

ЛАБОРАТОРНА РОБОТА № 5

ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ

Мета роботи: використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити методи ансамблів у машинному навчанні.

1. ТЕОРЕТИЧНІ ВІДОМОСТІ

Теоретичні відомості подані на лекціях. Також доцільно вивчити матеріал поданий в літературі:

Джоши Пратик. Искусственный интеллект с примерами на Python. : Пер. с англ. - СПб. : ООО "Диалектика", 2019. - 448 с. - Парал. тит. англ. ISBN 978-5-907114-41-8 (рус.)

Можна використовувати Google Colab або Jupiter Notebook.

Термін ансамблеве навчання (ensemble learning) відноситься до процесу побудови множини моделей та пошуку такої їх комбінації, яка дозволяє отримати кращі результати ніж кожна з моделей окремо. У якості індивідуальних моделей можуть виступати класифікатори, регресори і інші об'єкти, що дозволяють моделювати дані тим чи іншим способом.

Ансамблеве навчання застосовується в багатьох сферах, наприклад, прогностичній класифікації, виявлення аномалій та інше.

Чому застосовується ансамблеве навчання? Щоб це зрозуміти, звернемося до реального прикладу. Припустимо, ви хочете купити новий телевізор, але про останні моделі вам нічого не відомо.

Ваше завдання - купити найкращий телевізор із тих, які пропонуються за доступною для вас ціною, але ви недостатньо добре знаєте ринок, щоб зробити обґрунтований вибір. У подібних випадках ви цікавитесь думкою кількох експертів у цій галузі. Так вам легше прийняти найвірніше рішення. У більшості випадків ви не прив'язуватиметеся до думки якогось одного фахівця і приймете остаточне рішення на основі узагальнення оцінок, зроблених різними людьми. Ми робимо так, тому що прагнемо звести до мінімуму ймовірність прийняття невірних чи недостатньо оптимальних рішень.

При виборі моделі найчастіше виходять з того, щоб вона призводила до найменших помилок на тренувальному наборі даних. Проблема полягає в тому, що такий підхід не завжди працює через можливий ефект перенавчання. Навіть якщо перехресна перевірка моделі підтверджує її адекватність вона може призводити до незадовільних результатів для невідомих даних.

Однією з основних причин ефективності ансамблевого навчання є те, що цей метод дозволяє знизити загальний ризик вибору невдалої моделі. Завдяки тому, що тренування здійснюється на широкій різноманітності навчальних наборів даних, ансамблевий підхід дозволяє отримувати непогані результати для невідомих даних. Якщо ми створюємо модель на основі ансамблевого навчання, то результати, отримані з використанням індивідуальних моделей, повинні виявляти певний розкид. Це дозволяє вловлювати всілякі нюанси, в результаті чого узагальнена модель виявляється більш точною. Зазначене розмаїття результатів досягається за рахунок використання різних навчальних параметрів для індивідуальних моделей, завдяки чому вони генерують різні межі рішень для тренувальних даних. Це означає, що кожна модель буде використовувати різні правила для логічного висновку, тим самим забезпечуючи більш ефективний спосіб валідації кінцевого результату. Якщо між моделями спостерігається узгодженість, то це означає, що модель коректна.

Випадкові та гранично випадкові ліси

Випадковий ліс (random forest) - окремий випадок ансамблевого навчання у якому індивідуальні моделі конструюються з використанням дерев рішень. Отриманий ансамбль використовується в подальшому для прогнозування результатів. При конструюванні окремих дерев використовують випадкові підмножини тренувальних даних. Це гарантує розкид даних між різними деревами рішень. Як зазначалося вище, в ансамблевому навчанні дуже важливо забезпечити різноманітність ансамблю індивідуальних моделей.

Однією з найбільших переваг випадкових лісів є те, що вони не перенавчаються. Як ви вже знаєте, у машинному навчанні ця проблема трапляється досить часто. Конструюючи неоднорідну множину дерев рішень за рахунок використання різних випадкових підмножин, ми гарантуємо відсутність перенавчання моделі на тренувальних даних.

У процесі конструювання дерева рішень його вузли послідовно розщеплюються, і їм вибираються найкращі порогові значення, що знижують ентропію кожному рівні. У процесі розщеплення вузлів враховуються в повному обсязі ознаки, що характеризують дані вхідного набору.

