Команда Tensor

Тума Анна, Малышкина Марина, Никитин Кирилл

15 ноября 2024 г.

II ВСЕРОССИЙСКИЙ КВАНТОВЫЙ ХАКАТОН
Проектные задачи

Содержание

1 Математическая оценка

1.1 Задача 1. Формирование инвестиционного портфеля

Количество акций, доступных к покупке: n, количество периодов: m. $\{n_{j,i}\}_j$ – распределение портфеля (число акций вида j за период i), $\omega_{j,i}$ – цена акций вида j за период i, $\sum_j \omega_{j,1} n_{j,1} \leq N$ – условие нормировки, начальная закупка портфеля, $N = 1000000\ USD$.

$P_i = \sum_j \omega_{j,i} n_{j,i}$ – стоимость портфеля в период i,

$r_i = \dfrac{P_{i+1} - P_i}{P_i}$ – доходность портфеля за i период,

$\overline{r} = \dfrac{1}{m} \sum_{i=1}^{m} r_i$ – средняя доходность за весь период,

$\sigma = \sqrt{m \sum_{i=1}^{m} \dfrac{(r_i - \overline{r})^2}{m-1}}$ – уровень риска за весь период,

$\sum_i (P_{i+1} - P_i) = P_{final}$ – целевая функция максимализации.

Граничные условия: $\sigma \leq 0.2$.

Сведение к задаче QUBO:

Из условия задачи следует, что закупка пакета акций происходит разово, т.е. $\forall i\ n_{j,i} = n_{j,1} = n_j$ Целевая функция к минимизации :

$$-\sum_i (P_{i+1} - P_i) = -\sum_{i,j} (\omega_{j,i+1} n_j - \omega_{j,i} n_j) = -\sum_j n_j \sum_i (\omega_{j,i+1} - \omega_{j,i})$$

Представим веса акций как D_n – диагональная матрица с элементами n_j на диагонали, стоимость акций на i итерации как Ω_i размера $1 \times n$. Тогда целевая функция будет иметь вид: $-(\Omega_n - \Omega_1)D_n\overline{x} \to min$, при условии, что

$$\forall j \ \sum_i x_{i,j} \leq 1, x_{i,j} \in \{0,1\}, \ (\Omega_1)D_n\overline{x} \leq N, \ \sqrt{m\sum_{i=1}^{m}\frac{(r_i - \overline{r})^2}{m-1}} \leq 0.2,$$

Квадратичный штраф для системы $\forall j \ \sum_i x_{ij} \leq 1$:

$$\frac{\alpha}{2}\sum_j \sum_{i_1 \neq i_2} x_{j,i_1}x_{j,i_2} = \frac{\alpha}{2}\overline{x}^T(E_m \otimes (I_{n\times n} - E_n))\overline{x}, \alpha > 0$$

Целевая функция с учётом штрафа:

$$\frac{\alpha}{2}\overline{x}^T(E_m \otimes (I_{n\times n} - E_n))\overline{x} - (\Omega_n - \Omega_1)D_n\overline{x} = \frac{\alpha}{2}\overline{x}^T(E_m \otimes (I_{n\times n} - E_n))\overline{x} - \overline{f}^T\overline{x} \to min.$$

при условии, что $(\Omega_1)D_n\overline{x} \leq N$:
положим $\overline{f}^T\overline{x} + z = N, z \in [0, N], pr = [log_2(N+1)], z = L\overline{y}$, y – бинарный вектор-столбец размера pr, L – диагональная матрица из PR_i, $PR = (1, 2, \dots 2^{pr-2}, N - 2^{pr-1} - 1)$,
пусть $u = (\overline{x}, \overline{y})$ – вектор столбец основных и вспомогательных переменных, тогда квадратичный штраф равен:

$$\frac{\beta}{2}||\overline{f}^T\overline{x} + L\overline{y} - N||^2 = \frac{\beta}{2}u^TQ_Nu + \beta v_N^Tu + \frac{\beta}{2}||N||^2, \beta > 0$$

$$Q_N = \begin{pmatrix} D_n^T\Omega^T\Omega D_n & D_n^T\Omega^T \otimes L \\ L^T \otimes \Omega D_n & L^TL \end{pmatrix}, \tag{1}$$

$$v_N = \begin{pmatrix} \Omega D_n \otimes (I_{n\times 1} \otimes N) \\ L^T \otimes (I_{n\times 1} \otimes N) \end{pmatrix}, \tag{2}$$

