

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»  
Институт компьютерных наук и технологий  
Высшая школа искусственного интеллекта



**ПОЛИТЕХ**  
Санкт-Петербургский  
политехнический университет  
Петра Великого

**КУРСОВАЯ РАБОТА**  
**«Разработка классификатора для базы данных»**  
по дисциплине «Машинное обучение»

Выполнил  
Студент 3540201/10301 группы

Ф.М. Титов

Руководитель  
профессор, д.т.н.

Л.В. Уткин

Санкт-Петербург  
2021 г.

# Содержание

<b>Введение .....</b>	<b>3</b>
<b>Цель работы .....</b>	<b>3</b>
<b>Постановка задачи .....</b>	<b>3</b>
<b>Основная часть .....</b>	<b>6</b>
<b>1. Методы классификации .....</b>	<b>6</b>
<b>Метод ближайших соседей .....</b>	<b>7</b>
<b>Наивный байесовский классификатор .....</b>	<b>8</b>
<b>Бустинг .....</b>	<b>9</b>
<b>Вывод к разделу .....</b>	<b>10</b>
<b>2. Методы кластеризации .....</b>	<b>12</b>
<b>Метод k средних .....</b>	<b>12</b>
<b>Метод медоидов .....</b>	<b>12</b>
<b>Иерархическая кластеризация .....</b>	<b>13</b>
<b>Вывод к разделу .....</b>	<b>14</b>
<b>3. Регрессия .....</b>	<b>15</b>
<b>Вывод к разделу .....</b>	<b>15</b>
<b>4. Автокодеры .....</b>	<b>16</b>
<b>Автокодер для сокращения размерности .....</b>	<b>16</b>
<b>Автокодер для реализации разреженного скрытого слоя .....</b>	<b>17</b>
<b>Зашумленный автокодер .....</b>	<b>18</b>
<b>Вывод к разделу .....</b>	<b>19</b>
<b>Заключение .....</b>	<b>21</b>
<b>Приложение .....</b>	<b>22</b>

# Введение

Машинное обучение — класс методов искусственного интеллекта, характерной чертой которых является не прямое решение задачи, а обучение за счёт применения решений множества сходных задач. Для построения таких методов используются средства математической статистики, численных методов, математического анализа, методов оптимизации, теории вероятностей, теории графов, различные техники работы с данными в цифровой форме.

## Цель работы

Целью данной работы является применение и анализ работы алгоритмов машинного обучения без учителя и с учителем на основе реальной базы данных.

Необходимо разработать алгоритмы классификации, кластеризации и проверить их работу на реальных данных. Кроме того, нужно применить методы логистической регрессии и оценить наиболее значимые параметры выбранного набора данных. Также необходимо провести классификацию с предварительной обработкой данных, основанной на использовании нейронных сетей – автокодиров.

## Постановка задачи

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

Список методов классификации:

1. Наивный байесовский классификатор.
2. Метод ближайших соседей.
3. Деревья решений.
4. Метод опорных векторов.
5. Нейронные сети.
6. Беггинг.
7. Бустинг.

Список методов кластеризации:

1. Метод k средних.
2. Метод медоидов.
3. Иерархическая кластеризация.

Базы данных (по вариантам, в скобках имена соответствующих файлов):

1. Protein Localization Sites (Ecoli)
2. Heart disease (heart)
3. Hepatitis Domain (Hepatitis)
4. Liver patient records (Indian Liver Patient Dataset)
5. Mammographic Mass Data (mammographic\_masses)
6. Parkinsons Disease Data Set (parkinsons)
7. Pima Indians Diabetes Database (pima-indians-diabetes)
8. Diagnosing of cardiac Single Proton Emission Computed Tomography images (SPECT)
9. Wisconsin Diagnostic Breast Cancer (wdbc)

Для каждой 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.

В нашем случае был выбран набор данных Wisconsin Diagnostic Breast Cancer (wdbc). Данные содержат следующие поля:

1. Номер (ID).
2. Диагноз (M = malignant (злокачественный), B = benign(доброкачественный)).

Для каждого клеточного ядра вычисляются десять вещественных признаков:

3. Радиус (radius).
4. Текстура (texture).
5. Периметр (perimeter).
6. Площадь (area).
7. Гладкость (smoothness).
8. Компактность (compactness).
9. Вогнутость (concavity).
10. Вогнутые точки (concave points).
11. Симметрия (symmetry).
12. Фрактальная размерность (fractal dimension).

Остальные признаки являются статистическими характеристиками данных.