Натомість вибирається найкращий спосіб розщеплення вузлів, заснований на поточному випадковому піднаборі ознак, що розглядаються. Включення фактора випадковості збільшує зміщення випадкового лісу, проте дисперсія зменшується завдяки усередненню. Це обумовлює робастність (стійкість до відхилень) результуючої моделі.

Гранично випадкові ліси (extremely random forests) ще більше посилюють роль фактора випадковості. Поряд із випадковим вибором ознак випадково вибираються також граничні значення. Ці випадково генеровані значення стають правилами розбиття, що додатково зменшують варіативність моделі. Тому використання гранично випадкових лісів зазвичай призводить до більш гладких меж прийняття рішень у порівнянні з тими, які вдається отримати за допомогою випадкових лісів.

2. ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ ТА МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ЙОГО ВИКОНАННЯ

Завдання 2.1. Створення класифікаторів на основі випадкових та гранично випадкових лісів

Використовувати файл вхідних даних: data_random_forests.txt, побудувати класифікатори на основі випадкових та гранично випадкових лісів.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Способи створення класифікаторів обох типів дуже схожі, тому для вказівки того, який класифікатор створюється, ми будемо використовувати вхідний прапор.

Створіть новий файл Python та імпортуйте такі пакети.

```
import argparse
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn import cross_validation
from sklearn.ensemble import RandomForestClassifier,
ExtraTreesClassifier
from sklearn import cross_validation
from sklearn.metrics import classification_report
from utilities import visualize_classifier
```

Визначимо синтаксичний аналізатор (парсер) аргументів для Python, щоб можна було приймати тип класифікатора як вхідний параметр. Задаючи відповідне значення цього параметра, ми зможемо вибрати тип класифікатора, що створюється.

```
# Парсер аргументів
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Classify \
        data using Ensemble Learning techniques')
    parser.add_argument('--classifier-type',
        dest='classifier_type', required=True,
        choices=['rf', 'erf'], help="Type of \
        classifier to use; can be either 'rf' or \
        'erf'")
    return parser
```

Визначимо основну функцію та вилучимо вхідні аргументи.

```

if __name__ == '__main__':
    # Вилучення вхідних аргументів
    args = build_arg_parser().parse_args()
    classifier_type = args.classifier_type

```

У файлі data_random_forests.txt кожен рядок містить значення розділені комою. Перші два значення відповідають вхідним даним, останнє – цільовій мітці. У цьому наборі даних містяться три різні класи. Завантажимо дані із цього файлу.

```

# Завантаження вхідних даних
input_file = 'data_random_forests.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]

```

Розіб'ємо вхідні дані на три класи.

```

# Розбиття вхідних даних на три класи
class_0 = np.array(X[y==0])
class_1 = np.array(X[y==1])
class_2 = np.array(X[y==2])

```

Візуалізуємо вхідні дані.

```

# Візуалізація вхідних даних
plt.figure()
plt.scatter(class_0[:, 0], class_0[:, 1], s=75,
            facecolors='white', edgecolors='black',
            linewidth=1, marker='s')
plt.scatter(class_1[:, 0], class_1[:, 1], s=75,
            facecolors='white', edgecolors='black',
            linewidth=1, marker='o')
plt.scatter(class_2[:, 0], class_2[:, 1], s=75,
            facecolors='white', edgecolors='black',
            linewidth=1, marker='^')
plt.title('Входные данные')

```

Розіб'ємо дані на навчальний та тестовий набори.

```

# Розбивка даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test =
    cross_validation.train_test_split(
        X, y, test_size=0.25, random_state=5)

```

Визначимо параметри, які використовуватимемо при конструюванні класифікатора. Параметр `n_estimator` - це кількість дерев. Параметр `max_depth` – це максимальна кількість рівнів у кожному дереві. Параметр `random_state` - це початкове значення для генератора випадкових чисел, необхідне ініціалізації алгоритму класифікатора з урахуванням випадкового лісу.

```
# Класифікатор на основі ансамблевого навчання
params = {'n_estimators': 100, 'max_depth': 4,
          'random_state': 0}
```

Залежно від того, яке значення вхідного параметра ми надали, класифікатор конструюється на основі випадкового або гранично випадкового лісу.

```
if classifier_type == 'rf':
    classifier = RandomForestClassifier(**params)
else:
    classifier = ExtraTreesClassifier(**params)
```

Навчимо та візуалізуємо класифікатор.

```
classifier.fit(X_train, y_train)
visualize_classifier(classifier, X_train, y_train,
                    'Training dataset')
```

