Machine Learning
Home assignment 1: Supervised learning

Solutions are by Yaroslava Lochman.

Problem 1 (Зважена лінійна регресія; 15 балів)**.** Зважена лінійна регресія — це регресія, у якій ми по-різному оцінюємо помилку для кожного з навчальних прикладів. Для навчання зваженої лінійної регресії нам потрібно мінімізувати функцію втрат виду:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m} \omega^{(i)} \left(\theta^\top x^{(i)} - y^{(i)}\right)^2$$

a. **(5 балів)** Покажіть, що функція втрат зваженої лінійної регресії $j(\theta)$ також може бути записана у такому вигляді:

$$J(\theta) = (X\theta - \mathbf{y})^\top W (X\theta - \mathbf{y})$$

де $X, \mathbf{y}$ визначені так само, як на лекції, а W — діагональна матриця. Поясніть, що таке матриця W та якими будуть її елементи.

b. **(10 балів)** Якщо всі $\omega^{(i)} = 1$, тоді, як ми бачили на лекції, нормальне рівняння має вигляд:

$$X^\top X\theta = X^\top \mathbf{y}$$

а значення θ, що мінімізує функцію втрат і дає найвищу точність передбачення:

$$\theta = \left(X^\top X\right)^{-1} X^\top \mathbf{y}$$

Виведіть вираз для знаходження θ у зваженій лінійній регресії. Знайдіть градієнт $\nabla_\theta J(\theta)$ і, прирівнявши його до нуля, виведіть нормальне рівняння для знаходження θ. Вираз буде залежати від X, W і $\mathbf{y}$.

Solution to the problem 1. .

a. Let's recall:

$$X = \begin{pmatrix} - & x^{(1)\top} & - \\ & \vdots & \\ - & x^{(m)\top} & - \end{pmatrix} \in \mathbb{R}^{m\times(n+1)} \quad \text{, where } x^{(i)} = \begin{pmatrix} 1 & x_1 & \cdots & x_n \end{pmatrix}^\top \in \mathbb{R}^{n+1} \qquad \mathbf{y} = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{pmatrix} \in \mathbb{R}^m$$

$$\theta = \begin{pmatrix} \theta_0 & \cdots & \theta_n \end{pmatrix}^\top \in \mathbb{R}^{n+1} \qquad W = diag\left(\omega^{(1)}, \ldots, \omega^{(m)}\right) \quad \text{, where } \omega^{(i)} = \omega(x^{(i)}, x) - \text{ weight f-n}$$

$$X\theta - \mathbf{y} = \begin{pmatrix} \theta^\top x^{(1)} - y^{(1)} \\ \vdots \\ \theta^\top x^{(m)} - y^{(m)} \end{pmatrix}$$

Let $X\theta - \mathbf{y} = \mathbf{z} = \begin{pmatrix} z_1 & \cdots & z_m \end{pmatrix}^\top$, so $z_i = \theta^\top x^{(i)} - y^{(i)}$. Then

$$W(X\theta - \mathbf{y}) = W\mathbf{z} = \begin{pmatrix} \omega^{(1)} z_1 & \cdots & \omega^{(m)} z_m \end{pmatrix}^\top$$

$$(X\theta - \mathbf{y})^\top W(X\theta - \mathbf{y}) = \mathbf{z}^\top W\mathbf{z} = \begin{pmatrix} z_1 & \cdots & z_m \end{pmatrix} \begin{pmatrix} \omega^{(1)} z_1 \\ \vdots \\ \omega^{(m)} z_m \end{pmatrix} =$$

$$= \sum_{i=1}^{m} z_i \omega^{(i)} z_i = \sum_{i=1}^{m} \omega^{(i)} z_i^2 = \sum_{i=1}^{m} \omega^{(i)} (\theta^\top x^{(i)} - y^{(i)}))^2 = 2J(\theta)$$

So we'll modify the construction of W s.t. $W = diag\left(\frac{\omega^{(1)}}{2}, \ldots \frac{\omega^{(m)}}{2}\right)$ and therefore:

$$J(\theta) = (X\theta - \mathbf{y})^\top W(X\theta - \mathbf{y})$$

So $W = W(x)$ is a diagonal matrix s.t. $W_{ii} = \frac{\omega^{(i)}}{2} = \frac{\omega(x^{(i)},\, x)}{2}$ is a half of the chosen weight function value in the observation point $x^{(i)}$ for the point of estimation x.

b. Since $W^\top = W$, and using the properties of trace in terms of the gradient we have:

$$\nabla_\theta J(\theta) = \nabla_\theta \left((X\theta - \mathbf{y})^\top W(X\theta - \mathbf{y})\right) = \nabla_\theta \left((\theta^\top X^\top - \mathbf{y}^\top)(WX\theta - W\mathbf{y})\right) =$$

$$= \nabla_\theta \left(\theta^\top X^\top WX\theta - \mathbf{y}^\top WX\theta - \theta^\top X^\top W\mathbf{y} + \mathbf{y}^\top W\mathbf{y}\right) =$$

$$= \nabla_\theta \operatorname{tr} \left(\theta^\top X^\top WX\theta - \mathbf{y}^\top WX\theta - \theta^\top X^\top W\mathbf{y} + \mathbf{y}^\top W\mathbf{y}\right) =$$

$$= \nabla_\theta \left(\operatorname{tr} \theta^\top X^\top WX\theta - 2 \operatorname{tr} \mathbf{y}^\top WX\theta\right) = (X^\top WX)^\top \theta + X^\top WX\theta - 2X^\top W\mathbf{y} =$$

$$= 2(X^\top WX\theta - X^\top W\mathbf{y}) = 0$$

Therefore the normal equation is:

$$X^\top WX\theta = X^\top W\mathbf{y}$$

So the value of θ that minimizes $J(\theta)$ is:

$$\theta = (X^\top WX)^{-1} X^\top W\mathbf{y}$$

Problem 2 (Регресія Пуассона та сімейство експоненціальних моделей; 25 балів)**.** Ви маєте задачу: передбачити кількість звернень у службу підтримки вашого сайту в певний день. У службу підтримки за день звертається цілочисельна кількість людей, яких зазвичай не більше 7–10, тому ви вирішили використовувати розподіл Пуассона для моделювання таких звернень.

