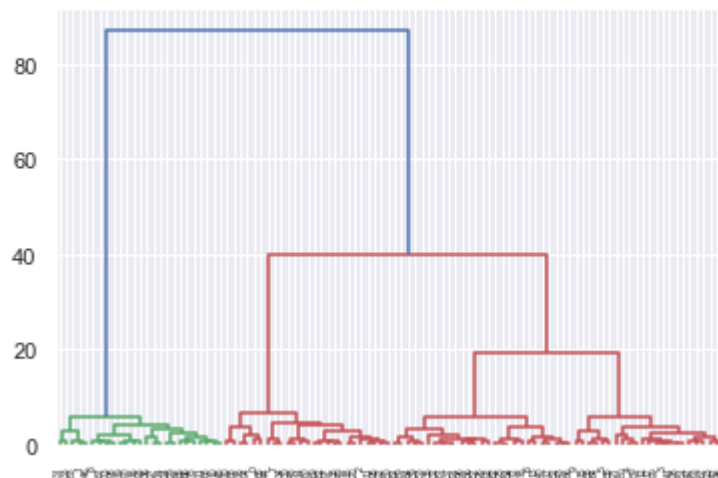
# plot the cost against K values
plt.plot(range(1, 11), cost, color = 'g', linewidth = '3')
plt.xlabel("Value of K")
plt.ylabel("Squared Error (Cost)")
plt.show() # clear the plot
# Точка ліктя є найбільш оптимальним значенням для вибору k
```

Ієрархічна кластеризація

Ієрархічна кластеризація - це загальне сімейство алгоритмів кластеризації, які будують вкладені кластери шляхом об'єднання або розділення їх послідовно. Ієрархія кластерів представляють у вигляді дерева (або дендрограми). Корінь дерева - це унікальний скупчення, яке збирає всі зразки, листя - це кластери лише з одним зразком.

```
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method='ward'))
```



DBSCAN

DBSCAN алгоритм розглядає кластери, як ділянки високої щільності, розділених районах з низькою щільністю. Завдяки цьому досить загальному погляду кластери, знайдені DBSCAN, можуть мати будь-яку форму, на відміну від k-засобів, що передбачає, що кластери мають опуклу форму. Центральним компонентом DBSCAN є концепція основних зразків, які є зразками, що знаходяться в районах високої щільності. Отже, кластер - це сукупність зразків ядра, кожна близька одна до одної (вимірюється деякою мірою відстані) та набір непрофільних зразків, близьких до основного зразка (але самі по собі не є основними зразками). Для алгоритму є два параметри, `min_samples` і `eps` які формально визначають щільність. Вища `min_samples` або нижча `eps` вказують на більш високу щільність, необхідну для формування кластера.

Більш формально ми визначаємо вибірку ядра як вибірку в наборі даних таким чином, що існують min_samples інші вибірки на відстані eps , які визначаються як сусіди основної вибірки. Це говорить про те, що основний зразок знаходиться у щільній області векторного простору. Кластер - це сукупність основних зразків, які можна побудувати шляхом рекурсивного взяття основного зразка, пошуку всіх його сусідів, які є основними зразками, знаходження всіх їх сусідів, які є основними зразками тощо. Кластер також має набір непрофільних зразків, які є зразками, які є сусідами основного зразка в кластері, але самі по собі не є основними вибірками. Інтуїтивно ці зразки знаходяться на межі скупчення.

Будь-який основний зразок є частиною кластеру за визначенням. Будь-який зразок, який не є основним зразком і знаходиться принаймні eps на відстані від будь-якого основного зразка, за алгоритмом вважається вичерпним.

Хоча параметр min_samples першу чергу керує наскільки толерантний алгоритм до шуму (у шумних та великих наборах даних може бути бажаним збільшити цей параметр), цей параметр eps має вирішальне значення для правильного вибору для набору даних та функції відстані, і зазвичай його не можна залишити на значення за замовчуванням. Він контролює місцеве сусідство пунктів. Якщо вибрано занадто мало, більшість даних взагалі не будуть кластеризовані. Якщо вибрано занадто велике, воно змушує об'єднати близькі кластери в один кластер, і в підсумку весь набір даних буде повернутий як єдиний кластер.

На малюнку нижче колір позначає приналежність кластеру, великі кола вказують основні зразки, знайдені алгоритмом. Менші кола - це непрофільні зразки, які все ще є частиною кластера. Більше того, внизу позначені чорними точками шуми.

```
from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
```

```
centers = [[1, 1], [-1, -1], [1, -1]]
X, labels_true = make_blobs(n_samples=750,
                           centers=centers,
                           cluster_std=0.4,
                           random_state=0)

X = StandardScaler().fit_transform(X)
```

```
db = DBSCAN(eps=0.2, min_samples=5).fit(X)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_

# Кількість кластерів у мітках, ігноруючи шум, якщо вони є.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)
```

```

print('Орієнтовна кількість кластерів: %d' % n_clusters_)
print('Орієнтовна кількість точок шуму: %d' % n_noise_)
print("Однорідність: %0.3f" % metrics.homogeneity_score(labels_true, labels))
print("Повнота: %0.3f" % metrics.completeness_score(labels_true, labels))
print("V-ступінь: %0.3f" % metrics.v_measure_score(labels_true, labels))
print("Коригуваний індекс Rand: %0.3f"
      % metrics.adjusted_rand_score(labels_true, labels))
print("Налагоджена взаємна інформація: %0.3f"
      % metrics.adjusted_mutual_info_score(labels_true, labels))
print("Коефіцієнт силуету: %0.3f"
      % metrics.silhouette_score(X, labels))

```

Орієнтовна кількість кластерів: 3
 Орієнтовна кількість точок шуму: 40
 Однорідність: 0.939
 Повнота: 0.827
 V-ступінь: 0.880
 Коригуваний індекс Rand: 0.914
 Налагоджена взаємна інформація: 0.826
 Коефіцієнт силуету: 0.600

```

unique_labels = set(labels)
colors = [plt.cm.Spectral(each)
          for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
        col = [0, 0, 0, 1]

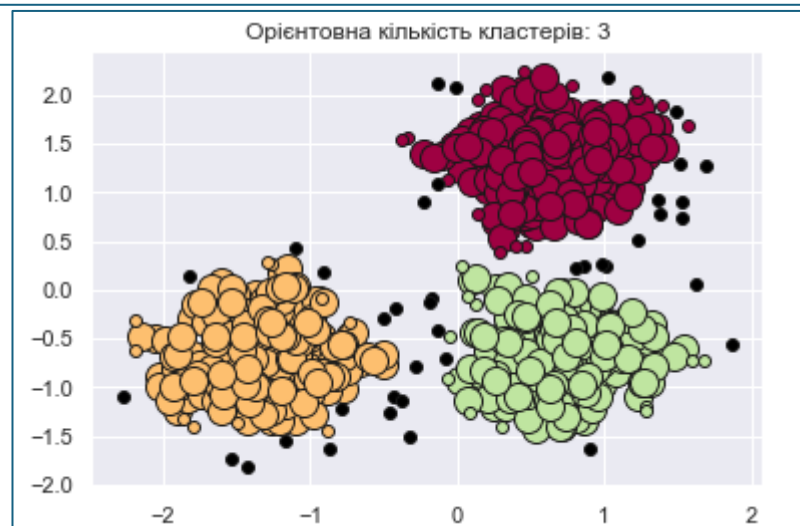
    class_member_mask = (labels == k)

    xy = X[class_member_mask & core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=14)

    xy = X[class_member_mask & ~core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=6)

plt.title('Орієнтовна кількість кластерів: %d' % n_clusters_)
plt.show()

```



OPTICS

В OPTICS алгоритму багато спільного з DBSCAN алгоритмом, і можна розглядати як узагальнення DBSCAN, що спрощує ϵ від одного значення до діапазону значень. Основна відмінність між DBSCAN і OPTICS є те, що алгоритм OPTICS будує граф, який присвоює кожен зразок на `reachability_`, в межах кластера `ordering_`; ці два атрибути присвоюються при встановленні моделі та використовуються для визначення належності кластеру. Якщо OPTICS запускається зі значенням за замовчуванням `inf`, встановленим для `max_eps`, витяг кластера стилю DBSCAN може бути здійснено повторно в лінійний час для будь-якого заданого ϵ значення за допомогою `cluster_optics_dbscan` методу. Налаштування `max_eps` до нижчого значення дасть можливість скоротити виконання, і його можна вважати максимальним радіусом сусідства від кожної точки, щоб знайти інші потенційні точки досяжності.

```
from matplotlib import gridspec
from sklearn.cluster import OPTICS, cluster_optics_dbscan
from sklearn.preprocessing import normalize, StandardScaler
```

```
t = h.iloc[0:93, 1:4]
t
```

	SO	VO	P
0	12291001	3091694	82927922
1	17702028	7408087	82240724

```
# Обробка відсутніх значень, якщо такі є
t.fillna(method = 'ffill', inplace = True)
t.head()
```

```
# Масштабування даних, щоб привести всі атрибути до порівняльного рівня
scaler = StandardScaler()

X_scaled = scaler.fit_transform(t)

# Нормалізація даних, щоб дані
# приблизно відповідає розподілу Гаусса
X_normalized = normalize(X_scaled)

# Перетворення масиву numpy в pandas DataFrame
X_normalized = pd.DataFrame(X_normalized)

# Перейменування стовпців
X_normalized.columns = t.columns

X_normalized.head()
```



```
# Побудова моделі кластеризації OPTICS
optics_model = OPTICS(min_samples = 10, xi = 0.05, min_cluster_size = 0.05)

# Навчання моделі
optics_model.fit(X_normalized)

OPTICS(algorithm='auto', cluster_method='xi', eps=None, leaf_size=30,
        max_eps=inf, metric='minkowski', metric_params=None,
        min_cluster_size=0.05, min_samples=10, n_jobs=None, p=2,
        predecessor_correction=True, xi=0.05)
```

```
# Виготовлення міток згідно з технікою DBSCAN з eps = 0.5
labels1 = cluster_optics_dbscan(reachability = optics_model.reachability_,
                                core_distances = optics_model.core_distances_,
                                ordering = optics_model.ordering_, eps = 0.5)

# Producing the labels according to the DBSCAN technique with eps = 2.0
labels2 = cluster_optics_dbscan(reachability = optics_model.reachability_,
                                core_distances = optics_model.core_distances_,
                                ordering = optics_model.ordering_, eps = 2)

# Створення масивного масиву з числами на рівних проміжках до
# вказаного діапазону
space = np.arange(len(X_normalized))

# Збереження відстані досяжності кожної точки
reachability = optics_model.reachability_[optics_model.ordering_]

# Зберігання міток кластера кожної точки
labels = optics_model.labels_[optics_model.ordering_]

print(labels)

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

```
# Визначення рамок візуалізації
plt.figure(figsize =(10, 7))
G = gridspec.GridSpec(2, 3)
ax1 = plt.subplot(G[0, :])
ax2 = plt.subplot(G[1, 0])
ax3 = plt.subplot(G[1, 1])
ax4 = plt.subplot(G[1, 2])
```

```

# Початок графіку доступності та відстані
colors = ['c.', 'b.', 'r.', 'y.', 'g.']
for Class, colour in zip(range(0, 5), colors):
    Xk = space[labels == Class]
    Rk = reachability[labels == Class]
    ax1.plot(Xk, Rk, colour, alpha = 0.3)
ax1.plot(space[labels == -1], reachability[labels == -1], 'k.', alpha = 0.3)
ax1.plot(space, np.full_like(space, 2., dtype = float), 'k-', alpha = 0.5)
ax1.plot(space, np.full_like(space, 0.5, dtype = float), 'k-.', alpha = 0.5)
ax1.set_ylabel('Початок графіку доступності та відстані')
ax1.set_title('Діапазон доступності')

# Побудова кластеризації OPTICS
colors = ['c.', 'b.', 'r.', 'y.', 'g.']
for Class, colour in zip(range(0, 5), colors):
    Xk = X_normalized[optics_model.labels_ == Class]
    ax2.plot(Xk.iloc[:, 0], Xk.iloc[:, 1], colour, alpha = 0.3)

ax2.plot(X_normalized.iloc[optics_model.labels_ == -1, 0],
        X_normalized.iloc[optics_model.labels_ == -1, 1],
        'k+', alpha = 0.1)
ax2.set_title('OPTICS кластеризація')

```

```

# Plotting the DBSCAN Clustering with eps = 0.5
colors = ['c', 'b', 'r', 'y', 'g', 'greenyellow']
for Class, colour in zip(range(0, 6), colors):
    Xk = X_normalized[labels1 == Class]
    ax3.plot(Xk.iloc[:, 0], Xk.iloc[:, 1], colour, alpha = 0.3, marker = '.')

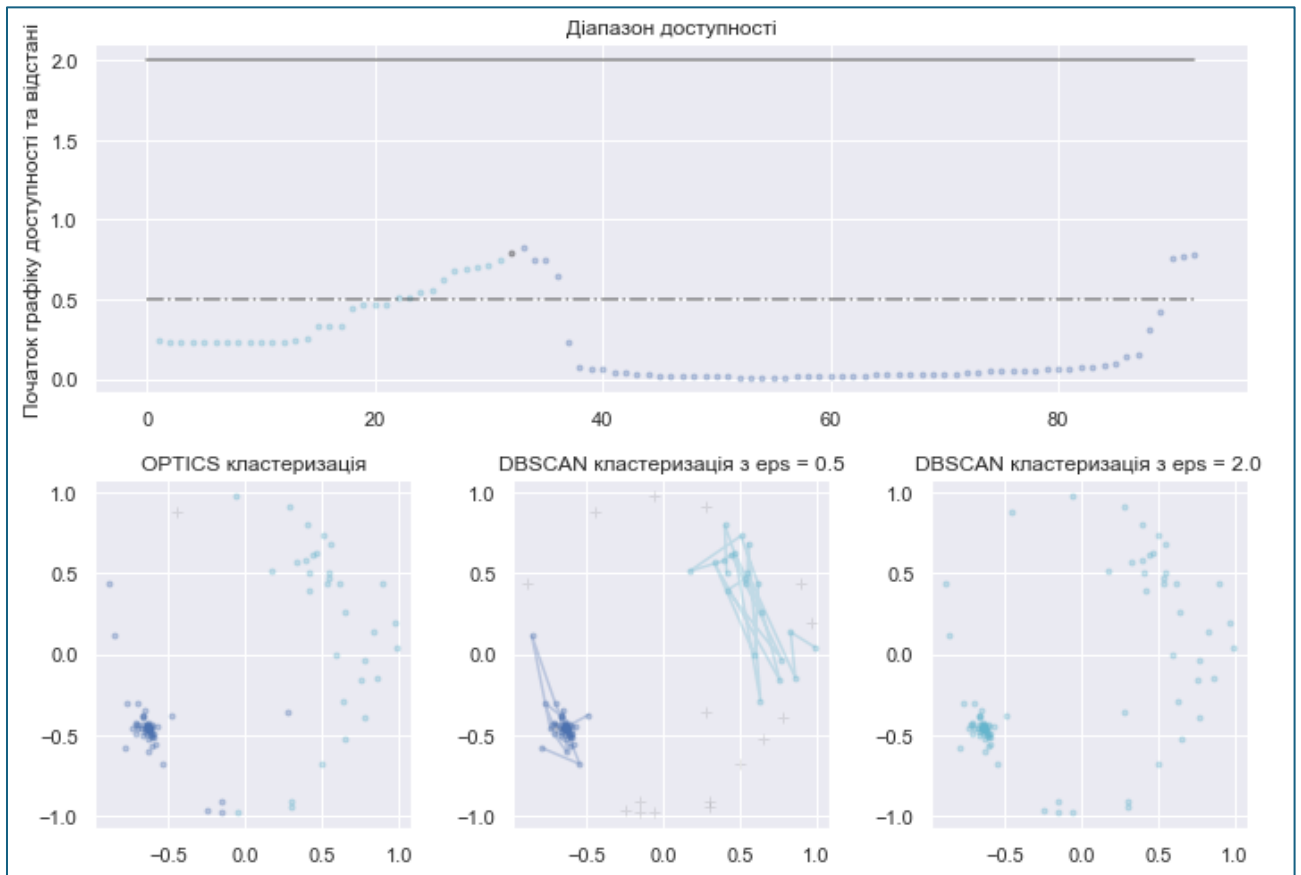
ax3.plot(X_normalized.iloc[labels1 == -1, 0],
        X_normalized.iloc[labels1 == -1, 1],
        'k+', alpha = 0.1)
ax3.set_title('DBSCAN кластеризація з eps = 0.5')

# Побудова кластеризації DBSCAN за допомогою eps = 2.0
colors = ['c.', 'y.', 'm.', 'g.']
for Class, colour in zip(range(0, 4), colors):
    Xk = X_normalized.iloc[labels2 == Class]
    ax4.plot(Xk.iloc[:, 0], Xk.iloc[:, 1], colour, alpha = 0.3)

ax4.plot(X_normalized.iloc[labels2 == -1, 0],
        X_normalized.iloc[labels2 == -1, 1],
        'k+', alpha = 0.1)
ax4.set_title('DBSCAN кластеризація з eps = 2.0')

plt.tight_layout()
plt.show()

```



Як показано на наведеному вище графіку, поєднання відстаней досяжності та набору даних `ordering_` створює графік досяжності, де щільність точок представлена на осі Y, а точки впорядковані таким чином, що точки поблизу є сусідніми. 'Розрізання' ділянки дає результат, схожий на DBSCAN; всі точки над 'розрізом' класифікуються як шум, і кожен раз, коли виникає перерва при читанні зліва направо, означає новий кластер. Вилучення кластера за замовчуванням за допомогою OPTICS розглядає круті схили всередині графіка, щоб знайти кластери, і користувач може визначити, що вважається крутим схилом, використовуючи параметри. Кольори кластерів у площинному просторі відповідають кластерам лінійних сегментів ділянки доступності.

Завдання

1 Завантаження та підготовка даних: Використовуючи дані з Вашого варіанту (Додаток А), виконайте попередню обробку даних, включаючи очищення від пропущених значень, масштабування числових ознак та нормалізацію даних (якщо це необхідно). Переконайтеся, що дані підготовлені для застосування алгоритмів кластеризації.

2 Вибір методу кластеризації:

- Виберіть один або кілька методів кластеризації, таких як **K-середніх (K-means)**, **ієрархічна кластеризація**, **DBSCAN** або **OPTICS**.
- Обґрунтуйте свій вибір, з огляду на характер даних, наприклад, кількість кластерів, розподіл даних або необхідність виявлення шуму.

3 Навчання моделі:

- Застосуйте обраний метод кластеризації до підготовлених даних.
- Якщо обрано K-середніх, знайдіть оптимальну кількість кластерів K за допомогою методу "ліктя" або коефіцієнта силуєту.
- Якщо обрано інші методи, налаштуйте відповідні параметри (наприклад, `eps` та `min_samples` для DBSCAN).

4 Візуалізація результатів:

- Побудуйте графіки для візуалізації результатів кластеризації, зокрема діаграми розсіяння з кольоровими мітками для кожного кластера.
- Для ієрархічної кластеризації побудуйте дендрограму.

5 Аналіз результатів:

- Оцініть якість кластеризації, використовуючи метрики, такі як коефіцієнт силуету або інші показники, які підходять для вибраного методу.
- Поясніть, як отримані кластери відрізняються між собою і що вони означають в контексті даних.

6 Порівняння методів (опціонально): Якщо ви використовували кілька методів кластеризації, порівняйте їх результати та зробіть висновки щодо найбільш придатного для ваших даних.

Оцінювання звіту представлено у Вступі даного збірника.

Лабораторна робота №2.

Регуляризаційні лінійні регресійні моделі

Теоретична частина

Регуляризація — це техніка в машинному навчанні, яка використовується для запобігання переобученню моделі. У лінійних регресійних моделях двома основними методами регуляризації є:

1. **Ridge регресія** (L2 регуляризація): додає штраф за суму квадратів коефіцієнтів до функції втрат.

Формула для Ridge-регресії:

$$L(\beta) = \sum_{i=1}^n (y_i - X_i\beta)^2 + \lambda \sum_{j=1}^p \beta_j^2$$

- де λ — параметр регуляризації, який контролює величину штрафу.

2. **Lasso регресія** (L1 регуляризація): додає штраф за суму абсолютних значень коефіцієнтів до функції втрат.

Формула для Lasso-регресії:

$$L(\beta) = \sum_{i=1}^n (y_i - X_i\beta)^2 + \lambda \sum_{j=1}^p |\beta_j|$$

де λ — параметр регуляризації.

Практична частина

Завантаження та підготовка даних

Для цієї лабораторної роботи використовуються дані про вартість житла у Бостоні (Boston housing dataset). Цей набір даних включає інформацію про різні характеристики житлових районів і відповідні їм ціни на житло.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_boston

# Завантаження даних
boston = load_boston()
```

```
X = pd.DataFrame(boston.data, columns=boston.feature_names)
y = pd.Series(boston.target)

# Розподіл на тренувальні і тестові набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Масштабування даних
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Навчання моделей

Навчаємо моделі Ridge та Lasso регресії на масштабованих даних. Ridge-регресія застосовує L2 регуляризацію, яка зменшує значення всіх коефіцієнтів. Lasso-регресія застосовує L1 регуляризацію, яка може зануляти деякі коефіцієнти.

```
from sklearn.linear_model import Ridge, Lasso

# Навчання Ridge-регресії
ridge = Ridge(alpha=1.0)
ridge.fit(X_train_scaled, y_train)

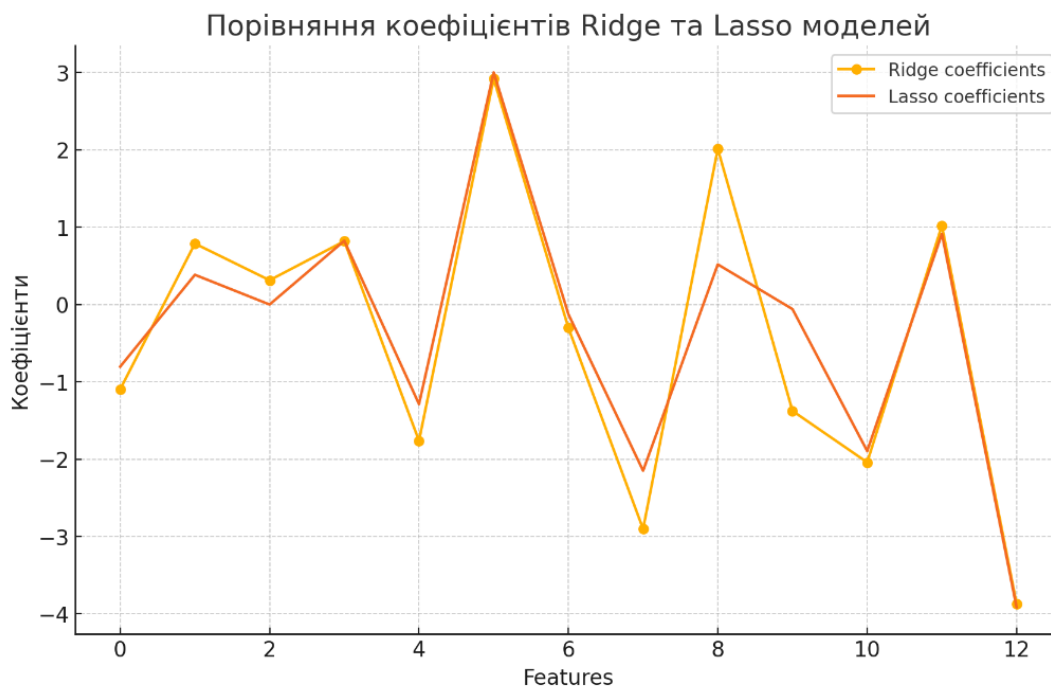
# Навчання Lasso-регресії
lasso = Lasso(alpha=0.1)
lasso.fit(X_train_scaled, y_train)
```

Порівняння коефіцієнтів

Порівнюємо коефіцієнти моделей Ridge та Lasso. Як видно з графіка, Ridge рівномірно зменшує всі коефіцієнти, тоді як Lasso обнуляє деякі з них.

```
import matplotlib.pyplot as plt
# Візуалізація коефіцієнтів
plt.figure(figsize=(10,6))
plt.plot(ridge.coef_, label="Ridge coefficients", marker='o')
plt.plot(lasso.coef_, label="Lasso coefficients", marker='x')
plt.title("Порівняння коефіцієнтів Ridge та Lasso моделей")
plt.xlabel("Features")
plt.ylabel("Коефіцієнти")
plt.legend()
plt.show()
```

Нижче представлений графік, що показує відмінності в коефіцієнтах моделей Ridge та Lasso.