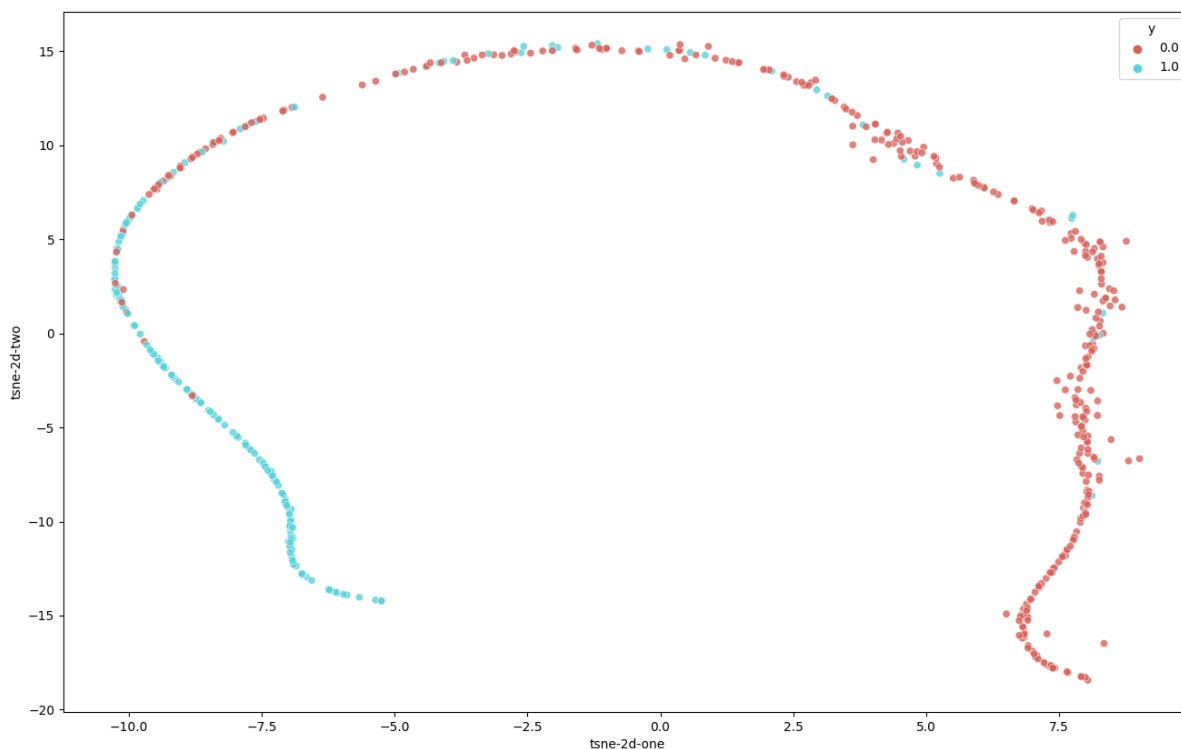
# Основная часть

## 1. Методы классификации

В качестве классификаторов были выбраны следующие методы:

- 1) Метод ближайших соседей
- 2) Наивный байесовский классификатор
- 3) Бустинг.

Согласно заданию, необходимо было визуализировать данные с помощью алгоритма t-SNE. Результат визуализации представлен на рис. 1. В данном случае синие точки обозначают доброкачественную опухоль, красные – злокачественную.



*Рис.1. Визуализация данных с помощью алгоритма t-SNE. Красные точки – злокачественная опухоль, синие – доброкачественная.*

На рис.2. представлена тепловая карта исходного набора данных, показывающая корреляцию между признаками.

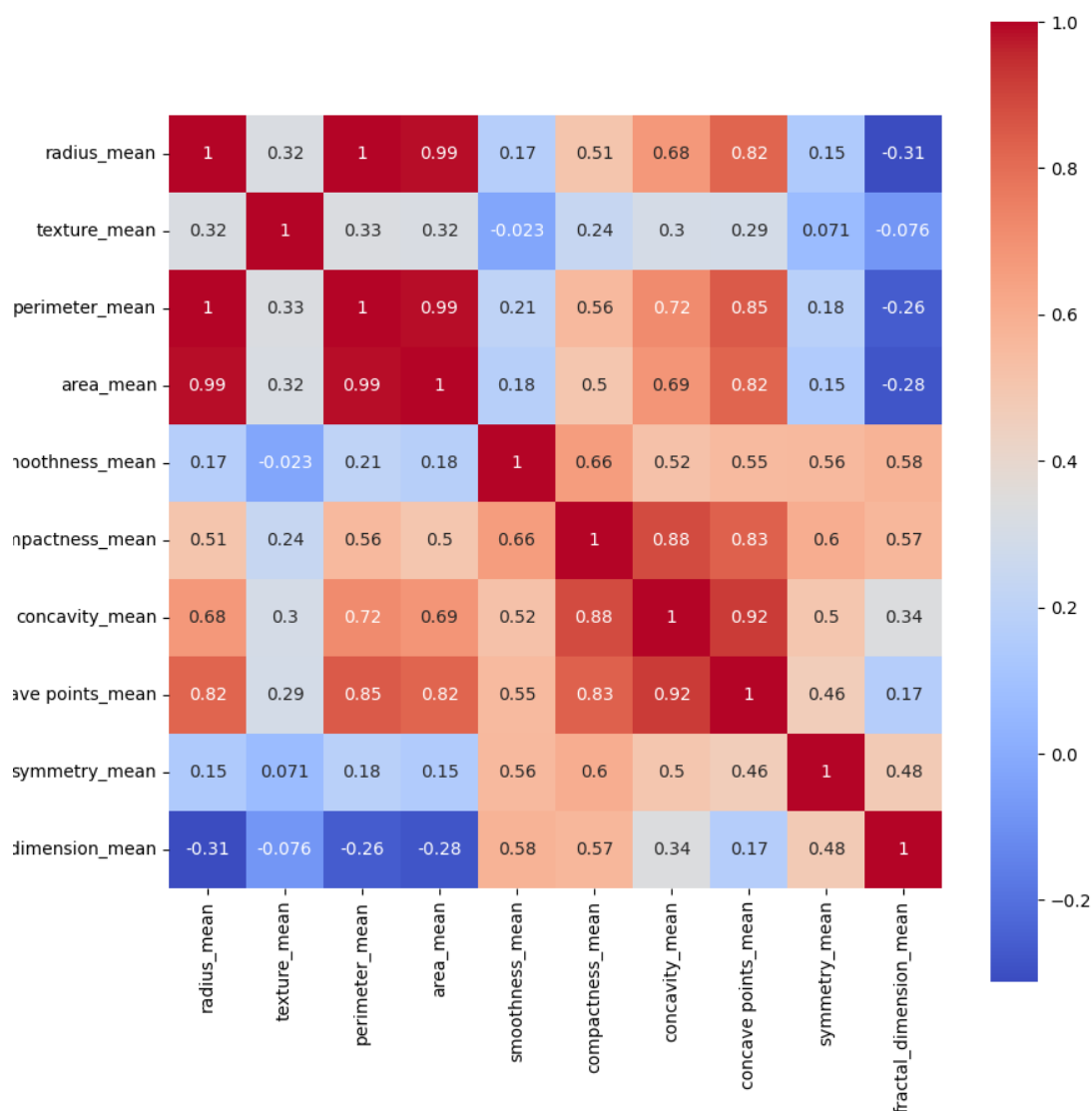


Рис.2. Тепловая карта исходных данных.

Для разработки классификаторов и оценки качества их работы была использована библиотека языка программирования Python scikit-learn. Подбор наилучших параметров осуществлялся с помощью алгоритма поиска по сетке (grid search), который реализован в методе GridSearchCV, импортируемом из той же библиотеки. Поиск по сетке принимает на вход модель и различные значения гиперпараметров (сетку гиперпараметров). Далее, для каждого возможного сочетания значений гиперпараметров, метод считает ошибку и в конце выбирает сочетание, при котором ошибка минимальна.

