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

КУРСОВАЯ РАБОТА

по дисциплине «Машинное обучение, часть1»

Выполнил: студент группы
5140201/30301 В.А. Ефременко
<подпись>

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

Санкт-Петербург
2023

1. Цель работы

Курсовой проект заключается в разработке классификаторов для реальной базы данных, визуализации данных, исследовании и настройки классификаторов.

2. Формулировка задания

Для выбранной 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.

6. Подготовить пояснительную записку по курсовому проекту и листинги программ.

3. Используемые данные и методы

В качестве датасета была выбрана база данных – Wisconsin Diagnostic Breast Cancer. Набор данных представляет собой результаты анализа биопсийных образцов молочных желез. Задача анализа заключается в диагностике рака молочной железы на основе характеристик ядер клеток, выделенных из дигитализированных изображений тонких игл. Основная информация о датасете:

- Предсказание: B – доброкачественная (benign), M – злокачественная (malignant).
- Множества линейно разделимы с использованием всех 30 входных признаков.
- Признаки вычисляются на основе цифрового изображения тонкой иглы аспирата молочной железы (FNA) и описывают характеристики ядер клеток, присутствующих на изображении.
- Число экземпляров: 569.
- Число атрибутов: 32 (ID, диагноз, веществennозначные признаки).
- В датасете отсутствуют пропущенные значения.
- Распределение классов: 357 – benign, 212 – malignant.

Для классификации данных были выбраны следующие классификаторы:

- Наивный байесовский классификатор
- Метод ближайших соседей
- Бустинг

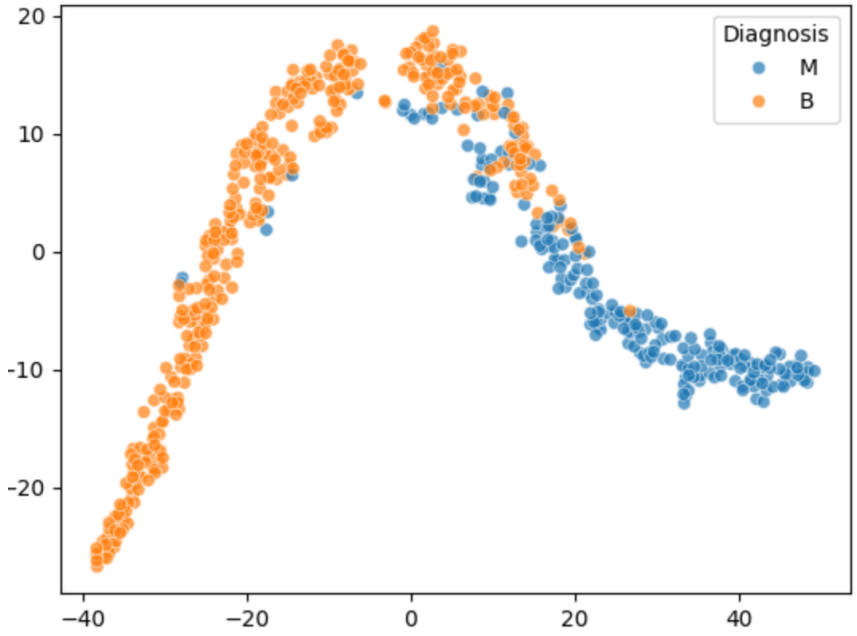
Метод кластеризации: метод k-средних.

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.

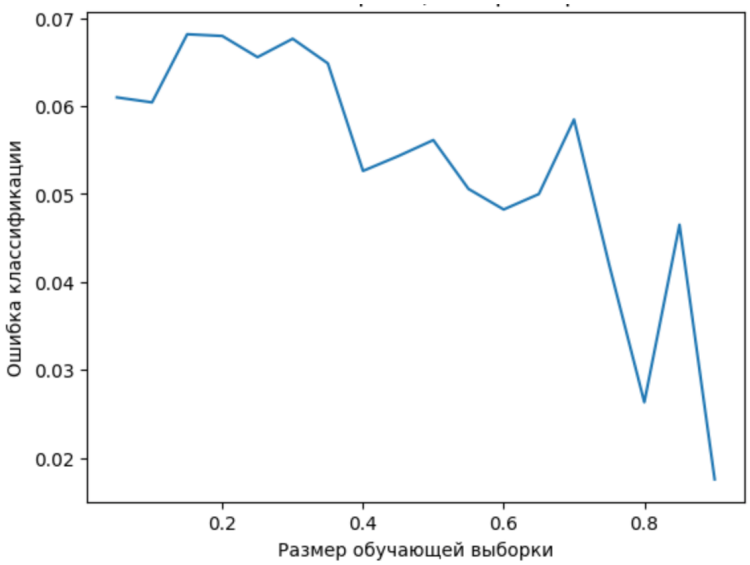
Выполним визуализацию данных при помощи метода t-SNE:



Видим, что классы имеют некоторые пересечения, но при этом визуально они хорошо разделимы.

Наивный байесовский классификатор

Для оценки работы байесовского классификатора брались разные доли обучающей выборки от общего объема данных. Была получена следующая зависимость ошибки классификации от размера обучающей выборки:



Соответственно, видим, что с увеличением обучающей выборки, ошибка классификации уменьшается и повышается точность классификации.

Наименьшая полученная ошибка классификации: 0.0195.

Метод ближайших соседей

Для подбора лучшей модели k-ближайших соседей был проведен эксперимент с разными настройками параметров: k – количества соседей, kernel – ядра, metric – метрики расстояния и distance – расстояния.

Таким образом, наибольшую точность достигла модель с параметрами:

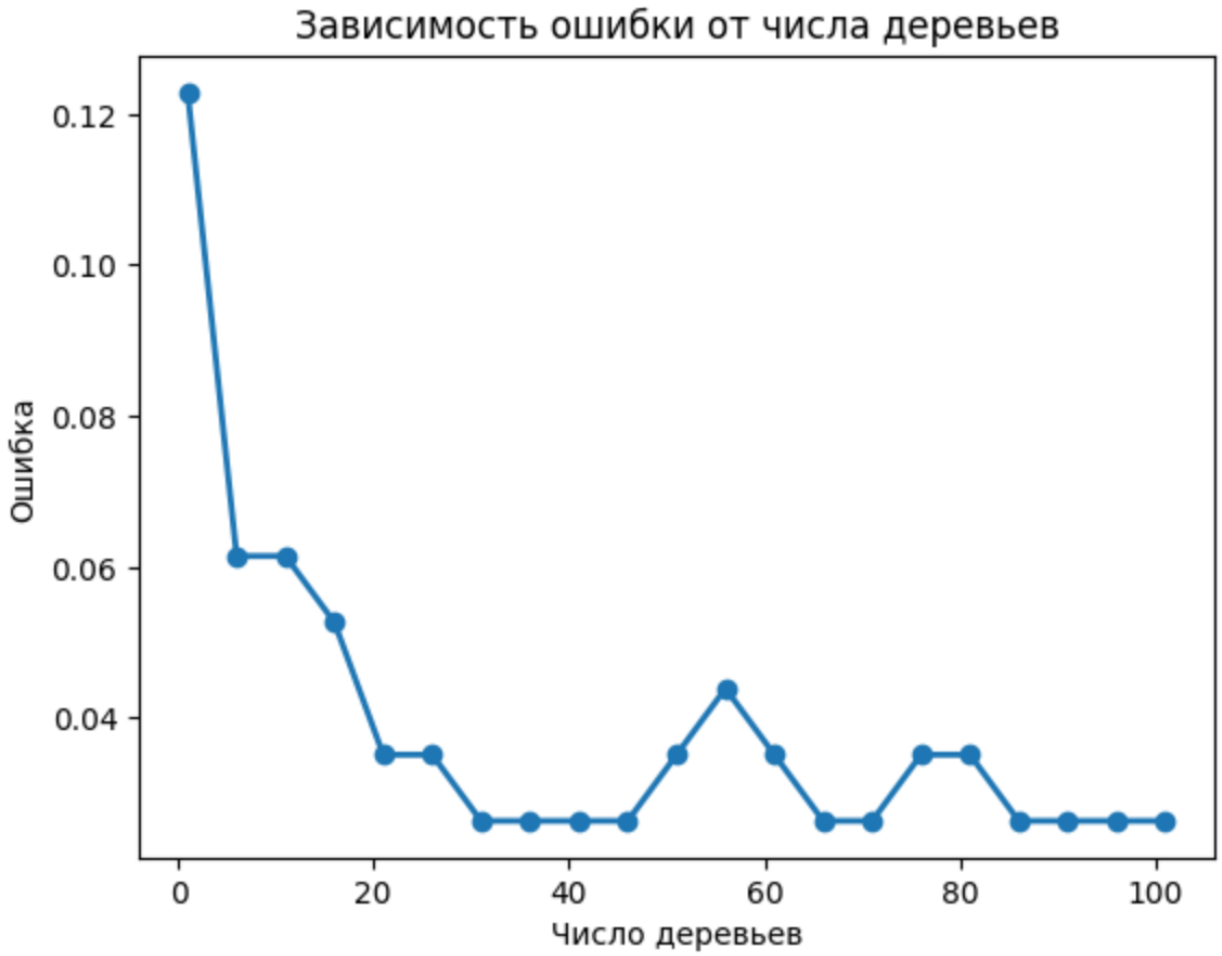
```
{'k': 3, 'kernel': 'uniform', 'distance': 1, 'metric': 'euclidean'}
```

