Практическая работа 4

Тема: Решение задачи классификации

Цель: закрепить навыки решения задач классификации с использованием логистической регрессии, метода k-ближайших соседей и метода опорных векторов (SVM). Освоить предобработку данных, оценку моделей с помощью метрик accuracy, F1, confusion matrix, PR-AUC, ROC-AUC, а также научиться оптимизировать порог классификации.

Задание:

1. Подготовка данных:
   - Найти и выбрать набор данных для решения задачи классификации (учитывая, что в группе не должно быть повторяющихся наборов данных).
   - Провести предобработку данных: устранить пропуски, нормализовать или стандартизировать данные, если это необходимо.
2. Реализация моделей классификации:
   - Реализовать следующие алгоритмы классификации:
   - Логистическая регрессия
   - Метод k-ближайших соседей
   - Метод опорных векторов (SVM)
   - Разделить набор данных на обучающую и тестовую выборки (например, 70% на обучение, 30% на тестирование).
3. Оценка моделей:
   - Сравнить результаты работы всех моделей на основе следующих метрик:
     - Accuracy (Точность)
     - F1-Score
     - Confusion matrix (Матрица ошибок)
     - PR-AUC (Precision-Recall AUC)
     - ROC-AUC (Receiver Operating Characteristic AUC)
   - Отрисовать PR и ROC кривые для каждого класса (если задача

мультиклассовая) или для классов (если бинарная).

4. Оптимизация порога классификации:
    - Если задача бинарная, найти оптимальный порог классификации на основе:
    - PR-кривой с точки зрения F1-метрики
    - ROC-кривой с использованием G_mean.

Порядок выполнения работы:

Этап 1. Подготовка данных:

1. Найти и загрузить набор данных для задачи классификации.
2. Провести необходимые шаги по предобработке данных:
   - Проверить на наличие пропущенных значений и обработать их.
   - Нормализовать или стандартизировать числовые признаки (если это необходимо для алгоритмов).

Этап 2. Реализация моделей:

1. Используя функцию `train_test_split`, разделить набор данных на обучающую и тестовую выборки.
2. Реализовать модели:
   - Логистическая регрессия (`LogisticRegression`).
   - Метод k-ближайших соседей (`KNeighborsClassifier`).
   - Метод опорных векторов (SVM) (`SVC`).
3. Настроить гиперпараметры моделей с помощью `GridSearchCV`.

Этап 3. Оценка качества моделей:

1. Оценить модели на основе метрик accuracy, F1, confusion matrix.
2. Построить PR и ROC кривые для каждой модели.
3. Рассчитать PR-AUC и ROC-AUC для каждой модели.
4. Сравнить результаты моделей по всем метрикам.

Этап 4. Оптимизация порога (для бинарной классификации):

1. Построить PR-кривую и найти оптимальный порог по F1-метрике.
2. Построить ROC-кривую и найти оптимальный порог по G_mean.

Отчет:

1. Описать ход выполнения работы.
2. Привести листинги кода с комментариями.
3. Вставить скриншоты графиков и результатов.
4. Подготовить к защите работу, рассказав о сравнении моделей и результатах оптимизации порога.

ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ ДЛЯ ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ РАБОТЫ

ЭТАП 1: ПОДГОТОВКА ДАННЫХ

1.1 Поиск и выбор набора данных:

Для задачи классификации данные могут быть получены из открытых источников, например, с сайтов:

- Kaggle
- UCI Machine Learning Repository
- Google Dataset Search

Важно, чтобы данные были четко размечены (имелись метки классов).

1.2 Предобработка данных:

Пропуски в данных: Пропущенные значения могут ухудшить работу моделей, поэтому их нужно корректно обработать.

Удаление строк с пропусками.

Когда в данных встречаются пропуски (например, значения “NaN” или отсутствующие записи), один из подходов к обработке таких данных — это полное удаление строк, в которых встречаются пропуски.

Преимущества:

- Удаление строк с пропусками упрощает данные, позволяя сразу использовать их для обучения моделей без дополнительных операций.
- Избегается внесение потенциальных искажений в данные (которые могут возникнуть при неправильной замене пропусков).

Недостатки:

- Если много строк имеют пропуски, то удаление может привести к значительной потере данных, что повлияет на точность моделей.
- Удаление строк с пропусками неэффективно, если пропущенные данные критически важны для модели.

Пример использования в Python

Листинг 1 – Удаление строк с пропусками пример

```
import pandas as pd
# Удаление строк с любыми пропущенными значениями
data = pd.read_csv('data.csv')
clean_data = data.dropna()
```

Функция `dropna()` удаляет все строки, где есть хотя бы одно пропущенное значение.