## Метод ближайших соседей

Данный метод реализован в классе KNeighborsClassifier библиотеки scikit-learn. Точность работы алгоритма зависит от следующих параметров:

- “n\_neighbors” – количество ближайших соседей;

- “algorithm” – используемый алгоритм. Параметр может принимать значения: “auto”, “ball\_tree”, “kd\_tree”, “brute”.
- “p” – параметр расстояния Миньковского. Когда  $p = 1$ , это эквивалентно использованию `manhattan_distance` (11) и `euclidean_distance` (12) для  $p = 2$ . Для произвольного  $p$  используется `minkowski_distance (1_p)`.

Поиск оптимальных параметров осуществлялся среди указанных выше параметров. Для метода ближайших соседей удалось достичь точности 95.61%, среднеквадратичной ошибки 4.39% (рис.3). В результате выполнения метода поиска по сетке были подобраны следующие параметры:

- “n\_neighbors” равно 4;
- “algorithm” – по умолчанию (“auto”);
- “p” равно 2 (`euclidean_distance`).

```
KNeighborsClassifier:
Accuracy: 95.61%
MSE: 4.39%
Cross validation score: 94.92% (+/- 2.87%)
Execution time: 5.029 seconds

Best parameters: {'algorithm': 'auto', 'n_neighbors': 4, 'p': 2}
```

*Рис.3. Результат поиска наилучших параметров для метода k ближайших соседей.*

## Наивный байесовский классификатор

Наивный байесовский классификатор был реализован с использованием класса `GaussianNB` библиотеки `scikit-learn`. Точность работы классификатора зависит от параметра “priors”, который определяет априорные вероятности классов. Если параметр не определен, его значение корректируется в соответствии с набором данных.

Классификатор был протестирован на следующем наборе данных для параметра “priors”: None, [0.01, 0.99], [0.1, 0.9], [0.2, 0.8], [0.25, 0.75], [0.3, 0.7], [0.35, 0.65], [0.4, 0.6].

В результате тестирования была достигнута наилучшая точность работы алгоритма (точность – 93.86%, среднеквадратичная ошибка – 6.14%) при



значении параметра “priors” равном None (рис. 4), то есть без явного указания значения параметра.

```
GaussianNB:
Accuracy: 93.86%
MSE: 6.14%
Cross validation score: 95.64% (+/- 7.73%)
Execution time: 0.25244 seconds

Best parameters: {'priors': None}
```

*Рис.4. Результат поиска наилучших параметров для наивного байесовского классификатора.*

## Бустинг

Алгоритм бустинга был реализован с использованием класса GradientBoostingClassifier библиотеки scikit-learn. Точность работы классификатора зависит от параметров:

- “max\_depth” – максимальная глубина отдельных регрессионных оценок.
- “criterion” – функция измерения качества разбиения. Поддерживаемые критерии: “friedman\_mse” для среднеквадратичной ошибки с оценкой улучшения по Фридману, “squared\_error” для среднеквадратичной ошибки и “mae” для средней абсолютной ошибки.
- “loss” – функция потерь. Поддерживаемые параметры: “deviance”, “exponential”.

Классификатор был протестирован с помощью алгоритма поиска по сетке на следующем наборе параметров:

- “max\_depth”: [3, 4, 5, 6, 7, 8, 9, 10].
- “criterion”: [«friedman\_mse», “squared\_error”] .
- “loss”: [“deviance”, “exponential”].

В результате тестирования была достигнута наилучшая точность работы алгоритма (точность – 95.61%, среднеквадратичная ошибка – 4.39% (рис.5)) при следующих значениях параметров:

- “max\_depth”: 3
- “criterion”: “friedman\_mse”
- “loss”: “deviance”

```
GradientBoostingClassifier:  
Accuracy: 95.61%  
MSE: 4.39%  
Cross validation score: 97.43% (+/- 3.40%)  
Execution time: 61.548 seconds  
  
Best parameters: {'criterion': 'friedman_mse', 'loss': 'deviance', 'max_depth': 3}
```

*Рис.5. Результат поиска наилучших параметров для бустинга.*

## Вывод к разделу

Были разработаны следующие методы:

- 1) Метод ближайших соседей
- 2) Наивный байесовский классификатор
- 3) Бустинг.

Для каждого были подобраны оптимальные параметры, дающие наименьшую ошибку классификации:

- 1) Метод k ближайших соседей:
  - a. “n\_neighbors”: 4;
  - b. “algorithm” – по умолчанию (“auto”);
  - c. “p”: 2.
- 2) Наивный байесовский классификатор:
  - a. “priors”: None
- 3) Бустинг:
  - a. “max\_depth”: 3
  - b. “criterion”: “friedman\_mse”
  - c. “loss”: “deviance”

Классификатор	Точность	Среднеквадратичная ошибка
Метод ближайших соседей	95.61%	4.39%
Наивный байесовский классификатор	93.86%	6.14%
Бустинг	95.61%	4.39%

*Таблица 1. Сравнение точности работы классификаторов.*

В табл.1 приведено сравнение алгоритмов классификации. Таким образом, для данного набора данных и подобранных параметров наименьшую ошибку классификации показали метод ближайших соседей и бустинг.

## 2. Методы кластеризации

Согласно заданию, было необходимо удалить метки классов исходного набора данных и выполнить кластеризацию. Кластеризация данных была выполнена с использованием следующих алгоритмов:

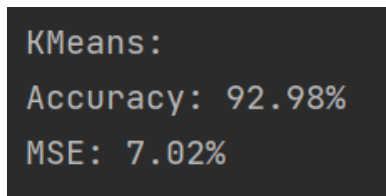
1. Метод k средних.
2. Метод медоидов.
3. Иерархическая кластеризация.

Предварительно данные были нормированы с помощью класса `MinMaxScaler` библиотеки `sklearn`, чтобы исходные данные (тестовые и тренировочные  $X$ ) находились в диапазоне от 0 до 1.