Постановка задачи в форме QUBO:

$$\frac{1}{2}u^TQu + v^Tu \to min,$$

где

$$Q = \begin{pmatrix} \alpha(E_m \otimes (I_{n\times n} - E_n)) + \beta(D_n^T\Omega^T\Omega D_n) & \beta(D_n^T\Omega^T \otimes L) \\ \beta(L^T \otimes \Omega D_n) & \beta(L^TL) \end{pmatrix}, \tag{3}$$

$$v = \begin{pmatrix} -(\Omega_n - \Omega)D_n - \beta(\Omega D_n \otimes (I_{n\times 1} \otimes N)) \\ \beta(L^T \otimes (I_{n\times 1} \otimes N)) \end{pmatrix}. \tag{4}$$

В случае без ограничения на σ риск становится значительно ниже поставленного в задаче (в среднем на выборке из 15 расчетов с рандомизированными начальными приближениями принимает значение 0.08). Введём поправку на σ:

При условии что $\sqrt{m\sum_{i=1}^{m}\frac{(r_i-\overline{r})^2}{m-1}}=\sigma$:

положим $\sum_{i=1}^{m}(r_i-\overline{r})^2=\frac{(m-1)\sigma^2}{m}$, квадратичный штраф равен:

$$\frac{\gamma}{2}u^T\sum_i(r_i-\overline{r})u,$$

$$Q=\begin{pmatrix}\alpha(E_m\otimes(I_{n\times n}-E_n))+\beta(D_n^T\Omega^T\Omega D_n) & \beta(D_n^T\Omega^T\otimes L)\\ \beta(L^T\otimes\Omega D_n) & \beta(L^TL)\end{pmatrix}+\frac{\gamma}{2}u^T\sum_i(r_i-\overline{r})u \tag{5}$$

$$v=\begin{pmatrix}-(\Omega_n-\Omega_1)D_n-\beta(\Omega_1D_n\otimes(I_{n\times1}\otimes N))\\ \beta(L^T\otimes(I_{n\times1}\otimes N))\end{pmatrix}. \tag{6}$$

Полученную матрицу $QUBO$ можно использовать для решения задачи методом квантового отжига.

1.1.1 Сложность и асимптотика

С помощью описанных преобразований нами был написан код, позволяющий получить матрицу QUBO из исходных данных. Данный алгоритм действий предполагает ограниченность по сложности матричным умножением. В качестве функции матричного умножения на практике использовались методы numpy (ex. numpy.mulmul ($O(N^3)$)).

Итерация	1	2	...	15
Время (real), с	1,974	1.878	...	1.742

Таблица 1: Caption

В дальнейшем матрица QUBO использовалась для решения задачи оптимизации методом квантового отжига с помощью эмулятора QiOpt.
Код для данной задачи

```
import pandas as pd
import numpy as np
import math
```

```
import numba as nb
import random
import csv
import random
import statistics
import sys

alpha = float(sys.argv[1]) #fee cjefficient 1
beta = float(sys.argv[2]) #fee coefficient 2
gamma=float(sys.argv[3])

@nb.njit(parallel=True)
def matrix_mult(A, B):
     m, n = A.shape
     _, p = B.shape
     C = np.zeros((m, p))
     for j in range(n):
       for k in range(p):
         for i in range(m):
           C[i, k] += A[i, j] * B[j, k]
     return C

table=pd.read_csv('data.csv')
full=table.to_numpy()
omega_1=full[0]
omega_n=full[99]
omega=[]
for i in range (100):
  omega.append(omega_n[i] - omega_1[i])

omega=np.array(omega).reshape(-1,1)
Dm=np.zeros((100,100))
L=np.ones((1,19))


for i in range(0,100):
    Dm[i][i]=bool(random.getrandbits(1))
    if i<int(math.log(1e6,2)):
       L[0][i]=2**(int(math.log(1e6,2)) - (int(math.log(1e6,2)-i) ))

left_element=beta*Dm.dot(omega.dot(omega.transpose().dot(Dm)))
```

```
ExI=alpha*np.ones(left_element.shape)
np.fill_diagonal(ExI, 0)

np.add(ExI, left_element)
#dummy.reshape(-1,1)
right_element=beta*matrix_mult(Dm, omega).dot(L)

lower_right_element=beta*matrix_mult(L.transpose(), L)
final = np.hstack((left_element, right_element))
final_2=np.hstack((right_element.transpose(), lower_right_element))
final=np.vstack((final,final_2))
dumm=list(final)
matrix=[]
for i in range(final.shape[0]):
  for j in range(final.shape[1]):
    if (final[i,j]!=0):
      arr_f=([int(i+1), int(j+1), final[i,j]])
      matrix.append(arr_f)

matrix=np.array(matrix)

with open("temp.csv", "w") as f:
  wr=csv.writer(f, delimiter = " ")
  wr.writerows(matrix)
```