Реальный пример

Предположим, у вас есть набор данных о клиентах банка, где могут быть пропуски в таких столбцах, как возраст, доход, город и т. д. Вот как можно удалить строки с пропусками

Листинг 2 – Реальный пример удаления строк

```
import pandas as pd
# Допустим, у нас есть CSV-файл с данными клиентов банка
data = pd.read_csv('bank_customers.csv')
# Посмотрим на первые несколько строк данных
print(data.head())
# Удаление всех строк, где есть хотя бы одно пропущенное значение
clean_data = data.dropna()
# Выведем информацию о данных до и после очистки
print("Размер исходных данных:", data.shape)
print("Размер данных после удаления строк с пропусками:", clean_data.shape)
```

Описание

В этом примере данные загружаются из файла bank_customers.csv. Сначала выводится количество строк и столбцов в исходных данных. Функция dropna() удаляет все строки, где есть хотя бы одно пропущенное значение.

В конце выводится размер данных после очистки, что позволяет оценить, сколько строк было удалено.

Этот подход полезен, когда пропуски встречаются случайно и их не слишком много, но он может оказаться неэффективным, если в данных слишком много пропусков. В таком случае лучше использовать другие методы обработки пропусков, например, заполнение медианой или модой.

Заполнение пропусков средним, медианой или модой (для числовых данных).

Если данные имеют пропуски, но их нельзя удалить (так как потеряется важная информация), альтернативой является заполнение пропусков (импутация).

Способы заполнения:

- **Среднее (Mean):** Для числовых данных, часто используется среднее значение по столбцу для заполнения пропущенных данных. Подходит, если распределение данных симметричное.

Пример:

```
data['Age'].fillna(data['Age'].mean(), inplace=True)
```

- **Медиана (Median):** Если данные содержат выбросы или асимметричны, лучше использовать медиану, так как она менее чувствительна к крайним значениям.

Пример:

```
data['Income'].fillna(data['Income'].median(), inplace=True)
```

- **Мода (Mode):** Для категориальных данных можно использовать наиболее часто встречающееся значение (моду).

Пример:

```
data['Gender'].fillna(data['Gender'].mode()[0], inplace=True)
```

Преимущества заполнения:

- Сохранение всех строк данных, что может быть полезно при небольшом объеме данных.
- Заполнение делает возможным использование алгоритмов, которые не поддерживают пропуски в данных.

Недостатки заполнения:

- Может исказить статистические свойства данных.
- В зависимости от метода заполнения, может вносить искажения, если распределение данных не симметрично (например, использование среднего на выборке с выбросами).

Для категориальных признаков — заполнение наиболее частым значением.

Для категориальных признаков (например, столбцы с текстовыми метками, такими как "Пол", "Город", "Тип клиента") пропуски можно заполнить несколькими способами:

1. Мода (наиболее частое значение): Этот метод хорошо работает, когда в данных нет значительного разброса категорий, и одна из категорий преобладает.

Пример:

```
data['City'].fillna(data['City'].mode()[0], inplace=True)
```

2. Создание нового уровня (например, "Неизвестно"): Этот метод полезен, когда пропуски могут быть информативными. Добавляется новый уровень категории, чтобы указать, что данные отсутствуют.

Пример:

```
data['City'].fillna('Unknown', inplace=True)
```

Заполнение на основе других признаков: В некоторых случаях можно попытаться предсказать пропущенные значения на основе других переменных. Например, если есть признаки, которые коррелируют с категорией, можно обучить модель для предсказания пропущенных категориальных значений.

Преимущества:

- Сохранение целостности набора данных без удаления строк.
- Возможность более точного анализа категориальных данных, когда пропуски заполнены логически правильным способом.

Недостатки:

- Заполнение может внести ошибку, если пропуски не случайны, что может снизить точность модели.
- Иногда пропуски могут содержать важную информацию, которую можно потерять при заполнении.

Нормализация и стандартизация: Некоторые алгоритмы, такие как метод k-ближайших соседей и SVM, чувствительны к масштабу признаков.

Поэтому данные могут требовать:

- **Нормализация:** Приведение значений к диапазону [0, 1].
- **Стандартизация:** Преобразование данных так, чтобы они имели среднее значение 0 и стандартное отклонение 1.

Полезные библиотеки:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

ЭТАП 2: РЕАЛИЗАЦИЯ МОДЕЛЕЙ КЛАССИФИКАЦИИ

2.1. Разделение данных на обучающую и тестовую выборки:

Перед обучением моделей важно разделить данные на тренировочную и тестовую выборки. Это позволяет проверить, насколько хорошо модель сможет обобщать данные, которые она еще не видела.

Часто используется функция train_test_split из библиотеки sklearn, которая автоматически разделяет данные в заданном соотношении (например, 70% на обучение и 30% на тестирование).

Листинг – Разделение данных

```
from sklearn.model_selection import train_test_split


# Разделение данных на 70% тренировочные и 30% тестовые
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
```

2.2 Логистическая регрессия:

Логистическая регрессия — это линейный классификатор, который предсказывает вероятность того, что объект принадлежит к одному из классов. Этот метод особенно хорошо подходит для бинарной классификации, где нужно выбрать один из двух классов. Для многоклассовой классификации используются такие подходы, как "One-vs-Rest" или "Softmax".