a. **(7 балів)** Розподіл імовірностей Пуассона має вигляд:

$$P(y; \lambda) = \frac{e^{-\lambda} \lambda^y}{y!}$$

Покажіть, що розподіл Пуассона належить до експоненціального сімейства і вкажіть, чому дорівнюють $b(y)$, η, $T(y)$, $a(\eta)$.

b. **(3 бали)** Якою буде канонічна функція відгуку (canonical response function) для цього розподілу? Ви можете використати той факт, що випадкова величина з розподілом Пуассона з параметром λ має середнє значення λ.

c. **(15 балів)** Для навчальної вибірки $\{\left(x^{(i)}, y^{(i)}\right);\ i = \overline{1,m}\}$ логарифмічна функція правдоподібності (log-likelihood) екземпляра буде:

$$l_i(\theta) = \log P(y^{(i)} \mid x^{(i)}; \theta)$$

Виведіть похідну $\frac{\partial}{\partial \theta_j} l(\theta)$ та сформулюйте правило оновлення ваги методом стохастичного градієнтного підйому, якщо y має розподіл Пуассона та канонічну функцію відгуку.

Solution to the problem 2. So we make an assumption that $y \mid x; \theta \sim Pois(\lambda)$.

a.

$$P(y;\lambda) = \frac{e^{-\lambda}\lambda^y}{y!} = \frac{1}{y!}e^{-\lambda}e^{y(\ln\lambda)} = \frac{1}{y!}\exp\left((\ln\lambda)y - \lambda\right) = b(y)\exp\left(\eta^\top T(y) - a(\eta)\right)$$

$T(y) = y;\ \ b(y) = \frac{1}{y!};\ \ \eta = \ln\lambda;\ \ a(\eta) = e^\eta = \lambda.$

b. The canonical response function:

$$g(\eta) = E(T(y)|\eta) = E(y|\eta) = \lambda = e^\eta \ \Rightarrow\ h(x) = g(\theta^\top x) = e^{\theta^\top x}$$

c. Log-likelihood of i^{th} training example:

$$l_i(\theta) = \ln\left(\frac{1}{y^{(i)}!}\exp\left(y^{(i)}(\ln\lambda) - \lambda\right)\right) = y^{(i)}(\ln\lambda) - \lambda - \ln\left(y^{(i)}!\right) = y^{(i)}\theta^\top x^{(i)} - e^{\theta^\top x} - \ln\left(y^{(i)}!\right)$$

It's derivative with respect to θ_j:

$$\frac{\partial}{\partial\theta_j} l_i(\theta) = y^{(i)}x_j^{(i)} - e^{\theta^\top x^{(i)}}x_j^{(i)}$$

So the gradient is:

$$\nabla_\theta l_i(\theta) = y^{(i)}x^{(i)} - e^{\theta^\top x^{(i)}}x^{(i)} = \left(y^{(i)} - e^{\theta^\top x^{(i)}}\right)x^{(i)}$$

Therefore in the stochastic gradient ascent (since the objective function is log-likelihood is to be maximized) the weights are updated this way:

$$\theta = \theta + \alpha\nabla_\theta l_i(\theta) = \theta + \alpha\left(y^{(i)} - e^{\theta^\top x^{(i)}}\right)x^{(i)} \ \ \left[\,= \theta + \alpha\left(y^{(i)} - h(x^{(i)})\right)x^{(i)}\,\right]$$

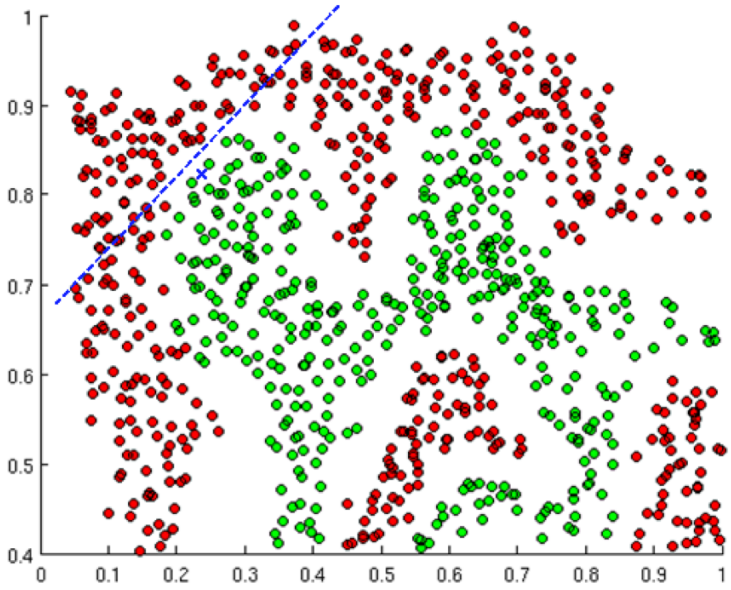
This update inside a single stochastic gradient ascent step is repeated for every i from 1 to m.

Problem 3 (Зважена логістична регресія; 30 балів)**.** Деякі моделі навчання з вчителем, такі як нейронні мережі, наприклад, мають функцію гіпотези дуже високої варіативності. Інтуїтивно, це означає, що вони мають дуже «гнучку» межу класів (decision boundary). Проте, їх передбачення часто важко пояснити. В деяких задачах (наприклад, при

автоматизованій діагностиці в медицині) пояснення рішень моделей машинного навчання так само важливо, як і їх точність. Без цього лікарі не довіряють висновкам машини.

Проте, ми можемо модифікувати логістичну регресію так, щоб вона була здатна робити якісні передбачення навіть в умовах дуже нелінійної межі класів (decision boundary). Передбачення логістичної регресії пояснювати доволі просто.

Ваша задача – модифікувати логістичну регресію таким чином, щоб вона передбачала якомога точніше найближчі точки до точки запиту (query point), аналогічно зваженій лінійній регресії.



Для обрахунку вагових коефіцієнтів ми будемо використовувати таку саму формулу, яку використовували для зваженої лінійної регресії, записану у векторній формі:

$$\omega^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^\top (x^{(i)} - x)}{2\tau^2}\right)$$

a. **(5 балів)** Виведіть формулу функції втрат (cost function) для зваженої логістичної регресії.

b. **(10 балів)** Виведіть правило оновлення ваг θ для зваженої логістичної регресії методом градієнтного спуску.

c. **(15 балів)** Виведіть рішення зваженої логістичної регресії методом нормальних рівнянь.

