Лабораторна робота №4

Класифікація текстових даних

Мета роботи: Ознайомитись з класифікацією документів за допомогою моделей машинного навчання.

Короткі теоретичні відомості

Для задачі класифікації з учителем потрібно мати певні позначені дані, які можна використовувати для навчання моделі класифікації тексту. Ці дані, по суті, є підібраними документами, які вже заздалегідь віднесені до певного класу чи категорії. Використовуючи їх, по суті, витягуємо ознаки та атрибути з кожного документа та змушуємо модель вивчати ці атрибути, що відповідають кожному конкретному документу та його класу/категорії. Це робиться шляхом передачі його алгоритму машинного навчання з учителем.

Звичайно, дані потрібно попередньо обробити та нормалізувати перед створенням моделі. Після цього виконується той самий процес нормалізації та виділення ознак, а потім результат передається в модель, щоб передбачити клас або категорію для нових документів.

Класифікація тексту або документа визначається як процес віднесення текстових документів до одного або кількох класів або категорій, припускаючи, що наявний заздалегідь визначений набір класів. Документи є текстовими документами, і кожен документ може містити речення або навіть абзац слів. Система класифікації тексту може успішно класифікувати кожен документ до його правильного класу (класів) на основі властивостей документа.

Існує кілька варіантів завдань на класифікацію тексту, які базуються на кількості класів для прогнозування та характері прогнозів:

• Бінарна класифікація

• Багатокласова класифікація

• Класифікація за кількома мітками

Наступні основні кроки описують типовий робочий процес для системи класифікації тексту, припускаючи, що набір даних уже готовий до використання:

• Підготовка навчального та тестового наборів даних

• Попередня обробка та нормалізація текстових документів

• Вилучення та розробка ознак

• Навчання моделі

• Прогнозування за допомогою моделі та оцінка моделі

• Впровадження моделі

Щоб побудувати систему машинного навчання, потрібно побудувати моделі на даних навчання, а потім перевірити та оцінити їх продуктивність на основі даних для тестування. Тому потрібно розділити набір даних на навчальні та тестові набори даних.

Для цього використовується функція train_test_split бібліотеки scikit-learn.

from sklearn.model_selection import train_test_split

train_corpus, test_corpus, train_label_nums, test_label_nums, train_label_names, test_label_names = train_test_split(data_df['Clean Article'], data_df['Target Label'], data_df['Target Name'],test_size=0.3, random_state=0)

train_corpus.shape, test_corpus.shape

У наборі даних є багато точок даних, якими зазвичай є рядки набору даних, а стовпці — це різні характеристики або ознаки набору даних із певними значеннями для кожного рядка чи спостереження.

Вилучення ознак з тексту включає створення моделі текстових даних.

from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(min_df=0.0, max_df=1.0)

cv_train_features = cv.fit_transform(train_corpus)

cv_test_features = cv.transform(test_corpus)

На етапі моделювання алгоритми класифікації проходять три стадії:

• Навчання

• Оцінка

• Налаштування

Ефективність моделей класифікації зазвичай залежить від того, наскільки добре вони прогнозують результати для нових точок даних. Зазвичай ця продуктивність вимірюється за допомогою тестового набору даних, який складається з точок даних, які жодним чином не використовувалися для впливу на класифікатор або його навчання.

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, яка дасть відсоток правильно класифікованих значень:

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

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

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

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

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

То ж потрібно перевірити чи не сильно відрізняється кількість об’єктів у різних класах.

data_df["Target Name"].value_counts()

Існує чимало алгоритмів класифікації. Для класифікації текстів найчастіше використовуються наступні:

- Наївний байєсів класифікатор з багатьма мітками;
- Логістична регресія;
- Опорні вектори;
- Випадкові ліси;
- Градієнтний бустинг.

Наївний алгоритм Байєса — це алгоритм навчання з учителем, який використовує теорему Байєса. Однак існує «наївне» припущення, що кожна ознака повністю незалежна від інших.