Код рассчитывает решение задачи инвестиций (таск 1), применяя штрафные коэффициенты alpha, beta, gamma. Если нужно их поменять – это делается путем редактирования кода (переменные alpha, beta, gamma соотв.)

Формат запуска: python task1.py alpha beta gamma Напр. python task1.py 0.8 0.8 0.5

На выходе получается матрица QUBO, в которой остается только написать шапку формата "размерность матрицы"(исходные параметры + вспомогательные) + "кол-во ненулевых элементов матрицы"(строк на выходе) и убрать .0 у индексов (превратить их в нормальные целочисленные значения). После преобразований матрицу можно отправлять решать методом solve (qiopt). Результат работы кода:

```
{"Solution": [0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0,
```

```
1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
"Objective": -120464},
```


Риск: $\sigma = 0.18136285$. Доходность: 4.5%.

1.2 Задача 2. Оптимизация туристических маршрутов

Введем бинарные переменные: $i \in \{1, n+1\}$, где $n+1$ и 1 – индекс вокзала, $(i,j) \in E$ – множество связей между узлами, v – автобусы, $x_{v,p,i}$ – остановки у достопримечательностей ($x_{vpi} = 1$, если автобус v должен высадить туристическую группу у достопримечательности i с позицией p или это перекресток,т.е. во всех точках маршрута автобуса v, в остальных случаях $x_{vpi} = 0$). Стоимость перехода (i,j): $c_{ij} > 0$, проходимая последовательность узлов: $P \leq S$, P – длина пути, S – максимальная возможное время пути.
Целевая функция минимизации для m автобусов:

$$\sum_{v=1}^{m} \sum_{p=1}^{P-1} \sum_{(i,j)\in E} c_{ij} x_{v,p,i} x_{v,p+1,j} \to min \tag{7}$$

Ограничения:

1. M – множество всех автобусов ($|M| = m = 15$ штук), вместимость одного автобуса $N = 10$ человек.

2. максимальное время пути $S = 15$ тактов, прохождение 1 ребра = 1 такт, 1 остановка в узле = 1 такт.

3. группы людей $G = \{2, 3, 5, 9\}$, $G_{min} = 2$ – минимальное количество людей в одной группе.

4. 1 группа – 1 достопримечательность, иными словами, каждой группе соответствует своя достопримечательность.

5. в одном автобусе может находиться несколько групп.

6. 2 автобуса не могут одновременно находиться в одном узле.

7. более, чем 1 автобус, может находиться только на вокзале.

8. автобусы могут ехать только по перекресткам и в достопримечательности групп, которые должны посетить группы, находящиеся в автобусе.

9. автобусы с вокзала выезжают одновременно.