### Метод k средних

Данный метод был реализован с использованием класса `KMeans` библиотеки `sklearn.cluster`, с параметром количества кластеров, равным 2.

Результат работы алгоритма (точность, среднеквадратичная ошибка) представлен на рис. 6.



```
KMeans:  
Accuracy: 92.98%  
MSE: 7.02%
```

*Рис. 6. Точность работы кластеризации методом k средних.*

Таким образом, удалось достичь точности классификации 92.98%. Ошибка кластеризации – 7,02%.

### Метод медоидов

Данный метод был реализован с использованием класса `KMedoids` библиотеки `sklearn_extra.cluster`. Было задано значение параметра `n_clusters`, равное 2.

В результате была достигнута точность 89.47%; ошибка кластеризации – 10.53% (рис. 7).



```
AgglomerativeClustering (tree):  
Accuracy: 81.90%  
MSE: 18.10%
```

*Рис. 9. Точность работы иерархической кластеризации.*

**Вывод к разделу**

В результате работы были реализованы алгоритмы кластеризации:

- 1. Метод k средних.
- 2. Метод медоидов.
- 3. Иерархическая кластеризация.

Данные алгоритмы были протестированы на указанном выше наборе данных. Результаты тестирования представлены в табл. 2.

Классификатор	Точность	Среднеквадратичная ошибка
Метод k средних	92.98%	7.02%
Метод медоидов	81.90%	18.10%
Иерархическая кластеризация	89.47%	10.53%

*Таблица 2. Сравнение точности работы алгоритмов кластеризации.*

Таким образом, наименьшую ошибку показал метод k средних для набора данных wdbs – 7.02%.

### 3. Регрессия

С помощью логистической регрессии в рамках метода Лассо необходимо было определить наиболее значимые признаки для данного набора данных (wdbc).

Для реализации логистической регрессии был использован класс Lasso библиотеки `sklearn.linear_model`. Исходные данные были предварительно обработаны с использованием класса `MinMaxScaler`.

Результат работы программы представлен на рис.10.

```
x: [ 5.93111864  0.64946504 -3.20750138 -2.38369913  0.24493519 -0.2886687
      0.27620703  1.27415003  0.20950913  0.02414282]
```

*Рис 10. Полученные коэффициенты регрессии.*

#### Вывод к разделу

В результате выполненной регрессии были получены значения коэффициентов регрессии, представленные в табл. 3.

Признаки	Критерий регрессии
radius (радиус)	5.93111864
texture (текстура)	0.64946504
perimeter (периметр)	-3.20750138
area (площадь)	-2.38369913
smoothness (гладкость)	0.24493519
compactness (компактность)	-0.2886687
concavity (вогнутость)	0.27620703
concave_points (вогнутые точки)	1.27415003
symmetry (симметрия)	0.20950913
fractal_dimension (фрактальная р-ть)	0.02414282

*Таблица 3. Значения коэффициентов регрессии для каждого признака.*

Таким образом, наиболее значимыми являются признаки: радиус, периметр, площадь; наименее значимыми: фрактальная размерность, симметрия, вогнутость.

## 4. Автокодеры

Согласно заданию, было необходимо разработать автокодеры для преобразования исходных данных. Автокодер является частным случаем нейронной сети. Требовалось разработать:

- 1) Автокодер для сокращения размерности;
- 2) Автокодер для реализации разреженного скрытого слоя;
- 3) Зашумленный автокодер.

После преобразования исходных данных необходимо выполнить классификацию. В качестве алгоритма классификации был выбран метод ближайших соседей. Кроме того, согласно заданию, необходимо предоставить визуализацию данных, преобразованных с помощью автокодера, с помощью метода t-SNE.

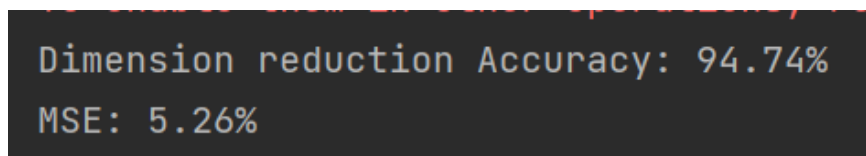
### Автокодер для сокращения размерности

Был реализован автокодер для сокращения размерности. При разработке была использована библиотека tensorflow keras. Автокодер имеет следующие параметры:

- 1) Число слоев – 3.
- 2) Число нейронов в скрытом слое – 8, функция активации – “relu”.
- 3) Функция активации в последнем слое – “sigmoid”.
- 4) Оптимизатор – “adam”.
- 5) Функция потерь – “mse”.
- 6) Количество эпох для обучения – 120.

Выборка была разбита на тестовые и тренировочные данные. Далее они были поданы на вход автокодеру для сокращения размерности. Визуализация полученных данных с помощью метода t-SNE представлена на рис.12.

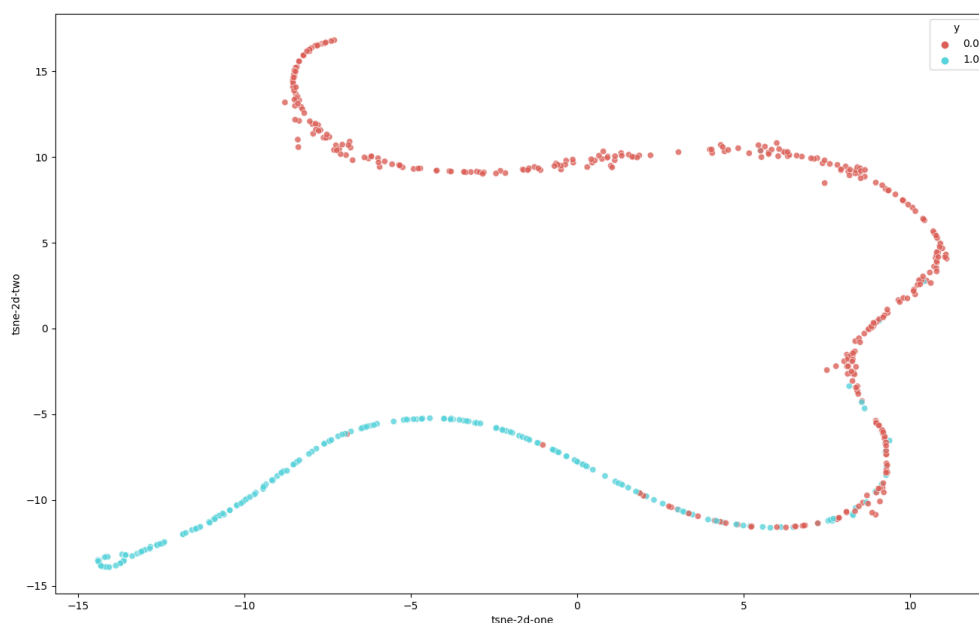
Далее была проведена классификация с помощью метода ближайших соседей. Результат представлен на рис. 11. Как можно наблюдать, точность классификации составила 94.74%, ошибка – 5.26%.



```
Dimension reduction Accuracy: 94.74%
MSE: 5.26%
```