Графік показує порівняння коефіцієнтів моделей **Ridge** та **Lasso** регресії.

- **Ridge регресія** (L2 регуляризація) рівномірно зменшує значення всіх коефіцієнтів. Це дозволяє моделі враховувати всі ознаки, навіть якщо вони менш важливі, зменшуючи ризик переобучення.
- **Lasso регресія** (L1 регуляризація) має здатність зануляти частину коефіцієнтів, що призводить до виключення неважливих ознак. Це робить модель більш інтерпретованою, але може призвести до втрати деякої інформації, особливо якщо певні ознаки є корисними для прогнозування.

Завдяки графіку можна побачити, як різні методи регуляризації впливають на модель і роблять її стійкішою до переобучення.

Завдання

1 Вибір даних для аналізу:

- Використайте дані, що містяться в Додатку А, для побудови прогностичної моделі, яка оцінює залежність цільової змінної (Y) від різних факторів (пояснювальних змінних).
- Визначте, які характеристики даних є найбільш значущими для побудови моделі, та підготуйте їх для подальшого аналізу.

2 Регуляризаційні лінійні регресійні моделі:

- Оберіть два методи регуляризації — Ridge регресію (L2 регуляризація) та Lasso регресію (L1 регуляризація) — для побудови прогностичних моделей.
- Ridge регресія рівномірно зменшує всі коефіцієнти, не зануляючи жоден з них, що дозволяє зберегти всі фактори в моделі.
- Lasso регресія має здатність зануляти частину коефіцієнтів, фактично виключаючи менш значущі змінні з моделі, що дозволяє виявити найважливіші фактори.

3 Порівняння моделей:

- Навчіть обидві моделі (Ridge та Lasso) на підготовлених даних та зробіть прогнози на основі тестових наборів даних.

- Порівняйте результати прогнозів обох моделей та оцініть їхню точність. Використовуйте відповідні метрики оцінки, такі як середньоквадратична помилка або середня абсолютна помилка.
- Проаналізуйте, як регуляризація впливає на прогнозування: чи є суттєві відмінності між точністю моделей Ridge та Lasso, і яка з них є кращою в контексті ваших даних?

4 Аналіз результатів:

- Проведіть порівняльний аналіз коефіцієнтів моделей Ridge та Lasso. Які змінні отримали більшу вагу у кожній моделі? Чи є змінні, які були виключені у моделі Lasso?
- На основі результатів побудуйте висновки про важливість окремих змінних та їхній вплив на цільову змінну (Y). Які фактори мають найбільший вплив на результат?

5 Висновки:

- На основі отриманих результатів, підсумуйте, яка модель виявилась найбільш ефективною для вирішення вашої задачі.
- Обґрунтуйте, чому обрана модель (Ridge чи Lasso) є кращою у контексті даних, з якими ви працюєте. Чи допомогла регуляризація уникнути переобучення та поліпшити точність прогнозів?
- Вкажіть, як регуляризаційні методи допомагають зробити моделі більш стійкими до зміщення і які з них варто використовувати для подальшого аналізу подібних даних.

Оцінювання звіту представлено у Вступі даного збірника.

Лабораторна робота №3.

Методи опорних векторів (SVM)

Теоретична частина:

Методи опорних векторів (Support Vector Machines, SVM) є потужними та універсальними алгоритмами для задач класифікації та регресії в машинному навчанні. Основна ідея SVM полягає в тому, щоб знайти гіперплощину, яка розділяє різні класи в просторі ознак з найбільшою відстанню до найближчих точок (опорних векторів) кожного класу.

Основні поняття

1. **Гіперплощина** — це геометрична площина, яка ділить простір ознак на дві частини. У випадку двовимірного простору, це лінія, в тривимірному — площина, а для вищих розмірностей — гіперплощина.
2. **Опорні вектори** — це точки даних, які знаходяться найближче до гіперплощини і визначають положення цієї гіперплощини. Вони є найбільш важливими точками, оскільки саме вони впливають на положення оптимальної гіперплощини.
3. **Маргінальний простір** — це відстань між гіперплощиною і найближчими до неї точками даних кожного класу. SVM прагне максимізувати цей простір, щоб забезпечити стійкість класифікації.

Основні особливості SVM:

1. **Лінійні SVM:**
 - **Лінійний поділ:** Лінійні SVM використовуються для класифікації даних, які можуть бути розділені лінійно, тобто коли існує пряма або площина, яка може чітко розділити класи.
 - **Оптимальна гіперплощина:** Алгоритм знаходить ту гіперплощину, яка забезпечує максимальне відокремлення класів (максимізує маргінальний простір). Це робить модель стійкою до шуму та більш здатною до узагальнення на нові дані.

Формально, задача SVM зводиться до знаходження коефіцієнтів w та b рівняння гіперплощини $w^T x + b = 0$, які максимізують маргінальний простір і мінімізують помилки класифікації.

2. **Нелінійні SVM:**

- **Класи, що не розділяються лінійно:** Коли дані не можна розділити лінійною гіперплощиною в початковому просторі ознак, використовуються ядрові методи. Ядрові функції дозволяють перетворити дані в простір вищої розмірності, де вони можуть бути лінійно розділені.
- **Ядрові функції (Kernel functions):** Нелінійні SVM використовують різні ядра для перетворення даних у новий простір:
 - **Поліноміальне ядро:** Перетворює простір ознак за допомогою поліноміальних функцій. Наприклад, якщо класи в двовимірному просторі не можна розділити лінійно, вони можуть бути розділені вищим ступенем поліному.
 - **Ядро Радіальної базисної функції (RBF):** Одне з найбільш часто використовуваних ядер. Це нелінійне ядро, яке дозволяє моделі знайти гіперплощину в просторі, де складні структури можуть бути розділені.
 - **Сигмоїдальне ядро:** В основному використовується для моделей, схожих на нейронні мережі.

Ядрові функції дозволяють працювати з високорозмірними даними без явного перетворення всіх ознак у вищий простір (так званий "трюк з ядром").

Параметри SVM:

1. **C (вартість помилок):**

- Це параметр регулювання, який контролює баланс між максимізацією маргінального простору та мінімізацією помилок класифікації.
- Якщо C велике, модель буде прагнути мінімізувати кількість помилок класифікації, навіть якщо це призведе до меншого маргінального простору. Це може призвести до переобучення.
- Якщо C маленьке, модель дозволяє більше помилок, але намагається максимізувати маргінальний простір, що робить модель стійкішою до переобучення.

2. **γ (для нелінійних SVM з RBF ядром):**

- Параметр γ визначає, як далеко впливає кожна окрема точка на класифікаційну гіперплощину. Високе значення γ означає, що кожна точка має велике локальне вплив на розділення класів, тоді як низьке значення γ вказує на ширший радіус впливу точок на класифікацію.

Практична частина:

У цій лабораторній роботі ми розглянемо методи опорних векторів (SVM) та їх використання для класифікації даних на прикладі набору даних Iris.

Завантаження та підготовка даних

Ми завантажуюємо набір даних Iris, поділяємо його на тренувальний і тестовий набори, а також масштабуємо дані для кращої роботи SVM.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import datasets

# Завантаження набору даних "Iris"
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Розділяємо дані на тренувальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=42)

# Масштабування даних
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Дані з набору Iris успішно завантажені. Дані розподілені на тренувальний (70%) та тестовий (30%) набори. Масштабування даних виконано для кращої роботи SVM.

Навчання моделі SVM

Моделі SVM з лінійним ядром була навчена на масштабованих тренувальних даних, після чого було виконано прогнозування на тестовому наборі.

```
from sklearn.svm import SVC

# Створення та навчання моделі SVM з лінійним ядром
svm_model = SVC(kernel='linear')
svm_model.fit(X_train_scaled, y_train)

# Прогнозування на тестовому наборі
y_pred = svm_model.predict(X_test_scaled)
```

Моделі SVM з лінійним ядром успішно навчена на тренувальних даних. Виконано прогнозування на тестовому наборі.

Оцінка якості моделі

Оцінка точності моделі та виведення класифікаційного звіту.

```
from sklearn.metrics import classification_report, accuracy_score

# Виведення звіту про класифікацію
print(classification_report(y_test, y_pred))

# Оцінка точності
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	19
1	1.00	0.96	0.98	23
2	0.95	1.00	0.98	13
accuracy			0.99	55
macro avg	0.98	0.99	0.99	55
weighted avg	0.99	0.99	0.99	55
Accuracy: 98.18%				

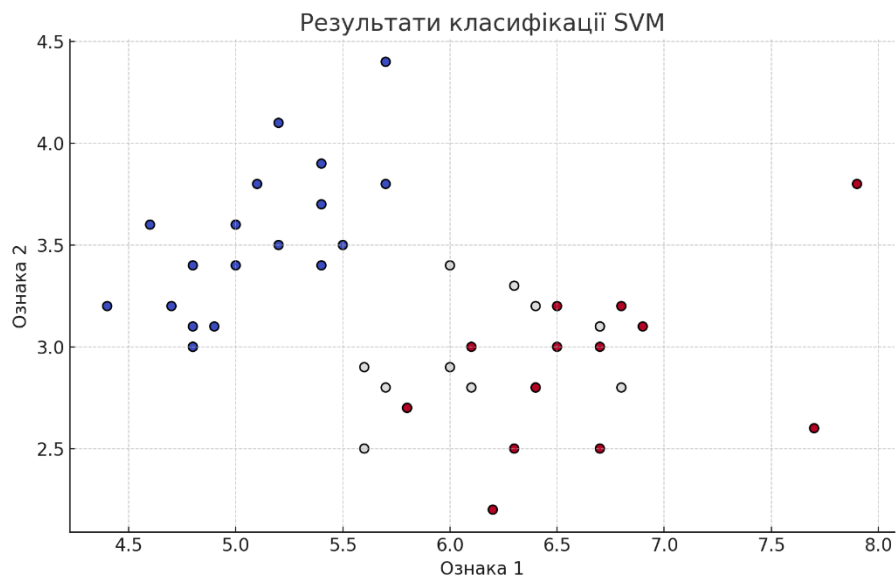
Моделі SVM з лінійним ядром досягла високої точності (98.18%). Усі класи були класифіковані з дуже високими показниками точності та повноти.

Візуалізація гіперплощини (для 2D-випадку)

Побудуємо графік для візуалізації результатів класифікації тестових даних за допомогою перших двох ознак.

```
import matplotlib.pyplot as plt

# Побудуємо графік для перших двох ознак
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred, cmap='coolwarm', marker='o',
            edgcolor='k')
plt.title("Результати класифікації SVM")
plt.xlabel("Ознака 1")
plt.ylabel("Ознака 2")
plt.show()
```



На графіку показані результати класифікації тестових даних за допомогою моделі SVM з лінійним ядром. Кожна точка на графіку представляє один об'єкт із тестового набору, а кольори позначають різні класи.

- **Різнокольорові точки:** представляють різні класи набору даних, що відображає здатність моделі розділяти дані на класи.
- **Гіперплощина:** хоча вона не відображена на графіку безпосередньо, модель SVM знаходить таку гіперплощину, що розділяє ці класи з максимальною відстанню до найближчих точок обох класів.
- **Правильна класифікація:** більшість точок знаходяться в окремих класах, що вказує на високу точність класифікації.

Модель змогла добре класифікувати об'єкти, демонструючи чітке розділення між класами за допомогою SVM, що підтверджується високою точністю (98.18%).

Завдання

1. Завантаження та підготовка даних:

- Використайте дані, наведені у Вашому варіанті (Додаток А), для аналізу за допомогою методу опорних векторів (SVM).

- Виконайте попередню обробку даних, включаючи очищення від пропущених значень та масштабування числових ознак для коректної роботи алгоритму SVM.
 - Підготуйте дані для навчання та тестування моделі, розділивши їх на тренувальний та тестовий набори.
2. **Вибір параметрів моделі SVM:**
- Оберіть відповідне ядро для моделі SVM. Якщо дані можуть бути лінійно розділені, використовуйте **лінійне ядро**. Якщо ж дані мають більш складну структуру, оберіть **нелінійне ядро**, наприклад, поліноміальне або RBF (Радіальної базисної функції).
 - Налаштуйте параметри моделі:
 - **Параметр C** контролює баланс між точністю класифікації і шириною маргінального простору. Визначте оптимальне значення параметра C, враховуючи можливість переобучення або недообучення.
 - **Параметр γ** (для нелінійних ядер) визначає радіус впливу точок на класифікацію. Експериментуйте з різними значеннями γ , щоб знайти оптимальне.
3. **Навчання та прогнозування:**
- Використайте обрану модель SVM для навчання на тренувальних даних.
 - Після навчання моделі зробіть прогноз на тестових даних і порівняйте результати з відомими відповідями.
 - Якщо це можливо, порівняйте результати з іншими моделями класифікації або з іншим типом ядра SVM.
4. **Оцінка якості моделі:**
- Оцініть якість класифікації за допомогою таких метрик, як **точність (accuracy)**, **повнота (recall)**, **точність (precision)** та **F1-оцінка**.
 - Порівняйте отримані результати з іншими моделями або з результатами при використанні інших ядер (лінійне, поліноміальне, RBF тощо).
5. **Візуалізація результатів:**
- Якщо кількість ознак дозволяє, візуалізуйте результати класифікації. Це можна зробити у випадку двовимірних або тривимірних даних. Покажіть, як модель розподілила дані між класами.
 - Для лінійних моделей побудуйте графік гіперплощини, яка розділяє класи, та опорних векторів.
6. **Аналіз параметрів та їхнього впливу:**
- Проаналізуйте, як зміна параметрів моделі (C і γ) впливає на результати класифікації. Оцініть, як ці параметри змінюють здатність моделі розрізняти класи.
 - Вкажіть, чи було досягнуто покращення точності моделі при зміні параметрів, та чи вдалося уникнути переобучення.

Оцінювання звіту представлено у Вступі даного збірника.

Лабораторна робота №4.

Дерева рішень

Теоретична частина:

Дерева рішень (Decision Trees) — це один із найпопулярніших алгоритмів машинного навчання для задач класифікації та регресії. Основною ідеєю дерева рішень є ієрархічна структура прийняття рішень, в якій дані діляться на підгрупи на основі певних умов. Кожна умова в дереві базується на ознаці, яка ділить набір даних на два або більше підмножин. Алгоритм продовжує цей процес ітеративно, створюючи дерево, яке може приймати рішення або робити прогнози на основі вхідних даних.

Основні компоненти дерева рішень:

- **Вузли (Nodes)** — це точки розгалуження дерева, які представляють умови або питання.
- **Гілки (Branches)** — це шляхи, що з'єднують вузли і вказують на можливі варіанти рішень.
- **Листя (Leaves)** — це кінцеві вузли дерева, які містять прогноз або категорію.

Алгоритм побудови дерева рішень

1. **Постановка задачі та вибір даних:** Перш за все, потрібно сформулювати питання, на яке повинна відповісти модель, і визначити дані, які будуть використані для навчання. У цьому випадку ми передбачаємо кількість туристів на основі різних факторів, таких як заробітна плата в туристичній галузі, інвестиції та кількість туристичних об'єктів.
2. **Попередня обробка даних:**
 - Завантажте дані у форматі, який підтримується Python (наприклад, CSV).
 - Виконайте перевірку на наявність пропущених або аномальних значень і виправте їх.
 - Визначте цільову змінну (Y — потік туристів) і пояснювальні змінні (фактори, що впливають на кількість туристів).
3. **Поділ даних на тренувальний і тестовий набори:**
 - Для навчання моделі використовуються тренувальні дані, а тестовий набір даних використовується для оцінки точності моделі. Зазвичай, дані діляться на 70% для тренування та 30% для тестування.
 - Встановіть випадкове значення (наприклад, 42), щоб забезпечити відтворюваність результатів.
4. **Створення та навчання моделі дерева рішень:**
 - Використовуючи бібліотеку **Scikit-learn**, імпортуйте клас **DecisionTreeClassifier** для задач класифікації або **DecisionTreeRegressor** для задач регресії.
 - Навчіть модель на тренувальному наборі даних, передавши їй цільові змінні та дані для навчання.
5. **Отримання прогнозів:** Після навчання моделі зробіть прогнози на тестових даних. Модель, базуючись на структурі дерева, прийматиме рішення для кожної нової точки даних.

6. **Оцінка точності моделі:** Використовуйте метрики точності, такі як **середня абсолютна похибка (MAE)** або **середньоквадратична похибка (RMSE)** для регресійних задач, або **точність класифікації** для класифікаційних задач, щоб оцінити, наскільки точно модель передбачає результати.
7. **Поліпшення моделі** (якщо потрібно): Якщо точність моделі недостатня, можливо, потрібно змінити глибину дерева, підлаштувати інші параметри або спробувати інший алгоритм.
8. **Візуалізація дерева:** Однією з переваг дерева рішень є можливість його візуалізації. За допомогою **Scikit-learn** можна відобразити структуру дерева та проаналізувати, як модель приймає рішення на різних етапах.

Приклад алгоритму дерева рішень:

1. Модель отримує дані про кількість суб'єктів туристичної діяльності (P), капітальні інвестиції (I), витрати на туризм (V) тощо.
2. Модель аналізує кожен фактор і приймає рішення щодо того, чи впливає цей фактор на прогноз потоку туристів (Y).
3. Після обробки всіх вузлів дерево надає прогноз.

Практична частина:

Алгоритм побудови дерева

Перш ніж зануритися в програмування, потрібна інструкція щоб розуміти, куди рухатись. Перелічені нижче кроки описують основу процесу машинного навчання в тому випадку, коли є завдання і модель:

1. Поставити питання і визначити необхідні дані
2. Отримати дані в доступному форматі
3. Довизначити/виправити відсутні точки даних / аномальні значення при необхідності
4. Підготувати дані для моделі машинного навчання
5. Встановити базову модель
6. Навчити модель на тренувальних даних
7. Отримати прогнози по тренувальних даними
8. Порівняйте прогнози з відомими цільовими наборами тестів і розрахувати коефіцієнт продуктивності
9. Якщо продуктивність не задовільна, відрегулювати модель, використовувати більше даних або спробувати використовувати іншу техніку моделювання
10. Інтерпретувати модель і представити результати в візуальній та чисельній формі

Дані

Для аналізу використаємо дані туристичної діяльності України та спрогнозуємо кількість туристів (Y).

```
In [2]: tur = pd.read_csv('TURUA3.csv', sep=',')
```

```
In [3]: tur
```

```
Out[3]:
```

	Year	Y	YP	S	V	R	P	C	I	K
0	2012	286797	296797	75994	5375459	6041	5346	3150653	273256	846
1	2013	342817	362817	272050	6544143	6411	5711	3189558	249873	862
2	2014	205988	398837	224246	4771043	4572	3885	3354027	219420	761

Позначення	Параметр	Одиниці вимірювання
Y	потік туристів	тис.осіб
Y_P	прогнозований потік туристів, лінійним методом	тис.осіб
S	середня заробітна плата на особу в туристичній галузі	грн
V	витрати на туризм	грн
P	кількість суб'єктів туристичної діяльності	од
R	кількість колективних засобів розміщування	од
K	кількість рекреацій	од
C	випуск в основних цінах та випуск за видами економічної діяльності	тис.грн
I	капітальні інвестиції за регіонами	тис.грн

Перетворення даних в масиви

Тепер потрібно розділити дані на функції і цілі. Мета, також звана міткою, є прогнозоване значення, в нашому випадку – потік туристів, а функції - це все стовпчики, які використовуються моделлю для прогнозування. Ми також перетворимо числові дейтафрейми Pandas в масиви [Numpy](#), згідно роботі алгоритму.

```
In [4]: # Використовуйте numpy для перетворення в масиви
import numpy as np
# Мітки - це значення, які ми хочемо передбачити
labels = np.array(tur['Y'])
# Видаляємо мітку з особливостей, вісь 1 відноситься до стовпців
tur = tur.drop('Y', axis = 1)
# Saving feature names for later use
tur_list = list(tur.columns)
# Convert to numpy array
tur = np.array(tur)
```

Формування набору даних для навчання і перевірки

Тепер приступимо до завершального етапу підготовки вихідних даних: поділ даних на навчальні та тестові набори. Під час навчання ми дозволяємо моделі «бачити» відповіді, в нашому випадку – потік туристів, щоб вона могла дізнатися, як передбачити потік туристів на основі функцій. Припускаємо, що є деякий взаємозв'язок між усіма функціями і цільовим значенням, і завдання моделі - вивчити ці співвідношення в процесі навчання. Потім, коли настає момент оцінки моделі, заставимо її зробити прогноз на тестовому наборі даних, в цьому випадку надано доступ тільки до функцій (але не до відповідей!). Оскільки у нас є відповіді для набору тестів, ми можемо порівняти отримані прогнози з істинними значеннями, щоб оцінити точність моделі. Як правило, при навчанні моделі ми довільно розбиваємо дані на навчальні та тестові набори, щоб отримати уявлення всіх точок даних (наприклад, якщо ми навчали модель на даних перших дев'яти місяців року, а потім використовували дані останніх трьох місяців для прогнозування, то наш алгоритм не зможе добре спрацювати, тому що він не навчався на даних останніх трьох місяців). Встановимо випадкове значення рівним 42, що означає, що результати будуть однаковими при кожному запуску розбиття для отримання відтворюваних результатів.

Нижченаведений код розбиває набір даних в ще один рядок:

```
In [5]: # Використання Scikit-навчання для поділу даних на навчальні та тестові набори
from sklearn.model_selection import train_test_split
# Розділіть дані на навчальні та тестові набори
train_tur, test_tur, train_labels, test_labels = train_test_split(tur, labels, test_size = 0.25, random_state = 42)
```

Подивитися на форму всіх даних, щоб переконатися, що все зроблено правильно. Кількість функцій навчання буде відповідати кількості стовпців і кількості рядків для відповідних функцій навчання, тестування і міток:

```
In [6]: print('Training Features Shape:', train_tur.shape)
        print('Training Labels Shape:', train_labels.shape)
        print('Testing Features Shape:', test_tur.shape)
        print('Testing Labels Shape:', test_labels.shape)
```

```
Training Features Shape: (6, 8)
Training Labels Shape: (6,)
Testing Features Shape: (2, 8)
Testing Labels Shape: (2,)
```

Організація основних даних

Перш ніж ми зможемо отримати і оцінити прогнози, необхідно встановити базовий рівень, розумну міру, яку ми сподіваємося перевершити за допомогою нашої моделі. Якщо модель не зможе перевершити базовий рівень, будемо розглядати це як невдалу спробу, тоді доведеться спробувати іншу модель або ж визнати, що машинне навчання не підходить для вирішення нашої задачі. Базовий прогноз для нашого випадку може бути прогнозом значення туристичного потоку (YP). Іншими словами, наша базова лінія - це помилка, яку ми отримали б, якби просто прогнозували туристичний потік.