Обчислимо результат на тестовому наборі даних та візуалізуємо його.

```
y_test_pred = classifier.predict(X_test)
visualize_classifier(classifier, X_test, y_test,
                    'Тестовый набор данных')
```

Перевіримо, як працює класифікатор, вивівши звіт із результатами класифікації.

```
# Перевірка роботи класифікатора
```

```

class_names = ['Class-0', 'Class-1', 'Class-2']
print("\n" + "#" * 40)
print("\nClassifier performance on training dataset\n")
print(classification_report(y_train,
                           classifier.predict(X_train), target_names=class_names))
print("#" * 40 + "\n")

print("#" * 40)
print("\nClassifier performance on test dataset\n")
print(classification_report(y_test, y_test_pred,
                           target_names=class_names))
print("#" * 40 + "\n")

```

Виконайте цей код, запросивши створення класифікатора на основі випадкового лісу за допомогою прапорця **rf** вхідного аргументу. Введіть у вікні терміналу наступну команду:

```
$ python3 random_forests.py --classifier-type rf
```

У процесі виконання цього коду отримайте **ряд зображень та занесіть їх у звіт**.

Графік вхідних даних. На графіку квадрати, кола та трикутники представляють три класи. Оцініть візуально, що класи значною мірою перекриваються, проте на цьому етапі це нормально. **Графік занесіть у звіт**.

Зображення на якому відображені границі класифікатора. **Графік занесіть у звіт**.

Тепер виконайте той самий код, запросивши створення класифікатора на основі гранично випадкового лісу за допомогою прапорця **erf** вхідного аргументу. Введіть у вікні терміналу наступну команду:

```
$ python3 random_forests.py --classifier-type erf
```

Отримайте зображення то порівняйте його з попереднім. **Графік занесіть у звіт**. Зверніть увагу, що в останньому випадку були отримані більш лагідні піки. Це обумовлено тим, що в процесі навчання гранично випадкові ліси мають більше можливостей для вибору оптимальних дерев рішень, тому, як правило, вони забезпечують отримання кращих границь.

Оцінка мір достовірності прогнозів

Якщо ви подивитеся на результати, що відображаються у вікні терміналу, побачите, що для кожної точки даних виводяться ймовірності. Цими ймовірностями вимірюються рівні довірливості (рівні довіри) для кожного класу. Оцінка рівнів довіри відіграє важливу роль у машинному

навчанні. Додайте в той же файл наступний рядок, який визначає масив тестових точок даних.

```
# Обчислення параметрів довірливості
test_datapoints = np.array([[5, 5], [3, 6], [6, 4],
                             [7, 2], [4, 4], [5, 2]])
```

Об'єкт класифікатора має убудований метод, призначений для обчислення рівнів довірливості. Класифікуємо кожну точку та обчислимо рівні довірливості.

```
print("\nConfidence measure:")
for datapoint in test_datapoints:
    probabilities =
classifier.predict_proba([datapoint])[0]
    predicted_class = 'Class-' +
str(np.argmax(probabilities))
    print('\nDatapoint:', datapoint)
    print('Predicted class:', predicted_class)
```

Візуалізуємо тестові точки даних на підставі меж класифікатора.

```
# Візуалізація точок даних
visualize_classifier(classifier, test_datapoints,
                     [0]*len(test_datapoints),
                     'Тестовые точки данных')

plt.show()
```

Результат виконання цього коду із прапором **rf** **занесіть у звіт**

У вікні терміналу з'явиться виведена інформація **Скріншот цієї інформації виріжте та занесіть у звіт.**

Для кожної точки даних обчислюється можливість її належності кожному з трьох класів. Ми вибираємо той клас, якому відповідає найвищий рівень довіри.

Результат виконання коду із прапором **erf** **занесіть у звіт.**

У вікні терміналу з'явиться виведена інформація **Скріншот цієї інформації виріжте та занесіть у звіт.**

Збережіть код робочої програми під назвою LR_5_task_1.py

Код програми, графік функції та результати оцінки якості занесіть у звіт.

Зробіть висновок

Завдання 2.2. Обробка дисбалансу класів

Використовуючи для аналізу дані, які містяться у файлі `data_imbalance.txt` проведіть обробку з урахуванням дисбалансу класів.