Solution to the problem 3. .

a. In the original logistic regression the cost function to be minimized is a binary cross entropy:

$$J(\theta) = -\log L(\theta) = -\sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log\left(1 - h(x^{(i)})\right)$$

where $h(x^{(i)}) = \sigma(\theta^\top x^{(i)})$, $\sigma(\cdot)$ – sigmoid function.
In the weighted logistic regression we use the weights s.t. to minimize:

$$J(\theta) = -\sum_{i=1}^{m} \omega^{(i)} \left(y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log \left(1 - h(x^{(i)})\right)\right)$$

b.

$$\frac{\partial}{\partial \theta_j} J(\theta) = -\sum_{i=1}^{m} \omega^{(i)} \left(y^{(i)} \frac{\partial}{\partial \theta_j} \log \sigma(\theta^\top x^{(i)}) + (1 - y^{(i)}) \frac{\partial}{\partial \theta_j} \log \left(1 - \sigma(\theta^\top x^{(i)})\right)\right) =$$

$$= -\sum_{i=1}^{m} \omega^{(i)} \left(y^{(i)} \frac{1}{\sigma(\theta^\top x^{(i)})} \frac{\partial}{\partial \theta_j} \sigma(\theta^\top x^{(i)}) - (1 - y^{(i)}) \frac{1}{1 - \sigma(\theta^\top x^{(i)})} \frac{\partial}{\partial \theta_j} \sigma(\theta^\top x^{(i)})\right) =$$

$$\left| \frac{\partial}{\partial \theta_j} \sigma(\theta^\top x^{(i)}) = \sigma(\theta^\top x^{(i)}) \left(1 - \sigma(\theta^\top x^{(i)})\right) x_j^{(i)} \right|$$

$$= -\sum_{i=1}^{m} \omega^{(i)} \left(y^{(i)} \left(1 - \sigma(\theta^\top x^{(i)})\right) - (1 - y^{(i)}) \sigma(\theta^\top x^{(i)})\right) x_j^{(i)} = -\sum_{i=1}^{m} \omega^{(i)} \left(y^{(i)} - \sigma(\theta^\top x^{(i)})\right) x_j^{(i)}$$

$$\nabla_\theta J(\theta) = -\sum_{i=1}^{m} \omega^{(i)} \left(y^{(i)} - \sigma(\theta^\top x^{(i)})\right) x^{(i)} = -X^\top W \left(\mathbf{y} - \sigma(X\theta)\right)$$

where $W = diag(\omega^{(1)}, \ldots, \omega^{(m)})$.

So the parameters update inside the batch gradient descent step is:

$$\theta = \theta - \alpha \nabla_\theta J(\theta) = \theta + \alpha X^\top W \left(\mathbf{y} - \sigma(X\theta)\right)$$

c. To find the minimum of the cost function we make it's gradient with respect to θ to be equal to zero:

$$\nabla_\theta J(\theta) = -X^\top W \left(\mathbf{y} - \sigma(X\theta)\right) = 0$$
$$\Rightarrow X^\top W \sigma(X\theta) = X^\top W \mathbf{y} =: \mathbf{b} \in \mathbb{R}^{n+1}$$
$$\begin{cases} \sum_{i=1}^{m} \omega^{(i)} \sigma(\theta^\top x^{(i)}) = b_0 \\ \sum_{i=1}^{m} x_1^{(i)} \omega^{(i)} \sigma(\theta^\top x^{(i)}) = b_1 \\ \vdots \\ \sum_{i=1}^{m} x_n^{(i)} \omega^{(i)} \sigma(\theta^\top x^{(i)}) x^{(i)} = b_n \end{cases}$$

So to find θ we need to solve the above system of nonlinear equations (since $\sigma(z) = \frac{1}{1+e^{-z}}$ is nonlinear function), but the closed-form solutioin can't be found so far.

Problem 4 (Масштабування ознак в лінійній регресії; 15 балів)**.** Ви побудували лінійну регресію для передбачення витрат палива на 100 км. шляху. Як ознаки (features) ви використали вагу автомобіля з пасажирами, об'єм двигуна та швидкість в кілометрах на годину. Після впровадження моделі у виробництво виявилось, що ознака швидкості приходить не в кілометрах на годину, а в милях на годину. Ваша модель вже реалізована на мікросхемі, змінити її чи формат даних, які поступають на вхід, дуже дорого. Проте, змінити передбачення перед тим, як видавати його користувачу, просто і дешево. Як можна модифікувати передбачення так, щоб воно було коректним?

Solution to the problem 4. . Prediction model for y:

$$h(\mathbf{x}) = \theta^\top \mathbf{x}; \qquad \mathbf{x} = \begin{pmatrix} 1 & x_1 & x_2 & x_3 \end{pmatrix}^\top$$

where units of the velocity x_3 are miles per hour.

Now we want to predict y based on the velocity in kilometers per hour. Let's denote transformed feature to change units from miles to kilometers: $\hat{x}_3 = ax_3$, $a = 1.609344$, $\hat{\mathbf{x}} = \begin{pmatrix} 1 & x_1 & x_2 & \hat{x}_3 \end{pmatrix}^\top$. So we want to get $\hat{h}(\hat{\mathbf{x}}) = f(h(\hat{\mathbf{x}}))$.

$$\mathbf{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \frac{1}{a}\hat{x}_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{1}{a} \end{pmatrix} \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \hat{x}_3 \end{pmatrix} = A\hat{\mathbf{x}}$$

$$\hat{h}(\hat{\mathbf{x}}) = h(\mathbf{x}) = \theta^\top \mathbf{x} = \theta^\top A\hat{\mathbf{x}} = (A^\top \theta)^\top \hat{\mathbf{x}}$$

$$h(\hat{\mathbf{x}}) = \theta^\top \hat{\mathbf{x}}$$

$$\hat{h}(\hat{\mathbf{x}}) - h(\hat{\mathbf{x}}) = (A^\top \theta)^\top \hat{\mathbf{x}} - \theta^\top \hat{\mathbf{x}} = ((A^\top - I)\theta)^\top \hat{\mathbf{x}} = \frac{1}{a}\theta_3 \hat{x}_3$$

$$\hat{h}(\hat{\mathbf{x}}) = h(\hat{\mathbf{x}}) + \frac{1}{a}\theta_3 \hat{x}_3$$

Problem 5 (Програмування: зважена лінійна регресія; 35 балів)**.** У цьому завданні ви працюватимете з реальними даними з зарплатного опитування DOU.ua за травень 2016р. Ви реалізуєте зважену лінійну регресію, яка передбачає зарплати Java-інженерів, та навчите свою модель за допомогою градієнтного спуску.