Задачи к решению: распределение людей по автобусам + пути автобусов. Математические ограничения:

1. последовательные переходы автобусов возможны только между соединенными узлами:

$$\sum_{v=1}^{m}\sum_{p=1}^{P-1}\sum_{(i,j)\notin E} x_{v,p,i}x_{v,p+1,j} = 0$$

2. каждый автобус после прибытия на вокзал остается там:

$$\sum_{v=1}^{m}\sum_{p=2}^{P-1}\sum_{j=1}^{n} x_{v,p,n+1}x_{v,p+1,j} = 0$$

3. последовательности узлов начинаются и заканчиваются на вокзале:

$$\sum_{v=1}^{m}(1 - x_{v,1,n+1}x_{v,P,n+1}) = 0$$

4. за один такт в одном узле может стоять не более одного автобуса, за исключением вокзала:

$$\sum_{v=1}^{m} x_{v,p,i} \leq 1, i \in \overline{2,n}$$

5. автобус может посетить только и только достопримечатетельности своих групп ровно 1 раз: $\forall v\ \exists!\ (g_v \in \mathbb{N})\&(g_v \leq \left[\frac{N}{G_{min}}\right])$, g_v – количество групп в v автобусе. Разобьем все узлы на 2 группы: перекрестки (l) и достопримечательности (h), их множества L и H соответственно, $|L| + |H| = n$, $|H| = 35, |L| = 22$.

$$\sum_{h\in H} x_{v,p,h} = g_v$$

6. Распределение $\{x_{v,p,h} = 1\}_{h \in H} = h_v$ порождает множество достопримечательностей $(h_v, |h_v| = g_v)$, которые должен посетить автобус v. Тогда $\forall((v,h) \in (M \otimes h_v) : x_{v,p,h} = 1) \& ((v,h) \notin (M \otimes h_v) : x_{v,p,h} = 0)$.

$$\sum_{(v,h) \in (M \otimes h_v)} x_{v,p,h} = |H|$$

Постановка задачи QUBO:
$\overline{x}^T (C \otimes D \otimes E_m)\overline{x} \to min$,
D – матрица размера $P \times P$, $D_{kr} = 1, |k-r| = 1, D_{kr} = 0, |k-r| \neq 1$, C – матрица размера $n \times n$

Квадратичные штрафы:

1.

$$\frac{\alpha_1}{2} \sum_{v=1}^{m} \sum_{p=1}^{P-1} \sum_{(i,j) \in E} x_{v,p,i} x_{v,p+1,j} = \frac{\alpha_1}{2} \overline{x}^T (\overline{A} \otimes D \otimes E_m)\overline{x}, \alpha_1 > 0, \quad (8)$$

$\overline{A}$ – логическое отрицание матрицы взаимодействия между узлами $(0 \to 1, 1 \to 0)$.

2.

$$\frac{\alpha_2}{2} \sum_{v=1}^{m} \sum_{p=2}^{P-1} \sum_{j=1}^{n} x_{v,p,n+1} x_{v,p+1,j} = \frac{\alpha_2}{2} \overline{x}^T (B \otimes E_m)\overline{x}, \alpha_2 > 0, \quad (9)$$

B – матрица $(n+1)P \times (n+1)P$ – в пространстве выбора путей из возможных подграфов.

3.

$$\frac{\alpha_3}{2} \sum_{v=1}^{m} (1 - x_{v,1,n+1} x_{v,P,n+1}) = \frac{\alpha_3}{2} \overline{x}^T (E''_{n+1} \otimes H \otimes E_m)\overline{x} + \frac{\alpha_3}{2} m, \quad (10)$$

элементы $H(1,P)$ и $H(P,1)$ матрицы H размера $P \times P$ равны 0.5, остальные элементы равны 0;
E''_{n+1} – матрица размера $(n+1) \times (n+1)$, $E''_{n+1}(n+1, n+1) = 1, \forall i \neq (n+1) E''_{n+1}(i,i) = 0, \alpha_3 > 0$.

4. при условии, что $\sum_{v=1}^{m} x_{v,p,i} \leq 1, i \in \overline{2,n}$:

положим $\sum_{v=1}^{m} x_{v,p,i} + z = E_{(n+1)P}, z \in [0,1], pr_i = [log_2(z_i + 1)], z = L\overline{y}$,

$pr = \sum_i pr_i$, y – бинарный вектор-столбец размера pr, L – диагональная матрица из PR_i, $PR = (1, 2, \dots 2^{pr-2}, N - 2^{pr-1} - 1)$,
пусть $u = (\overline{x}, \overline{y})$ – вектор столбец основных и вспомогательных переменных, тогда квадратичный штраф равен:

$$\frac{\alpha_4}{2}||\sum_{v=1}^{m} x_{v,p,i} + L\overline{y} - E_{(n+1)P}||^2 = \frac{\alpha_5}{2}u^T Q_4 u + \alpha v_4^T u + \frac{\alpha}{2}||N||^2, \alpha_4 > 0, \tag{11}$$