Якість класифікатора залежить від даних, що використовуються для навчання. Однією з найпоширеніших проблем, із якими доводиться зіштовхуватися у реальних завданнях, є якість даних. Щоб класифікатор працював надійно, йому необхідно надати рівну кількість точок даних для кожного класу. Однак у реальних умовах гарантувати дотримання цієї умови не завжди можливо. Якщо кількість точок даних для одного класу в 10 разів більше, ніж для іншого, то класифікатор віддаватиме перевагу першому класу. Отже, такий дисбаланс необхідно врахувати алгоритмічно.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі пакети.

```
import sys
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import ExtraTreesClassifier
from sklearn import cross_validation
from sklearn.metrics import classification_report
from utilities import visualize_classifier
```

Використовуємо для аналізу дані, які містяться у файлі `data_imbalance.txt`. У цьому файлі кожен рядок містить значення розділені комою. Перші два значення відповідають даним, останнє – цільовій мітці. У цьому наборі даних є два класи. Завантажимо дані із цього файлу.

```
# Завантаження вхідних даних
input_file = 'data_imbalance.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
```

Розіб'ємо вхідні дані на два класи.

```
# Поділ вхідних даних на два класи на підставі міток
class_0 = np.array(X[y==0])
class_1 = np.array(X[y==1])
```


Візуалізуємо вхідні дані, використовуючи точкову діаграму.

Візуалізація вхідних даних

```
plt.figure()
plt.scatter(class_0[:, 0], class_0[:, 1], s=75,
            facecolors='black', edgecolors='black',
            linewidth=1, marker='x')
plt.scatter(class_1[:, 0], class_1[:, 1], s=75,
            facecolors='white', edgecolors='black',
            linewidth=1, marker='o')
plt.title('Входные данные')
```

Розіб'ємо дані на навчальний та тестовий набори.

Розбиття даних на навчальний та тестовий набори

```
X_train, X_test, y_train, y_test =
    cross_validation.train_test_split(
        X, y, test_size=0.25, random_state=5)
```

Визначимо параметри для класифікатора з урахуванням гранично випадкових лісів. Зверніть увагу на вхідний параметр `balance`, який керує тим, чи враховуватиметься алгоритмічно дисбаланс класів.

У разі врахування цього фактора ми повинні додати ще один параметр, `class_weight`, що балансує ваги таким чином, щоб вони були пропорційні до кількості точок даних у кожному класі.

Класифікатор на основі гранично випадкових лісів

```
params = {'n_estimators': 100, 'max_depth': 4,
          'random_state': 0}
if len(sys.argv) > 1:
    if sys.argv[1] == 'balance':
        params = {'n_estimators': 100, 'max_depth': 4,
                  'random_state': 0, 'class_weight': 'balanced'}
    else:
        raise TypeError("Invalid input argument; should be 'balance'")
```

Створимо, навчимо і візуалізуємо класифікатор, використовуючи тренувальні дані.

```
classifier = ExtraTreesClassifier(**params)
classifier.fit(X_train, y_train)
visualize_classifier(classifier, X_train, y_train, 'Training dataset')
```

Передбачимо та візуалізуємо результат для тестового набору даних.

```

y_test_pred = classifier.predict(X_test)
visualize_classifier(classifier, X_test, y_test, 'Тестовый набор данных')

# Обчислення показників ефективності класифікатора
class_names = ['Class-0', 'Class-1']
print("\n" + "#" * 40)
print("\nClassifier performance on training dataset\n")
print(classification_report(y_train,
                             classifier.predict(X_train), target_names=class_names))
print("#" * 40 + "\n")

print("#" * 40)
print("\nClassifier performance on test dataset\n")
print(classification_report(y_test, y_test_pred,
                             target_names=class_names))
print("#" * 40 + "\n")

plt.show()

```

Графік вхідних даних занесіть у звіт.

Графік даних класифікатора для тестового набору занесіть у звіт.

Зверніть увагу що, класифікатору не вдалося визначити фактичну межу між двома класами. В даному випадку обчислену межу представляє чорна пляма у верхній частині малюнка.

У вікні терміналу також відобразиться інформація. **Скрін з інформацією виріжте та занесіть у звіт!**

Також буде виведено попередження про наявність нульових значень у першому рядку з числовими даними, що призводить до виникнення помилки поділу на нуль (виключення ZeroDivisionError) при спробі обчислення показника f1-score. Щоб це попередження не з'являлося, запустіть код у вікні терміналу із прапором ignore.

```
python3 --W ignore class_imbalance.py
```

Далі для врахування дисбалансу класів виконайте код:

```
python3 class_imbalance.py balance
```

Графік даних класифікатора занесіть у звіт.