```
In [7]: # Основні прогнози - це середні показники
        baseline_preds = test_tur[:, tur_list.index('YP')]
        # Базові помилки та відображати середню базову помилку
        baseline_errors = abs(baseline_preds - test_labels)
        print('Середня базова помилка: ', round(np.mean(baseline_errors), 2), 'осіб')
```

```
Середня базова помилка: 146012.5 осіб
```

У тепер у нас є мета. Якщо не зможемо перевершити середню помилку в 146012.5 осіб, то доведеться переглянути підхід в цілому.

Модельне навчання

Після проведення роботи з підготовки даних, створення і навчання моделі досить просто здійснити з [допомогою Scikit-learn](#). Імпортуємо модель випадкового регресійного лісу з scikit-learn, створюємо екземпляр моделі і тренуємо (ім'я scikit-learn для навчання) модель на наборі даних для навчання. (Знову встановлюємо випадковий стан для відтворюваних результатів). Процес описується трьома рядками в scikit-learn!

```
In [8]: # Імпортуйте модель, яку ми використовуємо
        from sklearn.ensemble import RandomForestRegressor
        # Моментальна модель з 1000 дерев рішень
        rf = RandomForestRegressor(n_estimators = 1000, random_state = 100)
        # Навчіть модель на даних про навчання
        rf.fit(train_tur, train_labels);
```

Отримання прогнозів на тестовому наборі

Поки наша модель навчилася пошуку співвідношень між функціями і цілями. Наступний крок - з'ясувати, наскільки точна модель. Для цього зробимо прогнози по тестовим функцій (моделі заборонено бачити тестові відповіді). Потім можна порівняти прогнози з відомими відповідями. При виконанні регресії ми повинні переконалися, що використовуємо абсолютну помилку, тому що очікуємо, що деякі наші відповіді будуть низькими, а деякі - високими. Нас цікавить, як далекий наш середній прогноз від фактичного значення, тому беремо абсолютне значення (як це робили і при складанні базової лінії).

Виконання прогнозів з використанням моделі - це ще одна однорядкова команда в Scikit-learn.

```
In [9]: # Використовуйте метод прогнозування лісу на даних тесту
predictions = rf.predict(test_tur)
# Обчисліть абсолютні помилки
errors = abs(predictions - test_labels)
# Обчислити середню абсолютну помилку (mae)
print('Середня абсолютна помилка:', round(np.mean(errors), 2), 'грн.')
```

Середня абсолютна помилка: 35240.66 грн.

Наша середня оцінка відхиляється на 35240.66 грн. Це набагато менше в порівнянні з вихідним рівнем. Хоча це може здатися незначним, але це на 24% краще базового рівня, який, в залежності від області і проблеми, може являти собою, наприклад, для великої компанії мільйони доларів.

Визначення показників ефективності

Щоб уявити наші прогнози в перспективі, можемо обчислити точність, використовуючи середню відносну похибку, виражену у відсотках, віднімається з 100%.

```
In [10]: # Обчислити середню абсолютну процентну помилку (MAPE)
mape = 100 * (errors / test_labels)
# Обчислити та відобразити точність
accuracy = 100 - np.mean(mape)
print('Точність:', round(accuracy, 2), '%.')
```

Точність: 89.16 %.

Наша модель навчилася прогнозувати туристичний потік в Україні на наступний рік з точністю в 89%.

Поліпшення моделі (при необхідності)

Сьогодні процес машинного навчання може здійснюватися за допомогою підстроювання гіперпараметрів. По-суті, це означає «підстроювання параметрів для підвищення продуктивності» (налаштування називаються [гіперпараметрами](#), щоб відрізнити їх від параметрів моделі, отриманих під час навчання). Найпоширеніший спосіб виконання такого налаштування - створити сукупність моделей з різними настройками, оцінити їх на одному наборі даних і подивитися, яка з них підходить найкраще. Звичайно, робити це вручну - досить утомливо, тому краще скористатися [автоматизованими методами](#) з Skicit-learn. Для настройки гіперпараметров (Hyperparameter) часто потрібно швидше [проявити винахідливість](#), а не спиратися на теоретичні знання. Точність 89% задовільна для розглянутої задачі, але врахуйте, що зазвичай перша побудована модель майже ніколи не буває найефективнішою моделлю.

Інтерпретація моделі і звіт про результати

Зараз вже відомо, що модель якісна, але все ж вона в значній мірі - чорний ящик. Введемо деякі масиви NumPy для навчання, просимо зробити прогнози, оцінити прогнози і переконаватися, що вони розумні. Виникає питання: як ця модель отримує результат? Є два підходи, що дозволяють заглянути в Random forest: по-перше, можемо подивитися на одне дерево в лісі, а по-друге, можемо подивитися на особливості, пов'язані з нашими пояснювальними змінними.

Візуалізація окремого дерева рішень

Одна з самих класних реалізацій Random forest в Skicit-learn із загального числа можливих реалізацій - можна вивчити будь-які дерева в лісі. Ми виберемо одне дерево і збережемо його як образ.

Наступний код бере одне дерево з лісу і зберігає його як зображення.

```
In [11]: conda install pydot graphviz

Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working... done

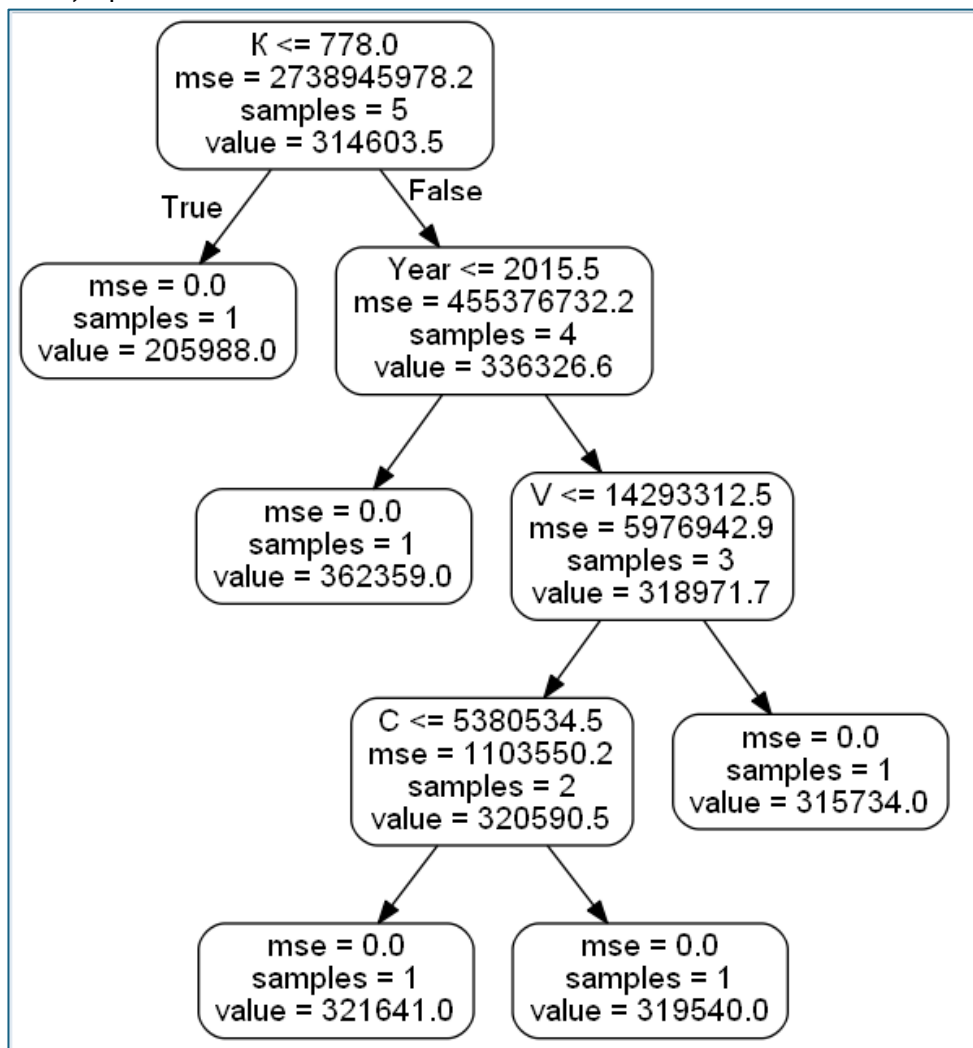
# All requested packages already installed.

Note: you may need to restart the kernel to use updated packages.

In [12]: # Імпорт інструментів, необхідних для візуалізації
from sklearn.tree import export_graphviz
import pydot

In [13]: # Витягни одне дерево з лісу
tree = rf.estimators_[5]
# Експорт зображення в крапковий файл
export_graphviz(tree, out_file = 'tree1.dot', feature_names = tur_list, rounded = True, precision = 1)
# Використовуйте крапковий файл для створення графіка
(graph, ) = pydot.graph_from_dot_file('tree1.dot')
# Записати графік у png-файл
graph.write_png('tree1.png')
```

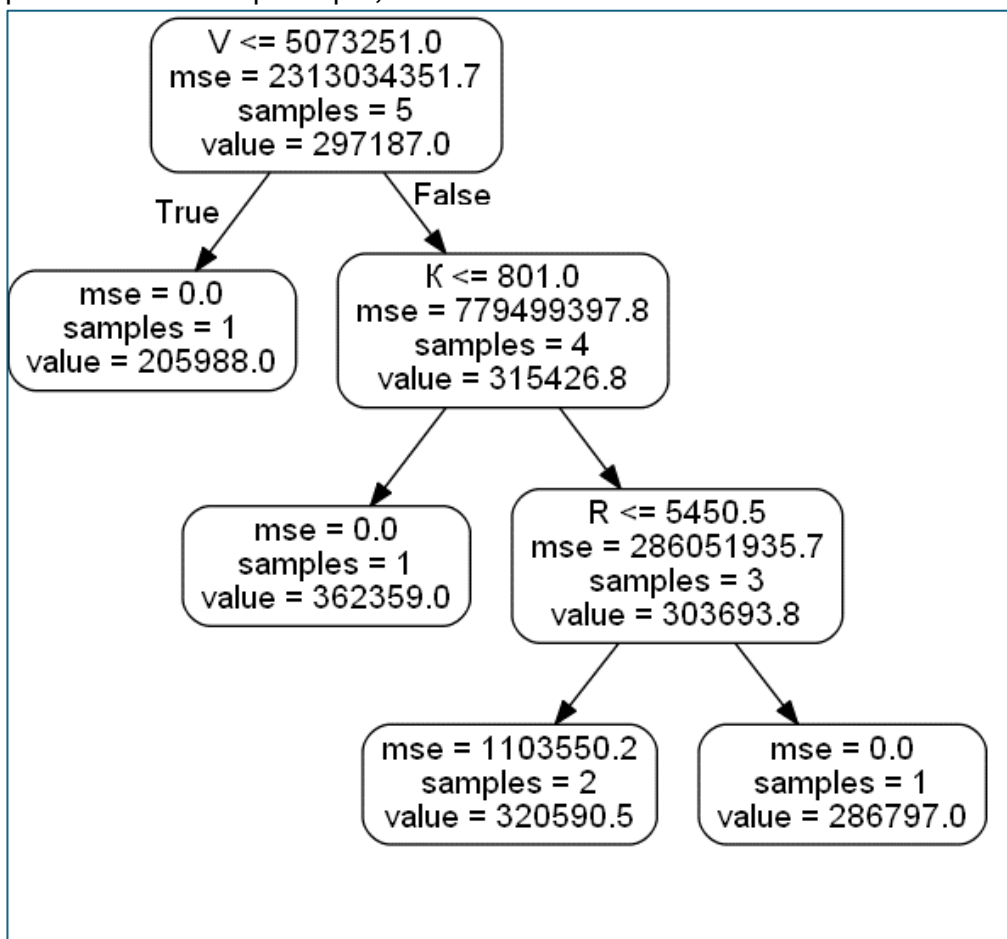
Подивимось, що вийшо:



Щоб спростити завдання, обмежимо глибину дерев в лісі, для створення читаного зображення.

```
In [14]: # Обмежте глибину дерева до 3-х рівнів
rf_small = RandomForestRegressor(n_estimators=10, max_depth = 3)
rf_small.fit(train_tur, train_labels)
# Витягніть маленьке дерево
tree_small = rf_small.estimators_[5]
# Збережіть дерево у форматі PNG
export_graphviz(tree_small, out_file = 'small_tree1.dot', feature_names = tur_list, rounded = True, precision = 1)
(graph, ) = pydot.graph_from_dot_file('small_tree1.dot')
graph.write_png('small_tree1.png');
```

Ось дерево зменшених розмірів, анотоване мітками:



Грунтуючись виключно на цьому дереві, можемо зробити прогноз для будь-якої нової точки даних.

Завдання

1 Підготовка даних:

- Використовуйте дані з вашого варіанту (Додаток А) для створення прогностичної моделі.
- Виділіть цільову змінну (Y — потік туристів) та всі інші пояснювальні змінні, які будуть використовуватися для побудови моделі дерева рішень.

2 Побудова моделі дерева рішень:

- Використовуючи алгоритм дерева рішень, спрогнозуйте кількість туристів на основі наданих даних.
- Навчіть модель на тренувальному наборі даних і зробіть прогнози на тестових даних.
- Визначте точність моделі, використовуючи відповідні метрики для оцінки якості прогнозів.

3 Аналіз результатів:

- Побудуйте графік дерева рішень для візуалізації моделі.
- Проаналізуйте вплив різних змінних на прогноз та зробіть висновки щодо їхнього значення.
- Оцініть, наскільки модель відповідає реальним даним.

4 Оптимізація та покращення (якщо потрібно):

- Використовуйте гіперпараметри для покращення точності моделі.
- Спробуйте інші методи моделювання, якщо точність моделі є незадовільною.

Оцінювання звіту представлено у Вступі даного збірника.

Лабораторна робота №5.

Ансамблеве навчання

Теоретична частина

Ансамблеве навчання — це підхід, що використовує кілька моделей для покращення точності прогнозу. Ідея полягає в тому, що поєднання кількох моделей може призвести до зменшення загальної похибки та підвищення точності прогнозу в порівнянні з використанням однієї моделі.

Основні види ансамблевих методів:

1. **Bagging (Bootstrap Aggregation):**

- Цей метод базується на побудові кількох незалежних моделей на різних підмножинах даних, обраних за допомогою бутстрепінгу.
- **Принцип:** Створюються кілька тренувальних наборів даних, використовуючи вибірку з поверненням, після чого моделі тренуються на кожному з них, а результати усереднюються (для регресії) або голосуються (для класифікації).
- **Приклад:** Один із найвідоміших прикладів — Random Forest, де комбінуються дерева рішень.

2. **Boosting:**

- Метод, який послідовно будує моделі, де кожна наступна модель намагається виправити помилки попередньої.
- **Принцип:** Кожна нова модель отримує більше ваг для прикладів, які були неправильно передбачені попередніми моделями, що дозволяє "зосередитись" на складних випадках.
- **Приклади:** AdaBoost, Gradient Boosting, XGBoost.

3. **Random Forest:**

- Це різновид методу Bagging, який використовує ансамбль дерев рішень для створення моделі. Кожне дерево тренується на випадковій підмножині ознак та прикладів.
- **Принцип:** Random Forest зменшує кореляцію між деревами, що підвищує стійкість до переобучення та підвищує точність моделі.

Практична частина:

У даній частині лабораторної роботи ми розглянемо три основні методи ансамблевого навчання — **Random Forest**, **AdaBoost**, та **Gradient Boosting** — для задачі класифікації на основі набору даних про рак молочної залози (**Breast Cancer dataset**). Ми навчимо моделі, оцінимо їхню точність та порівняємо результати.

Завантаження бібліотек та даних

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_breast_cancer

# Завантаження набору даних
data = load_breast_cancer()
X = data.data
y = data.target

# Розділяємо дані на тренувальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Масштабування даних
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Отже, набір даних Breast Cancer завантажено, містить 569 прикладів з 30 характеристиками. Дані розподілені на тренувальні (70%) та тестові (30%) набори. Виконано масштабування даних для нормалізації ознак.

Навчання моделей

Навчимо моделі **Random Forest**, **AdaBoost**, і **Gradient Boosting** на тренувальних даних і проведемо прогнозування на тестових даних.

```
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,
GradientBoostingClassifier

# Random Forest
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)
y_pred_rf = rf_model.predict(X_test_scaled)

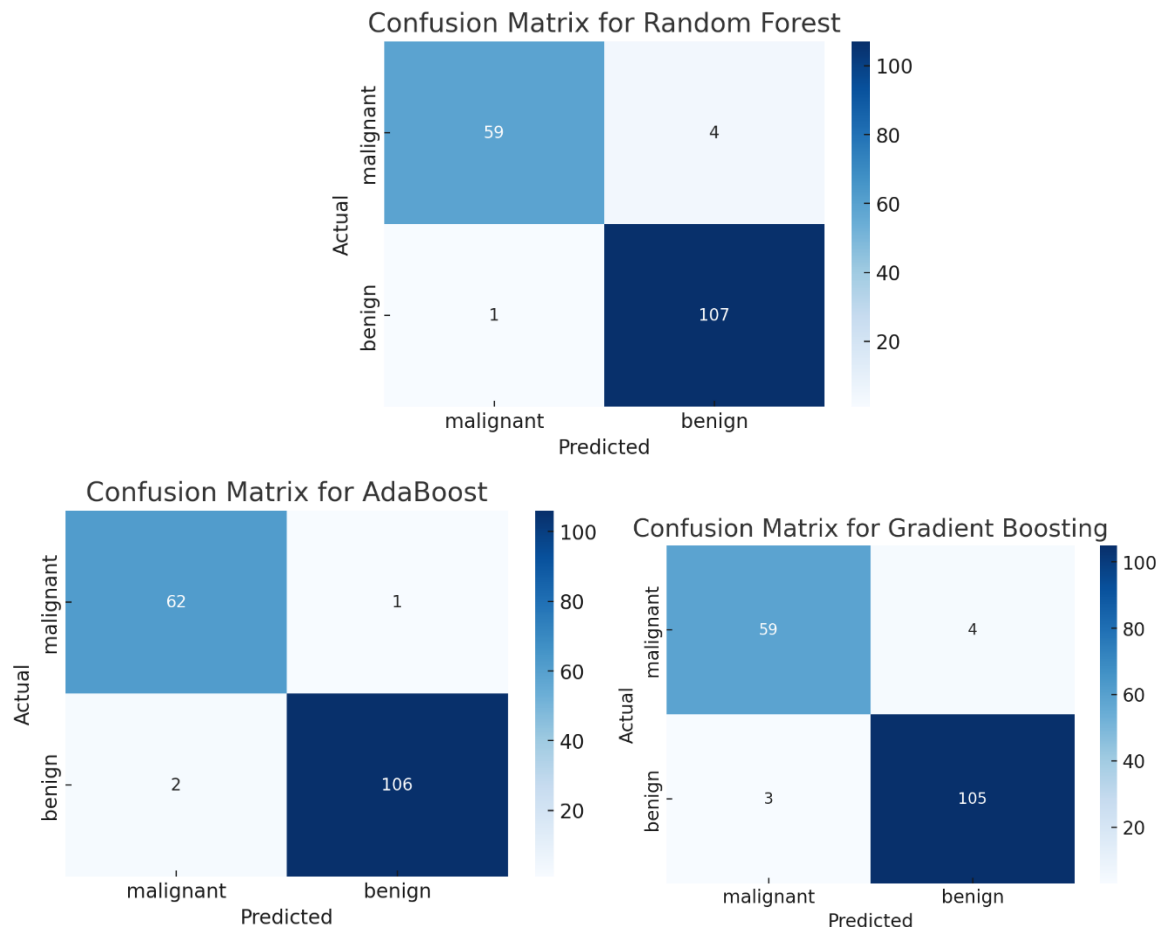
# AdaBoost
ada_model = AdaBoostClassifier(n_estimators=100, random_state=42)
ada_model.fit(X_train_scaled, y_train)
y_pred_ada = ada_model.predict(X_test_scaled)

# Gradient Boosting
gb_model = GradientBoostingClassifier(n_estimators=100, random_state=42)
gb_model.fit(X_train_scaled, y_train)
y_pred_gb = gb_model.predict(X_test_scaled)
```

Отже, три ансамблеві моделі були успішно навчені на тренувальних даних:

- **Random Forest** використовує 100 дерев.
- **AdaBoost** використовує 100 ітерацій для підсилення.
- **Gradient Boosting** використовує 100 кроків для підсилення градієнта.

Моделі зробили прогнози на тестовому наборі даних.



На графіках вище представлені матриці невідповідностей для трьох ансамблевих моделей:

1. **Random Forest:** матриця невідповідностей показує високу точність моделі, оскільки більшість точок правильно класифіковані.
2. **AdaBoost:** також показує гарний результат, але має трохи більше помилок класифікації, ніж Random Forest.
3. **Gradient Boosting:** продемонстрував подібні результати до Random Forest з мінімальними помилками класифікації.

Ці матриці невідповідностей наочно демонструють, наскільки добре кожна модель справляється з класифікацією класів на тестовому наборі даних.

Оцінка якості моделей

Оцінимо точність моделей, використовуючи **classification_report**, щоб оцінити метрики, такі як точність (precision), повнота (recall), F1-оцінка, та загальну точність (accuracy).

```

from sklearn.metrics import classification_report, accuracy_score

# Оцінка Random Forest
print("Random Forest:")
print(classification_report(y_test, y_pred_rf))
print(f"Точність: {accuracy_score(y_test, y_pred_rf):.2f}\n")

# Оцінка AdaBoost
print("AdaBoost:")
print(classification_report(y_test, y_pred_ada))
print(f"Точність: {accuracy_score(y_test, y_pred_ada):.2f}\n")

# Оцінка Gradient Boosting
print("Gradient Boosting:")
print(classification_report(y_test, y_pred_gb))
print(f"Точність: {accuracy_score(y_test, y_pred_gb):.2f}\n")

```

Результати:

- **Random Forest:**

Random Forest:					
	precision	recall	f1-score	support	
0	0.98	0.97	0.98	63	
1	0.99	0.99	0.99	108	
accuracy			0.98	171	
macro avg	0.98	0.98	0.98	171	
weighted avg	0.98	0.98	0.98	171	
Точність: 0.98					

AdaBoost:

AdaBoost:					
	precision	recall	f1-score	support	
0	0.94	0.95	0.94	63	
1	0.97	0.96	0.96	108	
accuracy			0.96	171	
macro avg	0.95	0.95	0.95	171	
weighted avg	0.96	0.96	0.96	171	
Точність: 0.96					

Gradient Boosting:

Gradient Boosting:					
	precision	recall	f1-score	support	
0	0.97	0.97	0.97		63
1	0.99	0.98	0.98		108
accuracy			0.98		171
macro avg	0.98	0.98	0.98		171
weighted avg	0.98	0.98	0.98		171
Точність: 0.98					

Висновки по результатам:

- **Random Forest** та **Gradient Boosting** показали найвищу точність — 98%. Ці моделі чудово справляються з класифікацією, особливо на складних наборах даних.
- **AdaBoost** трохи поступився за точністю (96%), але все ще показав добрі результати.

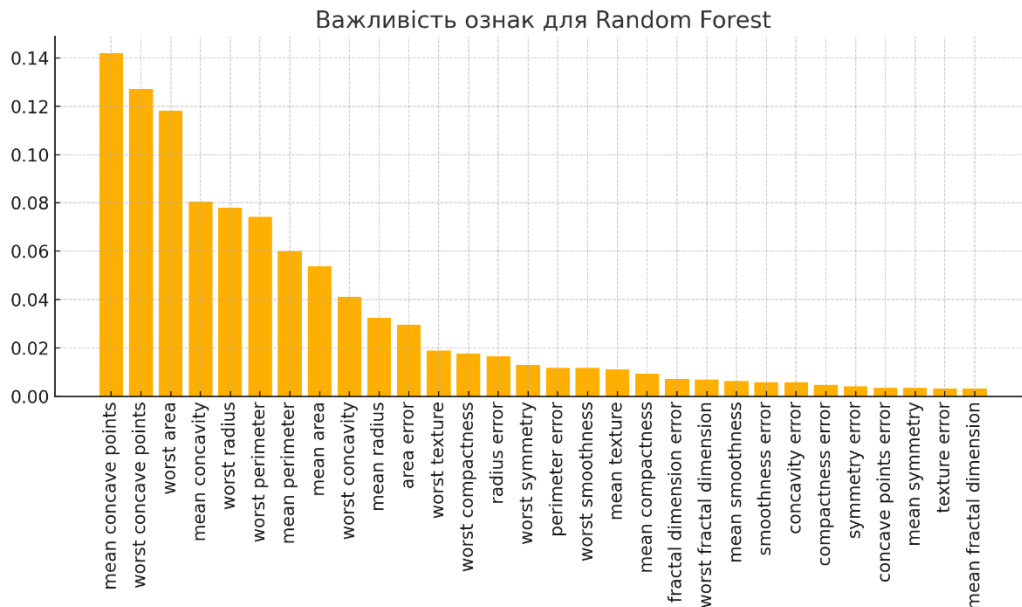
Візуалізація важливості ознак

Щоб зрозуміти, які ознаки найбільш важливі для моделі **Random Forest**, можна побудувати графік важливості ознак.

