|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА − Российский технологический университет»**  **РТУ МИРЭА** |

**Институт информационных технологий (ИИТ)**

**Кафедра прикладной математики (ПМ)**

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ**

по дисциплине «Технологии и инструментарий машинного обучения»

**Практическое занятие № 4**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент группы ИНБО-01-17 | *ИМБО-02-22, Ким Кирилл Сергеевич* | (подпись) | |
| Преподаватель | *Трушин Степан Михайлович, преподаватель* | (подпись) | |
| Отчет представлен | «\_\_\_»\_\_\_\_\_\_\_\_202\_\_г. | |  | |

Москва 2024 г.

**Цель занятия:**

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.

**Постановка задачи:**

**Этап 1. Подготовка данных:**

1. Найти и загрузить набор данных для задачи классификации.

Листинг 1 – Загрузка данных

|  |
| --- |
| import pandas as pd  from sklearn.datasets import fetch\_california\_housing  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import mean\_squared\_error, r2\_score  from sklearn.linear\_model import LinearRegression, Lasso, Ridge  from sklearn.metrics import mean\_squared\_error  f = ('/content/diabets-data-set/diabetes.csv')  data = pd.read\_csv(f) |

1. Провести необходимые шаги по предобработке данных:

* Проверить на наличие пропущенных значений и обработать их.
* Нормализовать или стандартизировать числовые признаки (если это необходимо для алгоритмов).

Листинг 2 – Проверка на наличие пропущенных значений

|  |
| --- |
| # Удаление строк с любыми пропущенными значениями  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)  # воспользуемся функциями isnull() и sum()  clean\_data.isnull().sum() |

Листинг 3 – Стандартизация числовых признаков

|  |
| --- |
| # импортируем необходимый класс из модуля preprocessing библиотеки sklearn  from sklearn.preprocessing import StandardScaler  # создадим объект этого класса  scaler = StandardScaler()    # приведем данные к единому масштабу  scaled\_data = scaler.fit\_transform(clean\_data)  # преобразуем scaled\_data обратно в датафрейм  df\_scaled = pd.DataFrame(scaled\_data, columns=['Pregnancies', 'Glucose', 'BloodPressure',                                                 'SkinThickness', 'Insulin', 'BMI',                                                 'DiabetesPedigreeFunction', 'Age', 'Outcome'])  # посмотрим на результат (только два первых значения)  df\_scaled |

**Этап 2. Реализация моделей:**

1. Используя функцию `train\_test\_split`, разделить набор данных на обучающую и тестовую выборки.

Листинг 4 – Разделение данных на обучающую и тестовую выборки

|  |
| --- |
| X = clean\_data.iloc[:,:-1].values  Y = clean\_data.iloc[:,-1:].values  # импортируем необходимый модуль  from sklearn.model\_selection import train\_test\_split    # размер тестовой выборки составит 30%  # также зададим точку отсчета для воспроизводимости результата  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y,                                                      test\_size = 0.3,                                                      random\_state = 42) |

1. Реализовать модели:

* Логистическая регрессия (`LogisticRegression`).

Листинг 5 – Логистическая регрессия

|  |
| --- |
| # импортируем логистическую регрессию из модуля linear\_model библиотеки sklearn  from sklearn.linear\_model import LogisticRegression    # создадим объект этого класса и запишем его в переменную model  model1 = LogisticRegression()    # обучим нашу модель  model1.fit(X\_train, y\_train)    # выполним предсказание класса на тестовой выборке  y\_pred1 = model1.predict(X\_test)  y\_pred1 |

* Метод k-ближайших соседей (`KNeighborsClassifier`).

Листинг 6 – Метод k-ближайших соседей

|  |
| --- |
| # Импортируйте необходимые модули  from sklearn.neighbors import KNeighborsClassifier  from sklearn.metrics import accuracy\_score  model2 = KNeighborsClassifier()  # обучим нашу модель  model2.fit(X\_train, y\_train)  # выполним предсказание класса на тестовой выборке  y\_pred2 = model2.predict(X\_test) |

* Метод опорных векторов (SVM) (`SVC`).

