Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский Политехнический Университет Петра Великого
—
Институт компьютерных наук и компьютерной безопасности
Высшая школа искусственного интеллекта

КУРСОВАЯ РАБОТА

«Разработка классификаторов для базы данных Heart disease»

по дисциплине «Машинное обучение»

Выполнила: студентка группы
5140201/30301 А.В. Фазылова
<подпись>

Проверил: Л.В. Уткин
д.т.н., профессор
<подпись>

Санкт-Петербург
2024

Содержание

1. Цель работы

Цель работы заключается в изучении и сравнении различных методов классификации и кластеризации на реальных данных, а также в определении наиболее эффективных и значимых признаков для принятия решений в контексте определенной базы данных.

2. Постановка задачи

Для базы данных необходимо:

1. Разработать 3 классификатора и осуществить настройку их параметров для минимизации ошибки классификации на тестовых данных. Выполнить визуализацию данных при помощи метода t-SNE.
2. Сравнить классификаторы (по критерию вероятность ошибки классификации для тестовых данных) и обосновать выбор наилучшего из них.
3. Удалить их базы метки классов и осуществить кластеризацию данных. Построить дендограмму. Сравнить полученные результаты с реальными метками данных. Определить долю ошибочно кластеризованных данных.
4. Используя логистическую регрессию в рамках метода Лассо, определить наиболее значимые признаки, влияющие на отнесение объектов к определенному классу.
5. Использовать автокодер для сокращения размерности или для реализации разреженного скрытого слоя нейронной сети. Преобразовать обучающую выборку при помощи автокодера и осуществить классификацию новых данных с оценкой ошибки классификации. Выполнить визуализацию новых обучающих данных при помощи метода t-SNE. Определить, когда качество классификации лучше, если использовать сокращение размерности или разреженность скрытого слоя. Выполнить классификацию с использованием зашумленного автокодера (denoising autoencoder). Сравнить полученные результаты с пп.1 и 2.

3. Исходные данные и методы работы

В качестве методов классификации были выбраны следующие методы:

1. Метод ближайших соседей.
2. Бэггинг.
3. Бустинг.

В качестве метода кластеризации был выбран метод k-means.

В качестве базы данных была выбрана Heart disease (heart).

База данных Heart disease содержит информацию о наличии сердечного заболевания у пациента. Целевая переменная может принимать два значения: 0 = нет заболевания и 1 = есть заболевание. Также датасет содержит 13 атрибутов.

Информация об атрибутах:

1. возраст
2. пол
3. тип боли в груди (4 значения)
4. кровяное давление в состоянии покоя
5. уровень холестерина в сыворотке крови в мг/дл
6. уровень сахара в крови натощак > 120 мг/дл
7. результаты электрокардиографии в состоянии покоя (значения 0,1,2)
8. достигнутая максимальная частота сердечных сокращений
9. стенокардия, вызванная физической нагрузкой
10. oldpeak = депрессия ST, вызванная физическими упражнениями, по сравнению с отдыхом
11. наклон пикового сегмента ST при нагрузке
12. количество крупных сосудов (0-3), окрашенных при флуороскопии
13. талассемия: 3 = нормальный; 6 = исправленный дефект; 7 = обратимый дефект

270 наблюдений. Пропущенных значений нет

4. Ход работы

Задание 1.

Разработать 3 классификатора и осуществить настройку их параметров для минимизации ошибки классификации на тестовых данных. Выполнить визуализацию данных при помощи метода t-SNE.

Построим при помощи t-SNE все исходные данные (рисунок 1). Синим цветом выделены люди без сердечного заболевания, а зеленым больные.

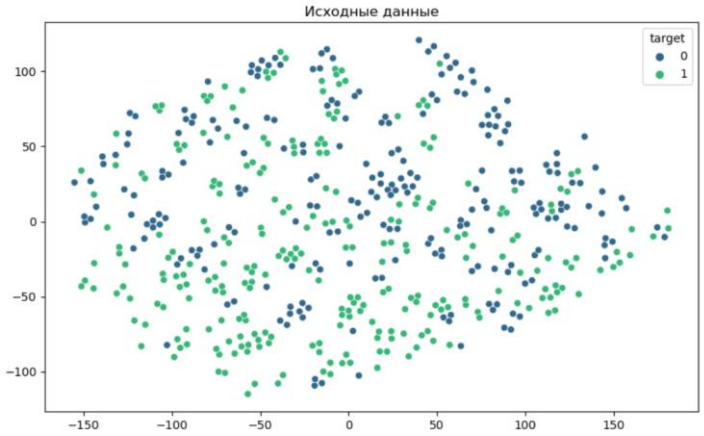

Рисунок 1 – Исходные данные

Метод k-ближайших соседей

Проведем классификацию методом k-ближайших соседей. Найдем оптимальный параметр k и оптимальное ядро.

Построим графики зависимости ошибок от количества k для ядер uniform и distance.