Рис. 11. Точность классификации при использовании автокодера сокращения размерности.





*Рис. 12. Визуализация данных, обработанных с помощью автокодера для сокращения размерности. Синие точки – доброкачественная опухоль; красные – злокачественная.*

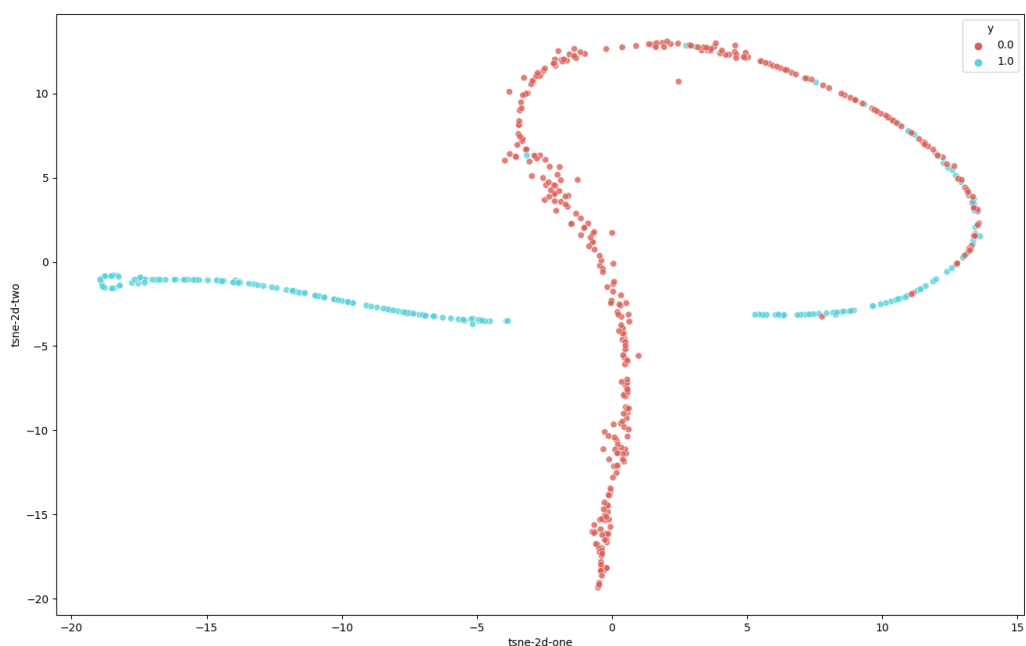
### **Автокодер для реализации разреженного скрытого слоя**

Был реализован автокодер для реализации разреженного скрытого слоя. Он имеет следующие параметры:

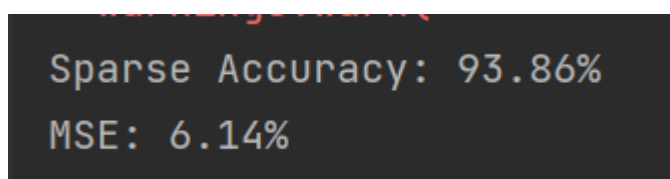
- 1) Число слоев – 3.
- 2) Число нейронов в скрытом слое – 8, функция активации – “relu”.  
К нему была добавлена функция регуляризатора для применения к выходным данным слоя. Это необходимо для реализации разреженного скрытого слоя.
- 3) Функция активации в последнем слое – “sigmoid”.
- 4) Оптимизатор – “adam”.
- 5) Функция потерь – “mse”.
- 6) Количество эпох для обучения – 120.

Визуализация данных, полученных с применением автокодера для реализации разреженного скрытого слоя, а также с помощью метода t-SNE, представлена на рис.13.

Результат классификации методом ближайших соседей представлен на рис. 14. Как можно наблюдать, точность классификации составила 93.86%, ошибка – 6.14%.



*Рис. 13. Визуализация данных, обработанных с помощью автокодера для реализации разреженного скрытого слоя. Синие точки – доброкачественная опухоль; красные – злокачественная.*



*Рис. 14. Точность классификации при использовании автокодера для реализации разреженного скрытого слоя.*

## **Зашумленный автокодер**

Был реализован зашумленный. Он имеет следующие параметры:

- 1) Число слоев – 4.
- 2) Имеется слой GaussianNoise из библиотеки keras, необходимый для зашумления входных данных.
- 3) Число нейронов в скрытом слое – 8, функция активации – “relu”.
- 4) Функция активации в последнем слое – “sigmoid”.
- 5) Оптимизатор – “adam”.
- 6) Функция потерь – “mse”.
- 7) Количество эпох для обучения – 120.

Визуализация данных, полученных с применением зашумленного автокодера, а также с помощью метода t-SNE, представлена на рис.15.

Результат классификации методом ближайших соседей представлен на рис. 16. Как можно наблюдать, точность классификации составила 97.37%, ошибка – 2.63%.