У записнику **salary-prediction-wlr.ipynb** уже реалізована підготовка даних, візуалізація та оцінка результатів моделі. Вам залишається реалізувати саму логіку зваженої лінійної регресії, заповнивши пропущені місця в коді.

Після того, як завершите, запустіть останню комірку — вона містить автоматичні тести, що перевірять правильність ваших обчислень. Ви повинні побачити повідомлення «OK», якщо все працює вірно.

a. **(3 бали)** Реалізуйте функцію гіпотези зваженої лінійної регресії.

b. **(10 балів)** Реалізуйте функцію зважування навчальних прикладів.

c. **(7 балів)** Реалізуйте функцію втрат зваженої лінійної регресії.

d. **(12 балів)** Обчисліть градієнт функції втрат зваженої лінійної регресії.

e. **(3 балів)** Реалізуйте правило оновлення ваг при градієнтному спуску.

Solution to the problem 5. (See also **salary-prediction-wlr.ipynb**)

1 Передбачення зарплат на IT-ринку України

У цьому завданні ви працюватимете з реальними даними з зарплатного опитування DOU.ua за травень 2016р. Ви реалізуєте зважену лінійну регресію, яка передбачає зарплати Java-інженерів, та навчите свою модель за допомогою градієнтного спуску.

Заповніть пропущений код в розділі «Моделювання» (позначено коментарями) та запустіть розділ «Тестування», щоб перевірити його правильність.

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
```

```
In [2]: np.random.seed(42)
```

```
In [3]: %matplotlib inline
```

```
In [4]: %load_ext ipython_unittest
```

1.1 Підготовка даних

```
In [5]: df_salaries = pd.read_csv("data/2016_may_final.csv")
```

Оберемо тільки Java-інженерів з-поміж усіх респондентів.

```
In [6]: df_java = pd.DataFrame(df_salaries[(df_salaries["Язык.программирования"] == \
                                                                    "Java") &
                                           (df_salaries["cls"] == "DEV")])
```

Перейменуємо деякі колонки:

```
In [7]: df_java.rename(
            columns={
                "exp": "TotalExperience",
                "loc": "Location"
            },
            inplace=True
        )
```

Закодуємо рівень англійської мови числами від 1 (найнижчий) до 5 (найвищий):

```
In [8]: df_java["EnglishLevel"] = df_java["Уровень.английского"].map({
            "элементарный": 1,
            "ниже среднего": 2,
            "средний": 3,
            "выше среднего": 4,
            "продвинутый": 5
        })
```

Закодуємо колонку Location (найбільші IT-міста або "other") за допомогою one-hot encoding:

```
In [9]: city_columns = [
            "LocationOther",
            "LocationDnipro",
            "LocationKyiv",
            "LocationLviv",
            "LocationOdesa",
            "LocationKharkiv"
        ]
        df_java[city_columns] = pd.get_dummies(df_java["Location"])
```

Відберемо такі ознаки:

- Загальна кількість років досвіду
- Рівень англійської мови
- Місто

```
In [10]: feature_columns = ["TotalExperience", "EnglishLevel"] + city_columns
         df_X = df_java[feature_columns]
         df_y = df_java[["salary"]]

In [11]: print("X shape:", df_X.shape)

X shape: (929, 8)


In [12]: df_X.head(10)

Out[12]:     TotalExperience  EnglishLevel  LocationOther  LocationDnipro  \
         5               0.5             3              1               0
         7               5.0             1              0               0
         17              0.0             3              0               0
         27              4.0             3              0               0
         28              6.0             4              0               0
         39              3.0             4              0               1
         46              2.0             4              0               0
         49              3.0             3              0               0
         59              2.0             3              0               0
         89              1.0             5              0               0

             LocationKyiv  LocationLviv  LocationOdesa  LocationKharkiv
         5               0             0              0                0
         7               1             0              0                0
         17              1             0              0                0
         27              1             0              0                0
```

```
28          1          0          0          0
39          0          0          0          0
46          0          0          0          1
49          1          0          0          0
59          0          0          0          1
89          1          0          0          0
```

```
In [13]: df_y.head(10)
```

```
Out[13]:     salary
         5      500
         7     1600
         17     600
         27    3400
         28    2880
         39    1425
         46    1700
         49    1800
         59    1235
         89    1200
```

Розділимо вибірку на навчальну та тестову:

```
In [14]: training_set_size = 0.8
```

```
In [15]: dataset_assignment = np.random.uniform(size=len(df_X))

         X_train = df_X[dataset_assignment <= training_set_size].values
         y_train = df_y[dataset_assignment <= training_set_size].values.flatten()

         X_test = df_X[dataset_assignment > training_set_size].values
         y_test = df_y[dataset_assignment > training_set_size].values.flatten()
```

Щоб градієнтний спуск швидше збігався, нормалізуємо навчальну вибірку так, щоб кожна ознака мала $\mu = 0, \sigma = 1$:

$$x' = \frac{x - \bar{x}}{\sigma}$$

```
In [16]: feature_means = np.average(X_train, axis=0)
         feature_sigmas = np.std(X_train, axis=0)
```

```
In [17]: X_train = (X_train - feature_means) / feature_sigmas
         X_test = (X_test - feature_means) / feature_sigmas
```

Додаємо уявну ознаку $x_0 = 1$ (intercept term).

```
In [18]: if not np.all(X_train[:, 0] == 1):
             X_train = np.insert(X_train, 0, values=1, axis=1)

         if not np.all(X_test[:, 0] == 1):
             X_test = np.insert(X_test, 0, values=1, axis=1)

In [19]: print("X train: ", X_train.shape)
         print("y train: ", y_train.shape)
         print()
         print("X test:  ", X_test.shape)
         print("y test:  ", y_test.shape)

X train:  (741, 9)
y train:  (741,)

X test:   (188, 9)
y test:   (188,)
```

1.2 Моделювання

Реалізуйте функцію гіпотези лінійної регресії в матричній формі.

```
In [20]: def predict_linear(theta, X):
             # =============== TODO: Your code here ===============
             # Compute the hypothesis function for linear regression.
             y_hat = X.dot(theta)
             return y_hat
             # ====================================================
```

Реалізуйте функцію зважування всіх навчальних прикладів $x^{(i)}$, якщо нам дана точка передбачення x.