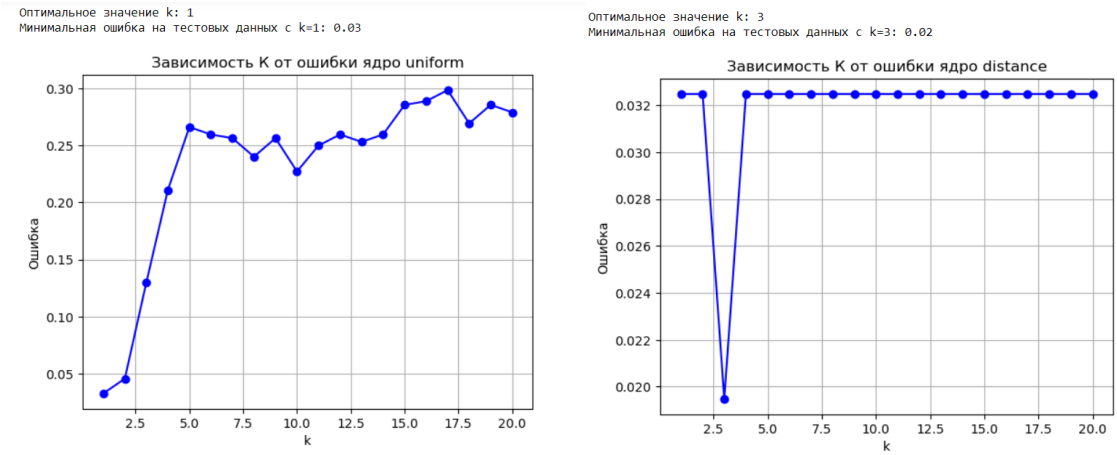

Рисунок 2 – Графики зависимости ошибок от количества k

Наименьшая ошибка 0.02 достигается при ядре distance и количестве k=0.02.

Используя t-SNE, визуализируем модель с оптимальными параметрами, а также приведем исходную тестовую выборку с правильно размеченными данными (рисунок 3).

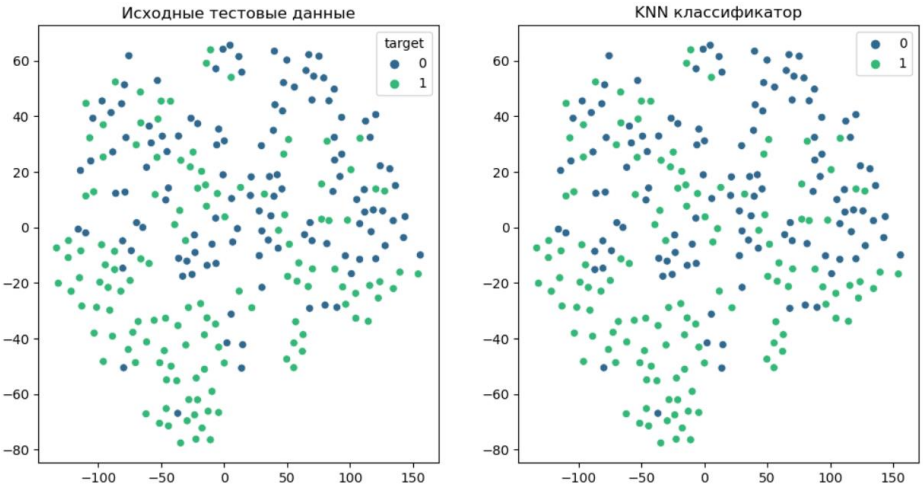

Рисунок 3 – Сравнение классифицированных и исходных данных

Бэггинг классификатор

Проведем классификацию при помощи ансамблевого метода Бэггинг. Подберем для него оптимальное количества базовых моделей (деревьев решений), которые будут обучаться на разных подвыборках данных.

Построим график (рисунок 4) зависимости ошибки от значения параметра n_estimators (количества деревьев).

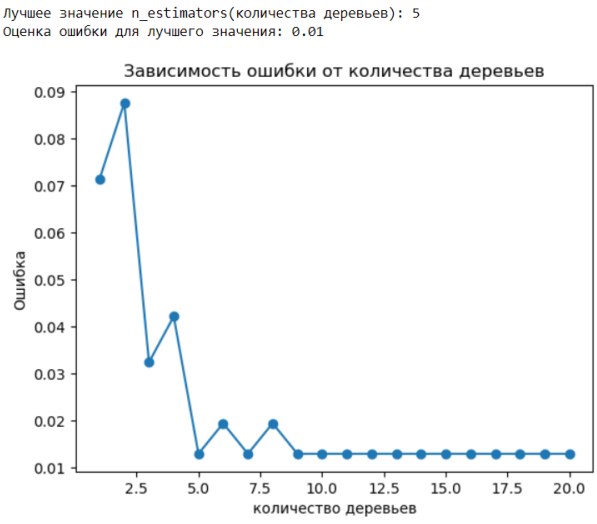

Рисунок 4 – Зависимость количества ошибок от параметра

Оптимальное количество деревьев 5 при нем ошибка равна 0.01.

Построим при помощи t-SNE графики (рисунок 5) сравнения исходных данный и выходных Бэггинг классификатора.