У вікні терміналу також відобразиться інформація. **Скрін з інформацією виріжте та занесіть у звіт!**

Збережіть код робочої програми з обов'язковими коментарям під назвою LR_5_task_2.py
Зробіть висновок

Завдання 2.3. Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку

Використовуючи дані, що містяться у файлі `data_random_forests.txt`, знайти оптимальних навчальних параметрів за допомогою сіткового пошуку.

У процесі роботи з класифікаторами вам не завжди відомо, які параметри є найкращими. Їх підбір вручну методом грубої сили (шляхом перебору всіх можливих комбінацій) практично нереалізований.

І тут на допомогу приходить сіточний пошук (grid search). Розглянемо як це робиться.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі пакети.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn import cross_validation, grid_search
from sklearn.ensemble import ExtraTreesClassifier
from sklearn import cross_validation
from sklearn.metrics import classification_report

from utilities import visualize_classifier
```

Використовуємо для нашого аналізу дані, що містяться у файлі `data_random_forests.txt`.

```
input_file = 'data_random_forests.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
```

Розіб'ємо дані на три класи.

```
# Розбиття даних на три класи на підставі міток
class_0 = np.array(X[y==0])
class_1 = np.array(X[y==1])
class_2 = np.array(X[y==2])
```

Розіб'ємо дані на навчальний та тестовий набори.

```
# Розбиття даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test =
    cross_validation.train_test_split(
        X, y, test_size=0.25, random_state=5)
```

Задамо сітку значень параметрів, де будемо тестувати класифікатор.

Зазвичай ми підтримуємо постійним значення одного параметра та варіюємо інші. Потім ця процедура повторюється кожного з параметрів.

На разі ми хочемо знайти найкращі значення параметрів `n_estimators` і `max_depth`. Визначимо сітку значень параметрів.

```
# Визначення сітки значень параметрів
parameter_grid = [ {'n_estimators': [100],
                    'max_depth': [2, 4, 7, 12, 16]},
                   {'max_depth': [4], 'n_estimators': [25, 50, 100, 250]}
                   ]
```

Визначимо метричні характеристики, які має використовувати класифікатор для знаходження найкращої комбінації параметрів.

```
metrics = ['precision_weighted', 'recall_weighted']
```

Для кожної метрики необхідно виконати сітковий пошук, під час якого ми навчатимемо класифікатор конкретної комбінації параметрів.

```
for metric in metrics:
    print("\n##### Searching optimal parameters for", metric)

    classifier = grid_search.GridSearchCV(
        ExtraTreesClassifier(random_state=0),
        parameter_grid, cv=5, scoring=metric)
    classifier.fit(X_train, y_train)
```

Виведемо оцінку для кожної комбінації параметрів.

```
print("\nGrid scores for the parameter grid:")
for params, avg_score, _ in classifier.grid_scores_:
    print(params, '-->', round(avg_score, 3))
print("\nBest parameters:", classifier.best_params_)
```

Виведемо звіт із результатами роботи класифікатора.