Точность лучшей модели: 0.9211.

Бустинг

Для подбора лучшей модели алгоритма бустинга оценивалась ошибка для разного числа деревьев, которое менялось от 1 до 101 с шагом 5. Средняя ошибка моделей была 0.0388.

Наименьшая ошибка была получена для моделей с числом деревьев 31, 36, 41, 46, 66, 71, 86, 91, 96, 101. Наилучшей будем считать модель с числом деревьев 31. Наименьшая полученная ошибка: 0.0263



Задание 2

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

Ошибки наилучших моделей:

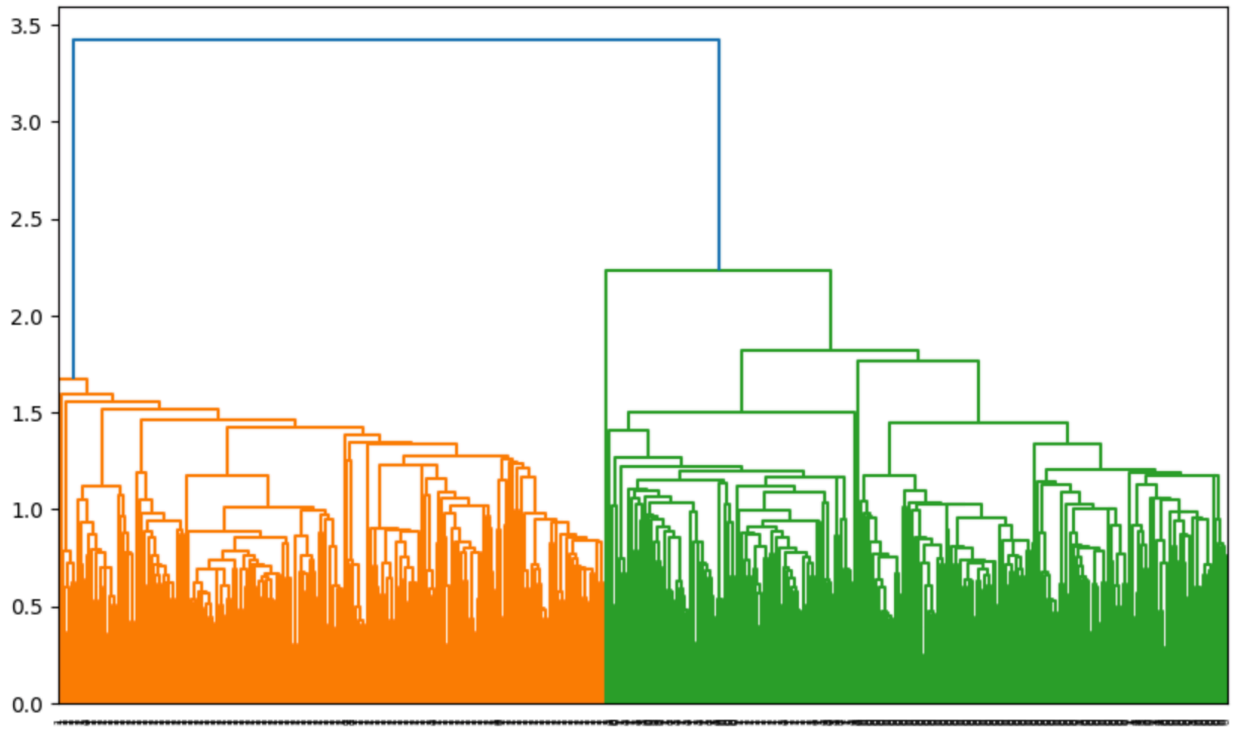
- Наивный байесовский классификатор: 0.0195
- Метод ближайших соседей: 0.0789
- Бустинг: 0.0263

