Содержание

Защита персональных данных клиентов

Вам нужно защитить данные клиентов страховой компании «Хоть потоп». Разработайте такой метод преобразования данных, чтобы по ним было сложно восстановить персональную информацию. Обоснуйте корректность его работы.

Нужно защитить данные, чтобы при преобразовании качество моделей машинного обучения не ухудшилось. Подбирать наилучшую модель не требуется.

Загрузка данных

In [1]:

```
import pandas as pd
import numpy as np

from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
```

In [2]:

```
data = pd.read_csv('/datasets/insurance.csv')
```

In [3]:

```
data.head()
```

Out[3]:

	Пол	Возраст	Зарплата	Члены семьи	Страховые выплаты
0	**1**	**41.0**	**49600.0**	**1**	**0**
1	**0**	**46.0**	**38000.0**	**1**	**1**
2	**0**	**29.0**	**21000.0**	**0**	**0**
3	**0**	**21.0**	**41700.0**	**2**	**0**
4	**1**	**28.0**	**26100.0**	**0**	**0**

In [4]:

```
# Разделим данные на признаки и целевые признаки

features = data.drop('Страховые выплаты', axis=1)
target = data['Страховые выплаты']
```

In [5]:

```
print(features.shape)
```

```
print(target.shape)
```

```
(5000, 4)

(5000,)
```

In [6]:

```
print(features.shape)
print(target.shape)
```

```
(5000, 4)

(5000,)
```

Умножение матриц

В этом задании вы можете записывать формулы в ***Jupyter Notebook.***

Чтобы записать формулу внутри текста, окружите её символами доллара **$;** если снаружи — двойными символами **\$$.** Эти формулы записываются на языке вёрстки ***LaTeX.***

Для примера мы записали формулы линейной регрессии. Можете их скопировать и отредактировать, чтобы решить задачу.

Работать в ***LaTeX*** необязательно.

Обозначения:

- X — матрица признаков (нулевой столбец состоит из единиц)
- y — вектор целевого признака
- P — матрица, на которую умножаются признаки
- w — вектор весов линейной регрессии (нулевой элемент равен сдвигу)

Предсказания:

$$a = Xw$$

Задача обучения:

$$w = \arg\min_w MSE(Xw, y)$$

Формула обучения:

$$w = (X^T X)^{-1} X^T y$$

Ответ: изменится.

Обоснование:

Используемые свойства:

$$(AB)^T = B^T A^T$$

$$(AB)^{-1} = B^{-1} A^{-1}$$

$$AA^{-1} = A^{-1}A = E$$

$$AE = EA = A$$

Доказать:

$$a = Xw$$
$$= XEw$$
$$= XPP^{-1}w$$
$$= (XP)P^{-1}w$$
$$= (XP)w'$$

Доказательство: \

$$w$$
$$= (X^T X$$
$$)^{-1}X^T y$$

\

$$w'$$
$$=$$
$$((XP)^T XP$$
$$)^{-1}(XP)^T y$$
$$w'$$
$$= (P^T(X^T X$$
$$)P)^{-1}(XP)^T y$$

Обновленные преобразования

$$w'$$
$$= P^{-1}(X^T X$$
$$)^{-1}(P^T$$
$$)^{-1}P^T X^T y$$
$$w'$$
$$= P^{-1}(X^T X$$
$$)^{-1}EX^T y$$
$$w'$$
$$= P^{-1}(X^T X$$
$$)^{-1}X^T y$$
$$w' = P^{-1}w$$

In [7]:

```
# Создадим случайную матрицу по кол-ву столбцов признака
random_matrix = np.random.normal(10, size=(4, 4))

print(random_matrix.shape)
```

```
(4, 4)
```

In [8]:

```
# Создадим класс линейной регрессии
class LinearRegression:
    def fit(self, train_features, train_target):
        X = np.concatenate((np.ones((train_features.shape[0], 1)), train_features), axis
=1)
        y = train_target
        w = np.linalg.inv(X.T @ X) @ X.T @ y
        self.w = w[1:]
        self.w0 = w[0]

    def predict(self, test_features):
        return test_features.dot(self.w) + self.w0
```

In [9]:

```
# Обучаем модель с сырыми признаками
```

```
model = LinearRegression()
model.fit(features, target)
predictions = model.predict(features)
print(r2_score(target, predictions))
```

```
0.42494550286668
```

In [10]:

```
# Обучаем модель, умножив признаки на обратную матрицу

features_inv = features @ np.linalg.inv(random_matrix)

model2 = LinearRegression()
model2.fit(features_inv, target)
predictions = model2.predict(features_inv)
print(r2_score(target, predictions))
```

```
0.4249435753006967
```

In [11]:

```
# Обучаем модель, умножив признаки на обычную матрицу

features_not_inv = features @ random_matrix

model2 = LinearRegression()
model2.fit(features_not_inv, target)
predictions = model2.predict(features_not_inv)
print(r2_score(target, predictions))
```

```
0.4249447510698463
```

Алгоритм преобразования

Алгоритм

Чтобы зашифровать данные, нужен ключ P (случайная матрица, которую мы умножаем), размер квадратной матрицы которого равен ширине матрицы признаков.

