Лабораторна робота №8

Класифікація в scikit-learn

Мета роботи: Ознайомитись з побудовою моделей для вирішення задачі класифікації в scikit-learn, оцінкою та способами налаштування цих моделей.

Короткі теоретичні відомості

Метою класифікації є визначення способу позначення даних за допомогою набору дискретних міток. Вона відрізняється від кластеризації при навчанні з учителем тим, що немає значення, наскільки близькі члени груп у просторі.

Одним з методі класифікації є логістична регресія, яка використовує логістичну сигмоїдну функцію для визначення ймовірностей у діапазоні [0, 1], які можна зіставити з мітками класів.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
df_y = df.pop('high_quality')
df_X = df.drop(columns='quality')
X_train, X_test,y_train, y_test = train_test_split(df_X, df_y, test_size=0.1, random_state=0,stratify=df_y)
wine_quality_lr = Pipeline([('scale', StandardScaler()),('lr', LogisticRegression(class_weight='balanced', random_state=0))])
wine_quality_lr.fit(X_train, y_train)
```

Оцінити ефективність моделей класифікації можна виходячи з того, наскільки добре кожен клас у даних був передбачений моделлю. У випадку бінарної класифікації, тобто коли мітки лише дві, виділяють позитивний та негативний класи.

Проблему класифікації можна оцінити, порівнявши передбачені мітки з фактичними мітками за допомогою матриці невідповідностей:

Істинна належність / Результати тесту	Позитивний	Негативний
Позитивний	TP	FP
Негативний	FN	TN

Де TP (True Positive) — кількість правильно визначених «позитивних» об’єктів;

FP (False Positive) — кількість «негативних» об’єктів, які хибно класифіковані як «позитивні»;

FN (False Negative) — кількість «позитивних» об’єктів, які хибно класифіковані як «негативні»;

TN (True Negative) — кількість правильно визначених «негативних» об’єктів.

Scikit-learn надає функцію confusion_matrix().

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, quality_preds)
```

Використовуючи значення в матриці невідповідностей, можна обчислити показники, які допоможуть оцінити ефективність класифікатора. Найкращі показники залежатимуть від мети, для якої будується модель, і від того, чи збалансовані класи.

Коли класи приблизно однакові за розміром, можна використовувати точність (accuracy), яка дасть відсоток правильно класифікованих значень:

$$A = \frac{TP + TN}{TP + FP + TN + FN}$$

```
wine_quality_lr.score(X_test, y_test)
```

Коли наявний дисбаланс класів, точність може стати ненадійним показником для вимірювання ефективності моделі. Тоді використовується влучність (precision) – це відношення правильно визначених позитивних результатів до всього, що позначено позитивним:

$$P = \frac{TP}{TP + FP}$$

Також використовується повнота (recall), що дає відношення правильно визначених позитивних об’єктів до всіх позитивних об’єктів:

$$R = \frac{TP}{TP + FN}$$

Scikit-learn має функцію classification_report(), яка обчислює влучність і повноту. На додаток до обчислення цих показників для мітки класу, вона також обчислює незважене середнє між класами і середньозважене (середнє між класами, зважене за кількістю об’єктів у кожному класі). Стовпець support вказує кількість об’єктів, які належать до кожного класу, використовуючи позначені дані.

Оцінка F1 допомагає збалансувати влучність і повноту, використовуючи середнє гармонійне з двох.

```
from sklearn.metrics import classification_report
print(classification_report(y_test, quality_preds))
```

Після створення та оцінки моделі, варто її налаштувати та обрати найкращу. Параметри моделі називаються гіперпараметрами.

Scikit-learn має клас GridSearchCV в модулі model_selection для налаштування моделі. GridSearch дозволяє визначити простір пошуку та перевірити всі комбінації гіперпараметрів у цьому просторі, зберігаючи ті, які призводять до найкращої моделі.Подібні класи використовують перехресну перевірку, тобто вони ділять навчальні дані на підмножини, деякі з яких будуть набором для оцінки моделі (без потреби в тестових дани допоки модель не буде навчена).

Одним із поширених методів перехресної перевірки є k-кратна перехресна перевірка, яка розбиває навчальні дані на k підмножин і тренує модель k разів, щоразу залишаючи одну підмножину для тестування. Оцінка моделі буде середнім значенням для k тестових наборів.

Щоб використовувати GridSearchCV, потрібно надати модель (або конвеєр) і простір пошуку, який буде словником, що відображає гіперпараметр для налаштування (за назвою) на список значень, які потрібно спробувати. За бажанням, можна надати для використання метрику оцінки, а також значення k для використання при перехресній перевірці.