Наилучшим оказался наивный байесовский классификатор, показав наименьшую ошибку классификации на тестовых данных.

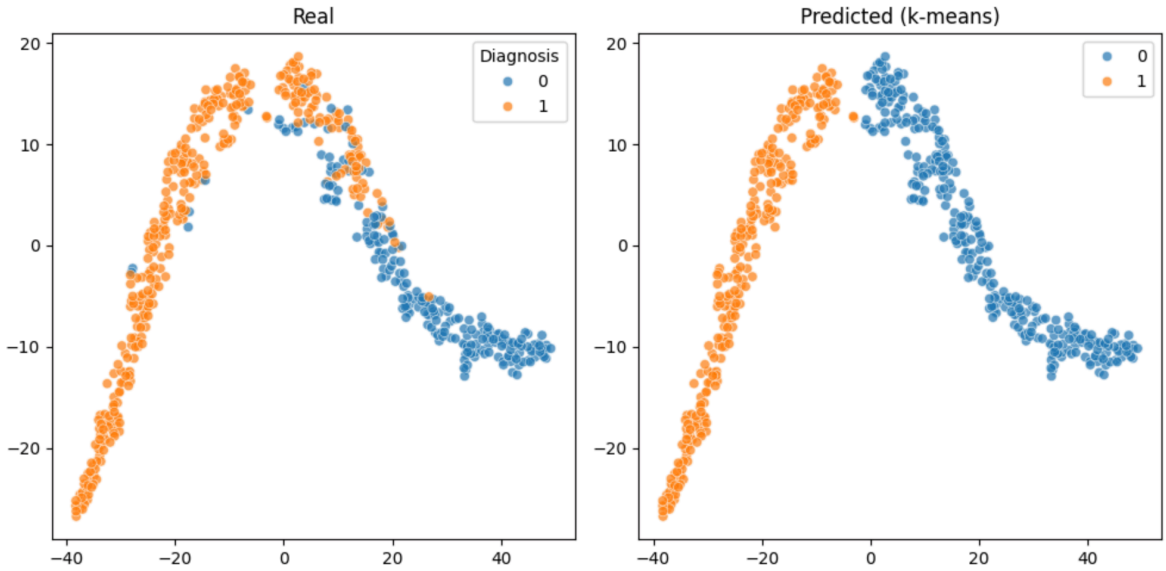
Задание 3

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

После удаления меток классов была построена дендрограмма:



Затем была проведена кластеризация данных методом k-средних. Ошибка, полученная с помощью данного метода, составила 17.75%. Ниже представлены графики сравнения данных с исходными метками классов и предсказанными.



Видим, что результат довольно неплохой, но модель плохо справляется с пересекающимися участками классов.

Задание 4

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

Определим наиболее значимые признаки и выявим неинформативные, которые лишь мешают работе модели. Сначала используем метод LassoCV для нахождения оптимального параметра alpha = 0.20182966045941297. Затем методом Лассо получим информацию о значимости признаков. Отсортировав признаки по значениям коэффициентов β, получили следующую градацию: признаки 24, 23, 5, 25 (именно в таком порядке) наиболее важные, все же остальные признаки не вносят значительного вклада в модель.