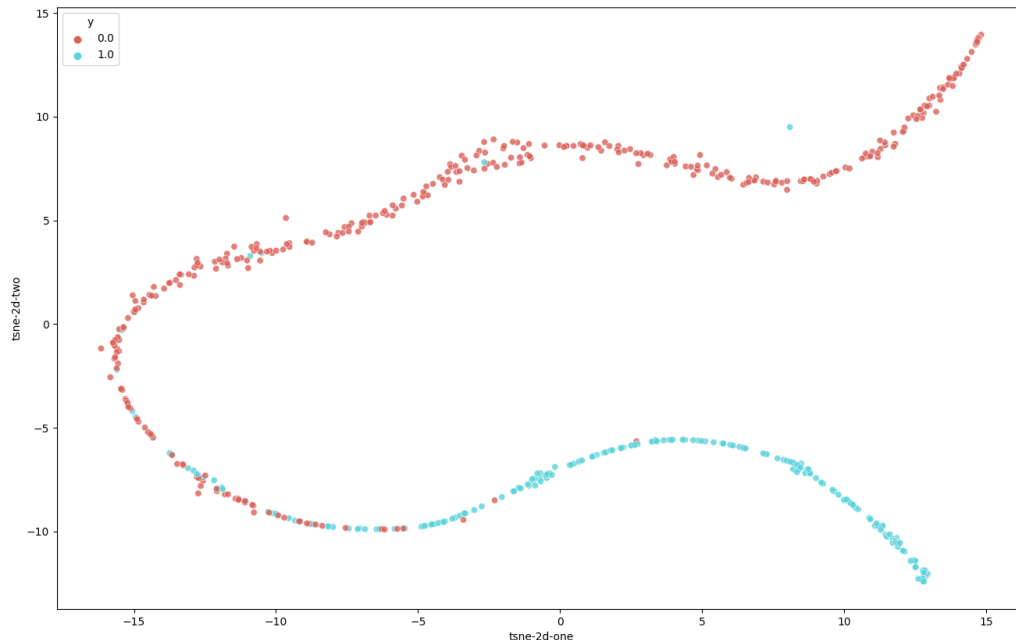


Рис. 15. Визуализация данных, обработанных с помощью автокодера для реализации разреженного скрытого слоя. Синие точки – доброкачественная опухоль; красные – злокачественная.

```
Denoising Accuracy: 97.37%  
MSE: 2.63%
```

Рис. 16. Точность классификации при использовании зашумленного автокодера.

## Вывод к разделу

Были разработаны следующие автокодеры:

- 1) Автокодер для сокращения размерности;
- 2) Автокодер для реализации разреженного скрытого слоя;
- 3) Зашумленный автокодер.

Результаты классификации методом ближайших соседей с использованием различных автокодеров представлены в табл. 4.

Можно сделать вывод, что качество классификации лучше при использовании зашумленного автокодера для предварительной обработки данных, так как с помощью него была достигнута наименьшая ошибка классификации.

<b>Классификатор</b>	<b>Точность</b>	<b>Среднеквадратичная ошибка</b>
Автокодер для сокращения размерности	94.74%	5.26%
Автокодер для реализации разреженного скрытого слоя	93.86%	6.14%
Зашумленный автокодер	97.37%	2.63%

*Таблица 4. Сравнение точности работы классификации с использованием различных автокодеров.*

## Заключение

В ходе работы для выбранной базы данных («Wisconsin Diagnostic Breast Cancer») были разработаны алгоритмы классификации и проведена настройка их параметров с помощью алгоритма поиска по сетке. Среди выбранных методов классификации (метод ближайших соседей, наивный байесовский классификатор, бустинг) наименьшую ошибку классификации (4.39%) показали метод ближайших соседей и бустинг.

Далее были разработаны алгоритмы кластеризации (метод k средних, метод медоидов, иерархическая кластеризация). Наименьшую ошибку кластеризации показал метод k средних (7.02%).

Также с помощью логистической регрессии в рамках метода Лассо были определены наиболее и наименее значимые признаки, влияющие на отнесение объектов к определенному классу. Наиболее значимыми являются признаки: радиус, периметр, площадь; наименее значимыми: фрактальная размерность, симметрия, вогнутость.

Далее были разработаны и применены автокодеры (сокращения размерности, для реализации разреженного скрытого слоя нейронной сети, зашумленный). Каждый автокодер был использован для преобразования входных данных, после чего была проведена оценка качества классификация методом ближайших соседей. Наилучший результат показал зашумленный автокодер (ошибка – 2.63%).

## Приложение

Листинг программы, написанной на языке программирования Python.

*classifiers.py:*

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from pandas.plotting import scatter_matrix
from sklearn.model_selection import train_test_split, cross_val_score,
GridSearchCV
from sklearn.metrics import accuracy_score, mean_squared_error
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import GradientBoostingClassifier
from sklearn import preprocessing
import time

def test_classifier(clf, X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
    start = time.time()

    clf.fit(X_train, y_train)
    prediction = clf.predict(X_test)
    scores = cross_val_score(clf, X, y, cv=5)

    end = time.time()
```

```
print("Accuracy: {0:.2%}".format(accuracy_score(prediction, y_test)))
print("MSE: {0:.2%}".format(mean_squared_error(y_test, prediction)))
print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(np.mean(scores),
np.std(scores) * 2))
print("Execution time: {0:.5} seconds \n".format(end - start))
```

```
def get_best_parameters_clf(clf, X, y, parameters):
```

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

```
    start = time.time()
```

```
    clf = GridSearchCV(clf, parameters, scoring='average_precision', n_jobs=-1)
```

```
    clf.fit(X_train, y_train)
```

```
    prediction = clf.predict(X_test)
```

```
    scores = cross_val_score(clf, X, y, cv=5)
```

```
    end = time.time()
```

```
    print("Accuracy: {0:.2%}".format(accuracy_score(prediction, y_test)))
```

```
    print("MSE: {0:.2%}".format(mean_squared_error(y_test, prediction)))
```

```
    print("Cross validation score: {0:.2%} (+/- {1:.2%})".format(np.mean(scores),
np.std(scores) * 2))
```

```
    print("Execution time: {0:.5} seconds \n".format(end - start))
```

```
    print("Best parameters: {0}".format(clf.best_params_))
```

```
    return clf.best_params_
```

```
data = pd.read_csv('C:/Users/FedozZz/PycharmProjects/machine-learning-  
coursework/data/data.csv')
```