```
import matplotlib.pyplot as plt

# Важливість ознак для Random Forest
importances = rf_model.feature_importances_
indices = np.argsort(importances)[::-1]

plt.figure(figsize=(10,6))
plt.title("Важливість ознак для Random Forest")
plt.bar(range(X_train.shape[1]), importances[indices], align="center")
plt.xticks(range(X_train.shape[1]), data.feature_names[indices], rotation=90)
plt.tight_layout()
plt.show()
```



Висновок по графіку важливості ознак для моделі Random Forest:

Графік ілюструє, які ознаки набору даних **Breast Cancer** найбільше впливають на прийняття рішень моделлю **Random Forest**. Найважливіші ознаки мають більший вплив на результати класифікації, тоді як ознаки з меншою важливістю мають слабший вплив.

- **Найважливіші ознаки:** Деякі медичні характеристики, наприклад, *mean area*, *mean concave points* та *worst radius*, виявилися найбільш важливими для моделі. Це означає, що саме ці ознаки найбільше впливають на правильну класифікацію пацієнтів.
- **Менш важливі ознаки:** Є ознаки, які мають незначний вплив на рішення моделі, і їх можна було б потенційно виключити для спрощення моделі без втрати точності.

Цей аналіз важливості ознак може допомогти зробити висновки щодо того, які медичні показники є найбільш важливими для діагностики раку молочної залози в контексті використання методу **Random Forest**.

Завдання

Створіть прогноз на основі даних із завдання 1 відносно Вашого варіанту, використовуючи метод ансамблевого навчання – **Random Forest**, **AdaBoost**, або **Gradient Boosting**. Для вашого аналізу оберіть один із цих методів, побудуйте модель, обчисліть точність прогнозу та порівняйте результати з іншими моделями.

- Оцініть якість моделі за допомогою метрик точності (accuracy), повноти (recall), точності (precision), F1-оцінки.
- Проведіть аналіз важливості ознак для обраної моделі.
- Візуалізуйте результати, використовуючи матриці невідповідностей та графік важливості ознак.

Оцінювання звіту представлено у Вступі даного збірника.

Лабораторна робота №6.

Добування ознак із зображень

Теоретична частина:

Як машини зберігають зображення?

Почнемо з основ. Важливо зрозуміти, як ми можемо читати та зберігати зображення на наших машинах, перш ніж дивитися на щось інше. Вважайте це функцією '`pd.read_`', але для зображень.

Почну з простого прикладу. Подивіться на зображення нижче:



У нас є зображення числа 8. Подивіться уважно на зображення – ви помітите, що воно складається з маленьких квадратних коробок. **Вони називаються пікселями.**

Однак є застереження. Ми бачимо зображення такими, якими вони є – у їх візуальній формі. Ми можемо легко розрізнити краї та кольори, щоб визначити, що на зображенні. З іншого боку, машинам важко це зробити. Вони зберігають зображення у вигляді чисел. Подивіться на зображення нижче:

```
0 2 15 0 0 11 10 0 0 0 0 9 9 0 0 0
0 0 0 4 60 157 236 255 255 177 95 61 32 0 0 29
0 10 16 119 238 255 244 245 243 250 249 255 222 103 10 0
0 14 170 255 255 244 254 255 253 245 255 249 253 251 124 1
2 98 255 228 255 251 254 211 141 116 122 215 251 238 255 49
13 217 243 255 155 33 226 52 2 0 10 13 232 255 255 36
16 229 252 254 49 12 0 0 7 7 0 70 237 252 235 62
6 141 245 255 212 25 11 9 3 0 115 236 243 255 137 0
0 87 252 250 248 215 60 0 1 121 252 255 248 144 6 0
0 13 113 255 255 245 255 182 181 248 252 242 208 36 0 19
1 0 5 117 251 255 241 255 247 255 241 162 17 0 7 0
0 0 0 4 58 251 255 246 254 253 255 120 11 0 1 0
0 0 4 97 255 255 255 248 252 255 244 255 182 10 0 4
0 22 206 252 246 251 241 100 24 113 255 245 255 194 9 0
0 111 255 242 255 168 24 0 0 6 39 255 232 230 56 0
0 218 251 250 137 7 11 0 0 0 2 62 255 250 125 3
0 173 255 255 101 9 20 0 13 3 13 182 251 245 61 0
0 107 251 241 255 230 98 55 19 118 217 248 253 255 52 4
0 18 146 250 255 247 255 255 255 249 255 240 255 129 0 5
0 0 23 113 215 255 250 248 255 255 248 248 118 14 12 0
0 0 6 1 0 52 153 233 255 252 147 37 0 0 4 1
0 0 5 5 0 0 0 0 0 14 1 0 6 6 0 0
```

Машини зберігають зображення у вигляді матриці чисел. Розмір цієї матриці залежить від кількості пікселів у будь-якому даному зображенні.

Скажімо, розмір зображення становить 180 x 200 або $n \times m$. Ці розміри в основному є кількістю пікселів у зображенні (висота x ширина).

Ці числа або піксельні значення позначають інтенсивність або яскравість пікселя. Менші числа (ближче до нуля) позначають чорний колір, а більші числа (ближче

~ 40 ~

до 255) позначають білий. Ви зрозумієте все, про що ми дізналися, проаналізувавши зображення нижче.

Розміри зображення нижче становлять 22 x 16, що можна перевірити, підрахувавши кількість пікселів:



0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	15	110	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	4	0
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	2	62	255	250	125	3	0
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	65	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	14	1	0	6	6	0	0	0



Приклад, який ми щойно обговорювали, це чорно-біле зображення. А як щодо кольорових зображень (які набагато більш поширені в реальному світі)? Як ви думаєте, кольорові зображення також зберігаються у формі двовимірної матриці?

Кольорове зображення зазвичай складається з кількох кольорів, і майже всі кольори можуть бути створені з трьох основних кольорів – червоного, зеленого та синього.

Отже, у випадку кольорового зображення є три матриці (або канали) – червоний, зелений і синій. Кожна матриця має значення від 0 до 255, які представляють інтенсивність кольору для цього пікселя. Розгляньте зображення нижче, щоб зрозуміти цю концепцію:



Colour Image

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

У нас є кольорове зображення ліворуч (як би ми його бачили, люди). Праворуч ми маємо три матриці для трьох кольорних каналів – червоного, зеленого та синього. Три канали накладаються для формування кольорового зображення.

Практична частина:

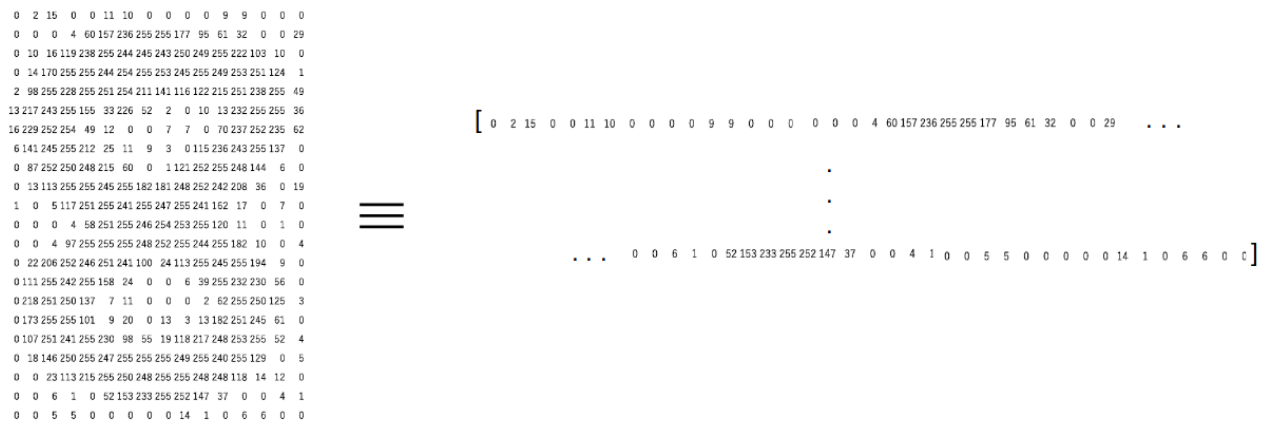
Спосіб №1: піксельні значення відтінків сірого як функції

Найпростіший спосіб створити об'єкти із зображення – використовувати ці необроблені значення пікселів як окремі об'єкти.

Розглянемо той самий приклад для нашого зображення вище (цифра «8») – розмір зображення 28 x 28.

Чи можете ви вгадати кількість функцій цього зображення? Кількість функцій буде такою ж, як і кількість пікселів! Отже, це число буде 784.

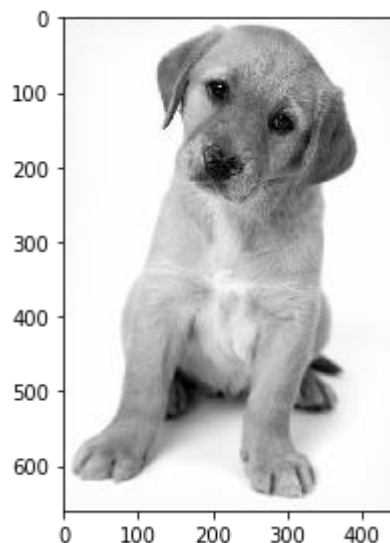
Тепер ось ще одне цікаве запитання – як ми розташуємо ці 784 пікселя як функції? Що ж, ми можемо просто додати значення кожного пікселя одне за одним, щоб створити вектор ознак. Це показано на зображенні нижче:



Давайте візьмемо зображення на Python і створимо для цього зображення такі функції:

```
image = imread('puppy.jpeg', as_gray=True)
image.shape, imshow(image)
```

(650, 450)



Форма зображення тут 650 x 450. Отже, кількість функцій має бути 297 000. Ми можемо створити це за допомогою функції *зміни форми* з NumPy, де ми вказуємо розмір зображення:

```
#pixel features
features = np.reshape(image, (660*450))

features.shape, features
```

(297000,)

масив ([0,96470588, 0,96470588, 0,96470588, ..., 0,96862745, 0,96470588, 0.96470588])

Тут у нас є наша особливість – це одновимірний масив довжиною 297 000. Легко, правда? Спробуйте цей метод вилучення функцій у наведеному нижче вікні живого кодування:

Але тут у нас був лише один канал або зображення у відтінках сірого. Чи можемо ми зробити те саме для кольорового зображення? Давай дізнаємось!

Спосіб №2: середнє значення пікселів каналів

Читаючи зображення в попередньому розділі, ми встановили параметр *as_gray = True*. Тож у нас був лише один канал на зображенні, і ми могли легко додати значення пікселів. Давайте видалимо параметр і знову завантажимо зображення:

```
image = imread('puppy.jpeg')
image.shape
```

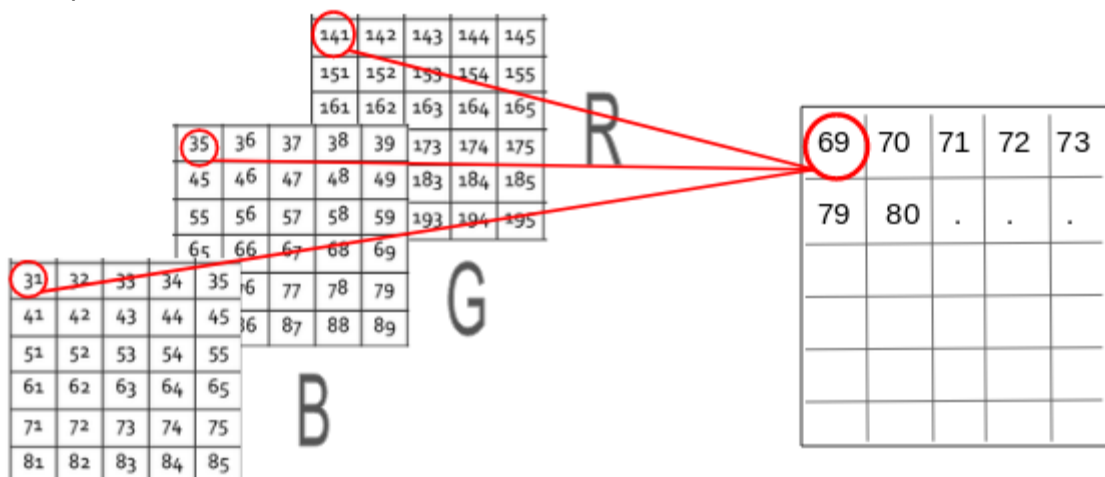
(660, 450, 3)

Цього разу зображення має розмірність (660, 450, 3), де 3 — це кількість каналів. Ми можемо продовжити створення функцій, як ми це робили раніше. Кількість функцій у цьому випадку буде $660 \times 450 \times 3 = 891\,000$.

Крім того, ось інший підхід, який ми можемо використати:

Замість того, щоб використовувати значення пікселів з трьох каналів окремо, ми можемо створити нову матрицю, яка має середнє значення пікселів з усіх трьох каналів.

Зображення нижче дасть вам ще більше ясності щодо цієї ідеї:



При цьому кількість функцій залишається незмінною, і ми також беремо до уваги значення пікселів з усіх трьох каналів зображення. Давайте закодуємо це на Python. Ми створимо нову матрицю з тим самим розміром 660 x 450, де всі значення ініціалізуються рівними 0. Ця матриця зберігатиме середні значення пікселів для трьох каналів:

```
image = imread('puppy.jpeg')
```

```
feature_matrix = np.zeros((660,450))
feature_matrix.shape
```

(660, 450)

У нас є тривимірна матриця розмірів (660 x 450 x 3), де 660 — висота, 450 — ширина, а 3 — кількість каналів. Щоб отримати середні значення пікселів, ми використаємо цикл *for* :

```
for i in range(0,image.shape[0]):
    for j in range(0,image.shape[1]):
        feature_matrix[i][j] = ((int(image[i,j,0]) + int(image[i,j,1]) + int(image[i,j,2]))/3)
```

Нова матриця матиме ту саму висоту та ширину, але лише 1 канал. Тепер ми можемо виконувати ті самі кроки, що й у попередньому розділі. Ми додаємо значення пікселів одне за одним, щоб отримати 1D-масив:

```
features = np.reshape(feature_matrix, (660*450))
features.shape
```

(297000,)

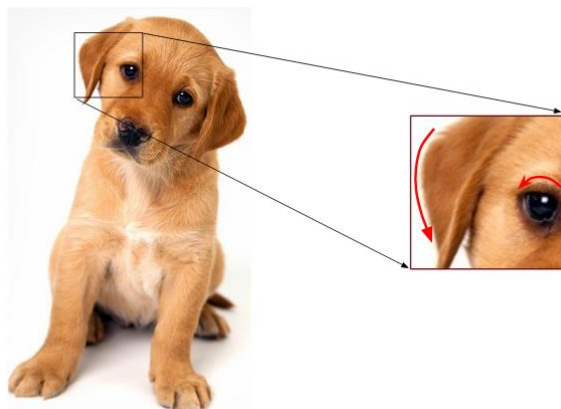
Спосіб № 3: Вилучення елементів Edge

Вважайте, що нам дано зображення нижче, і нам потрібно ідентифікувати присутні на ньому об'єкти:



Ви, мабуть, миттєво впізнали об'єкти – собаку, машину та kota. Які особливості ви врахували, виділяючи кожне з цих зображень? Одним із важливих факторів може бути форма, а потім колір або розмір. Що, якби машина також могла визначити форму, як ми?

Подібна ідея полягає в тому, щоб виділити ребра як об'єкти та використовувати їх як вхідні дані для моделі. Я хочу, щоб ви на мить подумали над цим: як ми можемо визначити краї на зображенні? Край – це в основному місце різкої зміни кольору. Подивіться на зображення нижче:



Я виділив тут два ребра. Ми могли ідентифікувати край, оскільки відбулася зміна кольору з білого на коричневий (на правому зображенні) і коричневого на чорний (на

лівому). А як відомо, зображення представляється у вигляді чисел. Отже, ми будемо шукати пікселі, навколо яких спостерігається різка зміна значень пікселів.

Припустимо, у нас є наступна матриця для зображення:

121	10	78	96	125
48	152	68	125	111
145	78	85	89	65
154	214	56	200	66
214	87	45	102	45

Щоб визначити, чи є піксель краєм чи ні, ми просто віднімаємо значення з обох боків пікселя. У цьому прикладі ми маємо виділене значення 85. Ми знайдемо різницю між значеннями 89 і 78. Оскільки ця різниця не дуже велика, ми можемо сказати, що навколо цього пікселя немає краю.

Тепер розглянемо піксель 125, виділений на зображенні нижче:

121	10	78	96	125
48	152	68	125	111
145	78	85	89	65
154	214	56	200	66
214	87	45	102	45

Оскільки різниця між значеннями по обидві сторони цього пікселя є великою, ми можемо зробити висновок, що в цьому пікселі є значний перехід і, отже, це край. Тепер постає питання, чи потрібно робити цей крок вручну?

Немає! Існують різні ядра, які можна використовувати для виділення країв на зображенні. Метод, який ми щойно обговорили, також може бути досягнутий за допомогою ядра Prewitt (у напрямку x). Нижче наведено ядро Prewitt:

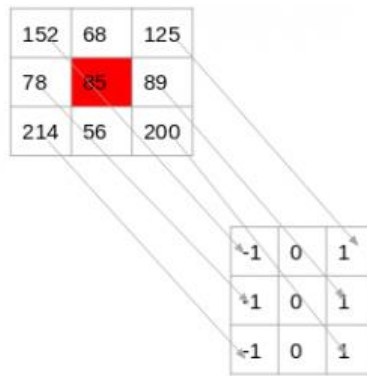
-1	0	1
-1	0	1
-1	0	1

Prewitt Kernel
X Direction

-1	-1	-1
0	0	0
1	1	1

Prewitt Kernel
Y Direction

Ми беремо значення, що оточують вибраний піксель, і множимо його на вибране ядро (ядро Превітта). Потім ми можемо додати отримані значення, щоб отримати остаточне значення. Оскільки ми вже маємо -1 в одному стовпці та 1 в іншому стовпці, додавання значень еквівалентно отриманню різниці.



Є багато інших ядер, і нижче я згадав чотири найпопулярніші:

-1	0	1
-1	0	1
-1	0	1

Prewitt Kernel
X Direction

-1	0	1
-2	0	2
-1	0	1

Sobel Kernel
X Direction

-1	-1	-1
0	0	0
1	1	1

Prewitt Kernel
Y Direction

-1	-2	-1
0	0	0
1	2	1

Sobel Kernel
Y Direction



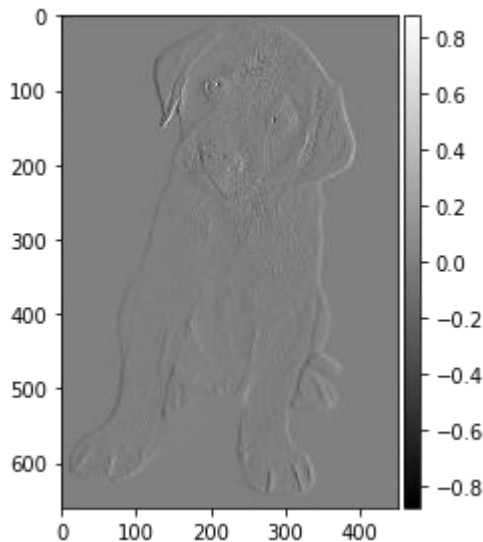
Давайте повернемося до блокнота та згенеруємо крайові елементи для того самого зображення:

```
#importing the required libraries
import numpy as np
from skimage.io import imread, imshow
from skimage.filters import prewitt_h, prewitt_v
import matplotlib.pyplot as plt
%matplotlib inline

#reading the image
image = imread('puppy.jpeg', as_gray=True)