```
y_pred = classifier.predict(X_test)
print("\nPerformance report:\n")
print(classification_report(y_test, y_pred))
```

Після виконання цього коду у вікні терміналу з'явиться інформація.

Скріншот цієї інформації занесіть у звіт.

Виходячи з комбінацій значень параметрів, використаних у сіточному пошуку, тут виведені результати, що відповідають найбільш оптимальній комбінації для показника precision.

Отримайте іншу комбінацію значень параметрів, що забезпечує отримання найкращого значення показника recall. **Скріншот цієї інформації занесіть у звіт.**

Вона відрізняється від першої, що цілком зрозуміло, оскільки precision і recall - різні метричні характеристики, що вимагають використання різних комбінацій параметрів.

Збережіть код робочої програми з обов'язковими коментарям під назвою LR_5_task_3.py

Код програми та рисунок занесіть у звіт.

Зробіть висновок

Завдання 2.4. Обчислення відносної важливості ознак

Коли ми працюємо з наборами даних, що містять N-вимірні точки даних, необхідно розуміти, що не всі ознаки однаково важливі. Одні з них відіграють більшу роль, ніж інші. Маючи в своєму розпорядженні цю інформацією, можна зменшити кількість розмірностей, що враховуються. Ми можемо використовувати цю можливість зниження складності алгоритму та його прискорення. Іноді деякі ознаки виявляються зайвими. Отже, їх можна безболісно виключити із набору даних.

Для обчислення важливості ознак будемо використовувати регресор AdaBoost. Скорочення походить від алгоритму Adaptive Boosting (адаптивна підтримка), який часто застосовується у поєднанні з іншими алгоритмами машинного навчання для підвищення їх ефективності. AdaBoost витягує

навчальні точки даних для тренування поточного класифікатора, використовуючи деякий розподіл їх ваг. Цей розподіл ітеративно оновлюється, тому наступні класифікатори фокусуються на складніших точках. (Важкі точки - це точки, які були класифіковані неправильно.) Завдяки цьому точки даних, які раніше були неправильно класифіковані, отримують великі ваги у вибіркового наборі даних, що використовується для навчання класифікаторів. Алгоритм об'єднує ці класифікатори в "комітет", який приймає остаточне рішення на підставі виваженої більшості голосів.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі пакети.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn import datasets
from sklearn.metrics import mean_squared_error,
    explained_variance_score
from sklearn import cross_validation
from sklearn.utils import shuffle
```

Ми будемо використовувати вбудований набір даних із цінами на нерухомість, доступний у бібліотеці scikit-learn.

```
# Завантаження даних із цінами на нерухомість
housing_data = datasets.load_boston()
```

Перемішаємо дані, щоб підвищити об'єктивність нашого аналізу.

```
# Перемішування даних
X, y = shuffle(housing_data.data, housing_data.target,
               random_state=7)
```

Розіб'ємо дані на навчальний та тестовий набори.

```
# Розбиття даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test =
    cross_validation.train_test_split(
        X, y, test_size=0.2, random_state=7)
```

Визначимо і навчимо регресор AdaBoost, застосовуючи регресор на основі дерева рішень у якості індивідуальної моделі.

```
# Модель на основі регресора AdaBoost
regressor = AdaBoostRegressor(
    DecisionTreeRegressor(max_depth=4),
    n_estimators=400, random_state=7)
regressor.fit(X_train, y_train)
```

Оцінимо ефективність регресора.

```
# Обчислення показників ефективності регресора AdaBoost
y_pred = regressor.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
evs = explained_variance_score(y_test, y_pred)
print("\nADABOOST REGRESSOR")
print("Mean squared error =", round(mse, 2))
print("Explained variance score =", round(evs, 2))
```

Цей регресор має вбудований метод, який можна викликати для обчислення відносної важливості ознак.

```
# Вилучення важливості ознак
feature_importances = regressor.feature_importances_
feature_names = housing_data.feature_names
```

Нормалізуємо значення відносної ваги ознак.

```
# Нормалізація значень важливості ознак
feature_importances = 100.0 * (feature_importances /
max(feature_importances))
```

Відсортуємо ці значення відображення у вигляді діаграми.

```
# Сортування та перестановка значень
index_sorted = np.flipud(np.argsort(feature_importances))
```

Розставимо мітки вздовж осі X для побудови стовпчастої діаграми.

```
# Розміщення міток уздовж осі X
pos = np.arange(index_sorted.shape[0]) + 0.5
```

Побудуємо стовпчасту діаграму.

```
# Побудова стовпчастої діаграми
```

```
plt.figure()
plt.bar(pos, feature_importances[index_sorted], align='center')
plt.xticks(pos, feature_names[index_sorted])
plt.ylabel('Relative Importance')
plt.title('Оценка важности признаков с использованием регрессора
          AdaBoost')
plt.show()
```

Після виконання цього коду на екрані з'явиться діаграма. **Діаграму занесіть у звіт та проаналізуйте.**

Відповідно до проведеного аналізу зробіть висновки, які ознаки мають найбільшу роль, а якими можна знехтувати. Висновки занесіть у звіт.

Код програми та результати занесіть у звіт.

Програмний код збережіть під назвою LR_4_task_4.py

Завдання 2.5. Прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів

Проведіть прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів. Використайте набір даних, доступний за адресою;

<https://archive.ics.uci.edu/ml/datasets/Dodgers+Loop+Sensor>.

Цей набір містить дані про інтенсивність дорожнього руху під час проведення бейсбольних матчів на стадіоні Доджер-стедіум у Лос-Анджелесі.