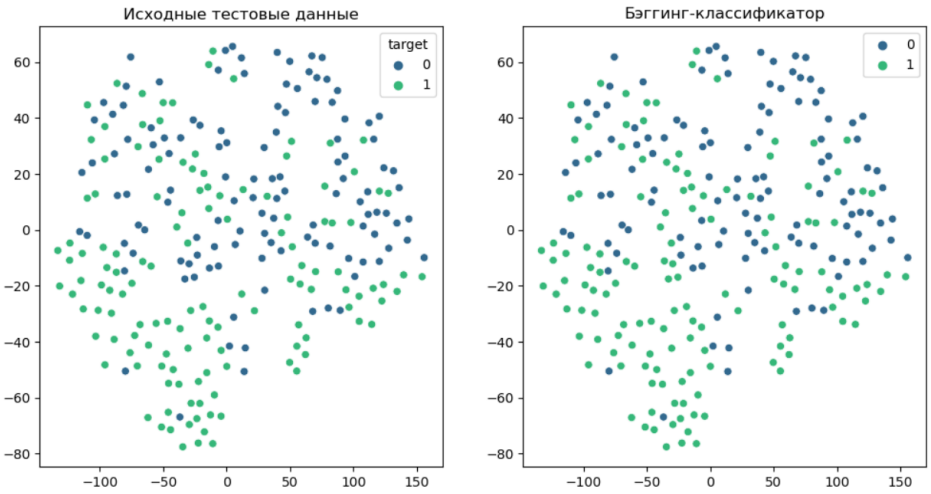

Рисунок 5 – Сравнение классифицированных и исходных данных

Бустинг алгоритм

Проведем классификацию при помощи Бустинг алгоритма. Подберем для него оптимальное количества базовых моделей (деревьев решений), которые в отличие от бэггинга, будут добавляются к композиции последовательно, а не независимо. Каждое новое дерево строится с учетом ошибок, сделанных предыдущими моделями.

Построим график (рисунок 6) зависимости ошибки от значения параметра n_estimators (количества деревьев).

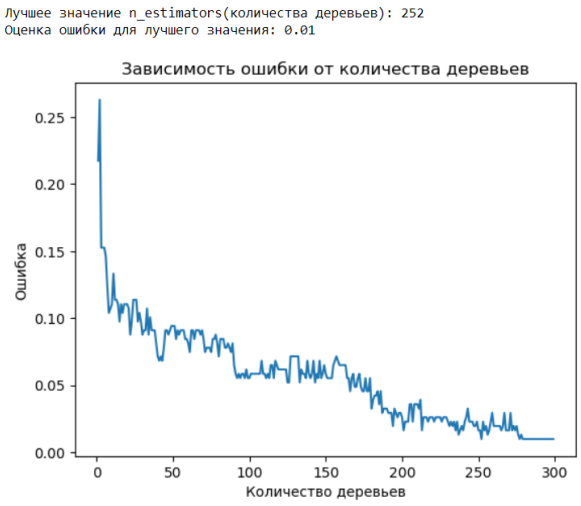

Рисунок 6 – Зависимость количества ошибок от параметра

Количество деревьев, при котором достигается наименьшая ошибка 0.01, равно 252.

Построим при помощи t-SNE графики (рисунок 7) сравнения исходных данный и выходных Бустинг классификатора.

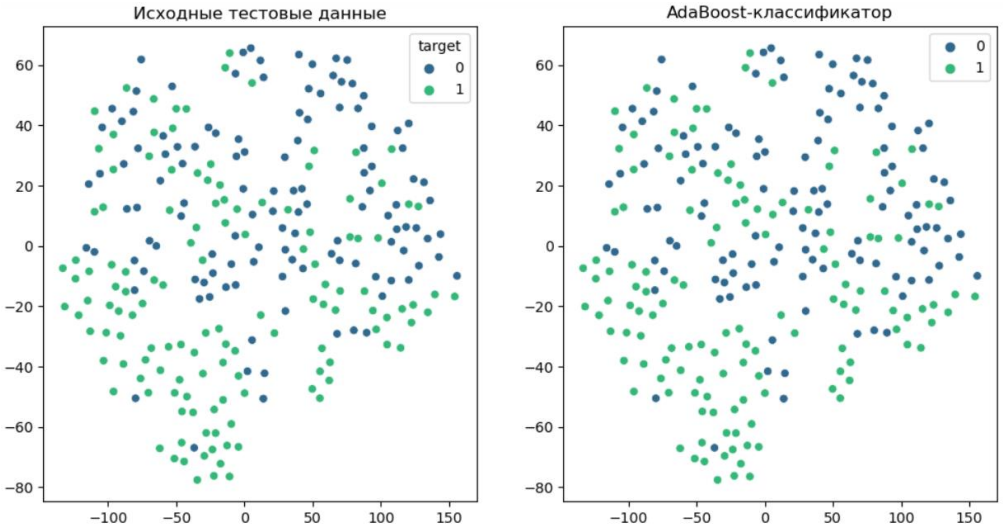

Рисунок 7 – Сравнение классифицированных и исходных данных

Задание 2.