Наївний байєсів класифікатор з багатьма мітками— це розширення алгоритму для прогнозування та класифікації точок даних, де кількість окремих класів або результатів перевищує два.

from sklearn.naive_bayes import MultinomialNB

mnb = MultinomialNB()

mnb.fit(cv_train_features, train_label_names)

print('Точність:' mnb.score(cv_test_features, test_label_names))

Вивести оцінку моделі за усіма метриками можна за допомогою

from sklearn.metrics import classification_report

predicted=mnb.predict(cv_test_features)

print(classification_report(test_label_names, predicted))

Логістична регресія використовує логістичну (відому також як сигмоїдну) математичну функцію для оцінки значень параметрів. Це коефіцієнти всіх ознак, які мінімізують загальні втрати під час прогнозування результату — у цьому випадку категорії груп новин. Алгоритм максимізує ймовірності прогнозованих значень до спостережуваних значень за допомогою оцінки максимальної ймовірності.

from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()

lr.fit(cv_train_features, train_label_names)

Алгоритм опорних векторів представляє навчальні дані у вигляді точок у просторі, так що точки, що належать до будь-якого класу, можуть бути розділені широким проміжком між ними (гіперплощиною), а новим точкам даних, які потрібно передбачити, призначаються класи залежно від того, з якого боку цієї гіперплощини вони розташовуються.

from sklearn.svm import LinearSVC

svm = LinearSVC()

svm.fit(cv_train_features, train_label_names)

Випадковий ліс — це модель-ансамбль, яка використовує кілька дерев рішень на різні підвибірки набору даних і усереднення для підвищення точності прогнозування та контролю за перенавчанням. Розмір підвибірки завжди такий самий, як вихідний розмір вхідної вибірки, але вибірки відбираються із заміною. У випадкових лісах усі дерева навчаються паралельно. Крім того, під час розбиття вузла в процесі побудови дерева вибране розбиття більше не є найкращим розбиттям серед усіх ознак. Натомість вибраний поділ є найкращим поділом серед випадкової підмножини ознак.

from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=10, random_state=0)

rfc.fit(cv_train_features, train_label_names)

Бустинг намагається виправити помилки попередніх моделей. Один із способів зробити це – рухатися в напрямку найбільш крутого зниження функції втрат для моделі. Оскільки градієнт (узагальнення похідної з багатьма змінними) є напрямком найбільш крутого підйому, це можна зробити шляхом розрахунку негативного градієнта, який дає напрямок найбільш крутого спуску, що означає найкраще покращення функції втрат від поточного результату. Ця техніка називається градієнтним спуском.

GradientBoostingClassifier також використовує багато дерев рішень, кожне з них будується таким чином, щоб зменшити помилку попередніх.

```
from sklearn.ensemble import GradientBoostingClassifier
gbc = GradientBoostingClassifier(n_estimators=10, random_state=0)
gbc.fit(cv_train_features, train_label_names)
```

Налаштування моделей:

```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer
mnb_pipeline = Pipeline([('tfidf', TfidfVectorizer()),('mnb', MultinomialNB())])
param_grid = {'tfidf__ngram_range': [(1, 1), (1, 2)], 'mnb__alpha': [1e-5, 1e-4, 1e-2, 1e-1, 1]}
gs_mnb = GridSearchCV(mnb_pipeline, param_grid, cv=5)
gs_mnb = gs_mnb.fit(train_corpus, train_label_names)
```

Можна одночасно налаштовувати як гіперпараметри моделей, так і попередню обробку, наприклад, представляти текст у вигляді TF-IDF моделей з уні- та біграмами.

Якщо використовувати у якості моделі тексту Word2Vec або FastText, то документи корпусу можна представити у вигляді усереднених векторів:

```
def document_vectorizer(corpus, model, num_features):
    vocabulary = set(model.wv.index_to_key)
    def average_word_vectors(words, model, vocabulary, num_features):
        feature_vector = np.zeros((num_features,), dtype="float64")
        nwords = 0.
        for word in words:
            if word in vocabulary:
                nwords = nwords + 1.
                feature_vector = np.add(feature_vector, model.wv[word])
        if nwords:
            feature_vector = np.divide(feature_vector, nwords)
```

return feature_vector

features = [average_word_vectors(tokenized_sentence, model, vocabulary, num_features) for tokenized_sentence in corpus]

return np.array(features)

avg_wv_train_features = document_vectorizer(corpus=tokenized_train, model=w2v_model, num_features=w2v_num_features)

avg_wv_test_features = document_vectorizer(corpus=tokenized_test, model=w2v_model, num_features=w2v_num_features)

Завдання до лабораторної роботи

Створити програму, яка зчитує заданий набір даних, виконує попередню обробку та класифікацію документів відповідно до варіанту. Якщо недостатньо ресурсів для роботи з повним набором даних, можна виділити частину, але таким чином, щоб були присутні усі класи.

Оформити звіт. Звіт повинен містити:

— титульний лист;
— код програми;
— результати виконання коду;
— порівняння оцінок моделей до і після спроби покращення.

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