```
In [21]: def get_example_weights(X, x_pred, tau):
             # =============== TODO: Your code here ===============
             # Compute the weight for each example, given the
             # prediction point (x_pred).
             W = np.exp(- (X - x_pred).dot((X - x_pred).T) /(2 * tau ** 2))
             weights = np.diag(W)
             return weights
             # ====================================================
```

Реалізуйте функцію втрат зваженої лінійної регресії. Подумайте, як обчислити цей вираз відразу в матричному вигляді.

```
In [22]: def cost_function(theta, X, y, weights):
             # =============== TODO: Your code here ===============
```

```
            # Given the currently learned model weights (theta),
            # compute the overall loss on the training set (X),
            # taking the weights into account.
            W = np.diag(weights) / 2
            y_hat = predict_linear(theta, X)
            res = y_hat - y
            J = res.T.dot(W.dot(res)) / len(y)
            return J
            # ====================================================
```

Реалізуйте обчислення градієнта функції втрат зваженої лінійної регресії. Подумайте, як обчислити цей вираз відразу в матричному вигляді.

```
In [23]: def cost_function_gradient(theta, X, y, weights):
            # ================ TODO: Your code here ================
            # Given the currently learned model weights (theta),
            # compute the gradient of the cost function on the
            # training set (X), taking the weights into account.
            W = np.diag(weights)
            y_hat = predict_linear(theta, X)
            res = y_hat - y
            grad_J = X.T.dot(W.dot(res))
            return grad_J
            # ====================================================
```

Реалізуйте один крок градієнтного спуску.

```
In [24]: def update_model_weights(theta, learning_rate, cost_gradient):
            # ================ TODO: Your code here ================
            # Given the learning rate and the gradient of the
            # cost function, take one gradient descent step and
            # return the updated vector theta.
            return theta - learning_rate * cost_gradient
            # ====================================================
```

Навчаємо модель:

```
In [25]: def gradient_descent(X, y, weights, loss_fun, grad_fun, learning_rate,
                              convergence_threshold, max_iters, verbose=False):
            theta = np.zeros(X.shape[1])
            losses = []

            for i in range(max_iters):
                loss = loss_fun(theta, X, y, weights)
                losses.append(loss)

                if verbose:
```

```
            print("Iteration: {0:3} Loss: {1}".format(i + 1, loss))

        if len(losses) > 2 \
        and np.abs(losses[-1] - losses[-2]) <= convergence_threshold:
            break

        grad = grad_fun(theta, X, y, weights)
        theta = update_model_weights(theta, learning_rate, grad)

    return theta, np.array(losses)
```

1.3 Передбачення нових даних

```
In [43]: def predict_weighted_linear(X, y, x_pred, verbose=False):
             weights = get_example_weights(X, x_pred, tau=0.1)
             theta, losses = gradient_descent(
                 X,
                 y,
                 weights,
                 loss_fun=cost_function,
                 grad_fun=cost_function_gradient,
                 learning_rate=0.005,
                 convergence_threshold=0.0001,
                 max_iters=500,
                 verbose=verbose
             )
             return predict_linear(theta, x_pred)
```

```
In [44]: x_pred = X_train[4]
```

```
In [45]: predicted = predict_weighted_linear(X_train, y_train, x_pred, verbose=True)
```