#calculating horizontal edges using prewitt kernel
edges_prewitt_horizontal = prewitt_h(image)
#calculating vertical edges using prewitt kernel
edges_prewitt_vertical = prewitt_v(image)

imshow(edges_prewitt_vertical, cmap='gray')
```

Завдання

1. **Вибір зображення для аналізу:** Оберіть зображення з бази даних (Додаток Б), яке буде використане для вилучення ознак. Оцініть структуру обраного зображення, враховуючи його розмірність, кольоровість та складність об'єктів. Цей вибір допоможе визначити відповідний метод для вилучення ознак.

2. **Попередня обробка зображення:** Перед вилученням ознак виконайте попередню обробку зображення. Для цього необхідно перевести кольорове зображення в градації сірого (якщо це необхідно) або забезпечити коректну роботу з кольоровими зображеннями. Переконайтеся, що зображення має відповідну розмірність та формат для подальшої обробки.

3. Вилучення ознак із зображень:

- Виконайте вилучення ознак із зображення за допомогою трьох підходів:
 - **Піксельні значення:** Враховуйте кожне піксельне значення як окрему ознаку. Проаналізуйте, скільки ознак було створено, та оцініть, чи забезпечує цей підхід достатню кількість корисних даних для подальшої обробки.
 - **Середнє значення каналів:** Для кольорових зображень обчисліть середні значення пікселів із трьох каналів (червоного, зеленого, синього) для кожного пікселя. Це дозволить зменшити кількість ознак і спростити обробку, зберігаючи при цьому основні колірні характеристики зображення.
 - **Контури об'єктів:** Використовуйте методи виявлення країв (наприклад, ядро Prewitt) для визначення контурів об'єктів на зображенні. Цей підхід дозволяє виділити ключові об'єкти та зменшити кількість даних, що обробляються, зосередившись на структурі об'єктів.

4. Аналіз та порівняння методів:

- Порівняйте результати, отримані за допомогою трьох методів вилучення ознак. Оцініть, який із підходів був найефективнішим для вашого зображення. Зверніть увагу на кількість отриманих ознак, їхню інформативність і те, наскільки ефективно кожен метод виділяє ключові характеристики зображення.
- Поясніть, як вилучені ознаки можуть використовуватися для подальших етапів аналізу, таких як класифікація чи сегментація зображень. Розгляньте, чи доцільно

використовувати всі отримані ознаки, або ж можна вибрати найважливіші для підвищення ефективності подальшої обробки.

5. Аналіз результатів:

- Зробіть висновки про те, які методи вилучення ознак були найкращими для обраного вами зображення. Поясніть, які з отриманих ознак є найінформативнішими для подальшої обробки.
- Опишіть, чи були відмінності у вилученні ознак для кольорових та чорно-білих зображень. Це допоможе зрозуміти, які підходи найкраще підходять для аналізу різних типів зображень.

Оцінювання звіту представлене у Вступі даного збірника.

Лабораторна робота №7.

Обробка природної мови

Теоретична частина

Обробка природної мови (NLP) – це галузь штучного інтелекту, що спрямована на взаємодію між комп'ютерами і людською мовою. Мета NLP полягає у тому, щоб комп'ютери могли розуміти, інтерпретувати і ефективно використовувати природну мову, яку люди використовують у своєму повсякденному житті. Завдяки NLP комп'ютери можуть працювати з текстовими і мовними даними, проводити аналіз і отримувати значущу інформацію з неструктурованих текстових масивів.

Аналіз настрою (Sentiment Analysis) – це процес визначення емоційної тональності тексту, чи є він позитивним, негативним або нейтральним. Ця техніка широко застосовується у таких сферах, як маркетинг, обслуговування клієнтів та аналіз відгуків. Наприклад, за допомогою аналізу настрою можна визначити, чи є відгуки на продукт позитивними або негативними.

Для реалізації аналізу настроїв часто використовують бібліотеки Python, такі як **VADER** (Valence Aware Dictionary and Sentiment Reasoner), що є частиною пакету **NLTK**. Цей інструмент дозволяє аналізувати тональність тексту без попередньої підготовки чи маркування даних, що робить його придатним для роботи з текстами в соціальних мережах та коментарях.

Розпізнавання іменованих сутностей (Named Entity Recognition, NER) – це задача автоматичного виявлення сутностей у тексті та класифікації їх за типами, такими як особи, організації, географічні місця, дати, грошові одиниці тощо. NER широко застосовується для структурування неструктурованих текстових даних у пошукових системах, рекомендаційних системах та автоматизованих системах підтримки клієнтів.

У Python можна використовувати бібліотеку **spaCy**, яка забезпечує високоточну модель для розпізнавання іменованих сутностей. Ця бібліотека дозволяє швидко і ефективно класифікувати сутності у великих масивах тексту, надаючи при цьому потужні інструменти для подальшого аналізу та візуалізації результатів.

Стемінг і лематизація – це техніки нормалізації слів у тексті. Їхня мета полягає в тому, щоб зменшити різноманіття форм слів, перетворивши їх до базової форми.

- **Стемінг (Stemming)** – це процес скорочення слова до його кореня або основи. Наприклад, слова "друзі", "дружба" та "дружній" можуть бути скорочені до основи "друг". Ця техніка менш точна, оскільки не завжди дає граматично коректні слова.
- **Лематизація (Lemmatization)** – це процес перетворення слова до його словникової форми, з урахуванням частини мови. Наприклад, слова "бігти", "біжить", "біг" будуть перетворені на "біг". Лематизація є точнішою, але займає більше часу через необхідність доступу до словника.

У Python для реалізації стемінгу та лематизації можна використовувати бібліотеки **NLTK** або **spaCy**.

Модель "Сумка слів" (Bag of Words, BoW) – це базова техніка для перетворення тексту в числове представлення, яке можна використовувати для побудови моделей

машинного навчання. Модель BoW не враховує порядок слів, а лише частоту їхньої появи у тексті.

Кожен документ перетворюється на вектор, де кожен елемент вектора відповідає кількості появ певного слова в тексті. Це дозволяє використовувати ці вектори для подальшого аналізу, наприклад для класифікації текстів або пошуку схожих документів.

TF-IDF (Term Frequency – Inverse Document Frequency) – це техніка, яка дозволяє враховувати важливість слова в тексті або документі в залежності від його частоти в корпусі текстів. TF-IDF вказує на те, наскільки значущим є певне слово для конкретного документа, беручи до уваги його поширеність у всьому наборі текстів.

TF-IDF розраховується наступним чином:

- **TF (Термова частота)** визначає частоту появи слова в документі.
- **IDF (Зворотна частота документа)** показує, наскільки рідко зустрічається це слово у всьому наборі документів.

TF-IDF дозволяє знижувати вагу часто використовуваних слів (як-от "і", "в") та підвищувати важливість менш поширених, але значущих слів.

На відміну від CountVectorizer, TF-IDF обчислює «ваги», які показують, наскільки релевантним є слово для документа в колекції документів (він же корпус). Значення TF-IDF збільшується пропорційно кількості разів, коли слово з'являється в документі, і компенсується кількістю документів у корпусі, які містять це слово. **Простіше кажучи, що вищий показник TF-IDF, то рідкісніший, унікальніший чи цінніший термін, і навпаки.** У ньому є програми для пошуку інформації, такі як пошукові системи, які націлені на надання результатів, які найбільше відповідають тому, що ви шукаєте.

Перш ніж ми побачимо реалізацію Python, давайте розглянемо приклад, щоб ви мали уявлення про те, як обчислюються TF та IDF. Для наступного прикладу ми використаємо ті ж речення, що й для прикладу CountVectorizer.

Речення 1: «Я люблю писати код на Python. Я люблю код Python»

Речення 2: «Я ненавиджу писати код на Java. Я ненавиджу код Java»

Термінова частота (TF)

Існують різні способи визначення частоти терміну. Один пропонує сам підрахунок (тобто те, що робить Count Vectorizer), але інші припускають, що це частота слова в реченні, поділена на загальну кількість слів у реченні. Для цього простого прикладу ми використаємо перший критерій, тому термін частота показаний у наступній таблиці.

Words	TF1	TF2
love	2	0
writing	1	1
code	2	2
Python	2	0
hate	0	2
Java	0	2

Зображення автора

Як бачите, значення такі самі, як і раніше обчислені для CountVectorizer. Також не враховуються слова з 2 літер і менше.

Інверсна частота документів (IDF)

IDF також розраховується різними способами. Хоча стандартна нотація підручника визначає IDF як $\text{idf}(t) = \log [n / (\text{df}(t) + 1)]$, бібліотека sklearn, яку ми будемо використовувати пізніше в Python, обчислює формулу за замовчуванням наступним чином.

$$IDF = \log \left(\frac{\text{Total number of sentences} + 1}{\text{Number of sentences containing that word} + 1} \right) + 1$$

Крім того, sklearn передбачає натуральний логарифм ln замість log згладжування (*smooth_idf=True*). Давайте обчислимо значення IDF для кожного слова, як це зробить sklearn.

Text	IDF1	IDF2
love	$\ln(3/2)+1 = 1.4$	$\ln(3/2)+1 = 1.4$
writing	$\ln(3/3)+1 = 1$	$\ln(3/3)+1 = 1$
code	$\ln(3/3)+1 = 1$	$\ln(3/3)+1 = 1$
Python	$\ln(3/2)+1 = 1.4$	$\ln(3/2)+1 = 1.4$
hate	$\ln(3/2)+1 = 1.4$	$\ln(3/2)+1 = 1.4$
Java	$\ln(3/2)+1 = 1.4$	$\ln(3/2)+1 = 1.4$

Отримавши значення TF і IDF, ми можемо отримати TF-IDF шляхом множення обох значень (TF-IDF = TF * IDF). Значення наведено в таблиці нижче.

Text	TF-IDF1	TF-IDF2
love	2.8	0
writing	1	1
code	2	2
Python	2.8	0
hate	0	2.8
Java	0	2.8

Wordcloud (Хмара слів) – це візуалізація, яка допомагає ідентифікувати ключові слова в тексті, представивши їх у вигляді хмари, де найбільш часто вживані слова мають більший розмір і виділяються яскравими кольорами. Хмари слів є простим і зручним інструментом для візуального огляду текстових даних, що допомагає швидко виявити найбільш значущі терміни.

У Python для створення хмар слів можна використовувати бібліотеку **Wordcloud**, яка дозволяє легко генерувати подібні візуалізації на основі текстових даних.

Практична частина

Аналіз настрою

Для простих випадків у Python ми можемо використовувати VADER (Valence Aware Dictionary for Sentiment Reasoning), який доступний у пакеті NLTK і може бути застосований безпосередньо до немаркованих текстових даних. Як приклад, давайте отримаємо всі оцінки настроїв реплік, вимовлених персонажами телешоу.

Спочатку ми обговорюємо набір даних, доступний на [Kaggle](#) або моєму [Github](#) під назвою «avatar.csv», а потім за допомогою VADER обчислюємо оцінку кожного проголошеного рядка. Усе це зберігається у df_character_sentiment фреймі даних.

```
import pandas as pd
from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

```

# reading and wrangling data
df_avatar = pd.read_csv('avatar.csv', engine='python')
df_avatar_lines = df_avatar.groupby('character').count()
df_avatar_lines = df_avatar_lines.sort_values(by=['character_words'],
ascending=False)[:10]
top_character_names = df_avatar_lines.index.values

# filtering out non-top characters
df_character_sentiment =
df_avatar[df_avatar['character'].isin(top_character_names)]
df_character_sentiment = df_character_sentiment[['character', 'character_words']]

# calculating sentiment score
sid = SentimentIntensityAnalyzer()
df_character_sentiment.reset_index(inplace=True, drop=True)
df_character_sentiment[['neg', 'neu', 'pos', 'compound']] =
df_character_sentiment['character_words'].apply(sid.polarity_scores).apply(pd.Series)
df_character_sentiment

```

Нижче df_character_sentiment ми бачимо, що кожне речення отримує негативну, нейтральну та позитивну оцінку.

	character	character_words	neg	neu	pos
0	Katara	Water. Earth. Fire. Air. My grandmother used t...	0.130	0.804	0.066
1	Sokka	It's not getting away from me this time. Watc...	0.000	1.000	0.000
2	Katara	Sokka, look!	0.000	1.000	0.000
3	Sokka	Sshh! Katara, you're going to scare it away. ...	0.200	0.800	0.000
4	Katara	But, Sokka! I caught one!	0.000	1.000	0.000
...
7053	Zuko	At least you don't look like a boar-q-pine! My...	0.183	0.817	0.000
7054	Suki	And why did you paint me firebending?	0.000	1.000	0.000
7055	Sokka	I thought it looked more exciting that way. O...	0.000	0.687	0.313
7056	Iroh	Hey, my belly's not that big anymore. I've rea...	0.000	1.000	0.000
7057	Toph	Well I think you all look perfect!	0.000	0.396	0.604

Ми навіть могли б згрупувати бали за символами та обчислити середнє значення, щоб отримати оцінку настрою для персонажа, а потім представити його за допомогою горизонтальних смужкових графіків за допомогою бібліотеки matplotlib (результат цього показано в цій [статті](#))

Примітка: VADER оптимізовано для тексту в соціальних мережах, тому ми повинні сприймати результати з недовірою. Ви можете використовувати більш повний алгоритм або розробити власний за допомогою бібліотек машинного навчання. За посиланням нижче є повний посібник щодо того, як створити його з нуля за допомогою Python за допомогою бібліотеки sklearn.

Розпізнавання іменованих сутностей (NER)

У Python ми можемо використовувати розпізнавання іменованих сутностей SpaCy, яке підтримує такі типи сутностей.

TYPE	DESCRIPTION
PERSON	People, including fictional
NORP	Nationalities or religious or political groups
FACILITY	Buildings, airports, highways, bridges, etc
ORG	Companies, agencies, institutions, etc
GPE	Countries, cities, states
LOC	Non-GPE locations, mountain ranges, bodies of water
PRODUCT	Objects, vehicles, foods, etc (Not services)
EVENT	Named hurricanes, battles, wars, sports events, etc
WORK_OF_ART	Titles of books, songs, etc
LAW	Named documents made into laws
LANGUAGE	Any named language
DATE	Absolute or relative dates or periods.
TIME	Times smaller than a day
PERCENT	Percentage, including "%".
MONEY	Monetary values, including unit
QUANTITY	Measurements, as of weight or distance
ORDINAL	"first", "second", etc
CARDINAL	Numerals that do not fall under another type

Щоб побачити це в дії, ми спочатку імпортуємо `spacy`, а потім створюємо `nlp` змінну, яка зберігатиме `en_core_web_sm` конвеєр. Це невеликий конвеєр англійської мови, навчений на письмовому веб-тексті (блоги, новини, коментарі), який включає лексику, вектори, синтаксис та сутності. Щоб знайти сутності, ми застосовуємо `nlp` до речення.

```
import spacy\nlp = spacy.load("en_core_web_sm")\ndoc = nlp("Biden invites Ukrainian president to White House this summer")\nprint([(X.text, X.label_) for X in doc.ents])
```

Результат:

```
[('Biden', 'PERSON'), ('Ukrainian', 'GPE'), ('White House', 'ORG'), ('this summer', 'DATE')]
```

Спейсі виявив, що «Байден» — це особа, «Українець» — це GPE (країни, міста, штати, «Білий дім» — організація, а «цього літа» — дата.

Створення основи та лематизація

Бібліотека Python NLTK полегшує роботу з обома методами. Давайте подивимося на це в дії.

Реалізація Python (Stemming)

Для англійської мови в `nlTK` доступні дві популярні бібліотеки — Porter Stemmer і LancasterStemmer.


```
from nltk.stem import PorterStemmer
from nltk.stem import LancasterStemmer# PorterStemmer
porter = PorterStemmer()
# LancasterStemmer
lancaster = LancasterStemmer()print(porter.stem("friendship"))
print(lancaster.stem("friendship"))
```

Алгоритм PorterStemmer дотримується не лінгвістики, а набору з 5 правил для різних випадків, які застосовуються поетапно для створення основ. Код `print(porter.stem("friendship"))` виведе слово `friendship`

LancasterStemmer простий, але може виникнути важкий стемінг через ітерації та надмірний стеммінг. Це призводить до того, що основи не є лінгвістичними, або вони можуть не мати значення. Код `print(lancaster.stem("friendship"))` виведе слово `friend`.

Ви можете спробувати будь-яке інше слово, щоб побачити, чим відрізняються обидва алгоритми. У випадку інших мов ви можете імпортувати `SnowballStemmer` з `nltk.stem`

Ми знову використаємо NLTK, але цього разу ми імпортуємо `WordNetLemmatizer`, як показано в коді нижче.

```
from nltk import WordNetLemmatizerlemmatizer = WordNetLemmatizer()
words = ['articles', 'friendship', 'studies', 'phones']for word in words:
    print(lemmatizer.lemmatize(word))
```

Лематизація генерує різні результати для різних значень частини мови (POS). Деякі з найпоширеніших значень POS: дієслово (v), іменник (n), прикметник (a) і прислівник (r). Значенням POS за замовчуванням у лематизації є іменник, тому надрукованими значеннями для попереднього прикладу будуть `article`, та `friendshipstudyphone`

Давайте змінимо значення POS на дієслово (v).

```
from nltk import WordNetLemmatizerlemmatizer = WordNetLemmatizer()
words = ['be', 'is', 'are', 'were', 'was']for word in words:
    print(lemmatizer.lemmatize(word, pos='v'))
```

У цьому випадку Python надрукує слово безля всіх значень у списку.

Сумка слів

Бібліотека `sklearn` Python містить інструмент під назвою `CountVectorizer`, який виконує більшу частину робочого процесу BoW.

Давайте використаємо наступні 2 речення як приклади.

Речення 1: «Я люблю писати код на Python. Я люблю код Python»

Речення 2: «Я ненавиджу писати код на Java. Я ненавиджу код Java»

Обидва речення будуть збережені в списку під назвою text. Потім ми створимо фрейм даних df для зберігання цього text списку. Після цього ми запустимо екземпляр CountVectorizer (cv), а потім підберемо та перетворимо текстові дані, щоб отримати числове представлення. Це буде збережено в матриці термінів документа df_dtm.

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
text = ["I love writing code in Python. I love Python code",
        "I hate writing code in Java. I hate Java code"]
df = pd.DataFrame({'review': ['review1', 'review2'], 'text': text})
cv = CountVectorizer(stop_words='english')
cv_matrix = cv.fit_transform(df['text'])
df_dtm = pd.DataFrame(cv_matrix.toarray(),
                      index=df['review'].values,
                      columns=cv.get_feature_names())
df_dtm
```

Представлення BoW, створене за допомогою CountVectorizer, що зберігається в df_dtm виглядає як на малюнку нижче. Майте на увазі, що слова з 2 літер або менше не враховуються CountVectorizer.

	code	hate	java	love	python	writing
review1	2	0	0	2	2	1
review2	2	2	2	0	0	1

Як ви можете бачити, числа всередині матриці представляють кількість разів, коли кожне слово згадувалося в кожному огляді. Такі слова, як «любов», «ненависть» і «код» мають однакову частоту (2) у цьому прикладі.

Загалом, ми можемо сказати, що CountVectorizer добре виконує свою роботу, розміщуючи текст, створюючи словниковий запас і генеруючи вектори; однак це не очистить необроблені дані. Я створив інструкцію з очищення та підготовки даних у Python, перегляньте її, якщо ви хочете дізнатися найкращі практики.

Періодичність терміну – зворотна частота документа (TF-IDF)

Завдяки бібліотеці sklearn обчислення TF-IDF, наведеного в таблиці вище, у Python вимагає кількох рядків коду.

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
text = ["I love writing code in Python. I love Python code",
        "I hate writing code in Java. I hate Java code"]
df = pd.DataFrame({'review': ['review1', 'review2'], 'text': text})
```

```
tfidf = TfidfVectorizer(stop_words='english', norm=None)
tfidf_matrix = tfidf.fit_transform(df['text'])
df_dtm = pd.DataFrame(tfidf_matrix.toarray(),
                      index=df['review'].values,
                      columns=tfidf.get_feature_names())
df_dtm
```

представлення TF-IDF, збережене в `df_dtm` виглядає так, як показано на малюнку 1. Знову ж таки, слова з 2 літер або менше не враховуються TF-IDF.

	code	hate	java	love	python	writing
review1	2.0	0.00000	0.00000	2.81093	2.81093	1.0
review2	2.0	2.81093	2.81093	0.00000	0.00000	1.0

примітка. За замовчуванням `TfidfVectorizer()` використовує нормалізацію `l2`, але для виведення тих самих формул, які показані вище, ми встановлюємо `norm=None` як аргумент. Щоб дізнатися більше про формули, які за замовчуванням використовуються, і про те, як ви можете їх налаштувати, перегляньте його [документацію](#).

Wordcloud

[illegible]

де резултат коду вище.



Wordcloud настільки популярні, тому що вони привабливі, прості для розуміння та для створення.

и можете ще більше налаштувати, змінивши кольори, видаливши стоп-слова, змінивши своє зображення або навіть додавши власне зображення, щоб

використовувати його як маску Wordcloud. Щоб отримати докладнішу інформацію, перегляньте посібник нижче.

Завдання

1. **Вибір текстових даних для аналізу:** Виберіть текстовий набір даних із варіанту (Додаток В), який буде використано для аналізу. Цей текстовий масив може містити новини, відгуки, коментарі або інший тип тексту. Проаналізуйте структуру та обсяг обраних текстів, оскільки це вплине на підходи для їх обробки.
2. **Попередня обробка текстових даних:** Проведіть попередню обробку текстових даних, щоб підготувати їх до подальшого аналізу: Видаліть стоп-слова, Перетворіть текст у нижній регістр для уникнення дублювання, Токенізуйте текст, розбивши його на окремі слова або фрази, Видаліть спеціальні символи та пунктуацію.
3. **Застосування методів NLP:** Виконайте повний аналіз текстових даних за допомогою наступних методів: **Аналіз настрою, Розпізнавання іменованих сутностей (NER), Лематизація та стемінг, Частота термінів (Bag of Words), TF-IDF (Термова частота – зворотна частота документа, Wordcloud (Хмара слів).**
4. **Оцінка та аналіз результатів:** Проаналізуйте результати застосованих методів NLP і визначте, який метод найкраще підходить для вашого набору даних. Оцініть вплив попередньої обробки текстів на точність та якість аналізу настроїв, розпізнавання сутностей, частотного аналізу та інших методів.

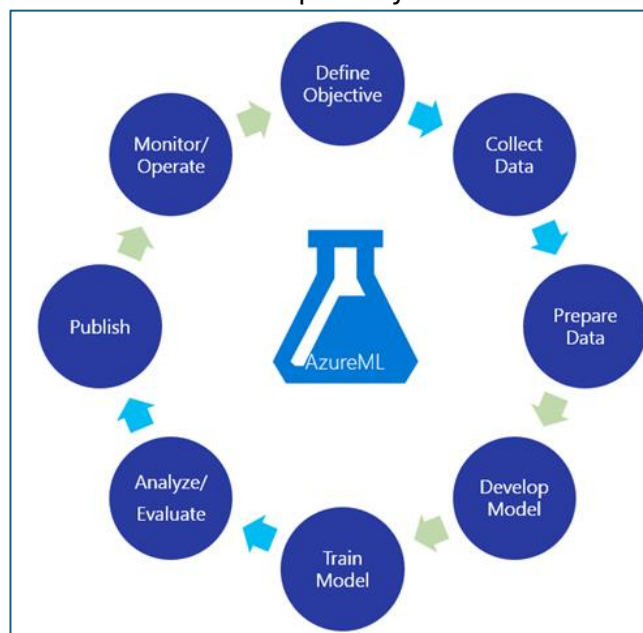
Оцінювання звіту представлено у Вступі даного збірника.

Лабораторна робота №8.

Масштабування процесу машинного навчання

Теоретична частина

Azure Machine Learning - один з найпростіших інструментів для використання машинного навчання, що прибирає вхідний бар'єр для всіх, хто вирішує використовувати його для своїх потреб. З Azure ML більше не треба бути математиком.



Студія машинного навчання Microsoft Azure (класична)

Студія машинного навчання Microsoft Azure (класична) - це інструмент для спільної роботи, що підтримує функцію перетягування об'єктів і призначений для створення, тестування і розгортання рішень для прогнозного аналізу даних. Студія машинного навчання Azure (класична) публікує моделі як веб-служби, які потім можна використовувати в призначених для користувача додатках і засобах бізнес-аналітики (наприклад, в Excel).

Студія машинного навчання (класична) - це місце, де перетинаються обробка і аналіз даних, прогнозна аналітика, хмарні ресурси і дані клієнта.

Для розробки моделі прогнозової аналітики зазвичай використовуються дані з одного або декількох джерел. Для отримання набору результатів ці дані перетворюються і аналізуються за допомогою різних операцій і статистичних функцій. Така розробка моделі - це ітеративний процес. Шляхом зміни різних функцій і їх параметрів виконується зведення результатів, поки не буде отримана підготовлена і ефективна модель.

Студія машинного навчання Azure (класична) надає інтерактивно-візуальний робочий простір, що спрощує створення, тестування і виконання ітерацій моделі прогнозової аналітики. Ви перетягуєте набори даних і модулі аналізу на інтерактивне

полотно і пов'язуєте їх разом, щоб створити експеримент, який потім виконується в Студії машинного навчання (класичної). Для виконання ітерацій в макеті моделі слід відредагувати експеримент, при необхідності зберегти копію і виконати його знову. Коли ви будете готові, навчальний експеримент можна перетворити в прогнозний, а потім опублікувати його як веб-службу, щоб модель стала доступна іншим користувачам.