Листинг 7 – Метод опорных векторов

|  |
| --- |
| from sklearn.svm import SVC  from sklearn.metrics import recall\_score  model3 = SVC(probability=True)  # Включаем вероятность для ROC  # обучим нашу модель  model3.fit(X\_train, y\_train)  # выполним предсказание класса на тестовой выборке  y\_pred3 = model3.predict(X\_test) |

1. Настроить гиперпараметры моделей с помощью `GridSearchCV`.

Листинг 8 – Настройка моделей с помощью “GridSearchCV”

|  |
| --- |
| params= {      'C': [0.01, 0.1, 1, 10]  }  gs1 = GridSearchCV(estimator=model1, param\_grid=params, cv=5)  # теперь сопоставьте набор данных с объектом GridSearchCV.  gs1 = gs1.fit(X\_train, y\_train)  # Вывод лучших параметров и тестовой точности  best\_params1 = gs1.best\_params\_  accuracy1 = gs1.best\_score\_  print("Лучшие параметры:", best\_params1)  print("Тестовая точность:", accuracy1)  params= {      'n\_neighbors': [3, 5, 7, 9]  }  gs2 = GridSearchCV(estimator=model2, param\_grid=params, cv=5)  # теперь сопоставьте набор данных с объектом GridSearchCV.  gs2 = gs2.fit(X\_train, y\_train)  # Вывод лучших параметров и тестовой точности  best\_params2 = gs2.best\_params\_  accuracy2 = gs2.best\_score\_  print("Лучшие параметры:", best\_params2)  print("Тестовая точность:", accuracy2)  params= {      'C': [0.01, 0.1, 1, 10]  }  gs3 = GridSearchCV(estimator=model1, param\_grid=params, cv=10)  # теперь сопоставьте набор данных с объектом GridSearchCV.  gs3 = gs3.fit(X\_train, y\_train)  # Вывод лучших параметров и тестовой точности  best\_params3 = gs3.best\_params\_  accuracy3 = gs3.best\_score\_  print("Лучшие параметры:", best\_params3)  print("Тестовая точность:", accuracy3) |

**Этап 3. Оценка качества моделей:**

1. Оценить модели на основе метрик accuracy, F1, confusion matrix.

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.

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).

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.

Листинг 9 – Оценка модели

|  |
| --- |
| # построим матрицу ошибок  from sklearn.metrics import confusion\_matrix    # передадим ей тестовые и прогнозные значения  # поменяем порядок так, чтобы злокачественные опухоли были положительным классом  model\_matrix = confusion\_matrix(y\_test, y\_pred1, labels = [1, 0])  # добавим подписи к столбцам и строкам через параметры columns и index  # столбец - это прогноз, строка - фактическое значение  # 0 - добр. образование, 1 - злок. образование (только в рамках матрицы ошибок!)  # для удобства создадим датафрейм  model\_matrix\_df = pd.DataFrame(model\_matrix, columns = ['Прогноз добр.', 'Прогноз злок.'], index = ['Факт добр.', 'Факт злок.'])  model\_matrix\_df  from sklearn.metrics import accuracy\_score, f1\_score  # Оценка метрик  accuracy = accuracy\_score(y\_test, y\_pred1)  f1 = f1\_score(y\_test, y\_pred1, average='weighted')  print("Accuracy:", accuracy)  print("F1:", f1) |

1. Построить PR и ROC кривые для каждой модели.

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.