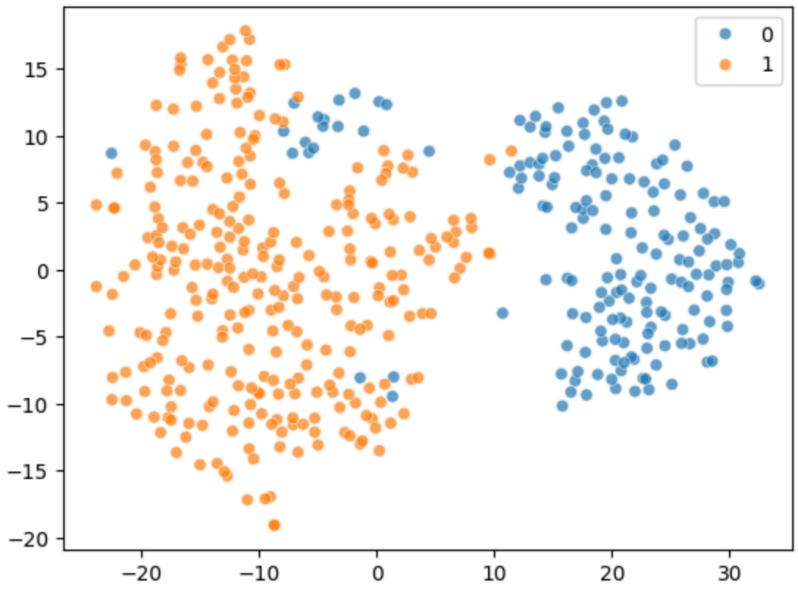
Задание 5

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.

Использование автокодера для сокращения размерности

Создадим модель автокодера с помощью функции H20DeepLearningEstimator из библиотеки h2o. Чтобы извлечь данные уменьшенной размерности, воспользуемся функцией deepfeatures().

Визуализация данных сокращенной размерности представлена на рисунке ниже:



Видим, что данные неплохо различимы, но всё еще присутствуют пересечения классов, что повлияет на точность классификации.

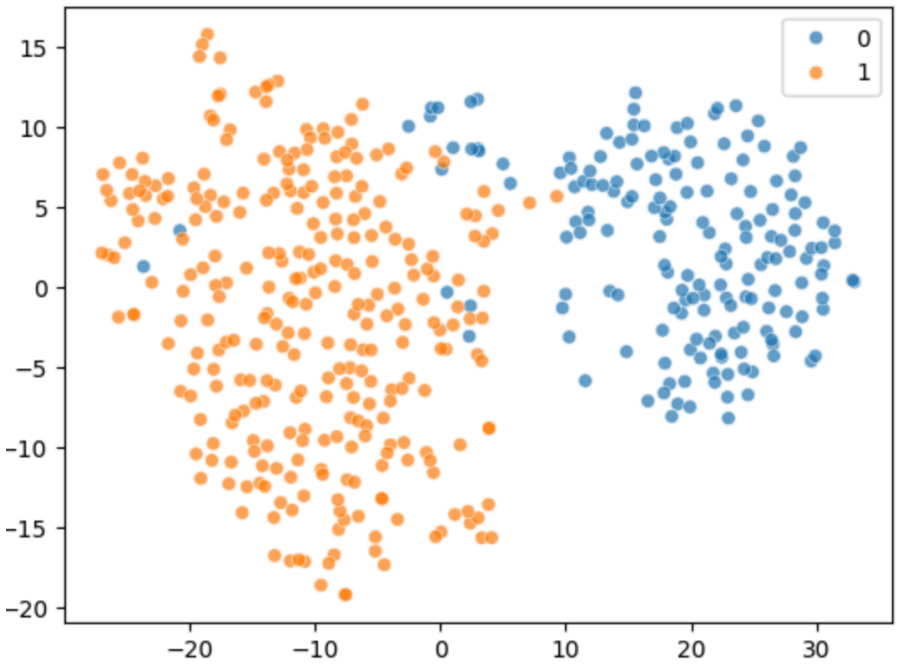
С помощью алгоритма бустинга были получены следующие результаты классификации – ошибка составила 0.0216, что немногим меньше, чем ошибка до уменьшения размерности.