```
Iteration:   1 Loss: 2051.275762424122
Iteration:   2 Loss: 1513.80754533939
Iteration:   3 Loss: 1121.5661412085203
Iteration:   4 Loss: 835.3104813807897
Iteration:   5 Loss: 626.4026457393463
Iteration:   6 Loss: 473.942834742058
Iteration:   7 Loss: 362.6784874795104
Iteration:   8 Loss: 281.4783673182373
Iteration:   9 Loss: 222.21895783853518
Iteration:  10 Loss: 178.97176060000322
Iteration:  11 Loss: 147.410189437005
Iteration:  12 Loss: 124.37672511648795
Iteration:  13 Loss: 107.56702714333422
Iteration:  14 Loss: 95.29940026075593
Iteration:  15 Loss: 86.3465523464754
```

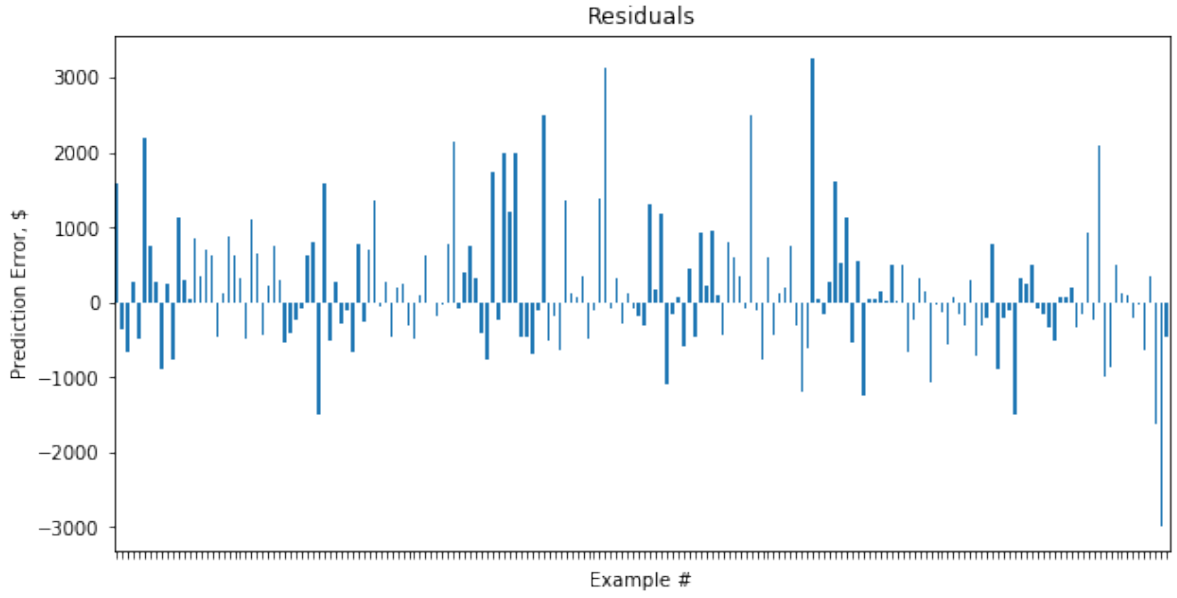
```
Iteration:  16 Loss: 79.8128122333628
Iteration:  17 Loss: 75.04452393759317
Iteration:  18 Loss: 71.56465284995903
Iteration:  19 Loss: 69.02506168722135
Iteration:  20 Loss: 67.1716817389729
Iteration:  21 Loss: 65.8190950315699
Iteration:  22 Loss: 64.83198452854961
Iteration:  23 Loss: 64.11159658700915
Iteration:  24 Loss: 63.5858613313323
Iteration:  25 Loss: 63.2021825552563
Iteration:  26 Loss: 62.92217583131277
Iteration:  27 Loss: 62.71782841149648
Iteration:  28 Loss: 62.56869674349827
Iteration:  29 Loss: 62.45986123317697
Iteration:  30 Loss: 62.3804336412496
Iteration:  31 Loss: 62.322467789451316
Iteration:  32 Loss: 62.2801645997701
Iteration:  33 Loss: 62.24929193637043
Iteration:  34 Loss: 62.22676120936045
Iteration:  35 Loss: 62.2103183825062
Iteration:  36 Loss: 62.19831847232921
Iteration:  37 Loss: 62.189560978764874
Iteration:  38 Loss: 62.18316978334519
Iteration:  39 Loss: 62.178505499540584
Iteration:  40 Loss: 62.175101506512576
Iteration:  41 Loss: 62.172617266891585
Iteration:  42 Loss: 62.17080425834318
Iteration:  43 Loss: 62.16948111060899
Iteration:  44 Loss: 62.1685154606533
Iteration:  45 Loss: 62.16781071064699
Iteration:  46 Loss: 62.167296364015726
Iteration:  47 Loss: 62.16692097274004
Iteration:  48 Loss: 62.166646990332964
Iteration:  49 Loss: 62.16644701557097
Iteration:  50 Loss: 62.166301051189
Iteration:  51 Loss: 62.166194503290946
Iteration:  52 Loss: 62.16611672133018
```

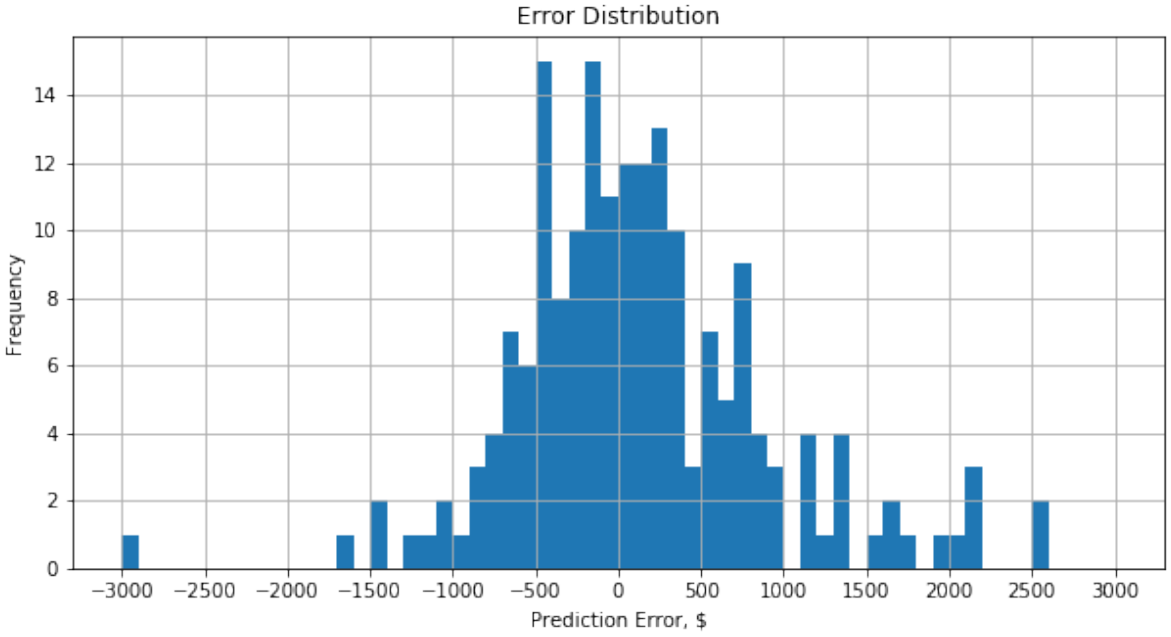
Оцінимо, наскільки модель помиляється на тестовій вибірці:

```
In [46]: y_test_pred = np.array([predict_weighted_linear(X_train, y_train, X_test[i]) \
                                 for i in range(len(y_test))])

In [47]: df_residuals = pd.DataFrame(y_test - y_test_pred, columns=["residual"])

In [48]: ax = df_residuals.plot.bar(figsize=(10, 5))
         ax.legend().remove()
```

```
ax.get_xaxis().set(ticklabels=[])
ax.set(xlabel="Example #")
ax.set(ylabel="Prediction Error, $")
ax.set(title="Residuals")

bins = np.arange(-3000, 3001, 100)
ticks = np.arange(-3000, 3001, 500)

ax = df_residuals.hist(bins=bins, figsize=(10, 5))[0][0]
ax.set(xticks=ticks)
ax.set(xlabel="Prediction Error, $")
ax.set(ylabel="Frequency")
ax.set(title="Error Distribution")
```

Out[48]: [Text(0.5, 1.0, 'Error Distribution')]





1.4 Тестування

Запустіть комірку нижче, щоб перевірити правильність вашого коду:

In [49]:
```
%%unittest_main

class WLRTests(unittest.TestCase):

    x = np.array([1, -0.5, 3, 1])
    X = np.array([
        [1, -0.5, 3, 1],
        [2, 8, -0.33, 5],
        [0, 0, 0, 0]
    ])
    y = np.array([40, 100, 12])
    theta = np.array([2, 5, 7, 9])
    eps = 0.001

    def assertFloatEquals(self, a, b):
        self.assertTrue(np.abs(a - b) < self.eps)

    def assertArrayEquals(self, a, b):
        a = np.array(a)
        b = np.array(b)
        self.assertEqual(a.shape, b.shape)
```