Практична частина:

Побудова прогностичної моделі

Для початку перейдіть за посиланням <https://azure.microsoft.com/en-us/services/machine-learning/#product-overview>, в пункті меню Products виберіть Analytics і натисніть Machine Learning.

Для роботи з Azure ML вам необхідна активна підписка Microsoft Azure. Якщо вона у вас вже є, то просто увійдіть в Azure Management Portal, інакше - попередньо зареєструйте безкоштовну пробну обліковий запис, перейшовши по <https://azure.microsoft.com/en-us/free/>.

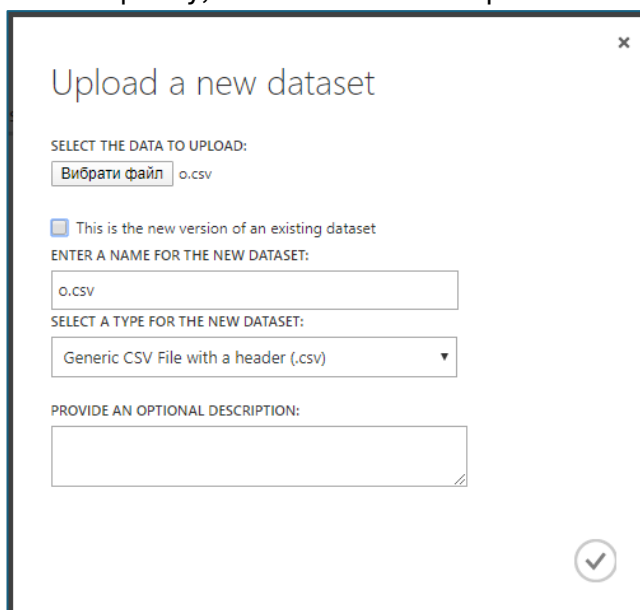
Після успішної авторизації ви опинитеся на головній сторінці Azure ML Studio (<https://studio.azureml.net/>) - середовищі, де буде відбуватися вся подальша робота.

Щоб розробити модель прогнозування населення, будуть потрібні дані, які можна використовувати для навчання і подальшої перевірки моделі. Використаємо набір даних з теми 6 (метод головних компонент).

Студія машинного навчання (класична) краще працює з файлами, в яких використовуються роздільники-коми (CSV-файлами), тому ви перетворюєте набір даних, замінивши прогалини запитом.

Передача набору даних в Студію машинного навчання (класичну)

Для завантаження цього файлу в Azure ML Studio натисніть на New в нижній частині сторінки і в панелі, послідовно виберіть Dataset і From Local File. В меню завантаження вкажіть шлях до завантаженого файлу, назва і як тип виберіть CSV.



Upload a new dataset

SELECT THE DATA TO UPLOAD:

o.csv

☐ This is the new version of an existing dataset

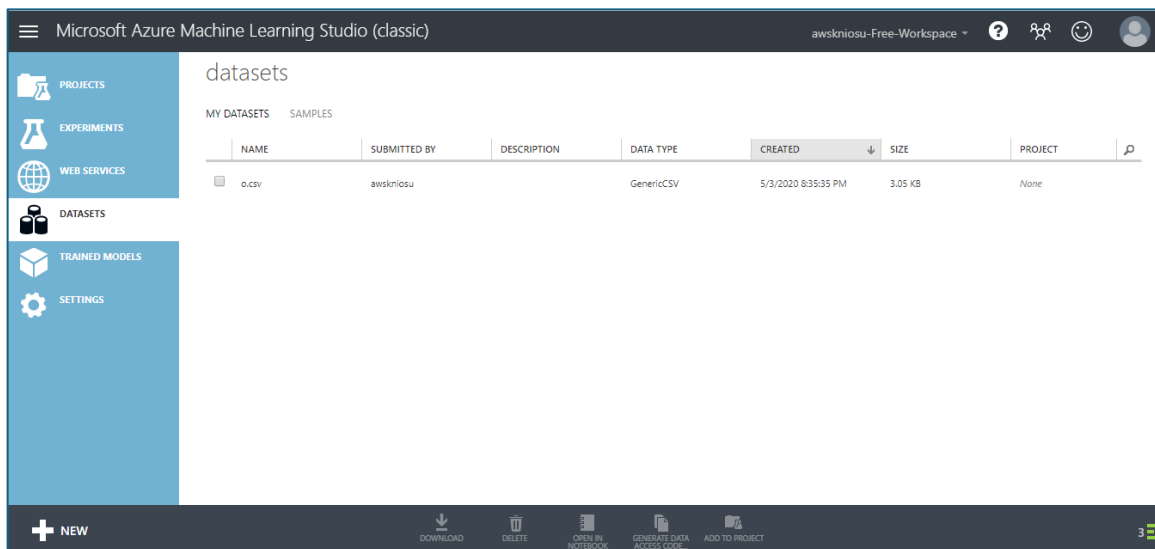
ENTER A NAME FOR THE NEW DATASET:

o.csv

SELECT A TYPE FOR THE NEW DATASET:

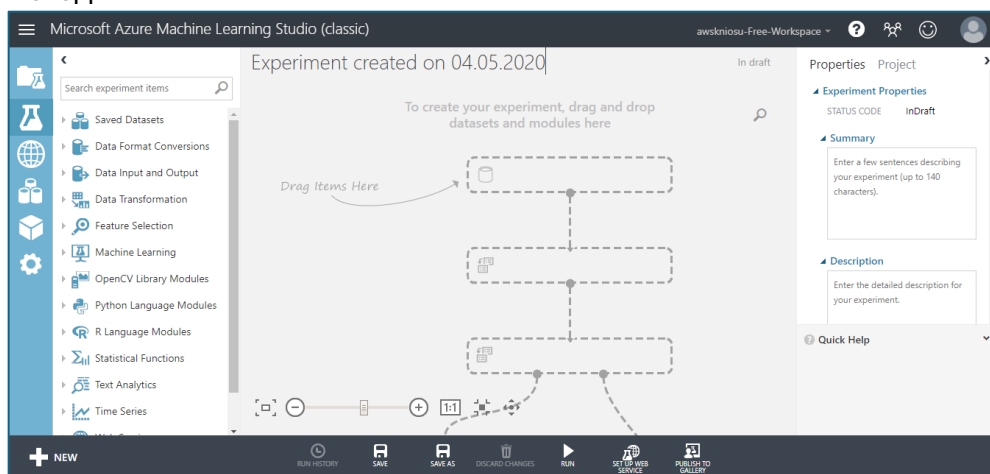
Generic CSV File with a header (.csv)

PROVIDE AN OPTIONAL DESCRIPTION:

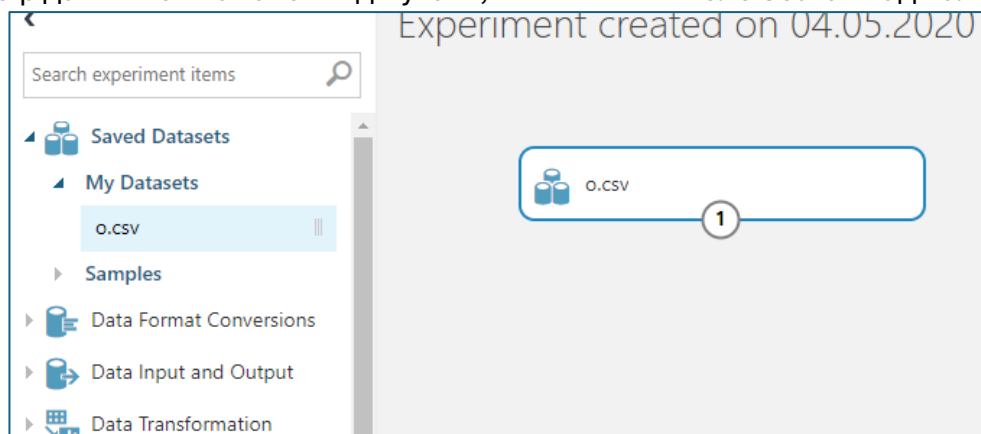


Створення експерименту

Наступний етап є створення в Студії машинного навчання (класичної) експерименту, який використовує набір даних. Для цього у розділі Properties > Experiments оберіть Blank Experiment. Рекомендуємо заповнити поля Summary і Description для експерименту. Ці властивості дозволять вам задокументувати експеримент, щоб будь-хто, хто відкриє його пізніше, зміг зрозуміти ваші цілі і методи.



На палітрі модулів зліва від полотна експерименту розгорніть Saved Datasets. Знайдіть набір даних, створений в розділі My Datasets, і перетягніть його на полотно. Набір даних можна також відшукати, ввівши ім'я в поле Search над палітрою.



Підготовка даних

Ви можете переглянути перші 100 рядків даних і деяку статистичну інформацію про набір даних в цілому. Клацніть вихідний порт набору даних (маленький гурток внизу) і виберіть Visualize.

The screenshot shows the Microsoft Azure Machine Learning Studio interface. On the left, the 'My Datasets' section lists 'o1.csv'. A context menu is open over the dataset, with the 'Visualize' option selected. The main workspace displays the dataset's first 100 rows and 4 columns: Country, SO, VO, and P. To the right, a 'MultiboxPlot' visualization is shown, comparing the 'Country' variable. The plot shows a distribution of values for each country, with the y-axis ranging from 1.0e+7 to 4.0e+7.

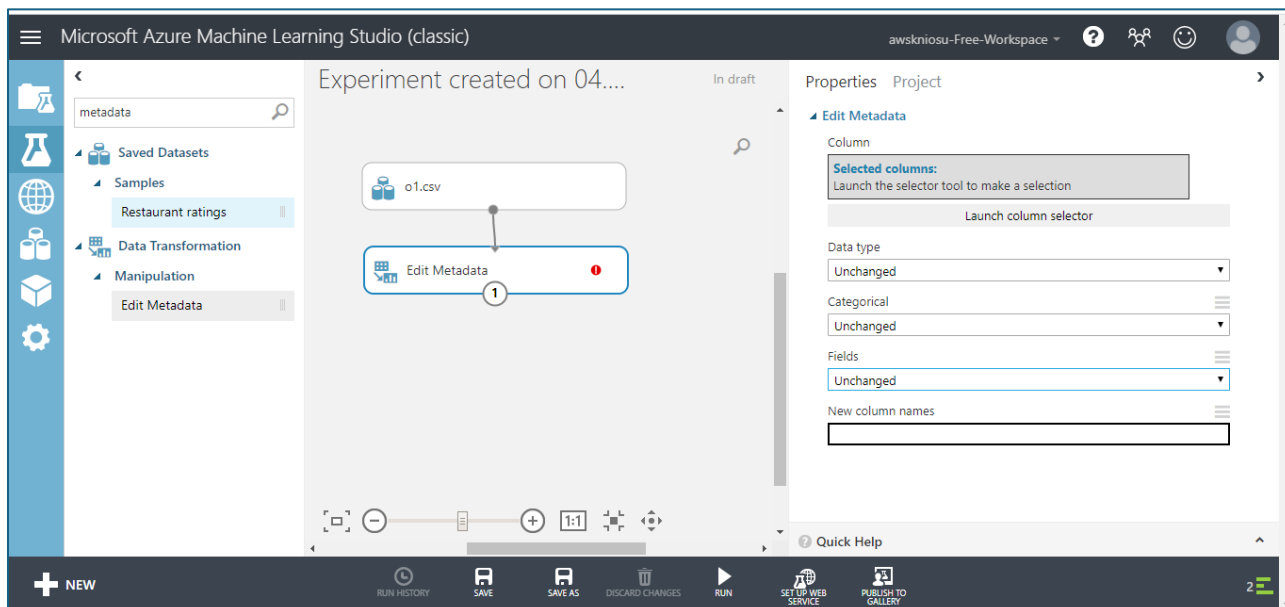
Country	SO	VO	P
Afghanistan	9608795	370610	37172386
Albania	520759	131833	2866376
Algeria	9492542	1600676	42228429
Argentina	11061186	3140963	44494502
Armenia	437612	102891	2951776
Austria	1278170	430370	8847037
Azerbaijan	1783390	200609	9942334
Bahrain	247489	44940	1569439
Bangladesh	36786304	3150539	161356039

Змінити назви заголовків стовпців можна за допомогою модуля Edit Metadata. Модуль Edit Metadata використовується для зміни метаданих, пов'язаних з набором даних.

Для використання модуля Edit Metadata спочатку необхідно вказати, які стовпці будуть змінені. Далі слід вказати дію, яка буде виконана з цими стовпцями. У палітрі модуля введіть "metadata" в полі Пошук. Модуль Edit Metadata з'являється в списку модулів. Перетягніть модуль Edit Metadata на полотно і вставте його під набором даних, доданим раніше.

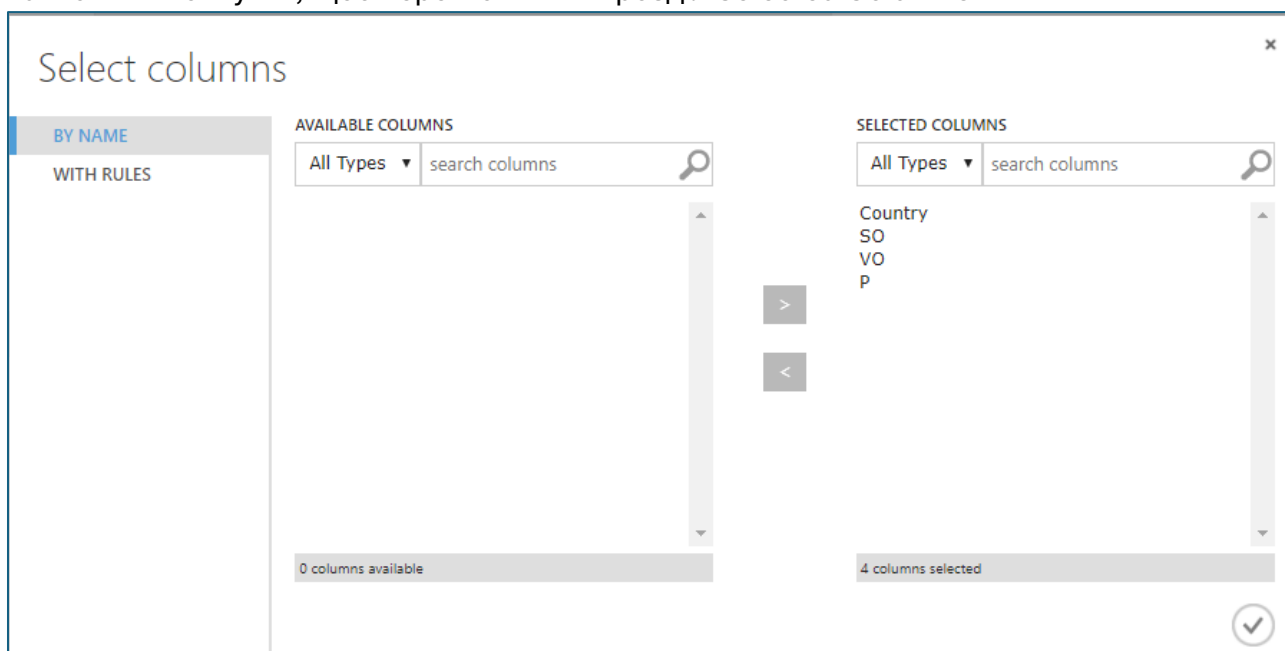
Підключіть набір даних до модуля Edit Metadata. Клацніть порт виведення набору даних (маленький кружок в нижній частині набору даних), перетягніть в порт введення модуля Edit Metadata (маленький кружок у верхній частині модуля) і відпустіть кнопку миші. Набір даних і модуль залишаються підключеними навіть при їх переміщенні на полотні.

Тепер наш експеримент повинен виглядати наступним чином:



Червоний знак оклику вказує на те, що для цього модуля ще не задані властивості. Тому ви зробите це зараз. Двічі клацнувши на модуль можна додати коментар.

Далі в області Properties праворуч від полотна клацніть Launch column selector. У діалоговому вікні Select columns виберіть всі рядки в розділі Available Columns і натисніть кнопку ">", щоб перемістити їх в розділ Selected Columns.



На панелі Properties знайдіть параметр New column names. У цьому полі введіть список імен для стовпчикsd набору даних, розділені комами і в порядку розташування стовпців. Якщо ви хочете перевірити заголовки стовпців, запустіть експеримент (клацніть RUN під полотном експерименту). Після його завершення в модулі Edit Metadata з'явиться зелена галочка, а потім виберіть Visualize.

Edit Metadata

Column

Selected columns:

Column names: Country,SO,VO,P

Launch column selector

Data type

Unchanged

Categorical

Unchanged

Fields

Unchanged

New column names

Country,Serednya_osvita,Vucha_osvita,Naselennya

Experiment created on 04.05.2020 > Edit Metadata > Results dataset

rows 100 columns 4

Country	Serednya_osvita	Vucha_osvita	Naselennya
Afghanistan	9608795	370610	37172386
Albania	520759	131833	2866376
Algeria	9492542	1600676	42228429
Argentina	11061186	3140963	44494502
Armenia	437612	102891	2951776
Austria	1278170	430370	8847037

view as

Statistics

Visualizations

To view, select a column in the table.

Створення навчальних та тестових наборів даних

Далі необхідно дані розділити для навчання моделі і дані для її тестування. Для цього використовується модуль Split Data. За замовчуванням коефіцієнт поділу дорівнює 0,5 і заданий параметр Randomized split. Це означає, що випадково обрана половина даних буде виводитися через один порт модуля Split Data, а інша половина - через інший. Ці параметри можна відрегулювати, а також використовувати параметр Випадкове початкове значення, щоб змінити співвідношення між даними для навчання і тестування.

Microsoft Azure Machine Learning Studio (classic)

aws-kniosk-Free-Workspace

Experiment created on 04...

In draft

Saving...

Properties Project

Split Data

Splitting mode

Split Rows

Fraction of rows in the first output dataset

0.8

☒ Randomized split

Random seed

0

Stratified split

False

o1.csv

Edit Metadata

Зміна назви стовпців

Split Data

1 2

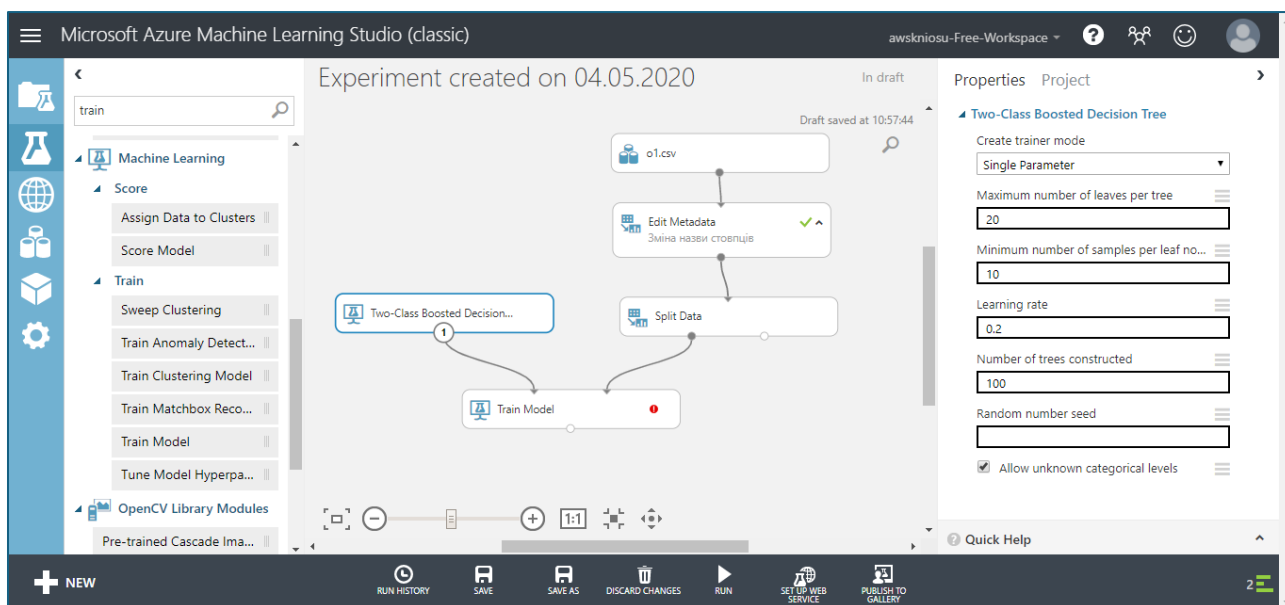
Використовувати вихідні дані модуля Split Data можна на свій розсуд. Але в цьому випадку рекомендується використовувати лівий вихід для навчання даних, а правий - для їх оцінки.

Навчання моделі

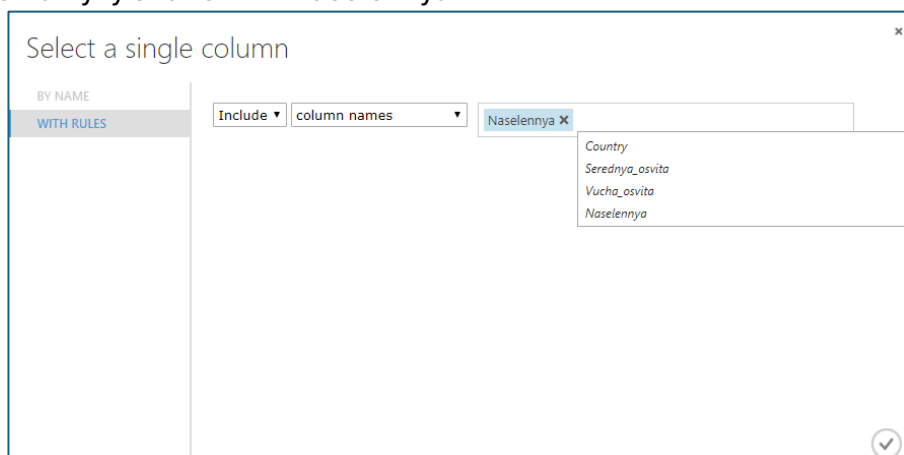
Одна з переваг використання Студії машинного навчання Azure (класичної) для створення моделей машинного навчання полягає в тому, що ви можете перевірити кілька типів моделей і порівняти їх результати в рамках одного експерименту. Цей тип експериментів допомагає знайти найкраще вирішення проблеми.

Існує безліч моделей, які можна вибрати. Щоб переглянути доступні моделі, розгорніть вузол Машинне навчання в палітрі модулів, а потім Initialize Model (Ініціалізація моделі) і вузли під ним. Для цього експерименту необхідно вибрати модулі Two-Class Support Vector Machine (Двокласовий метод опорних векторів) і Two-Class Boosted Decision Tree (Двокласовий метод розширене дерево рішень).

Далі знайдіть модуль Train Model, перетягніть його на полотно, а потім підключіть висновок модуля Two-Class Boosted Decision Tree до лівого порту введення модуля Train Model. З'єднайте лівий вихід лівого модуля Split Data з правим портом введення модуля Train Model.