Сравнить классификаторы (по критерию вероятность ошибки классификации для тестовых данных) и обосновать выбор наилучшего из них.

Для оптимальных параметров моделей ошибки составляют.

Ошибка для KNN (при к =3): 0.02

Ошибка для Бэггинг алгоритма (при количестве деревьев 5): 0.01

Ошибка для Бустинг алгоритма (при количестве деревьев 252): 0.01

Исходя из полученных данных лучшим классификатором можно отметить Бэггинг классификатор, так как с небольшим значением параметра, он смог достигнуть минимальной ошибки. Несмотря на то, что ошибки у Бустинг и Бэггинг алгоритма одинаковы, Бустинг алгоритм затратил большое количество времени на его реализацию и потребовал существенное количество деревьев.

Задание 3.

Удалить их базы метки классов и осуществить кластеризацию данных. Построить дендрограмму. Сравнить полученные результаты с реальными метками данных. Определить долю ошибочно кластеризованных данных.

Удалим метки классов из базы и для кластеризации воспользуемся методом k-means. Ошибка с использованием данного метода составила 0.43, что можно охарактеризовать плохим результатом. Однако, посмотрев на данные, можно сказать, что они довольно хаотично расположены и выделить кластеры в них действительно довольно сложно. Посмотрим на визуализации сравнений полученных результатов с реальными метками данных (рисунок 8).

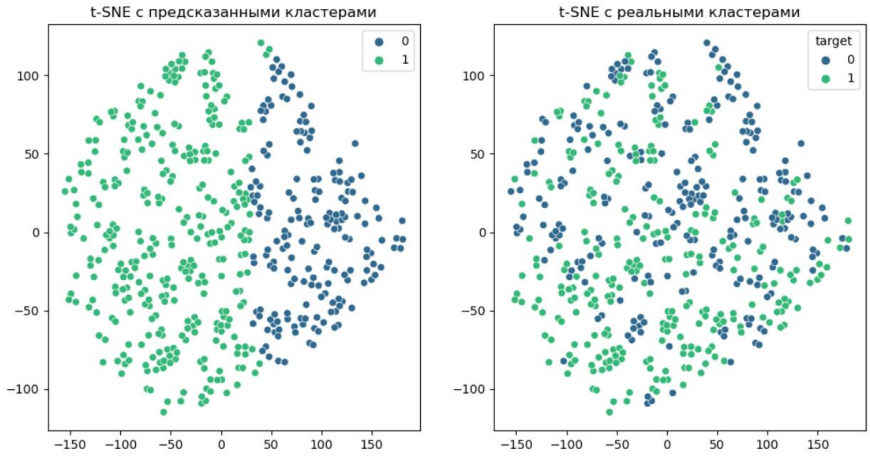

Рисунок 8 – Сравнение предсказанных и реальных данных

Также в ходе работы была построена дендрограмма (рисунок 9)

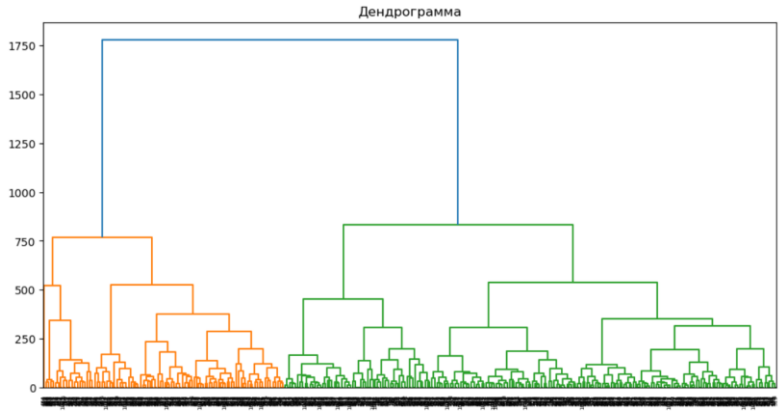

Рисунок 9 – Дендрограмма

Задание 4.

Используя логистическую регрессию в рамках метода Лассо, определить наиболее значимые признаки, влияющие на отнесение объектов к определенному классу.

Построим регрессию методом Лассо и визуализируем значимость признаков (рисунок 10).