Щоб зробити дані більш придатними для аналізу, їх необхідно піддати попередній обробці. Попередньо оброблені дані містяться у файлі `traffic_data.txt`. У цьому файлі кожен рядок містить рядкові значення, розділені комою. Як приклад розглянемо перший рядок. Значення в цьому рядку відформатовані наступним чином: день тижня, час доби, команда суперника, двійкове значення, що вказує, чи проходить матч (yes/no), кількість транспортних засобів, що проїжджають.

Метою завдання є прогнозування кількості транспортних засобів, що проїжджають дорогою, на підставі наданої інформації.

Отже, необхідно створити регресор, здатний прогнозувати вихідний результат. Створіть такий регресор на основі гранично випадкових лісів.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та імпортуйте такі пакети.


```

import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report,
    mean_absolute_error
from sklearn import cross_validation, preprocessing
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.metrics import classification_report

```

Завантажимо дані із файлу traffic_data.txt.

```

input_file = 'traffic_data.txt'
data = []
with open(input_file, 'r') as f:
    for line in f.readlines():
        items = line[:-1].split(',')
        data.append(items)

data = np.array(data)

```

Нечислові ознаки, що містяться серед цих даних, потребують кодування. Крім того, ми повинні простежити за тим, щоб числові ознаки не піддавалися кодуванню. Для кожної ознаки, що потребує кодування необхідно передбачити окремий кодувальник. Ми повинні відстежувати ці кодувальники, оскільки вони знадобляться нам, коли ми захочемо вирахувати результат для невідомої точки даних. Створимо вказані кодувальники.

```

# Перетворення рядкових даних на числові
label_encoder = []
X_encoded = np.empty(data.shape)
for i, item in enumerate(data[0]):
    if item.isdigit():
        X_encoded[:, i] = data[:, i]
    else:
        label_encoder.append(preprocessing.LabelEncoder())
        X_encoded[:, i] = label_encoder[-1]
            .fit_transform(data[:, i])
X = X_encoded[:, :-1].astype(int)
y = X_encoded[:, -1].astype(int)

```

Розіб'ємо дані на навчальний та тестовий набори.

```
# Розбиття даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test =
    cross_validation.train_test_split(
        X, y, test_size=0.25, random_state=5)
```

Навчимо регресор на основі гранично випадкових лісів.

```
# Регресор на основі гранично випадкових лісів
params = {'n_estimators': 100, 'max_depth': 4,
          'random_state': 0}
regressor = ExtraTreesRegressor(**params)
regressor.fit(X_train, y_train)
```

Обчислимо показники ефективності регресора на тестових даних.

```
# Обчислення характеристик ефективності
# регресора на тестових даних
y_pred = regressor.predict(X_test)
print("Mean absolute error:",
      round(mean_absolute_error(y_test, y_pred), 2))
```

Розглянемо як обчислюється результат для невідомої точки даних.

Для перетворення нечислових ознак на числові значення ми використовуємо кодувальники.

```
# Тестування кодування на одиночному прикладі
test_datapoint = ['Saturday', '10:20', 'Atlanta', 'no']
test_datapoint_encoded = [-1] * len(test_datapoint)
count = 0
for i, item in enumerate(test_datapoint):
    if item.isdigit():
        test_datapoint_encoded[i] = int(test_datapoint[i])
    else:
        test_datapoint_encoded[i] =
int(label_encoder[count].transform(test_datapoint[i]))
        count = count + 1

test_datapoint_encoded = np.array(test_datapoint_encoded)
```

Спрогнозуємо результат.

```
# Прогнозування результату для тестової точки даних
print("Predicted traffic:",
      int(regressor.predict([test_datapoint_encoded])[0]))
```

Повний код цього прикладу міститься у файлі `traffic_prediction.py`. Виконавши цей код, ви отримаєте як вихідний результат значення 26, яке дуже близько до фактичного значення. У цьому не важко переконатися, звернувшись до файлу даних.

Код програми та результати занесіть у звіт.

Програмний код збережіть під назвою `LR_5_task_5.py`

Коди комітити на GitHub. У кожному звіті повинно бути посилання на GitHub.

Назвіть бланк звіту `СШІ-ЛР-5-NNN-XXXXX.doc`

де `NNN` – позначення групи

`XXXXX` – позначення прізвища студента.

Переконвертуйте файл звіту в `СШІ-ЛР-5-NNN-XXXXX.pdf`