Использование автокодера для реализации разреженного скрытого слоя

Автокодер с разреженным скрытым слоем используется для извлечения более информативных и компактных представлений данных. Разреженный скрытый слой в автокодере означает, что только небольшое количество нейронов в этом слое активированы для каждого входного примера, что делает его представление более разреженным.

Разреженные представления позволяют автокодеру извлекать только наиболее важные признаки из данных.

Ниже представлена визуализация данных:

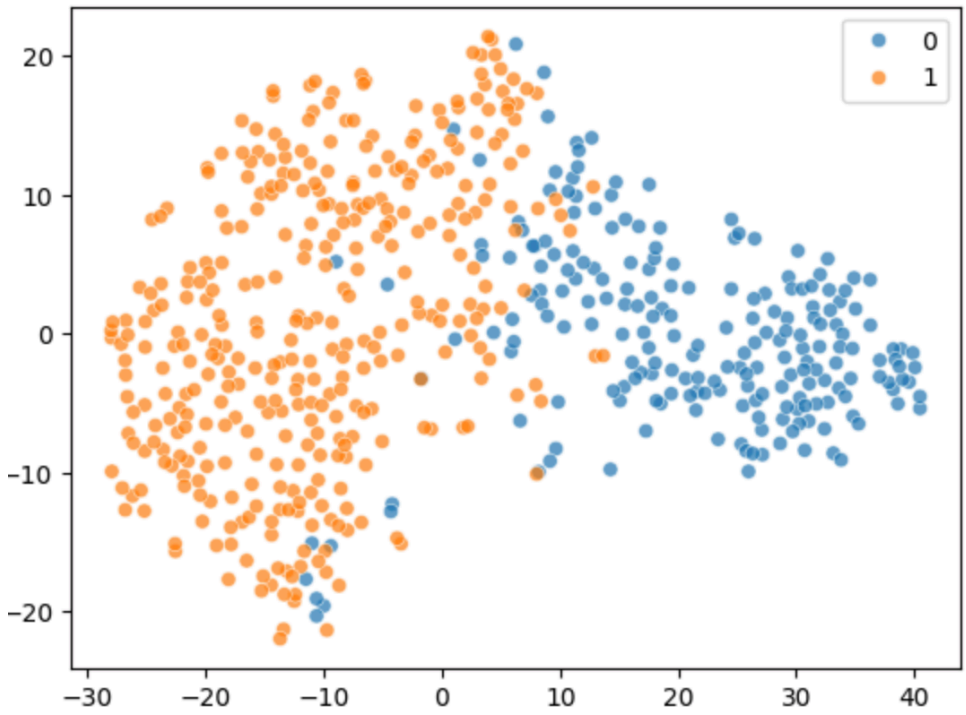


С помощью алгоритма бустинга были получены следующие результаты классификации – ошибка составила 0.01075. С использованием разреженности скрытого слоя получили лучшее качество классификации, чем с использованием сокращения размерности.

Использование зашумленного автокодера

Шумоподавляющий автокодер (Denoising Autoencoder) — это тип автокодера, который обучается удалять шум из входных данных. Основная цель шумоподавляющего автокодера - изучение более устойчивых и информативных представлений данных путем выделения существенных признаков и подавления влияния шумовых компонентов, поэтому для изучения его влияния предварительно зашумляем исходные данные путем добавления случайного шума.

Ниже представлена визуализация данных после прохождения зашумленным автокодером:



С помощью алгоритма бустинга были получены следующие результаты классификации – ошибка составила 0.0526. Шумоподавляющий автокодер дает самый низкий результат.

Сравнивая полученные результаты, можно сделать вывод, что на данном наборе данных лучше работать с разряженным автокодером.