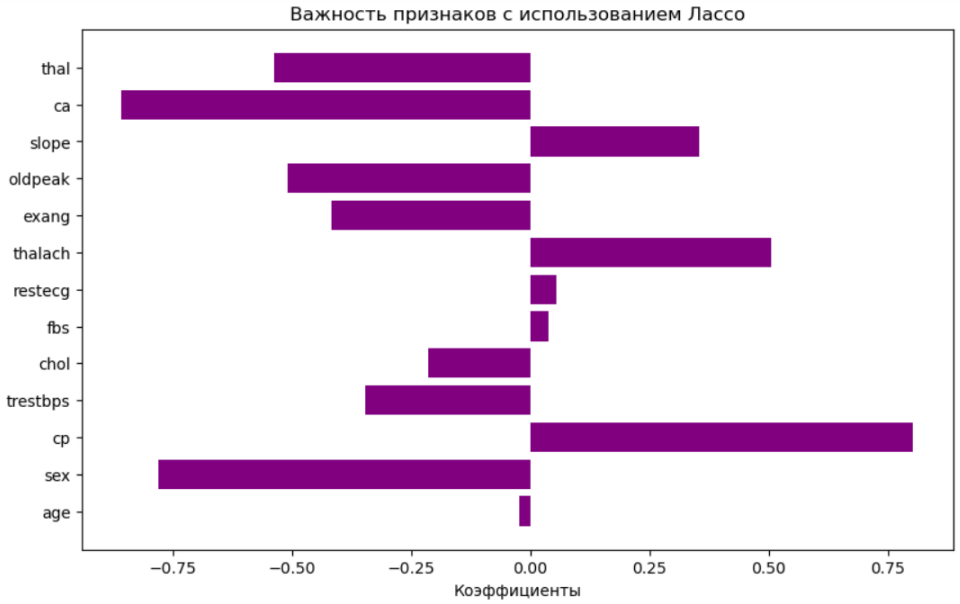

Рисунок 10 – Значимость признаков

На отнесение к классу не больных наибольшее влияние оказывают признаки ca (количество магистральных сосудов, окрашенных флюороскопией) и пол, а на класс заболевших наибольшее влияние оказывает параметр cp (тип боли в груди).

Задание 5.

Использовать автокодер для сокращения размерности или для реализации разреженного скрытого слоя нейронной сети. Преобразовать обучающую выборку при помощи автокодера и осуществить классификацию новых данных с оценкой ошибки классификации. Выполнить визуализацию новых обучающих данных при помощи метода t-SNE. Определить, когда качество классификации лучше, если использовать сокращение размерности

или разреженность скрытого слоя. Выполнить классификацию с использованием зашумленного автокодера (denoising autoencoder). Сравнить полученные результаты с пп.1 и 2.

Для реализации автокодеров будем использовать библиотеку h2o.

Автокодер для сокращения размерности

Используем автокодер для сокращения размерности, преобразуя обучающую выборку при помощи него (рисунок 11), а также осуществим классификацию новых данных при помощи модели AdaBoost с параметром по умолчанию.

Рисунок 11 – Тренировочные данные обработанные автокодером для сокращения размерности

Ошибка Бустинг классификации данных составила 0.029

Автокодер с разряженным скрытым слоем

Далее реализуем разреженный скрытый слой нейронной сети, для этого зададим параметр регуляризации L1= 1e-4. Полученные данные представлены на рисунке 12.

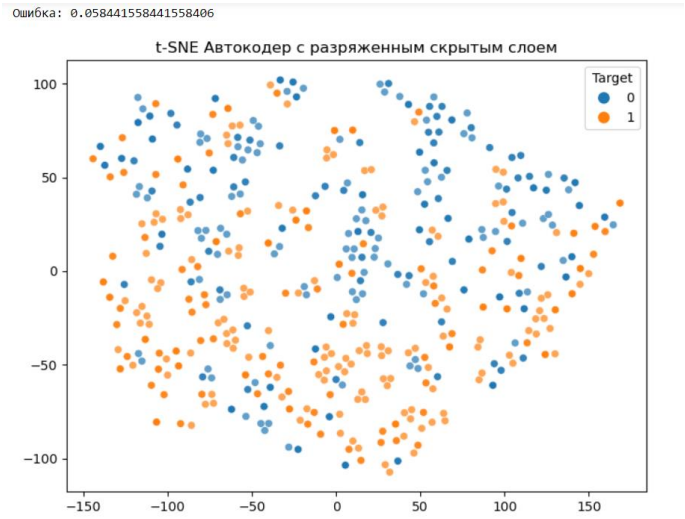

Рисунок 11 – Тренировочные данные обработанные автокодером с разряженным скрытым слоем

Ошибка Бустинг классификатора (с параметрами по умолчанию) на полученных данных составляет 0.0584.

Зашумленный автокодер

Построим зашумленный автокодер, визуализация его данных представлена на рисунке 12.

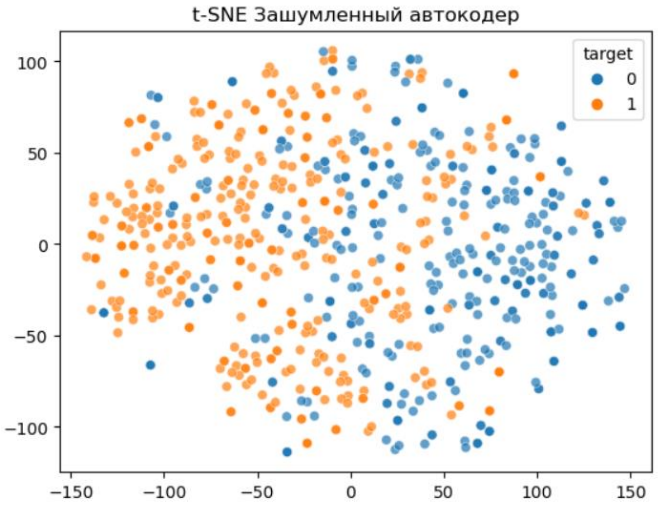

Рисунок 11 – Тренировочные данные обработанные зашумленным автокодером

Ошибка Бустинг классификатора (с параметрами по умолчанию) на полученных данных составила 0.0519

Сравним ошибки, полученные в результате Бустинг классификации после использования автокодеров на тренировочных данных.