Листинг 10 – PR и ROC кривые

|  |
| --- |
| import matplotlib.pyplot as plt  from sklearn.metrics import precision\_recall\_curve, roc\_curve, auc  y\_scores1 = model1.predict\_proba(X\_test)[:, 1]  precision, recall, \_ = precision\_recall\_curve(y\_test, y\_scores1)  plt.plot(recall, precision)  plt.xlabel('Recall')  plt.ylabel('Precision')  plt.title('Precision-Recall Curve')  plt.legend()  plt.show()  fpr, tpr, \_ = roc\_curve(y\_test, y\_scores1)  plt.plot(fpr, tpr)  plt.xlabel('False Positive Rate')  plt.ylabel('True Positive Rate')  plt.title('ROC Curve')  plt.legend()  plt.show()  import matplotlib.pyplot as plt  from sklearn.metrics import precision\_recall\_curve, roc\_curve, auc  y\_scores2 = model2.predict\_proba(X\_test)[:, 1]  precision, recall, \_ = precision\_recall\_curve(y\_test, y\_scores2)  plt.plot(recall, precision)  plt.xlabel('Recall')  plt.ylabel('Precision')  plt.title('Precision-Recall Curve')  plt.legend()  plt.show()  fpr, tpr, \_ = roc\_curve(y\_test, y\_scores2)  plt.plot(fpr, tpr)  plt.xlabel('False Positive Rate')  plt.ylabel('True Positive Rate')  plt.title('ROC Curve')  plt.legend()  plt.show()  import matplotlib.pyplot as plt  from sklearn.metrics import precision\_recall\_curve, roc\_curve, auc  y\_scores3 = model3.predict\_proba(X\_test)[:, 1]  precision, recall, \_ = precision\_recall\_curve(y\_test, y\_scores3)  plt.plot(recall, precision)  plt.xlabel('Recall')  plt.ylabel('Precision')  plt.title('Precision-Recall Curve')  plt.legend()  plt.show()  fpr, tpr, \_ = roc\_curve(y\_test, y\_scores3)  plt.plot(fpr, tpr)  plt.xlabel('False Positive Rate')  plt.ylabel('True Positive Rate')  plt.title('ROC Curve')  plt.legend()  plt.show() |

1. Рассчитать PR-AUC и ROC-AUC для каждой модели.

Листинг 11 – PR-AUC и ROC-AUC кривые

|  |
| --- |
| from sklearn.metrics import average\_precision\_score, roc\_auc\_score  y\_scores1 = model1.predict\_proba(X\_test)[:, 1]  pr\_auc1 = average\_precision\_score(y\_test, y\_scores1)  roc\_auc1 = roc\_auc\_score(y\_test, y\_scores1)  print("PR AUC:", pr\_auc1)  print("ROC AUC:", roc\_auc1)  from sklearn.metrics import average\_precision\_score, roc\_auc\_score  y\_scores2 = model2.predict\_proba(X\_test)[:, 1]  pr\_auc2 = average\_precision\_score(y\_test, y\_scores2)  roc\_auc2 = roc\_auc\_score(y\_test, y\_scores2)  print("PR AUC:", pr\_auc2)  print("ROC AUC:", roc\_auc2)  from sklearn.metrics import average\_precision\_score, roc\_auc\_score  y\_scores3 = model3.predict\_proba(X\_test)[:, 1]  pr\_auc3 = average\_precision\_score(y\_test, y\_scores3)  roc\_auc3 = roc\_auc\_score(y\_test, y\_scores3)  print("PR AUC:", pr\_auc3)  print("ROC AUC:", roc\_auc3) |

**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.

Листинг 13 – PR-кривая

|  |
| --- |
| from numpy import argmax  import numpy as np  # Вычисление F1-метрики для каждого порога  precision, recall, thresholds = precision\_recall\_curve(y\_test, y\_scores1)  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)]  print(optimal\_threshold\_pr)  print(gmean)  from numpy import argmax  import numpy as np  # Вычисление F1-метрики для каждого порога  precision, recall, thresholds = precision\_recall\_curve(y\_test, y\_scores2)  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))  print(optimal\_threshold\_pr)  print(gmean)  from numpy import argmax  import numpy as np  # Вычисление F1-метрики для каждого порога  precision, recall, thresholds = precision\_recall\_curve(y\_test, y\_scores3)  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)]  print(optimal\_threshold\_pr)  print(gmean) |

**Результат работы:**

Данную работу можете увидеть в блокноте Google Colab:

<https://colab.research.google.com/drive/1h_RCWf41VokI3QvxyoBZc1Zm3g4-IXzs?usp=sharing>

**Список использованных источников и литературы:**

1. Ростовцев В.С. Искусственные нейронные сети,   
   Издательство "Лань", 2019. — 216 с. — URL: https://e.lanbook.com/book/122180
2. Араки М. Манга: Машинное обучение,   
   Издательство "ДМК Пресс", 2020. — 214 с. — URL: <https://e.lanbook.com/book/179473>
3. Платонов, А. В. Машинное обучение : учебное пособие для вузов / А. В. Платонов. — Москва : Издательство Юрайт, 2022. — 85 с. — (Высшее образование). — ISBN 978-5-534-15561-7. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/508804