Пример кода

Листинг – Логистическая регрессия

```
from sklearn.linear_model import LogisticRegression
```

```
# Создание и обучение модели
model = LogisticRegression()
model.fit(X_train, y_train)
```

2.3 Метод k-ближайших соседей:

Метод k-ближайших соседей (k-Nearest Neighbors, k-NN) — это алгоритм, который классифицирует новый объект по его ближайшим соседям. Алгоритм работает по принципу: если большинство ближайших соседей принадлежат к какому-то классу, то и новый объект относится к этому классу. Количество соседей (параметр k) может быть выбрано с помощью подбора гиперпараметров, например, через GridSearchCV.

Пример кода:

Листинг – Метод К-ближайших соседей

```
from sklearn.neighbors import KNeighborsClassifier


# Создание и обучение модели
model = KNeighborsClassifier(n_neighbors=5)
model.fit(X_train, y_train)
```

2.4 Метод опорных векторов (SVM):

Метод опорных векторов (Support Vector Machine, SVM) — это мощный алгоритм, который пытается найти гиперплоскость, максимально разделяющую классы. SVM хорошо работает с небольшими наборами данных и может использовать различные ядра для работы с нелинейными задачами.

Пример кода:

Листинг – Метод опорных векторов

```
from sklearn.svm import SVC
```

```
model = SVC(kernel='linear')
model.fit(X_train, y_train)
```

ЭТАП 3: ОЦЕНКА МОДЕЛЕЙ

3.1 Метрики для оценки моделей:

1. Accuracy (Точность): Это одна из самых простых метрик для оценки классификатора. Точность вычисляется как отношение числа правильных предсказаний к общему числу предсказаний. Она подходит для задач, где классы сбалансированы, но может давать искаженные результаты на несбалансированных данных.

2. F1-Score: Это гармоническое среднее между precision (точность) и recall (полнота). F1-метрика особенно полезна для задач с несбалансированными классами, так как учитывает как false positives (ложные срабатывания), так и false negatives (ложные пропуски).

3. Confusion matrix (Матрица ошибок): Матрица ошибок отображает количество правильных и неправильных предсказаний для каждого класса. Она помогает оценить производительность модели в случае многоклассовых задач, показывая распределение ошибок.

Пример кода:

Листинг – Метрики

```
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix


# Предсказание
y_pred = model.predict(X_test)


# Оценка метрик
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred, average='weighted')
conf_matrix = confusion_matrix(y_test, y_pred)
```

3.2 Визуализация ROC и PR кривых:

ROC-кривая (Receiver Operating Characteristic) — это график, который показывает соотношение между True Positive Rate (чувствительность) и False Positive Rate (1 - специфичность) для разных порогов классификации.

Площадь под ROC-кривой (AUC) показывает, насколько хорошо модель различает классы.

PR-кривая (Precision-Recall) демонстрирует зависимость между точностью и полнотой для разных порогов.

Эта кривая особенно полезна для задач с несбалансированными данными, когда нужно больше внимания уделить false negatives.

Пример кода:

Листинг – Визуализация ROC, PR кривых

```
import matplotlib.pyplot as plt
# Построение ROC-кривой
plt.plot(fpr, tpr, label=f'ROC curve (area = {roc_auc:.2f})')
# Построение PR-кривой
plt.plot(recall, precision, label=f'PR curve (area = {pr_auc:.2f})')
plt.legend()
plt.show()
```

ЭТАП 4: ОПТИМИЗАЦИЯ ПОРОГА КЛАССИФИКАЦИИ

При бинарной классификации важно выбрать оптимальный порог для перевода вероятности в метку класса. Это особенно актуально, когда модель предсказывает вероятность принадлежности объекта к одному из классов, но нужно принять решение, к какому классу отнести объект на основе порогового значения (threshold). Порог классификации определяет, при каком значении вероятности объект относится к одному из классов.

PR-кривая (F1-метрика).

PR-кривая (Precision-Recall кривая) отображает зависимость между точностью (precision) и полнотой (recall). F1-метрика — это гармоническое среднее между precision и recall. Оптимальный порог выбирается на основе того, при каком значении порога F1-метрика достигает максимума.

ROC-кривая (G_mean).

ROC-кривая (Receiver Operating Characteristic) используется для визуализации работы бинарного классификатора, демонстрируя соотношение между True Positive Rate (чувствительность) и False Positive Rate. Для выбора оптимального порога можно использовать метрику G_mean, которая является геометрическим средним между True Positive Rate и True Negative Rate. Оптимальный порог выбирается при максимальном значении G_mean.

Пример кода:

Листинг – бинарная классификация

```
from numpy import argmax
# Вычисление F1-метрики для каждого порога
f1_scores = 2 * (precision * recall) / (precision + recall)
optimal_threshold_pr = thresholds[argmax(f1_scores)]
# Вычисление G_mean для каждого порога на основе ROC-кривой
gmean = np.sqrt(tpr * (1 - fpr))
optimal_threshold_roc = thresholds[argmax(gmean)]
```