5. Выводы

В данной работе для базы данных «Wisconsin Diagnostic Breast Cancer»:

1. Реализованы 3 классификатора, такие как наивный байесовский классификатор, k-ближайших соседей и бустинг. Для данных классификаторов были подобраны параметры для получения наименьшей ошибки классификации. Наилучшим классификатором оказался наивный байесовский.
2. Реализован метод кластеризации k-средних. Точность кластеризации составляет 82.25%, что является неплохим результатом.
3. Осуществлена градация признаков, влияющих на отнесение объектов к определенному классу, с помощью логистической регрессии в рамках метода Лассо.
4. Исследовано применение разных автокодеров. В качестве метода классификации новых данных был использован алгоритм бустинга с выявленными ранее оптимальными параметрами. Использование разреженного автокодера дало наименьшую ошибку классификации - 0.01075. Худшую ошибку классификации показало использование шумоподавляющего автокодера.

5. Листинг кода

Задание 1

```
############ Наивный байесовский классификатор
def train_and_evaluate(test_size):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=42)
    nb_classifier = GaussianNB()
    nb_classifier.fit(X_train, y_train)
    y_pred = nb_classifier.predict(X_test)
    error_rate = np.mean(y_pred != y_test)
    return error_rate, nb_classifier, X_test, y_test, y_pred


test_sizes = np.arange(0.1, 1.0, 0.05)
error_rates = []
best_model = None
best_error_rate = float('inf')

for test_size in test_sizes:
    error_rate, model, X_test, y_test, y_pred = train_and_evaluate(test_size)
    error_rates.append(error_rate)

    if error_rate < best_error_rate:
        best_error_rate = error_rate
        best_model = model
        best_X_test = X_test
        best_y_test = y_test
        best_y_pred = y_pred

############ Метод ближайших соседей
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)


k_values = list(range(1, 11))
kernel_values = ["uniform", "distance"]
distance_values = list(range(1, 11))
metric_values = ["euclidean", "manhattan", "chebyshev"]


best_accuracy = 0
best_params = {}


for k, kernel, distance, metric in product(k_values, kernel_values, distance_values, metric_values):
    model = KNeighborsClassifier(n_neighbors=k, weights=kernel, p=distance, metric=metric)
    model.fit(X_train, y_train)
    pred = model.predict(X_test)
    accuracy = (pred == y_test).mean()


    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_params = {'k': k, 'kernel': kernel, 'distance': distance, 'metric': metric}


best_model = KNeighborsClassifier(n_neighbors=best_params['k'],
                                  weights=best_params['kernel'],
                                  p=best_params['distance'],
                                  metric=best_params['metric'])
best_model.fit(X_train, y_train)
pred = best_model.predict(X_test)


final_accuracy = (pred == y_test).mean()
```

```
print(f"Точность лучшей модели: {final_accuracy:.4f}")
print("Лучшие параметры:")
print(best_params)

############ Бустинг
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

y_train = y_train.astype('category')
y_test = y_test.astype('category')
tree_num = np.arange(1, 102, 5)
error_boost = []

for t in tree_num:
    err_boost = []
    for i in range(5):
        clf = AdaBoostClassifier(n_estimators=t, random_state=i)
        clf.fit(X_train, y_train)
        pred = clf.predict(X_test)
        err_boost.append(1 - accuracy_score(y_test, pred))

    error_boost.append(np.mean(err_boost))

print("Средняя ошибка:", np.mean(error_boost))
print("Ошибки для каждого числа деревьев:", error_boost)
```

Задание 3

```
############ Дендрограмма
distance_matrix = hierarchy.distance.pdist(X_emb)
linkage_matrix = hierarchy.linkage(distance_matrix, method='single')
```

```
plt.figure(figsize=(10, 6))
dendrogram = hierarchy.dendrogram(linkage_matrix, labels=y.values)
plt.show()

############ k-средних
file_path_data = 'wdbc.data'
df = pd.read_csv(file_path_data, header=None, delimiter=',')
df.columns = ['ID', 'Diagnosis'] + list(df.columns[2:])
df['Diagnosis'] = df['Diagnosis'].replace({'M': 0, 'B': 1})
X = df.drop(columns=['ID', 'Diagnosis'])
y = df['Diagnosis']
X_emb = TSNE(n_components=2).fit_transform(X)

kmeans = KMeans(n_clusters=2, random_state=42)
predict_kmeans = kmeans.fit_predict(X_emb)
errors_kmeans = (predict_kmeans != y).sum() / len(y)
print(f'Error (k-means): {errors_kmeans * 100:.2f}%')

fig, ax = plt.subplots(1, 2, figsize=(10, 5))

sns.scatterplot(ax=ax[0], x=X_emb[:, 0], y=X_emb[:, 1], hue=y, alpha=0.7)
ax[0].set_title('Real')

sns.scatterplot(ax=ax[1], x=X_emb[:, 0], y=X_emb[:, 1], hue=predict_kmeans,
alpha=0.7)
ax[1].set_title('Predicted (k-means)')

fig.tight_layout()
plt.show()
```