Ошибка для автокодера сокращения размерности: 0.029.

Ошибка для автокодера с разряженным слоем: 0.0584.

Ошибка для шумоподавляющего автокодера: 0.0519.

Автокодер сокращения размерности показал себя лучше всех, затем идет зашумленный автокодер, хуже всех показал себя автокодер с разряженным слоем.

5. Выводы

В результате работы были выполнены все поставленные цели и задачи. Используя набор данных по заболеваниям сердца Heart disease, были достигнуты следующие результаты.

— Были реализованы классификаторы KNN, Bagging, Busting. Лучший результат из которых показал классификатор на основе Бэггинг алгоритма, достигнув точности 0.01

— На основе k-means был построен кластеризатор, однако точность его оказалась не высокой из-за повышенной хаотичности данных.

— При помощи регрессии Лассо удалось оценить наиболее влиятельные параметры модели, такие как количество магистральных сосудов, окрашенных флюороскопией, пол и тип боли в груди.

— Были реализованы автокодеры для сокращения размерности, с разряженным скрытым слоем и зашумленный. Лучший результат из которых показал автокодер для сокращения размерности.

6. Листинг кода

Задание 1.

```
#Импорт
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
import seaborn as sns
import h2o
from sklearn.manifold import TSNE
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.cluster import KMeans
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.metrics import accuracy_score
from sklearn.datasets import fetch_openml
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import adjusted_rand_score
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from h2o.estimators.deeplearning import H2OAutoEncoderEstimator
###### Инициализация данных
data = pd.read_csv("heart.csv", sep=',')
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
# Инициализация t-SNE
tsne = TSNE(n_components=2, random_state=7567)
plt.figure(figsize=(10, 6))
X_tsne = tsne.fit_transform(X)
```

```
sns.scatterplot(x=X_tsne[:, 0], y=X_tsne[:, 1], hue=y, palette='viridis', legend='full')
plt.title('Исходные данные')
plt.show()
# Разделим на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, random_state=7567, stratify=y)
#######метод k-ближайших соседей
k_values = list(range(1, 21))
error_values = []
best_k = None
best_error = float('inf')
for k in k_values:
    knn_classifier = KNeighborsClassifier(n_neighbors=k, weights='uniform')
    knn_classifier.fit(X_train, y_train)
    y_pred = knn_classifier.predict(X_test)
    error = 1 - accuracy_score(y_test, y_pred)
    if error < best_error:
        best_error = error
        best_k = k
    error_values.append(error)
print(f"Оптимальное значение k: {best_k}")
print(f"Минимальная ошибка на тестовых данных с k={best_k}: {best_error:.2f}")
plt.plot(k_values, error_values, marker='o', linestyle='-', color='b')
plt.title('Зависимость К от ошибки ядро uniform')
plt.xlabel('k')
plt.ylabel('Ошибка')
plt.grid(True)
plt.show()

k_values = list(range(1, 21))
error_values = []
best_k = None
best_error = float('inf')
```

```
for k in k_values:
    knn_classifier = KNeighborsClassifier(n_neighbors=k, weights='distance')
    knn_classifier.fit(X_train, y_train)
    y_pred = knn_classifier.predict(X_test)
    error = 1 - accuracy_score(y_test, y_pred)
    if error < best_error:
        best_error = error
        best_k = k
    error_values.append(error)
print(f"Оптимальное значение k: {best_k}")
print(f"Минимальная ошибка на тестовых данных с k={best_k}: {best_error:.2f}")
plt.plot(k_values, error_values, marker='o', linestyle='-', color='b')
plt.title('Зависимость К от ошибки ядро distance')
plt.xlabel('k')
plt.ylabel('Ошибка')
plt.grid(True)
plt.show()

knn_classifier = KNeighborsClassifier(n_neighbors=3, weights='distance')
knn_classifier.fit(X_train, y_train)
y_pred = knn_classifier.predict(X_test)
# Применение t-SNE для визуализации данных
tsne = TSNE(n_components=2, random_state=7567)
X_tsne = tsne.fit_transform(X)
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
X_test_tsne = tsne.fit_transform(X_test)
sns.scatterplot(x=X_test_tsne[:, 0], y=X_test_tsne[:, 1], hue=y_test, palette='viridis', legend='full')
plt.title('Исходные тестовые данные')
plt.subplot(1, 2, 2)
X_test_tsne = tsne.fit_transform(X_test)
sns.scatterplot(x=X_test_tsne[:, 0], y=X_test_tsne[:, 1], hue=y_pred, palette='viridis', legend='full')
```

```
plt.title('KNN классификатор')
plt.show()

####Бэггинг
# Создание и обучение бэггинг-классификатора
n_estimators_range = range(1, 21)
errors = []
for n_estimators in n_estimators_range:
    bagging_classifier = BaggingClassifier(n_estimators=n_estimators, random_state=7567)
    bagging_classifier.fit(X_train, y_train)
    y_pred = bagging_classifier.predict(X_test)
    error = 1 - accuracy_score(y_test, y_pred)
    errors.append(error)
best_n_estimators = n_estimators_range[np.argmin(errors)]
best_error = min(errors)
print(f"Лучшее значение n_estimators(количества деревьев): {best_n_estimators}")
print(f"Оценка ошибки для лучшего значения: {best_error:.2f}")
plt.plot(n_estimators_range, errors, marker='o')
plt.title('Зависимость ошибки от количества деревьев')
plt.xlabel('количество деревьев')
plt.ylabel('Ошибка')
plt.show()
best_bagging_classifier = BaggingClassifier(n_estimators=best_n_estimators, random_state=7567)
best_bagging_classifier.fit(X_train, y_train)
y_pred_best = best_bagging_classifier.predict(X_test)
# Применение t-SNE для визуализации данных
tsne = TSNE(n_components=2, random_state=7567)
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
X_test_tsne = tsne.fit_transform(X_test)
sns.scatterplot(x=X_test_tsne[:, 0], y=X_test_tsne[:, 1], hue=y_test, palette='viridis', legend='full')
plt.title('Исходные тестовые данные')
```

```
plt.subplot(1, 2, 2)

X_test_tsne_best = tsne.fit_transform(X_test)

sns.scatterplot(x=X_test_tsne_best[:, 0], y=X_test_tsne_best[:, 1], hue=y_pred_best, palette='viridis', legend='full')

plt.title('Бэггинг-классификатор')

plt.show()


####Бустинг

print(f"Лучшее значение n_estimators(количества деревьев): {best_n_estimators}")

print(f"Оценка ошибки для лучшего значения: {best_error:.2f}")

plt.plot(n_estimators_range, errors)

plt.title('Зависимость ошибки от количества деревьев')

plt.xlabel('Количество деревьев')

plt.ylabel('Ошибка')

plt.show()

# Создание и обучение AdaBoost-классификатора с лучшим значением n_estimators

best_adaboost_classifier = AdaBoostClassifier(n_estimators=best_n_estimators, random_state=7567)

best_adaboost_classifier.fit(X_train, y_train)

y_pred_best = best_adaboost_classifier.predict(X_test)

# Применение t-SNE для визуализации данных

tsne = TSNE(n_components=2, random_state=7567)

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

X_test_tsne = tsne.fit_transform(X_test)  # Применение t-SNE к тестовым данным

sns.scatterplot(x=X_test_tsne[:, 0], y=X_test_tsne[:, 1], hue=y_test, palette='viridis', legend='full')

plt.title('Исходные тестовые данные')


plt.subplot(1, 2, 2)

X_test_tsne_best = tsne.fit_transform(X_test)

sns.scatterplot(x=X_test_tsne_best[:, 0], y=X_test_tsne_best[:, 1], hue=y_pred_best, palette='viridis', legend='full')

plt.title('AdaBoost-классификатор')

plt.show()
```