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import MinMaxScaler
pipeline = Pipeline([('scale', MinMaxScaler()),('lr',
LogisticRegression(class_weight='balanced',random_state=0))])
search_space = {'lr__C': np.logspace(-1, 1, num=10),'lr__fit_intercept': [True,
False]}
lr_grid = GridSearchCV(pipeline, search_space, scoring='f1_macro',
cv=5).fit(X_train, y_train)
```

Гіперпараметри логістичної регресії:

penalty - відповідає за спосіб регуляризації (використовується для уникнення перенавчання моделі), можливі значення ‘l1’, ‘l2’, ‘elasticnet’, ‘none’.

C – обернене значення до сили регуляризації, повинно бути дійсним числом.

fit_intercept – булевий параметр, визначає чи буде додаватись до результуючої функції константа.

class_weight – ваги класів. Можна задати словника клас:вага або використати значення ‘balanced’ для автоматичного підлаштування ваг відповідно до пропорцій класів у даних.

Для класифікації також можна використовувати дерева рішень, випадкові ліси та інші моделі. Дерева рішень рекурсивно розбивають дані, приймаючи рішення про те, які ознаки використовувати для кожного поділу.

```
from sklearn.tree import DecisionTreeClassifier
pipeline = Pipeline([('scale',
StandardScaler()),('dt',DecisionTreeClassifier(random_state=0))])
pipeline.fit(X_train, y_train)
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
rf = RandomForestClassifier(n_estimators=100, random_state=0)
search_space = {'max_depth': [4, 8],'min_samples_leaf': [4, 6]}
rf_grid = GridSearchCV(rf, search_space, cv=5, scoring='precision').fit(X_train,
y_train)
```

Завдання до лабораторної роботи

Написати програму, яка навчає та тестує модель, що виконує задачу бінарної класифікації відповідно до варіанту, оцінити модель за допомогою відповідних метрик та спробувати її покращити (підібрати інші гіперпараметри або використати інший алгоритм класифікації, або змінити комбінацію ознак).

Оформити звіт. Звіт повинен містити:

— титульний лист;
— код програми;
— результати виконання коду.

Продемонструвати роботу програми та відповісти на питання стосовно теоретичних відомостей та роботи програми.

Варіант 1: lab2/possum.csv. Класифікувати опосумів за статтю, використовуючи будь-які незалежні змінні.
Варіант 2: Titanic.csv. Класифікувати пасажирів на тих, хто вижив, та інших, використовуючи будь-які незалежні змінні.
Варіант 3: employee.csv. Визначити за допомогою класифікації чи піде працівник з роботи (останній стовпець), використовуючи будь-які незалежні змінні.
Варіант 4: lab4/diamonds.csv. Класифікувати діаманти на діаманти ідеальної якості (‘Ideal’) та всі інші, використовуючи будь-які незалежні змінні.
Варіант 5: lab4/Stars.csv. Класифікувати зорі на маленькі тьмяні зорі (червоні та коричневі карлики, типи 0 та 1) та всі інші, використовуючи будь-які незалежні змінні.
Варіант 6: blood.csv. Класифікувати донорів крові за двома типами (останній стовпець), використовуючи будь-які незалежні змінні.
Варіант 7: lab2/Birthweight.csv. Класифікувати дітей на тих, що мають низьку вагу, та інших, використовуючи будь-які незалежні змінні.
Варіант 8: fetal_health.csv. Класифікувати ембріони на здорові (тип 1) та всі інші, використовуючи будь-які незалежні змінні.
Варіант 9: glass.csv. Класифікувати скло на скло для будівель (тип 1 та 2) та всі інші типи, використовуючи хімічні характеристики скла.
Варіант 10: lab4/penguins.csv. Класифікувати пінгвінів за статтю, використовуючи будь-які незалежні змінні.
Варіант 11: lab4/Stars.csv. Класифікувати зорі на гіганти (типи 4 та 5) та всі інші, використовуючи будь-які незалежні змінні.
Варіант 12: diabetes.csv. Класифікувати пацієнтів за наявністю чи відсутністю діабету (останній стовпець), використовуючи будь-які незалежні змінні.
Варіант 13: lab4/telecom.csv. Класифікувати клієнтів на лояльних та нелояльних, використовуючи будь-які незалежні змінні.

Варіант 14: glass.csv. Класифікувати скло на скло для ламп (тип 7) та всі інші типи, використовуючи хімічні характеристики скла.
Варіант 15: lab4/diamonds.csv. Класифікувати діаманти на преміум якість (‘Premium’) та всі інші, використовуючи будь-які незалежні змінні.