```
        self.assertTrue(np.all(np.abs(a - b) < self.eps))

    def test_predict_linear_should_compute_correct_prediction_for_1_example(self):
        expected = 29.5
        actual = predict_linear(self.theta, self.x)
        self.assertEqual(actual, expected)

    def test_predict_linear_should_compute_correct_predictions_for_multiple_example
        expected = [29.5, 86.69, 0]
        actual = (predict_linear(self.theta, self.X))
        self.assertArrayEquals(actual, expected)

    def test_get_example_weights_should_return_properly_shaped_vector(self):
        weights = get_example_weights(self.X, self.x, tau=5)
        self.assertTrue(weights.shape[0] == self.X.shape[0])

    def test_get_example_weights_should_compute_correct_weights(self):
        expected = [1.000, 0.134, 0.798]
        actual = get_example_weights(self.X, self.x, tau=5)
        self.assertArrayEquals(actual, expected)

    def test_cost_function_should_compute_correct_cost_unweighted(self):
        weights = np.ones(self.X.shape[0])
        expected = 71.901
        actual = cost_function(self.theta, self.X, self.y, weights)
        self.assertFloatEquals(actual, expected)

    def test_cost_function_should_compute_correct_cost_weighted(self):
        weights = np.array([0.5, 0.1, 0.28])
        expected = 18.860
        actual = cost_function(self.theta, self.X, self.y, weights)
        self.assertFloatEquals(actual, expected)

    def test_cost_gradient_should_return_properly_shaped_vector(self):
        weights = np.ones(self.X.shape[0])
        grad = cost_function_gradient(self.theta, self.X, self.y, weights)
        self.assertTrue(grad.shape == self.theta.shape)

    def test_cost_gradient_should_compute_correct_gradient_unweighted(self):
        weights = np.ones(self.X.shape[0])
        expected = [-37.12, -101.23, -27.108, -77.05]
        actual = cost_function_gradient(self.theta, self.X, self.y, weights)
        self.assertArrayEquals(actual, expected)

    def test_cost_gradient_should_compute_correct_gradient_weighted(self):
        weights = np.array([0.5, 0.1, 0.28])
```

```
            expected = [-7.912, -8.023, -15.311, -11.905]
            actual = cost_function_gradient(self.theta, self.X, self.y, weights)
            self.assertArrayEquals(actual, expected)

        def test_update_model_weights_should_not_update_when_gradient_is_zero(self):
            grad = np.zeros(self.theta.shape[0])
            theta_new = update_model_weights(self.theta, learning_rate=1, cost_gradient
            self.assertArrayEquals(theta_new, self.theta)

        def test_update_model_weights_should_update_with_complete_gradient_if_learning_
            grad = np.array([1.35, -0.89, 0.16, 0.98])
            expected = [0.65, 5.89, 6.84, 8.02]
            actual = update_model_weights(self.theta, learning_rate=1, cost_gradient=gr
            self.assertArrayEquals(actual, expected)

        def test_update_model_weights_should_take_learning_rate_into_account(self):
            grad = np.array([1.35, -0.89, 0.16, 0.98])
            expected = [1.730, 5.178, 6.968, 8.804]
            actual = update_model_weights(self.theta, learning_rate=0.2, cost_gradient=
            self.assertArrayEquals(actual, expected)


Success


...
----------------------------------------------------------------------
Ran 12 tests in 0.002s

OK


Out[49]: <unittest.runner.TextTestResult run=12 errors=0 failures=0>
```

Problem 6 (Програмування: класифікація спаму методом Баєса; 25 балів)**.** Ви застосуєте наївний Баєсівський класифікатор зі згладжуванням Лапласа для навчання спам-фільтру (на основі даних SpamAssassin Public Corpus).

Пригадайте, що ми розглядали дві моделі подій для класифікації текстів – **multivariate Bernoulli event model** та **multinomial event model**. Ваше завдання – реалізувати класифікацію методом multinomial event model.

Заповніть пропущений код у записнику **spam-bayes-multinomial.ipynb**. У вас повинен вийти кращий результат, ніж при multivariate Bernoulli ($> 95\%$ точності).

a. **(2 бали)** Закодуйте лист у вигляді вектора ознак.

b. **(3 бали)** Підрахуйте кількість слів у ham- та spam- листах.

c. **(3 балів)** Обчисліть апріорні ймовірності для класів ham та spam.

d. **(10 балів)** Обчисліть імовірності появи слів в рамках кожного класу.

e. **(7 балів)** Реалізуйте функцію класифікації нового листа.

Solution to the problem 6. (See also **spam-bayes-multinomial.ipynb**)

2 Класифікація спаму: Naive Bayes

Мультиноміальна модель подій

Ми застосуємо наївний Баєсівський класифікатор зі згладжуванням Лапласа для навчання спам-фільтру на основі даних SpamAssassin Public Corpus.

Заповніть пропущений код (позначено коментарями) та визначте точність передбачення. У вас повинен вийти кращий результат, ніж при моделі багатовимірного розподілу Бернуллі.

```
In [1]: import json
```

```
In [2]: import numpy as np

        from sklearn.model_selection import train_test_split
        from tqdm import tqdm_notebook as progressbar
```

2.1 Завантаження даних

Щоб краще зрозуміти, яким чином були очищені дані, див. spam-data-preparation.ipynb.

```
In [3]: def load_json_from_file(filename):
            with open(filename, "r", encoding="utf-8") as f:
                return json.load(f)
```

```
In [4]: emails_tokenized_ham = load_json_from_file("emails-tokenized-ham.json")
        emails_tokenized_spam = load_json_from_file("emails-tokenized-spam.json")
```

```
In [5]: vocab = load_json_from_file("vocab.json")
```

2.2 Кодування даних

Представте кожен лист як n-вимірний вектор $[x_1, x_2, ..., x_n]$, де x_i — це індекс i-го слова даного листа у словнику V, а n — кількість слів у листі.

Наприклад, лист *"Buy gold watches. Buy now."* міг би бути закодований так: $[3953, 11890, 32213, 3953, 20330]$.