Алгоритм шифроки следующий:

$$C = X * P$$

Где X - исходные признаки,

P - случайная матрица, являющейся ключом,

а C - зашифрованные признаки

Для расшифровки закодированных признаков потребуется следующий шаг:

$$X = C * P^{-1}$$

Обоснование

код ниже.

In [12]:

```
# Создадим функцию зашифровки данных

def to_encode_features(features):
    return features @ random_matrix
```

In [13]:

```
encode_features = to_encode_features(features)
print(encode_features)
```

	0	1	2	3
0	479111.169620	488077.191309	538048.261279	534063.000039
1	367216.264781	374039.958713	412359.336633	409301.561877
2	202968.923423	206729.781258	227914.282182	226223.361597
3	402656.426489	410234.498886	452215.829764	448869.477557
4	252182.977233	256881.776121	283191.649528	281092.581344
...	...	...	...	...
4995	344833.595932	351289.651123	387254.534988	384386.542375
4996	506043.904456	515545.528066	568315.767500	564108.434811
4997	327375.549263	333526.375817	367662.361888	364941.022691
4998	315838.204863	321760.581343	354695.234331	352069.349229
4999	392118.196923	399474.197589	440364.839939	437104.436914

[5000 rows x 4 columns]

In [14]:

```
def to_decode_features(encoded_features):
    return encoded_features @ np.linalg.inv(random_matrix)
```

In [15]:

```
print(to_decode_features(encode_features))
```

	0	1	2	3
0	1.000000e+00	41.0	49600.0	1.000000e+00
1	-2.469197e-11	46.0	38000.0	1.000000e+00
2	3.915663e-11	29.0	21000.0	2.654278e-10
3	-2.602776e-10	21.0	41700.0	2.000000e+00
4	1.000000e+00	28.0	26100.0	2.774740e-10
...	...	...	...	...
4995	-2.762713e-10	28.0	35700.0	2.000000e+00
4996	1.978730e-10	34.0	52400.0	1.000000e+00
4997	-2.062390e-10	20.0	33900.0	2.000000e+00
4998	1.000000e+00	22.0	32700.0	3.000000e+00
4999	1.000000e+00	28.0	40600.0	1.000000e+00

[5000 rows x 4 columns]

In [16]:

```
features
```

Out[16]:

	Пол	Возраст	Зарплата	Члены семьи
0	1	41.0	49600.0	1
1	0	46.0	38000.0	1
2	0	29.0	21000.0	0
3	0	21.0	41700.0	2
4	1	28.0	26100.0	0
...	...	...	...	...
4995	0	28.0	35700.0	2
4996	0	34.0	52400.0	1
4997	0	20.0	33900.0	2
4998	1	22.0	32700.0	3
4999	1	28.0	40600.0	1

5000 rows × 4 columns

Проверка алгоритма

In [17]:

```
# Обучим модель с незашифрованными данными

model = LinearRegression()
model.fit(features, target)
predictions = model.predict(features)
print(r2_score(target, predictions))
```

```
0.42494550286668
```

In [18]:

```
# Обучим модель с зашифрованными данными

model_encode = LinearRegression()
model_encode.fit(encode_features, target)
predictions_encode = model_encode.predict(encode_features)
print(r2_score(target, predictions_encode))
```

```
0.4249447510698463
```

Вывод

Удалось построить алгоритм зашифровки данных без потери качества метрики, применив шифр Хилла.

Чек-лист проверки

Поставьте **'x'** в выполненных пунктах. Далее нажмите **Shift+Enter.**

- **[x] Jupyter Notebook** открыт
- **[x]** Весь код выполняется без ошибок
- **[x]** Ячейки с кодом расположены в порядке исполнения
- **[x]** Выполнен шаг **1:** данные загружены
- **[x]** Выполнен шаг **2:** получен ответ на вопрос об умножении матриц
  - **[x]** Указан правильный вариант ответа
  - **[x]** Вариант обоснован
- **[x]** Выполнен шаг **3:** предложен алгоритм преобразования
  - **[x]** Алгоритм описан

    - **[x]** Алгоритм обоснован
- **[x]** Выполнен шаг **4:** алгоритм проверен
    - **[x]** Алгоритм реализован
    - **[x]** Проведено сравнение качества моделей до и после преобразования