Задание 4

```
############ Лассо
lasso_cv = LassoCV(cv=100, random_state=25)
lasso_cv.fit(X, y)
optimal_alpha = lasso_cv.alpha_
print(optimal_alpha)


x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=25)


lasso_model = Lasso(alpha=optimal_alpha, random_state=25)
lasso_model.fit(x_train, y_train)


sorted_features = sorted(zip(lasso_model.coef_, X.columns), key=lambda x: abs(x[0]), reverse=True)
for coef, feature_name in sorted_features:
    print(f'{feature_name}: {coef}')
```

Задание 5

```
########## Сокращение размерности автокодером
data = h2o.H2OFrame(df)
data['Diagnosis'] = data['Diagnosis'].asfactor()
train, val = data.split_frame([0.8])


predictors = [f"{i}" for i in range(2, 32)]
response = "Diagnosis"


mdl = H2ODeepLearningEstimator(activation='tanh', autoencoder=True)
mdl.train(predictors, response, training_frame=train)


features = mdl.deepfeatures(train, layer=1)
x_new = features.as_data_frame()
```

```
y_new = train[response].as_data_frame().values.reshape(1, -1)
emb = TSNE(n_components=2).fit_transform(x_new)

x_train, x_test, y_train, y_test = train_test_split(x_new, y_new[0], test_size=0.8, stratify=y_new[0], random_state=25)
Ada_model = AdaBoostClassifier(n_estimators=31).fit(x_train, y_train)
y_pred = Ada_model.predict(x_test)
print(f'Ошибка: {(y_pred != y_test).sum()/len(y_test)}')

########## разряж автокодер
mdl = H2ODeepLearningEstimator(activation='tanh', autoencoder=True, hidden=[500,500])
mdl.train(predictors, response, training_frame=train)

features = mdl1.deepfeatures(train, layer=1)
x_new = features.as_data_frame()
emb = TSNE(n_components=2).fit_transform(x_new)

sns.scatterplot(x=emb[:,0], y=emb[:,1], hue=y_new[0], alpha=0.7)

x_train, x_test, y_train, y_test = train_test_split(x_new, y_new[0], test_size=0.2, stratify=y_new[0], random_state=25)
Ada_model = AdaBoostClassifier(n_estimators=31).fit(x_train, y_train)
y_pred = Ada_model.predict(x_test)
print(f'Error: {(y_pred != y_test).sum()/len(y_test)}')

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

```
X = df.drop(columns=['ID', 'Diagnosis'])
y = df['Diagnosis'].tolist()
X_subset = X.iloc[subset_indices]
y_subset = [y[i] for i in 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

train = h2o.H2OFrame(pd.concat([X_with_noise, pd.Series(y, name='Diagnosis')], axis=1))

mdl1 = H2ODeepLearningEstimator(activation='tanh', autoencoder=True)
mdl1.train(predictors, response, training_frame=train, validation_frame=val)

features = mdl1.deepfeatures(train, layer=1)
a = train['Diagnosis'].as_data_frame().values.reshape(1, -1)
data_new = features.as_data_frame()
emb = TSNE(n_components=2).fit_transform(data_new)

sns.scatterplot(x=emb[:, 0], y=emb[:, 1], hue=a[0], alpha=0.7)
plt.show()

x_train, x_test, y_train, y_test = train_test_split(data_new, a[0], test_size=0.2, stratify=a[0], random_state=25)

bag_model = AdaBoostClassifier(n_estimators=31).fit(x_train, y_train)
predict2 = bag_model.predict(x_test)
print(f'Error: {(predict2 != y_test).sum()/len(y_test)}')
```