Задание 3.

```
# Кластеризация данных методом k-средних
kmeans = KMeans(n_clusters=2)
predicted_clusters = kmeans.fit_predict(X)
errors = (predicted_clusters != y).sum() / len(y)
print(f'Ошибка: {errors:.2f}')

# Построение дендрограммы
linkage_matrix = linkage(X, method='ward')
plt.figure(figsize=(12, 6))
dendrogram(linkage_matrix)
plt.title('Дендрограмма')
plt.show()

# Применение t-SNE для визуализации данных
tsne = TSNE(n_components=2, random_state=7567)
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
X_tsne_predicted = tsne.fit_transform(X)
sns.scatterplot(x=X_tsne_predicted[:, 0], y=X_tsne_predicted[:, 1], hue=predicted_clusters, palette='viridis', legend='full')
plt.title('t-SNE с предсказанными кластерами')

plt.subplot(1, 2, 2)
sns.scatterplot(x=X_tsne_predicted[:, 0], y=X_tsne_predicted[:, 1], hue=y, palette='viridis', legend='full')
plt.title('t-SNE с реальными кластерами')
plt.show()
```

Задание 4.

```
# Масштабирование признаков

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Модель
lasso_model = LogisticRegression(penalty='l1', solver='liblinear')
lasso_model.fit(X_train_scaled, y_train)

# Коэффициенты
feature_names = X.columns
coefficients = lasso_model.coef_[0]

# Построение графика важности признаков
plt.figure(figsize=(10, 6))
plt.barh(feature_names, coefficients, color='purple')
plt.xlabel('Коэффициенты')
plt.title('Важность признаков с использованием Лассо')
plt.show()
```