```
In [6]: def email_to_vector_multinomial(email_words, vocab):
            # =============== TODO: Your code here ===============
            # Build a feature vector for a single email using the
            # multinomial event model.
            return np.array(list(map(vocab.get, email_words)))
            # ====================================================
```

Тепер закодуємо всі листи:

```
In [7]: X = [
            email_to_vector_multinomial(email, vocab)
            for email in emails_tokenized_ham + emails_tokenized_spam
        ]

In [8]: y = np.array([0] * len(emails_tokenized_ham) + \
                     [1] * len(emails_tokenized_spam))
```

Поглянемо на кілька випадкових листів:

```
In [9]: sample_emails = [emails_tokenized_ham[10], emails_tokenized_ham[70]]

In [10]: for email in sample_emails:
             print(email)
             print()

['hello', 'seen', 'discuss', 'articl', 'approach', 'thank', 'httpaddress', 'hell', 'rule', '

['fri', 'number', 'aug', 'number', 'tom', 'wrote', 'xvid', 'number', 'project', 'make', 'gpl



In [11]: for email in sample_emails:
             email_vec = email_to_vector_multinomial(email, vocab)

             print("Email vector:", email_vec)
             print("Dimensionality:", email_vec.shape)
             print()

Email vector: [12866 26186  7632  1574  1361 29410 13468 12862 25408 30214   173 27396
 29564   872  8567 26411 19758  8849 27722 21116 29770 20748  4549 22215
 11528 19855 10871 13468 27540  7314 17535 16947  8851 13468]
Dimensionality: (34,)

Email vector: [10946 20419  1845 20419 29891 33008 33256 20419 23298 17594 12021  7779
  5370 26756  7232 20419 27443 26756  7232 20419 20419 13468]
Dimensionality: (22,)
```

2.3 Розділення вибірок

```
In [12]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,
                                                              random_state=42)
```

```
In [13]: print("# Train:", len(X_train))
         print("# Test: ", len(X_test))

# Train: 4987
# Test:  555
```

2.4 Навчання наївного Баєсового класифікатора

Підрахуйте сумарну кількість слів у ham- і spam-листах відповідно.

```
In [14]: from operator import itemgetter
         # =============== TODO: Your code here ===============
         # Count the total number of words in ham and spam emails.
         # Store these counts into the two variables defined below.
         ham_indices = np.where(y_train == 0)[0]
         spam_indices = np.where(y_train == 1)[0]
         X_train_ham = itemgetter(*ham_indices)(X_train)
         X_train_spam = itemgetter(*spam_indices)(X_train)
         ham_total_words_train = len(np.hstack(X_train_ham))
         spam_total_words_train = len(np.hstack(X_train_spam))
         # ====================================================
```

Тепер обчисліть апріорні імовірності для класів ham і spam. Зауважте, що добуток імовірностей може переповнити тип даних змінної, тому ми будемо використовувати логарифми.

```
In [15]: # =============== TODO: Your code here ===============
         # Compute the class priors for ham and spam emails.
         ham_log_prior = np.log(np.sum(y_train == 0) / len(y_train))
         spam_log_prior = np.log(np.sum(y_train == 1) / len(y_train))
         # total_words_train = ham_total_words_train + spam_total_words_train
         # ham_log_prior = np.log(ham_total_words_train / total_words_train)
         # spam_log_prior = np.log(spam_total_words_train / total_words_train)
         # ====================================================
```

Обчисліть правдоподібності (likelihood) для кожного слова. Також, застосуйте згладжування Лапласа, щоб уникнути ділення на нуль.

Створимо порожні вектори $\log \phi_{word\,|\,ham}$ та $\log \phi_{word\,|\,spam}$ і заповнимо їх для кожного слова зі словника.

```
In [16]: ham_log_phi = np.zeros(len(vocab), dtype="float64")
         spam_log_phi = np.zeros(len(vocab), dtype="float64")
```

```
In [17]: ham_word_counts = np.zeros(len(vocab))
         spam_word_counts = np.zeros(len(vocab))
```

In [33]:
```
from tqdm import tqdm_notebook as tqdm
from collections import defaultdict
# =============== TODO: Your code here ===============
# Compute log phi(word | class) for each word in the vocabulary.
# Fill out the `ham_log_phi` and `spam_log_phi` arrays below.
V = len(vocab)
alpha = 1e-8

X_train_ham_freq = []
X_train_ham_n = []
for email in X_train_ham:
    ham_freq = defaultdict(int)
    for w in email:
        ham_freq[w] += 1
    X_train_ham_freq.append(dict(ham_freq))
    X_train_ham_n.append(len(email))

X_train_spam_freq = []
X_train_spam_n = []
for email in X_train_spam:
    spam_freq = defaultdict(int)
    for w in email:
        spam_freq[w] += 1
    X_train_spam_freq.append(dict(spam_freq))
    X_train_spam_n.append(len(email))

for ham_freq, n in tqdm(zip(X_train_ham_freq, X_train_ham_n)):
    ham_word_counts[list(ham_freq.keys())] += \
                    np.array(list(ham_freq.values())) / n

for spam_freq, n in tqdm(zip(X_train_spam_freq, X_train_spam_n)):
    spam_word_counts[list(spam_freq.keys())] += \
                    np.array(list(spam_freq.values())) / n

ham_log_phi = np.log((ham_word_counts + alpha) / \
                     (np.sum(y_train == 0) + alpha*V))
spam_log_phi = np.log((spam_word_counts + alpha) / \
                      (np.sum(y_train == 1) + alpha*V))
# ====================================================
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

2.5 Передбачення

Реалізуйте функцію передбачення. Пригадайте, що знаменник $P(words)$ — один і той самий для обох класів, тому для передбачення його можна проігнорувати.

```
In [34]: def predict(X):
             # =============== TODO: Your code here ===============
             # Implement the prediction of target classes, given
             # a feature dataset X. You should return a response
             # vector containing n {0, 1} values, where n is the
             # number of examples in X.
             y_pred = []
             for email in X:
                 log_prob_ham = 0
                 log_prob_spam = 0
                 for w in email:
                     log_prob_ham += ham_log_phi[w]
                     log_prob_spam += spam_log_phi[w]
                 log_prob_ham += ham_log_prior
                 log_prob_spam += spam_log_prior
                 pred = 1 if log_prob_spam > log_prob_ham else 0
                 y_pred.append(pred)
             return y_pred
             # ====================================================
```

2.6 Оцінка точності передбачення

```
In [35]: pred_train = predict(X_train)
         pred_test = predict(X_test)

In [36]: accuracy_train = 1 - np.sum(pred_train != y_train) / len(y_train)
         accuracy_test = 1 - np.sum(pred_test != y_test) / len(y_test)

In [37]: print("Training accuracy:   {0:.3f}%".format(accuracy_train * 100))
         print("Test accuracy:       {0:.3f}%".format(accuracy_test * 100))

Training accuracy:   98.276%
Test accuracy:       97.477%
```