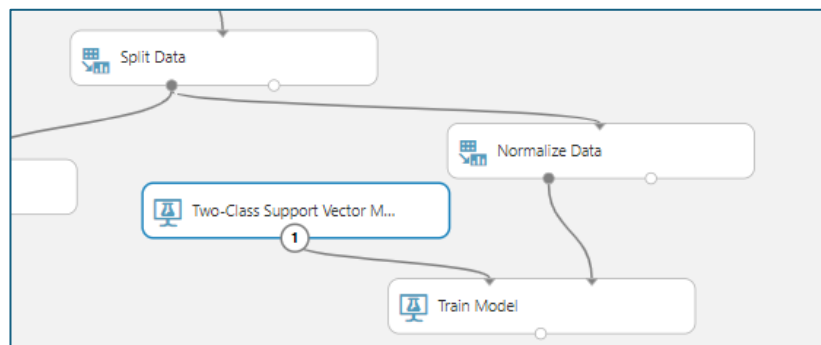


Тепер необхідно повідомити модуль Train Model, що модель повинна прогнозувати значення Naselennya. Виберіть модуль Train Model. В області Properties клацніть Launch column selector. У діалоговому вікні Select a single column введіть в розділі Available Columns в поле пошуку значення "Naselennya".



Тепер налаштуємо модель SVM. Знайдіть модуль Two-Class Support Vector Machine на палітрі модулів і перетягніть його на полотно. Клацніть правою кнопкою миші модуль Train Model, виберіть Копіювати, а потім клацніть полотно правою кнопкою миші і

виберіть Вставити . Зверніть увагу, що копія модуля Train Model має той же набір обраних стовпців, що й оригінал. Знайдіть модуль Normalize Data і перетягніть його на полотно. З'єднайте лівий порт виводу модуля Normalize Data з правим портом введення другого модуля Train Model. Тепер ця частина експерименту повинна виглядати наступним чином.

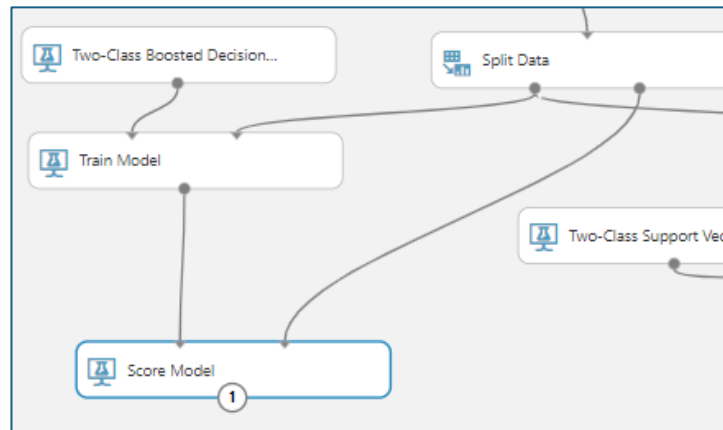


Тепер налаштуйте модуль Normalize Data. На панелі Properties виберіть для параметра Transformation method значення Tanh. Клацніть Launch column selector, вкажіть для параметра Begin With значення No columns. У першому списку виберіть пункт Включення, в другому - column type (тип стовпчика), а в третьому - Числовий . Це вказує, що будуть перетворені всі стовпчики чисел (і тільки числові).

Клацніть знак плюса (+) в правій частині цього рядка. Виберіть в першому списку значення Exclude, у другому списку виберіть column names, а потім прописуємо "Naselennya" в текстовому полі. Так ми повідомимо модулю, що стовпець Naselennya потрібно ігнорувати. В іншому випадку цей стовпець буде перетворений, так як є числовим.

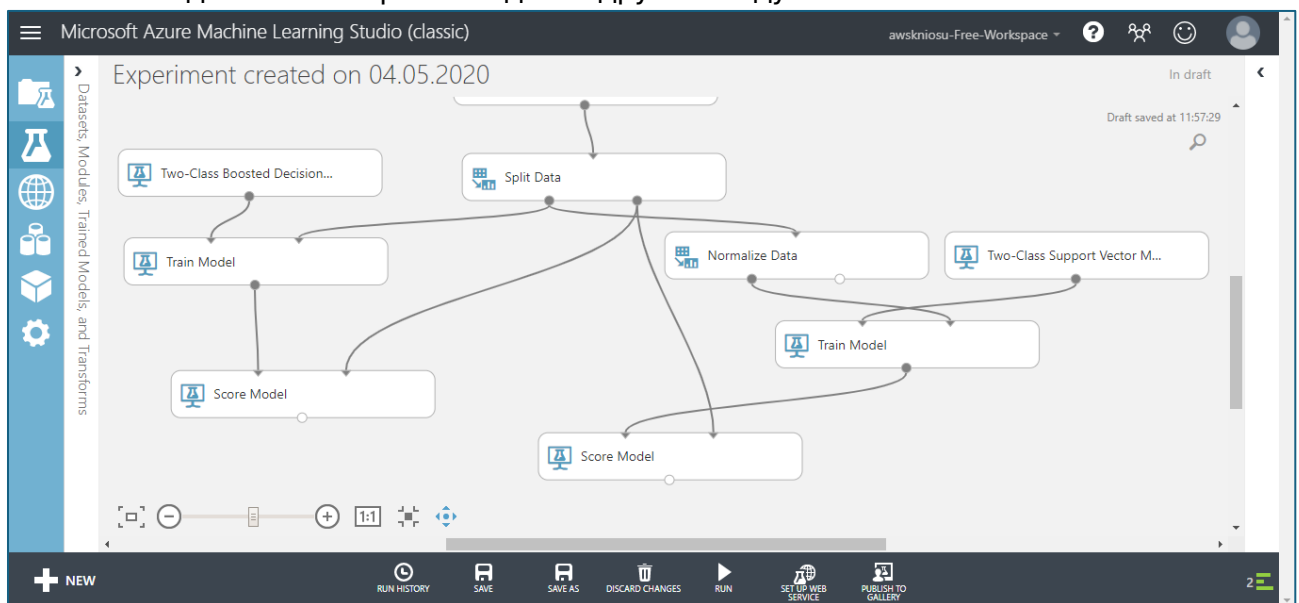
Оцінка і аналіз моделей

Можна порівняти результати двох моделей, щоб визначити, яка з них створює кращий результат за допомогою модулів "Score Model". Підключіть до модуля Train Model, який підключений до модуля Two-Class Boosted Decision Tree, нового модуля Score Model. З'єднайте правий модуль Split Data, який повертає дані для тестування, з правим портом введення модуля Score Model.



Тепер модуль Score Model може отримати відомості про населення з тестових даних, обробити їх за допомогою моделі і порівняти створені моделлю прогнози з фактичними даними стовпчика кредитного ризику з тестових даних.

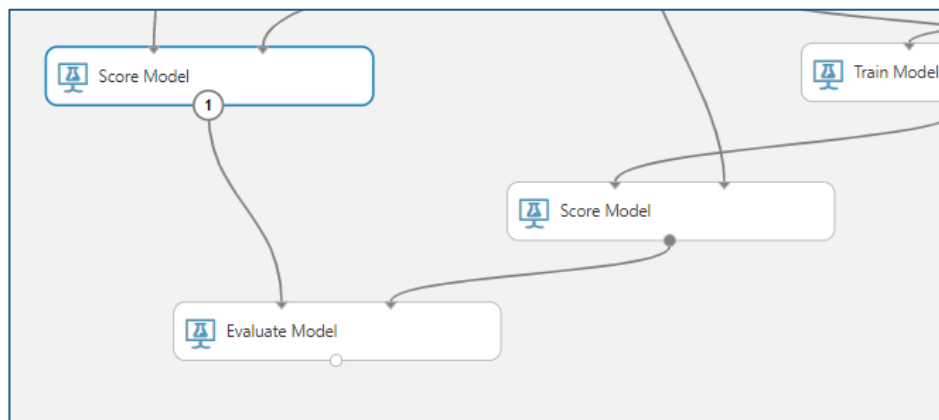
Скопіюйте та вставте модуль Score Model, щоб створити другу копію. З'єднайте висновок моделі SVM з портом введення другого модуля Score Model.



Аналіз моделі

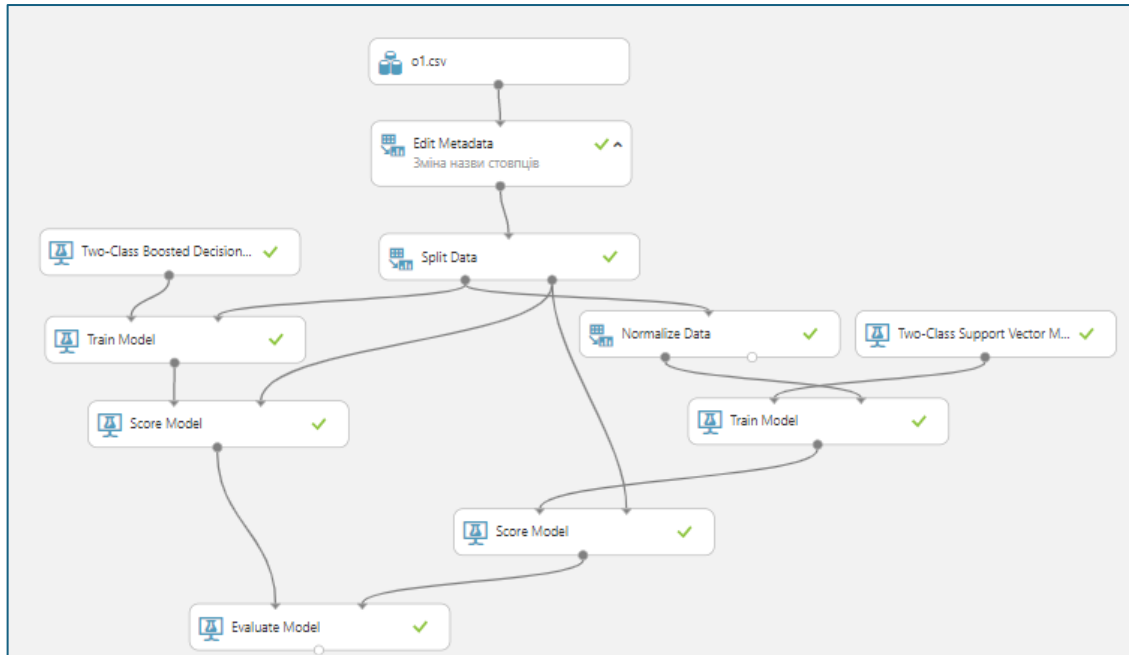
Щоб проаналізувати і порівняти два результату оцінки, використовуйте модуль Evaluate Model.

Знайдіть модуль Evaluate Model і перетягніть його на полотно. Підключіть порт виведення модуля Score Model, пов'язаний з модулем дерева прийняття рішень, до лівого порту введення модуля Evaluate Model. Другий модуль Score Model з'єднайте з правим портом введення.



Запуск експерименту і перевірка результатів

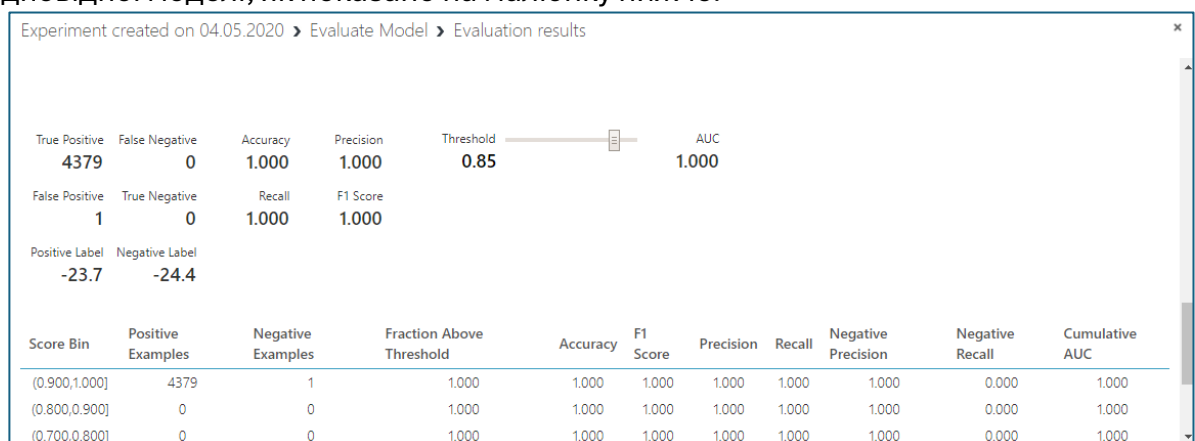
Натисніть кнопку RUN внизу полотна, щоб запустити експеримент. Це може зайняти кілька хвилин. Коли модуль завершить роботу, з'явиться зелений прапорець. Якщо ця галочка з'явилася у всіх модулях, значить експеримент завершив виконання.



Щоб перевірити результати, клацніть правою кнопкою миші порт виводу модуля Evaluate Model і виберіть Visualize.

Модуль Evaluate Model створює пару кривих і метрики, які дозволяють порівняти результати двох оцінюваних моделей. Результати можна бачити у вигляді кривих ROC (робочих характеристик приймача), кривих точності / повноти або кривих точності прогнозу. Додаткові відображаються дані включають матрицю невідповідностей, накопичувальні значення для області під кривою (AUC) та інші метрики. Ви можете змінити порогове значення, переміщаючи повзунок вліво або вправо, щоб подивитися, як це впливає на набір метрик.

У правій частині графіка натисніть Scored dataset або Scored dataset to compare, щоб виділити відповідну криву і відобразити внизу відповідні метрики. В умовних позначеннях для кривих "Scored dataset" відповідає лівому порту введення модуля Evaluate Model. У нашому випадку це модель дерева прийняття рішень. "Оцінений набір даних для порівняння" відповідає правому вхідному порту - в нашому випадку це модель SVM. Клацніть одну з цих міток, щоб виділити криву і відобразити внизу метрики для відповідної моделі, як показано на малюнку нижче.



Розгортання моделі в веб-службі

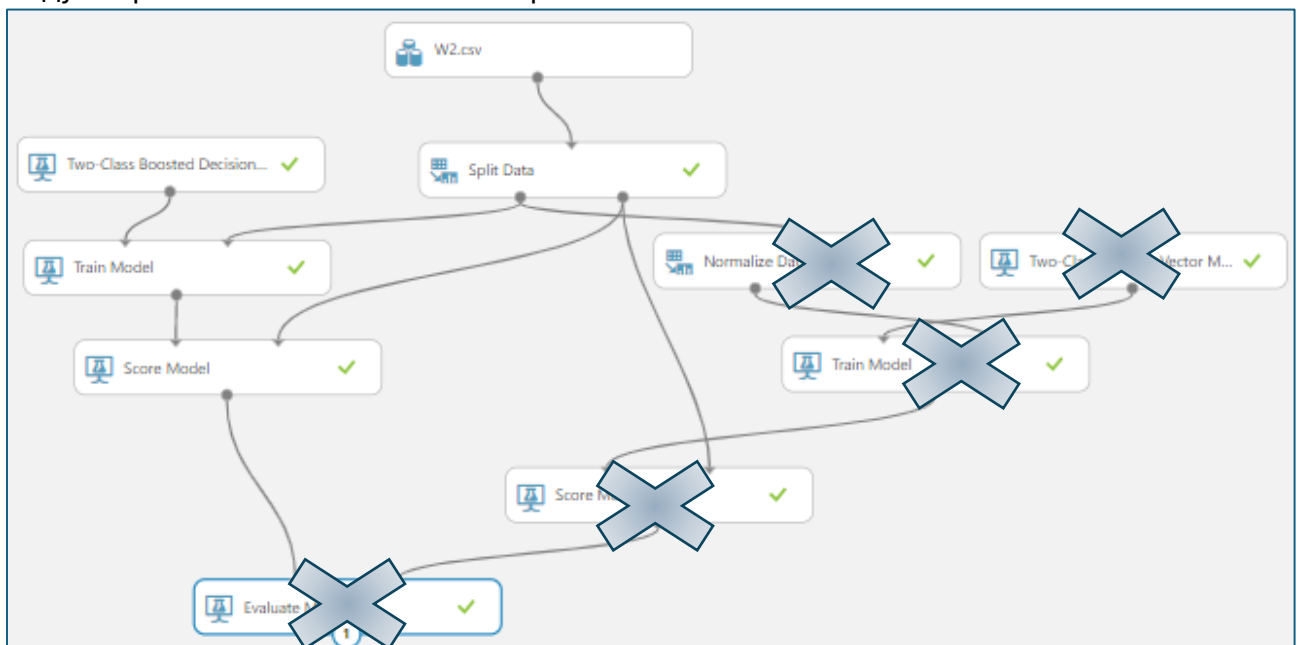
Підготовка до розгортання

Щоб інші користувачі могли застосувати модель прогнозування її можна розгорнути як веб-службу в Azure. Що дасть можливість створювати нові прогнози на основі розробленої моделі, оцінюючи введені користувачем дані. Перш за все необхідно трохи скоротити розроблений експеримент. Зараз в експерименті є дві різні моделі, але для розгортання в якості веб-служби використовується тільки одна.

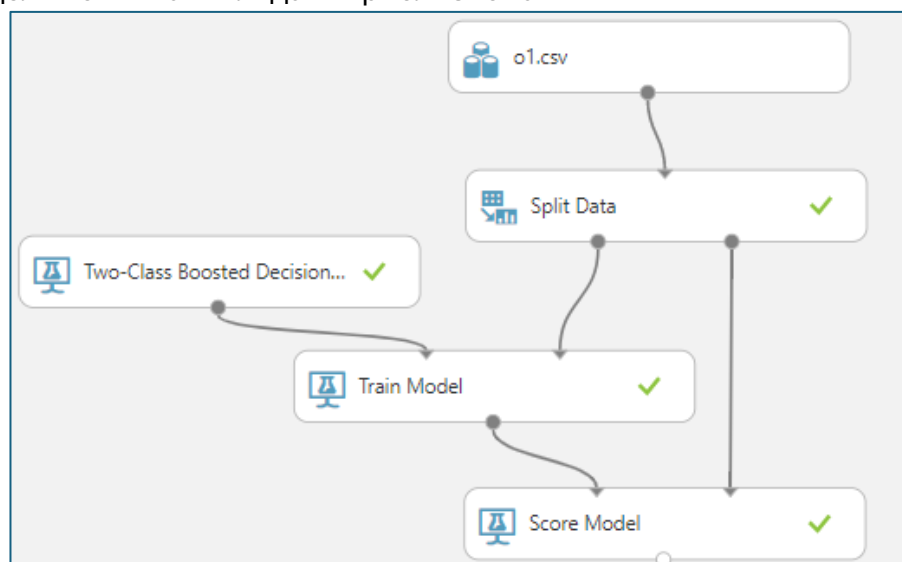
Припустимо, що модель дерева з посиленням підходить для експерименту краще, ніж модель опорних векторів. Тому спочатку ми видалимо модуль Two-Class Support Vector Machine і всі модулі, які використовувалися для його навчання. Спочатку створіть копію експерименту, натиснувши кнопку Зберегти як в нижній частині полотна експерименту.

Видалимо наступні модулі: Two-Class Support Vector Machine, модулі Train Model і Score Model і модуль Normalize Data, Evaluate Model - так як процес аналізу вже завершено.

По черзі виберіть кожен з цих модулів і натисніть клавішу DELETE або клацніть модуль правою кнопкою миші і виберіть Delete.



Тепер модель повинна виглядати приблизно так:

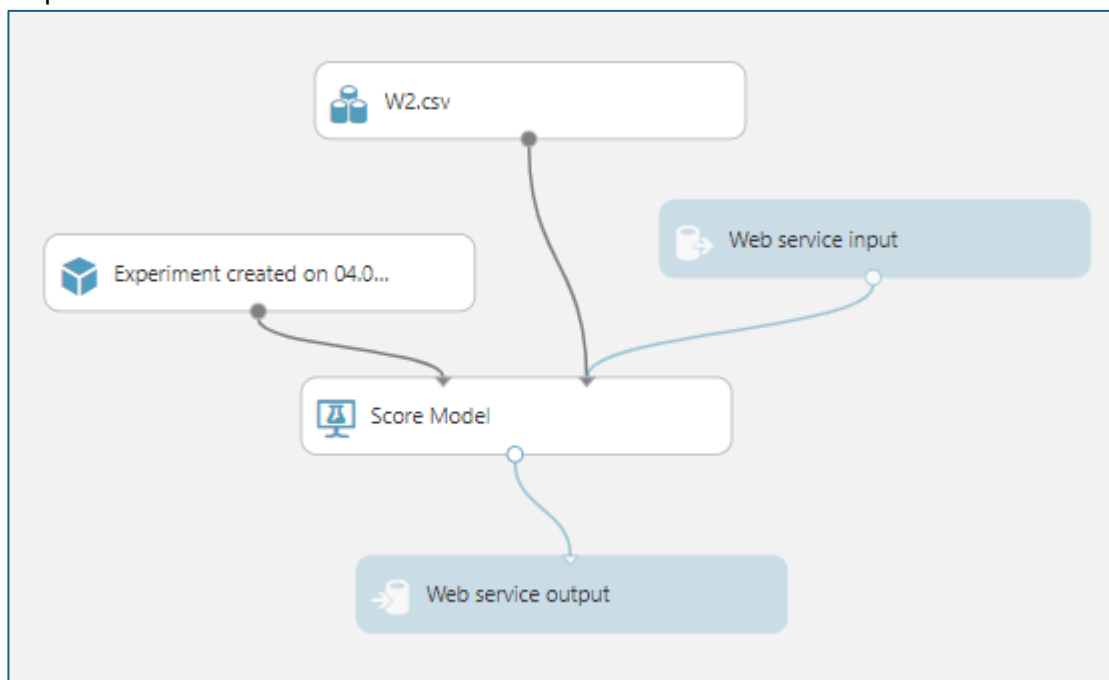


Перетворення навчального експерименту в прогностичний експеримент

Щоб підготувати модель до розгортання, слід перетворити навчальний експеримент в прогностичний. Для цього клацніть в нижній частині полотна експерименту Set Up Web Service і виберіть Predictive Web Service .

При натисканні Set Up Web Service відбувається наступне:

- Навчена модель зберігається як окремий модуль Trained Model і доступна в палітрі модулів зліва від полотна експерименту.
- Видаляються наступні модулі, які ми використовували для навчання:
 - Two-Class Boosted Decision Tree ;
 - Train Model ;
 - Split Data.
- Збережена навчена модель додається назад в експеримент.
- Додаються модулі Web service input і Web service output . Вони визначають, звідки надходять призначені для користувача дані при зверненні до веб-служби і які дані повертаються.



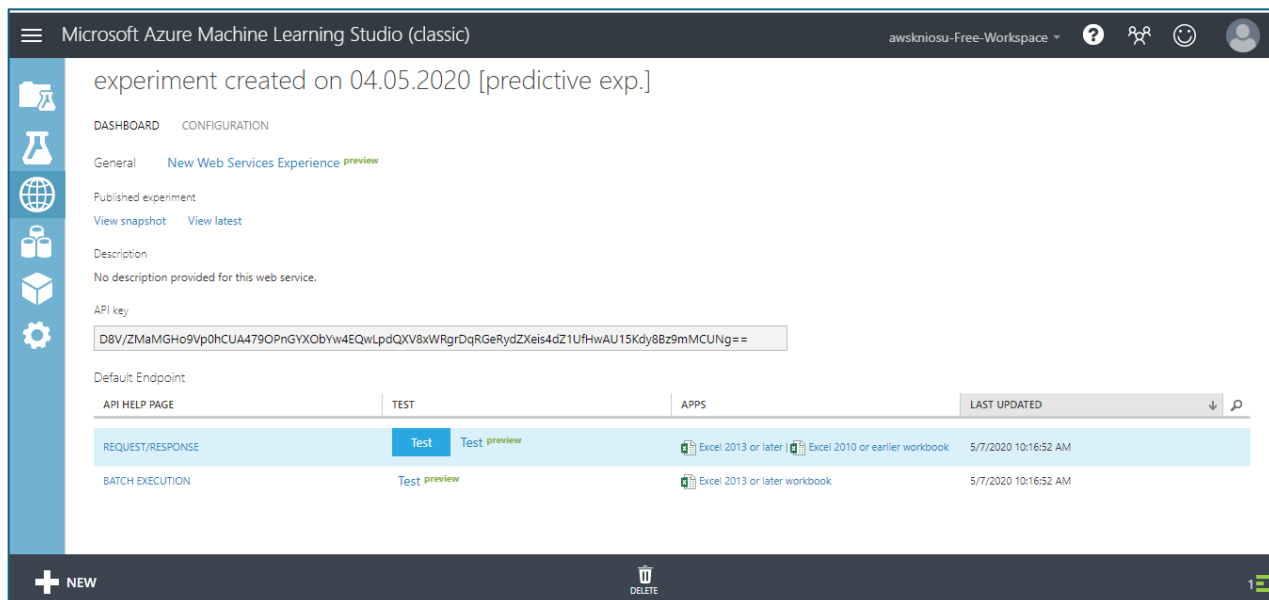
Запустіть експеримент в останній раз (натисніть RUN). Щоб переконатися в тому, що модуль як і раніше працює, клацніть вихідні дані модуля Score Model і виберіть View Results. Ви побачите, що початкові дані відображаються.