Задание 5.

```
###Автокодер для понижения размерности
data = pd.read_csv("heart.csv", sep=',')
X = data.drop('target', axis=1)
y = data['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, random_state=7567, stratify=y)

h2o.init()
h2o_train = h2o.H2OFrame(pd.concat([X_train, y_train.to_frame()], axis=1))

autoencoder = H2OAutoEncoderEstimator(activation="Tanh",autoencoder=True)
autoencoder.train(x=list(range(X_train.shape[1])), training_frame=h2o_train)

encoded_train = autoencoder.predict(h2o_train)
encoded_train = encoded_train.as_data_frame().values

adaboost_classifier = AdaBoostClassifier()
adaboost_classifier.fit(encoded_train, y_train)

encoded_test = autoencoder.predict(h2o.H2OFrame(X_test)).as_data_frame().values
accuracy = adaboost_classifier.score(encoded_test, y_test)
print("ошибка:", 1-accuracy)

tsne = TSNE(n_components=2, random_state=7567)
encoded_tsne = tsne.fit_transform(encoded_train)

plt.figure(figsize=(8, 6))
sns.scatterplot(x=encoded_tsne[:, 0], y=encoded_tsne[:, 1], hue=y_train, palette={0: '#1f77b4', 1: '#ff7f0e'}, alpha=0.7)
plt.title('t-SNE Автокодер для сокращения размерности')
plt.legend(title='Target', loc='best', labels=['0', '1'],
        handles=[plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=color, markersize=10) for color in ['#1f77b4', '#ff7f0e']])
plt.show()

###Автокодер со скрытым слоем
data = pd.read_csv("heart.csv", sep=',')
X = data.drop('target', axis=1)
y = data['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, random_state=7567, stratify=y)

h2o.init()
h2o_train = h2o.H2OFrame(pd.concat([X_train, y_train.to_frame()], axis=1))

# Инициализация автокодера с разреженным скрытым слоем
autoencoder = H2OAutoEncoderEstimator(activation="Tanh", autoencoder=True, l1=1e-4)  # Регуляризация L1 для разреженности
autoencoder.train(x=list(range(X_train.shape[1])), training_frame=h2o_train)

encoded_train = autoencoder.predict(h2o_train)
encoded_train = encoded_train.as_data_frame().values
```

```
adaboost_classifier = AdaBoostClassifier()
adaboost_classifier.fit(encoded_train, y_train)

encoded_test = autoencoder.predict(h2o.H2OFrame(X_test)).as_data_frame().values
accuracy = adaboost_classifier.score(encoded_test, y_test)
print("Ошибка:", 1-accuracy)

# Визуализация новых обучающих данных при помощи t-SNE
tsne = TSNE(n_components=2, random_state=7567)
encoded_tsne = tsne.fit_transform(encoded_train)

plt.figure(figsize=(8, 6))
sns.scatterplot(x=encoded_tsne[:, 0], y=encoded_tsne[:, 1], hue=y_train, palette={0: '#1f77b4', 1: '#ff7f0e'}, alpha=0.7)
plt.title('t-SNE Автокодер с разряженным скрытым слоем')
plt.legend(title='Target', loc='best', labels=['0', '1'],
        handles=[plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=color, markersize=10) for color in ['#1f77b4', '#ff7f0e']])
plt.show()


###Зашумленный автокодер
data = pd.read_csv("heart.csv", sep=',')

# Часть данных которая будет зашумлена
subset_size = 100
subset_indices = np.random.choice(data.shape[0], size=subset_size, replace=False)

X = data.drop('target', axis=1)
y = data['target']

X_subset = X.iloc[subset_indices]
y_subset = y.iloc[subset_indices]

noise_subset = np.random.normal(0, 1, size=(subset_size, X.shape[1]))
X_subset_with_noise = X_subset + noise_subset

X_with_noise = X.copy()
X_with_noise.iloc[subset_indices] = X_subset_with_noise

X_train, X_test, y_train, y_test = train_test_split(X_with_noise, y, train_size=0.7, random_state=7567, stratify=y)

h2o.init()
h2o_train = h2o.H2OFrame(pd.concat([X_train, y_train.to_frame()], axis=1))

autoencoder = H2OAutoEncoderEstimator(activation="Tanh",autoencoder=True)
autoencoder.train(x=list(range(X_train.shape[1])), training_frame=h2o_train)

sparse_layer_train = autoencoder.deepfeatures(h2o_train, layer=1)
sparse_layer_train = sparse_layer_train.as_data_frame().values

tsne = TSNE(n_components=2, random_state=7567)
```

```
sparse_tsne = tsne.fit_transform(sparse_layer_train)

sns.scatterplot(x=sparse_tsne[:, 0], y=sparse_tsne[:, 1], hue=y_train, alpha=0.7)
plt.title('t-SNE Зашумленный автокодер')
plt.show()

adaboost_classifier = AdaBoostClassifier()
adaboost_classifier.fit(sparse_layer_train, y_train)

sparse_layer_test = autoencoder.deepfeatures(h2o.H2OFrame(X_test), layer=1)
sparse_layer_test = sparse_layer_test.as_data_frame().values
accuracy = adaboost_classifier.score(sparse_layer_test, y_test)
print("Ошибка:", 1-accuracy)
```