```
print("\n \t The data frame has {0[0]} rows and {0[1]} columns.  
\n".format(data.shape))
```

```
data.info()
```

```
data.head(3)
```

```
data.drop(data.columns[[-1, 0]], axis=1, inplace=True)
```

```
data.info()
```

```
features_mean = list(data.columns[1:11])
```

```
plt.figure(figsize=(10, 10))
```

```
sns.heatmap(data[features_mean].corr(), annot=True, square=True,  
cmap='coolwarm')
```

```
plt.show()
```

```
color_dic = {'M': 'red', 'B': 'blue'}
```

```
colors = data['diagnosis'].map(lambda x: color_dic.get(x))
```

```
sm = scatter_matrix(data[features_mean], c=colors, alpha=0.4, figsize=((15, 15)))
```

```
# plt.show()
```

```
diag_map = {'M': 1, 'B': 0}
```

```
data['diagnosis'] = data['diagnosis'].map(diag_map)
```



```
X = data.loc[:, features_mean]
```

```
y = data.loc[:, 'diagnosis']
```

```
X = X.to_numpy()
```

```
y = y.to_numpy()
```

```
min_max_scaler = preprocessing.MinMaxScaler()
```

```
X = min_max_scaler.fit_transform(X)
```

```
print('-----TESTING CLASSIFIERS-----')
```

```
print('KNeighborsClassifier:')
```

```
test_classifier(KNeighborsClassifier(), X, y)
```

```
print('-----')
```

```
print('GaussianNB:')
```

```
# test_classifier(GaussianNB(priors=[0.2, 0.8]), X, y)
```

```
test_classifier(GaussianNB(), X, y)
```

```
print('-----')
```

```
print('GradientBoostingClassifier:')
```

```
test_classifier(GradientBoostingClassifier(), X, y)
```

```
print('-----')
```

```
print('-----CALCULATING THE BEST PARAMETERS FOR  
CLASSIFIERS-----')
```

```
print('KNeighborsClassifier:')
```

```
parameters = {'n_neighbors': list(range(1, 5)), 'algorithm': ['auto', 'ball_tree',  
'kd_tree', 'brute'], 'p': [1, 2, 3]}
```

```
bp_k_neib = get_best_parameters_clf(KNeighborsClassifier(), X, y, parameters)
```

```
print('-----')
```

```
print('GaussianNB:')
```

```

parameters = {
    'priors': [None, [0.01, 0.99], [0.1, 0.9], [0.2, 0.8], [0.25, 0.75], [0.3, 0.7], [0.35,
0.65], [0.4, 0.6]]}

bp_gauss = get_best_parameters_clf(GaussianNB(), X, y, parameters)
print('-----')

print('GradientBoostingClassifier:')

parameters = {'max_depth': list(range(3, 10)), 'criterion': ['friedman_mse',
'squared_error'],

               'loss': ['deviance', 'exponential']}

bp_boost = get_best_parameters_clf(GradientBoostingClassifier(), X, y,
parameters)

print('-----')


print('-----TESTING WITH CALCULATED PARAMETERS-----')

print('KNeighborsClassifier:')

test_classifier(KNeighborsClassifier(algorithm='auto', n_neighbors=4, p=2), X, y)

print('-----')

print('GaussianNB:')

test_classifier(GaussianNB(priors=None), X, y)

print('-----')

print('GradientBoostingClassifier:')

test_classifier(GradientBoostingClassifier(criterion='friedman_mse',
max_depth=3), X, y)

print('-----')

```

### ***clustering.py:***

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

```

```
from sklearn.metrics import accuracy_score, mean_squared_error
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn import preprocessing
from sklearn_extra.cluster import KMedoids
from scipy.cluster.hierarchy import dendrogram

data = pd.read_csv('C:/Users/FedozZz/PycharmProjects/machine-learning-
coursework/data/data.csv')

data.drop(data.columns[[-1, 0]], axis=1, inplace=True)

features_mean = list(data.columns[1:11])

diag_map = {'M': 1, 'B': 0}
data['diagnosis'] = data['diagnosis'].map(diag_map)

X = data.loc[:, features_mean]
y = data.loc[:, 'diagnosis']

X = X.to_numpy()
y = y.to_numpy()

min_max_scaler = preprocessing.MinMaxScaler()
X = min_max_scaler.fit_transform(X)

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

print('KMeans:')
kmeans = KMeans(n_clusters=2, random_state=0)
```

```

kmeans.fit(X_train)
prediction = kmeans.predict(X_test)

print("Accuracy: {0:.2% }".format(accuracy_score(prediction, y_test)))
print("MSE: {0:.2% }".format(mean_squared_error(y_test, prediction)))
print('-----')

```

```

def plot_dendrogram(model, **kwargs):
    # Create linkage matrix and then plot the dendrogram

    # create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1 # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack([model.children_, model.distances_,
                                      counts]).astype(float)

    dendrogram(linkage_matrix, **kwargs)

```

```
model = AgglomerativeClustering(distance_threshold=0, n_clusters=None)
model = model.fit(X)
```

```
# plt.figure(figsize=(15, 15))
plt.figure()
plt.grid(True)
plt.title('Dendrogram')
plot_dendrogram(model, truncate_mode='level', p=15)
plt.show()
```

```
print('AgglomerativeClustering (tree): ')
model = AgglomerativeClustering(n_clusters=2)
prediction = model.fit_predict(X)
print("Accuracy: {0:.2%}".format(accuracy_score(prediction, y)))
print("MSE: {0:.2%}".format(mean_squared_error(y, prediction)))
print('-----')
```

```
print('KMedoids: ')
kmedoids = KMedoids(n_clusters=2, random_state=0)
kmedoids.fit(X_train)
prediction = kmedoids.predict(X_test)
print("Accuracy: {0:.2%}".format(accuracy_score(prediction, y_test)))
print("MSE: {0:.2%}".format(mean_squared_error(y_test, prediction)))
print('-----')
```