Розгортання веб-служби

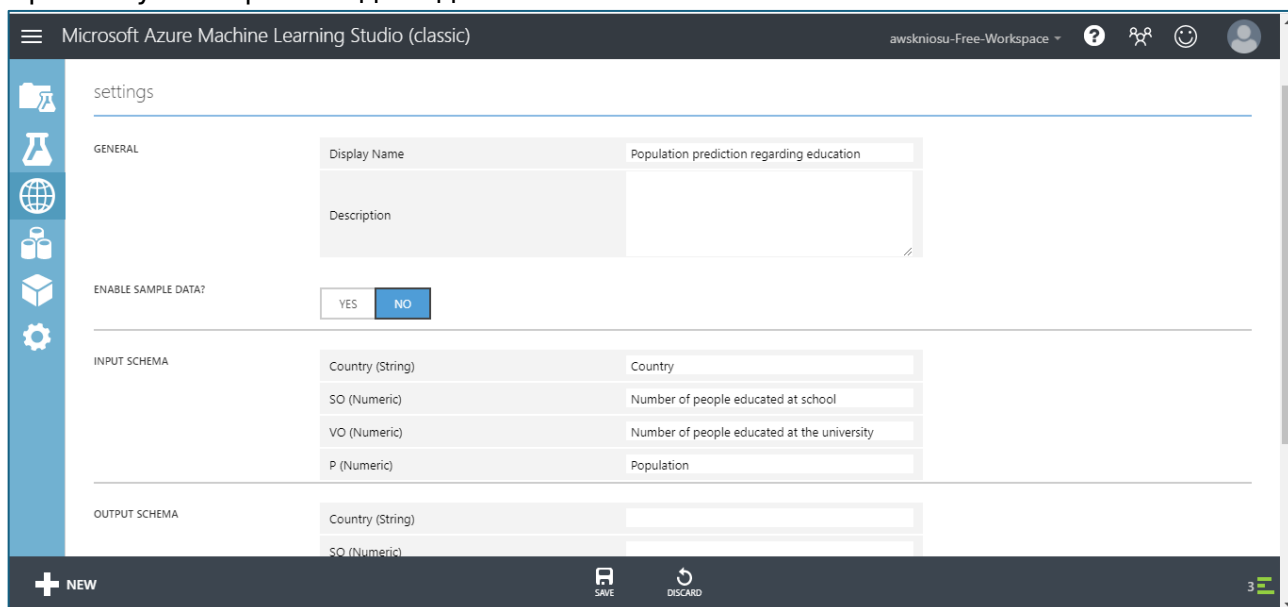
Експеримент можна розгорнути у вигляді класичної веб-служби або нової веб-служби на основі Azure Resource Manager.

Розгортання в вигляді класичної веб-служби

Щоб розгорнути класичну веб-службу на основі нашого експерименту, клацніть Deploy Web Service під полотном і виберіть пункт Deploy Web Service [Classic]. Студія машинного навчання (класична) розгорне експеримент як веб-службу і відкриє панель моніторингу для цієї веб-служби. Звідси можна повернутися до експерименту (View snapshot або View latest) і виконати простий тест веб-служби. На цій сторінці також наведено відомості про створення додатків, які можуть звертатися до веб-служби (докладніше про це в наступному кроці керівництва).



Службу можна налаштовувати на вкладці CONFIGURATION. На цій вкладці можна змінити ім'я служби (за замовчуванням вона отримує ім'я експерименту) і надати її опис. Можна також задати більш легким для читання мітку для вхідних і вихідних даних. Тут і перейменуємо отримане дослідження.



Щоб перевірити роботу веб-служби, клацніть вкладку Test. Щоб отримати відомості про створення додатків, які можуть звертатися до веб-служби, клацніть вкладку Consume.

Тестування веб-служби

При зверненні до веб-служби призначені для користувача дані надходять в модуль Web service input, а звіди передаються для оцінки в модуль Score Model. Прогнозний експеримент налаштований так, щоб модель отримала надходження даних в тому ж форматі, в якому був вихідний набір даних. Результати повертаються з веб-служби користувачеві через модуль Web service output.

Класичну веб-службу можна протестувати в Студії машинного навчання (класичної) або на порталі веб-служб Машинного навчання Azure.

Тестування в Студії машинного навчання (класичної)

На сторінці DASHBOARD натисніть кнопку Test в розділі Default Endpoint. З'явиться діалогове вікно, яке пропонує ввести вхідні дані для служби. Це ті ж стовпчики, які були в вихідному наборі даних про кредитні ризики. Введіть набір даних і натисніть кнопку OK.

Тестування на порталі веб-служб машинного навчання

На сторінці DASHBOARD натисніть кнопку Test preview в розділі Default Endpoint. На порталі веб-служб машинного навчання Azure відкриється сторінка тестування кінцевої точки веб-служби, де буде запропоновано ввести вхідні дані для служби. Це ті ж стовпчики, які були в вихідному наборі даних про кредитні ризики. Клацніть Test Request-Response.

Управління веб-службою

Розгорнутої веб-службою (класичної або нової) можна керувати за допомогою portalу веб-служб Машинного навчання Microsoft Azure .

Ось як можна відстежувати продуктивність веб-служби.

1. Ввійдіть на портал веб-служб Машинного навчання Microsoft Azure.
2. Клацніть Веб-служби .
3. Клацніть потрібну веб-службу.
4. Клацніть Панель моніторингу .

Доступ до веб-служби

На попередньому етапі цього керівництва було розгорнуто веб-службу, яка використовує модель прогнозування кількості населення. Тепер користувачі можуть відправляти в неї дані і отримувати результати.

Це веб-служба Azure, яка може отримувати і повертати дані за допомогою REST API одним з двох способів:

- Запит і відповідь - користувач відправляє одну або кілька рядків даних про кількість населення в службу за допомогою протоколу HTTP, а служба в якості відповіді повертає один або кілька наборів результатів.

- Пакетне виконання - користувач зберігає одну або кілька рядків даних в BLOB-об'єкт Azure, а потім відправляє адреса BLOB-об'єкта в Azure. Служба оцінює всі рядки даних у вхідному BLOB-об'єкті, зберігає результати в іншому BLOB-об'єкті і повертає URL-адресу даного контейнера.

Можна також розробити додатки для доступу до веб-службі за допомогою мов програмування R, C # і Python.

Очищення ресурсів

Якщо не плануєте далі використовувати ресурси, створені при роботі, видаліть їх, щоб не стягувалася плата.

Завдання

1 Оберіть задачу з попередніх лабораторних робіт: Виберіть одну з задач, виконаних у попередніх лабораторних роботах (кластерний аналіз, регресія, дерева рішень або обробка природної мови). Використайте її для створення моделі в середовищі Azure Machine Learning Studio.

2 Створіть і налаштуйте експеримент: Завантажте набір даних у середовище Azure ML Studio та налаштуйте експеримент для навчання обраної моделі. Використовуйте інструменти масштабування даних та оптимізації для підвищення продуктивності моделі.

3 Порівняйте результати: Порівняйте результати моделі, створеної в Azure ML Studio, з результатами попередніх лабораторних робіт. Проаналізуйте, як використання хмарних ресурсів та автоматизованих процесів вплинуло на ефективність і точність моделей.

Оцінювання звіту представлено у Вступі даного збірника.

Додаток А. Перелік варіантів для лабораторних робіт 1-5

№	Назва даних	набору	Посилання на набір даних	Опис
1	Iris Species		https://www.kaggle.com/uciml/iris	Класичний набір даних для класифікації квіток ірису
2	Wine Quality		https://www.kaggle.com/uciml/wine-quality-red-white	Прогнозування якості вина на основі фізико-хімічних властивостей
3	Breast Cancer Wisconsin (Diagnostic)		https://www.kaggle.com/uciml/breast-cancer-wisconsin-data	Класифікація типів раку грудей
4	Titanic: Machine Learning from Disaster		https://www.kaggle.com/c/titanic	Прогнозування виживання на основі даних про пасажирів
5	Heart Disease UCI		https://www.kaggle.com/ronitf/heart-disease-uci	Прогнозування наявності серцевих захворювань
6	House Prices: Advanced Regression Techniques		https://www.kaggle.com/c/house-prices-advanced-regression-techniques	Прогнозування цін на нерухомість
7	Pima Indians Diabetes Database		https://www.kaggle.com/uciml/pima-indians-diabetes-database	Класифікація наявності діабету
8	Boston House Prices		https://www.kaggle.com/c/boston-housing	Прогнозування цін на житло
9	Heart Failure Prediction		https://www.kaggle.com/andrewmvd/heart-failure-clinical-data	Прогнозування ризику серцевої недостатності
10	Diabetes 130-US hospitals		https://www.kaggle.com/uciml/diabetes-130-us-hospitals-for-years-1999-2008	Прогнозування та аналіз діабету
11	Stroke Prediction Dataset		https://www.kaggle.com/fedesoriano/stroke-prediction-dataset	Прогнозування інсульту на основі факторів ризику
12	Student Performance Data		https://www.kaggle.com/aljarah/xAPI-Edu-Data	Прогнозування успішності студентів
13	Credit Card Fraud Detection		https://www.kaggle.com/mlg-ulb/creditcardfraud	Виявлення шахрайських операцій з кредитними картами
14	Adult Income Dataset		https://www.kaggle.com/uciml/adult-census-income	Прогнозування рівня доходу на основі демографічних даних
15	Car Evaluation Dataset		https://www.kaggle.com/uciml/car-evaluation-dataset	Оцінка якості автомобілів на основі різних характеристик
16	Abalone Age Prediction		https://www.kaggle.com/rodolfomendes/abalone-dataset	Прогнозування віку морських молюсків (абалонів)
17	Loan Prediction Dataset		https://www.kaggle.com/altruistdelhite04/loan-prediction-problem-dataset	Прогнозування видачі позик на основі клієнтських даних
18	Marketing Campaign Data		https://www.kaggle.com/rikdifos/marketing-data	Аналіз поведінки клієнтів у маркетингових кампаніях
19	World Happiness Report		https://www.kaggle.com/unsdsn/world-happiness	Аналіз індексу щастя країн
20	Human Activity Recognition Using Smartphones		https://www.kaggle.com/uciml/human-activity-recognition-with-smartphones	Класифікація видів активностей людей за даними зі смартфонів
21	Bank Marketing Data		https://www.kaggle.com/uciml/bank-marketing	Прогнозування ефективності банківських маркетингових кампаній

22	California Housing Prices	https://www.kaggle.com/camnugent/california-housing-prices	Прогнозування цін на нерухомість в Каліфорнії
23	Seoul Bike Sharing Demand	https://www.kaggle.com/soonhyeok/seoul-bike-sharing-demand-prediction	Прогнозування попиту на прокат велосипедів у Сеулі
24	Forest Cover Type Prediction	https://www.kaggle.com/uciml/forest-cover-type-dataset	Прогнозування типів покриття лісу на основі географічних даних
25	Flight Delay Prediction	https://www.kaggle.com/usdot/flight-delays	Прогнозування затримок рейсів
26	NFL Big Data Bowl	https://www.kaggle.com/c/nfl-big-data-bowl-2021	Аналіз спортивних даних і прогнозування в американському футболі
27	Weather Dataset	https://www.kaggle.com/jsphyg/weather-dataset-rattle-package	Прогнозування погодних умов на основі історичних даних
28	Air Quality Data	https://www.kaggle.com/rohanrao/air-quality-data-in-india	Аналіз рівня забруднення повітря в різних регіонах Індії
29	Housing Prices in King County, USA	https://www.kaggle.com/harlfoxem/housesalesprediction	Прогнозування цін на житло в окрузі Кінг, США
30	Forest Fires	https://www.kaggle.com/elikplim/forest-fires-data-set	Прогнозування площі лісових пожеж на основі погодних умов

Додаток Б. Перелік варіантів для лабораторної роботи 6

№	Назва набору даних	Посилання на набір даних	Опис
1	CIFAR-10	https://www.kaggle.com/c/cifar-10	Набір даних для класифікації зображень 10 різних класів (тварини, транспорт, тощо).
2	Fashion MNIST	https://www.kaggle.com/zalando-research/fashionmnist	Набір даних з зображеннями одягу для класифікації різних типів (сукні, взуття, тощо).
3	MNIST (Handwritten Digits)	https://www.kaggle.com/oddrational/mnist-in-csv	Класичний набір рукописних цифр для класифікації.
4	Dogs vs Cats	https://www.kaggle.com/c/dogs-vs-cats	Задача класифікації зображень собак та котів.
5	Intel Image Classification	https://www.kaggle.com/puneet6060/intel-image-classification	Зображення місцевості для класифікації.
6	Chest Images (Pneumonia)	https://www.kaggle.com/paultimothy-mooney/chest-xray-pneumonia	Набір даних рентгенівських знімків для виявлення пневмонії.
7	Caltech Birds 200	https://www.kaggle.com/veeralakrishna/caltech-birds-2011-dataset	Набір зображень 200 видів птахів для класифікації.
8	Plant Pathology 2020	https://www.kaggle.com/c/plant-pathology-2020-fgvc7	Зображення листя для визначення хвороб рослин.
9	Lung and Colon Cancer Histopathological Images	https://www.kaggle.com/andrewmvd/ung-and-colon-cancer-histopathological-images	Набір даних гістопатологічних зображень для виявлення раку легень та кишківника.
10	Brain Tumor MRI Dataset	https://www.kaggle.com/ahmedhama-da0/brain-tumor-detection	Зображення МРТ для виявлення пухлин мозку.
11	Skin Cancer MNIST: HAM10000	https://www.kaggle.com/kmader/skin-cancer-mnist-ham10000	Набір зображень для виявлення раку шкіри.
12	Face Mask Detection	https://www.kaggle.com/andrewmvd/face-mask-detection	Набір зображень для класифікації людей з/без масок.
13	Car Damage Dataset	https://www.kaggle.com/pauljeffress/car-damage-detection	Набір зображень автомобілів для визначення ступеню пошкоджень.
14	Retinal Images OCT	https://www.kaggle.com/paultimothy-mooney/kermany2018	Зображення очної сітківки для діагностики захворювань.
15	ASL Alphabet	https://www.kaggle.com/grassknotted/asl-alphabet	Зображення для розпізнавання алфавіту американської жестової мови.
16	Aerial Cactus Identification	https://www.kaggle.com/c/aerial-cactus-identification	Зображення кактусів для ідентифікації на супутникових знімках.
17	Cat and Dog Breeds Classification	https://www.kaggle.com/qihiro/cat-and-dog	Зображення різних порід собак і котів для класифікації.
18	COVID-19 Radiography Database	https://www.kaggle.com/tawsifurrahman/covid19-radiography-database	Рентген-зображення для діагностики COVID-19.
19	Fashion Product Images Dataset	https://www.kaggle.com/paramaggarwal/fashion-product-images-dataset	Зображення модних товарів для класифікації.
20	Flowers Recognition	https://www.kaggle.com/alxmamaev/flowers-recognition	Зображення квітів для класифікації.

21	Cassava Disease Classification	Leaf	https://www.kaggle.com/c/cassava-leaf-disease-classification	Зображення листя касави для визначення хвороб.
22	Fruits 360		https://www.kaggle.com/moltean/fruits	Зображення різних фруктів для класифікації.
23	Malaria Images Dataset	Cell	https://www.kaggle.com/iarunava/cell-images-for-detecting-malaria	Зображення клітин для виявлення малярії.
24	Pokemon Image Dataset	Image	https://www.kaggle.com/kvpratama/pokemon-images-dataset	Зображення покемонів для класифікації.
25	Digital Petroglyph Dataset	Petroglyph	https://www.kaggle.com/andrewmvd/digital-petroglyph-dataset	Зображення петрогліфів для ідентифікації.
26	Leaf Classification		https://www.kaggle.com/c/leaf-classification	Зображення листя для класифікації видів рослин.
27	Food-101		https://www.kaggle.com/dansbecker/food-101	Зображення страв для класифікації різних типів їжі.
28	Art Images Dataset		https://www.kaggle.com/c/art-images-dataset	Зображення художніх творів для класифікації стилів та авторів.
29	LFW People Faces		https://www.kaggle.com/jessicali9530/lfw-dataset	Зображення облич для задач розпізнавання осіб.
30	Vehicles Dataset		https://www.kaggle.com/tunguz/vehicles-dataset	Зображення різних транспортних засобів для класифікації.

Додаток В Перелік варіантів для лабораторної роботи 7

№	Назва даних	набору	Посилання на набір даних	Опис
1	Sentiment140		https://www.kaggle.com/kazanova/sentiment140	Твіттер-дані з мітками настроїв для аналізу настроїв.
2	Amazon Unlocked Phones	Reviews: Mobile	https://www.kaggle.com/PromptCloudHQ/amazon-reviews-unlocked-mobile-phones	Відгуки на мобільні телефони з оцінками для аналізу настроїв.
3	News Dataset	Category	https://www.kaggle.com/rmisra/news-category-dataset	Тексти новин для класифікації за категоріями.
4	Yelp Reviews		https://www.kaggle.com/yelp-dataset/yelp-dataset	Відгуки з Yelp для аналізу настроїв і класифікації.
5	IMDB Movie Reviews		https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews	Рецензії на фільми з мітками настроїв.
6	Twitter Sentiment	US Airline	https://www.kaggle.com/crowdflower/twitter-airline-sentiment	Твіттер-дані для аналізу настроїв стосовно авіаліній.
7	Wikipedia Plots	Movie	https://www.kaggle.com/jrobischon/wikipedia-movie-plots	Тексти сюжетів фільмів з Wikipedia для аналізу текстів.
8	SMS Collection	Spam	https://www.kaggle.com/uciml/sms-spam-collection-dataset	SMS-повідомлення для класифікації спаму і не спаму.
9	SQuAD 2.0		https://www.kaggle.com/rajpurkar/squad2	Запитання та відповіді для задач пошуку відповідей у текстах.
10	Quora Pairs	Question	https://www.kaggle.com/quora/question-pairs-dataset	Питання з Quora для визначення подібності між питаннями.
11	MovieLens Dataset	25M	https://www.kaggle.com/rounakbanik/the-movies-dataset	Дані про фільми для рекомендаційних систем і аналізу.
12	TED Talks Transcripts		https://www.kaggle.com/rounakbanik/ted-talks	Транскрипти виступів TED для аналізу текстів і настроїв.
13	Covid19 Sentiment Analysis	Tweet	https://www.kaggle.com/datatattle/covid-19-nlp-text-classification	Твіттер-дані для аналізу настроїв, пов'язаних з COVID-19.
14	All The News		https://www.kaggle.com/snapcrack/all-the-news	Новини з різних джерел для класифікації і аналізу текстів.
15	Global Database	Terrorism	https://www.kaggle.com/START-UMD/gtd	Тексти про тероризм для аналізу і класифікації.
16	The Guardian Headlines		https://www.kaggle.com/rmisra/the-guardian-headlines-dataset	Заголовки новин з The Guardian для класифікації і аналізу.
17	Reddit Comments		https://www.kaggle.com/reddit/reddit-comments-may-2015	Дані з Reddit для аналізу коментарів і настроїв.
18	Enron Email Dataset		https://www.kaggle.com/wcukierski/enron-email-dataset	Листи з Enron для аналізу текстів і виявлення шахрайства.
19	Hate Detection	Speech	https://www.kaggle.com/mrmorj/hate-speech-and-offensive-language-dataset	Тексти для виявлення мовлення ненависті.
20	BBC News Summary		https://www.kaggle.com/pariza/bbc-news-summary	Тексти новин BBC для аналізу текстів і класифікації.
21	Amazon Fine Food Reviews		https://www.kaggle.com/snap/amazon-fine-food-reviews	Відгуки на продукти з Amazon для аналізу настроїв.
22	Spam Detection on YouTube Comments		https://www.kaggle.com/datasnaek/youtube-new	Коментарі на YouTube для класифікації спаму.
23	Trump Tweets		https://www.kaggle.com/aashita/nyt-comments	Твіттер-дані Дональда Трампа для аналізу настроїв.
24	Russian Troll Tweets		https://www.kaggle.com/fivethirtyeight/russian-troll-tweets	Твіттер-дані для аналізу настроїв і класифікації тролінгу.

2 5	Wikipedia Comments	Toxic	https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge	Коментарі з Wikipedia для класифікації токсичності.
2 6	Customer Sentiment Analysis	Reviews	https://www.kaggle.com/lakshmi25npathi/customer-reviews-on-products	Відгуки клієнтів для аналізу настроїв.
2 7	Election Sentiment Analysis	Tweets	https://www.kaggle.com/manchunhui/us-election-2020-tweets	Твіттер-дані про вибори для аналізу настроїв.
2 8	Amazon Reviews	Alexa	https://www.kaggle.com/sid321axn/amazon-alexa-reviews	Відгуки на Alexa для аналізу настроїв.
2 9	Game of Thrones Scripts		https://www.kaggle.com/albenft/game-of-thrones-script-all-seasons	Транскрипти серіалу «Гра престолів» для аналізу персонажів і настроїв.
3 0	NY Times Comments		https://www.kaggle.com/aashita/nyt-comments	Коментарі з NY Times для аналізу настроїв і класифікації.

Література

1. Abu-Mostafa, Y. S., Magdon-Ismael, M., & Lin, H. T. (2012). *Learning from data* (Vol. 4). New York, NY, USA:: AMLBook.
2. Alpaydin, E. (2020). *Introduction to machine learning*. MIT press.
3. Baldi, P., Brunak, S., & Bach, F. (2001). *Bioinformatics: the machine learning approach*. MIT press.
4. Barber, D. (2012). *Bayesian reasoning and machine learning*. Cambridge University Press.
5. Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.
6. Harrington, P. (2012). *Machine learning in action*. Manning Publications Co..
7. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media.
8. Langley, P. (1996). *Elements of machine learning*. Morgan Kaufmann.
9. Manning, C. D., Manning, C. D., & Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT press.
10. Marsland, S. (2015). *Machine learning: an algorithmic perspective*. CRC press.
11. Mitchell, T. M. (1997). *Machine learning*.
12. Nilsson, N. J. (1996). *Introduction to machine learning: An early draft of a proposed textbook*.
13. Quinlan, J. R. (2014). *C4. 5: programs for machine learning*. Elsevier.
14. Rashid, T. (2016). *Make your own neural network*. CreateSpace Independent Publishing Platform.
15. Russell, S., & Norvig, P. (2002). *Artificial intelligence: a modern approach*.