***regression.py:***

```
import pandas as pd
from sklearn.linear_model import Lasso
```

```
from sklearn import preprocessing
```

```
data = pd.read_csv('C:/Users/FedozZz/PycharmProjects/machine-learning-  
coursework/data/data.csv')
```

```
data.drop(data.columns[[-1, 0]], axis=1, inplace=True)
```

```
features_mean = list(data.columns[1:11])
```

```
diag_map = {'M': 1, 'B': 0}
```

```
data['diagnosis'] = data['diagnosis'].map(diag_map)
```

```
X = data.loc[:, features_mean]
```

```
y = data.loc[:, 'diagnosis']
```

```
def get_regression_coeff(x, y):
```

```
    x = x.to_numpy()
```

```
    y = y.to_numpy()
```

```
    min_max_scaler = preprocessing.MinMaxScaler()
```

```
    x = min_max_scaler.fit_transform(x)
```

```
    log = Lasso(alpha=0.00001)
```

```
    log.fit(x, y)
```

```
    return log.coef_
```

```
print("X:{ }".format(get_regression_coeff(X, y)))
```

***autoencoders.py:***

```
from sklearn.metrics import accuracy_score, mean_squared_error
from sklearn.model_selection import train_test_split
import tensorflow as tf
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn import preprocessing
from keras import regularizers
import seaborn as sns
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
```

```
def plot_TNSE(X, y):
```

```
    tsne = TSNE(n_components=2, verbose=0, perplexity=40, n_iter=300)
    tsne_results = tsne.fit_transform(X)
```

```
    df_subset = pd.DataFrame()
    df_subset['tsne-2d-one'] = tsne_results[:, 0]
    df_subset['tsne-2d-two'] = tsne_results[:, 1]
    df_subset['y'] = y
```

```
    plt.figure(figsize=(16, 10))
```

```
    sns.scatterplot(
        x="tsne-2d-one", y="tsne-2d-two",
        hue="y",
        palette=sns.color_palette("hls", 2),
        data=df_subset,
```

```
    legend="full",  
    alpha=0.8  
)  
plt.show()
```

```
def get_KNeighborsClf_accuracy(X_train, X_test, y_train, y_test):  
    clf = KNeighborsClassifier()  
    clf.fit(X_train, y_train)  
    prediction = clf.predict(X_test)  
  
    return accuracy_score(prediction, y_test), mean_squared_error(y_test,  
prediction)
```

```
data = pd.read_csv('C:/Users/FedozZz/PycharmProjects/machine-learning-  
coursework/data/data.csv')
```

```
data.drop(data.columns[[-1, 0]], axis=1, inplace=True)
```

```
features_mean = list(data.columns[1:11])
```

```
diag_map = {'M': 1, 'B': 0}  
data['diagnosis'] = data['diagnosis'].map(diag_map)
```

```
X = data.loc[:, features_mean]  
y = data.loc[:, 'diagnosis']
```

```
X = X.to_numpy()
```



```
y = y.to_numpy().astype(float)
```

```
plot_TNSE(X, y)
```

```
min_max_scaler = preprocessing.MinMaxScaler()
```

```
X = min_max_scaler.fit_transform(X)
```

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,  
random_state=42)
```

```
# AUTOENCODER FOR DIMENSION REDUCTION
```

```
model = tf.keras.Sequential([  
    tf.keras.Input(shape=(10,)),  
    tf.keras.layers.Dense(8, activation='relu'),  
    tf.keras.layers.Dense(10, activation='sigmoid')  
])
```

```
model.compile(optimizer='adam',  
              loss='mse',  
              metrics=['accuracy'])
```

```
model.fit(x_train, y_train, epochs=120, verbose=0)
```

```
X_train_coded = model.predict(x_train)
```

```
X_test_coded = model.predict(x_test)
```

```
acc, mse = get_KNeighborsClf_accuracy(X_train_coded, X_test_coded, y_train,  
y_test)
```

```
print("Dimension reduction Accuracy: {0:.2%}".format(acc))
```

```
print("MSE: {0:.2%}".format(mse))
```

```
X_coded = model.predict(X)
```

```
plot_TNSE(X_coded, y)
```

```
# SPARSE AUTOENCODER
```

```
model = tf.keras.Sequential([  
    tf.keras.Input(shape=(10,)),  
    tf.keras.layers.Dense(8, activation='relu',  
activity_regularizer=regularizers.l1(10e-5)),  
    tf.keras.layers.Dense(10, activation='sigmoid')  
])
```

```
model.compile(optimizer='adam',  
              loss='mse',  
              metrics=['accuracy'])
```

```
model.fit(x_train, y_train, epochs=120, verbose=0)
```

```
X_train_coded = model.predict(x_train)
```

```
X_test_coded = model.predict(x_test)
```

```
acc, mse = get_KNeighborsClf_accuracy(X_train_coded, X_test_coded, y_train,  
y_test)
```

```
print("Sparse Accuracy: {0:.2%}".format(acc))
```

```
print("MSE: {0:.2%}".format(mse))
```

```
X_coded = model.predict(X)
```

```
plot_TNSE(X_coded, y)
```

```
# SPARSE AUTOENCODER
```

```
model = tf.keras.Sequential([  
    tf.keras.layers.GaussianNoise(0.2),  
    tf.keras.Input(shape=(10,)),  
    tf.keras.layers.Dense(8, activation='relu'),  
    tf.keras.layers.Dense(10, activation='sigmoid')  
])
```

```
model.compile(optimizer='adam',  
              loss='mse',  
              metrics=['accuracy'])
```

```
model.fit(x_train, y_train, epochs=120, verbose=0)
```

```
X_train_coded = model.predict(x_train)
```

```
X_test_coded = model.predict(x_test)
```

```
acc, mse = get_KNeighborsClf_accuracy(X_train_coded, X_test_coded, y_train,  
y_test)
```

```
print("Denoising Accuracy: {0:.2%}".format(acc))
```

```
print("MSE: {0:.2%}".format(mse))
```

```
X_coded = model.predict(X)
```

```
plot_TNSE(X_coded, y)
```

```
# RESULTS:
```

```
# Dimension reduction Accuracy: 93.86%
```

```
# Sparse Accuracy: 92.98%
```

```
# Denoising Accuracy: 96.49%
```