$(E_{n+1} \otimes E_P \otimes I_{m\times 1})\overline{x} = I_{(n+1)P}$

$$Q_4 = \begin{pmatrix} E_{n+1}^T \otimes E_P^T \otimes I_{m\times 1}^T E_{n+1} \otimes E_P \otimes I_{m\times 1} & E_{n+1}^T \otimes E_P^T \otimes I_{m\times 1}^T \otimes L \\ L^T \otimes E_{n+1} \otimes E_P \otimes I_{m\times 1} & L^T L \end{pmatrix}, \tag{12}$$

$$v_4 = \begin{pmatrix} E_{n+1} \otimes E_P \otimes I_{m\times 1} \otimes I_{(n+1)P} \\ L^T \otimes I_{(n+1)P}) \end{pmatrix}, \tag{13}$$

5.

$$\frac{\alpha_5}{2}x^T I_{mP\times mP}x - \alpha_5 g_v I_{mP\times 1}x + \frac{\alpha_5 g_v^2}{2}, \alpha_5 > 0 \tag{14}$$

6.

$$\frac{\alpha_6}{2}x^T I_{P\times P}x - \alpha_6|H|I_{P\times 1}x + \frac{\alpha_6|H|^2}{2}, \alpha_5 > 0 \tag{15}$$

Собрав все штрафные коэффициенты получаем:

$$F = \overline{x}^T C \otimes D \otimes E_m \overline{x} + \frac{\alpha_1}{2}\overline{x}^T(\overline{A} \otimes D \otimes E_m)\overline{x} + \frac{\alpha_2}{2}\overline{x}^T(B \otimes E_m)\overline{x} + \\ + \frac{\alpha_3}{2}\overline{x}^T(E''_{n+1} \otimes H \otimes E_m)\overline{x} + \frac{\alpha_3}{2}m + \frac{\alpha_5}{2}x^T I_{mP\times mP}x - \alpha_5 g_v I_{mP\times 1}x + \frac{\alpha_5 g_v^2}{2} + \\ + \frac{\alpha_6}{2}x^T I_{P\times P}x - \alpha_6|H|I_{P\times 1}x + \frac{\alpha_6|H|^2}{2} \to min \tag{16}$$

$$Q' = \frac{1}{2}(C \otimes D \otimes E_m + \alpha_1(\overline{A} \otimes D \otimes E_m) + \alpha_2(B \otimes E_m) + \\ + \alpha_3(E''_{n+1} \otimes H \otimes E_m) + \alpha_5 I_{mP\times mP} + \alpha_6 I_{P\times P}) \tag{17}$$

$$Q = \begin{pmatrix} E_{n+1}^T \otimes E_P^T \otimes I_{m\times 1}^T E_{n+1} \otimes E_P \otimes I_{m\times 1} + Q' & E_{n+1}^T \otimes E_P^T \otimes I_{m\times 1}^T \otimes L \\ L^T \otimes E_{n+1} \otimes E_P \otimes I_{m\times 1} & L^T L \end{pmatrix}, \tag{18}$$

$$v' = -\alpha_5 g_v I_{mP\times 1} - \alpha_6 |H| I_{P\times 1} \tag{19}$$

$$v = \begin{pmatrix} E_{n+1} \otimes E_P \otimes I_{m\times 1} \otimes I_{(n+1)P} + v' \\ L^T \otimes I_{(n+1)P}) \end{pmatrix}. \tag{20}$$

Постановка задачи QUBO выглядит следующим образом:

$$\frac{1}{2} u^T Q u + v^T u \to min.$$

1.2.1 Оценка асимптотики задачи QUBO в рамках решения задачи оптимизации маршрутов

Зависимость количества элементов матрицы Q от количества автобусов имеет асимптотику $O(m^2)$, зависимость от продолжительности пути имеет асимптотику $O(P^2)$, зависимость количества узлов имеет асимптотику $O(n^2)$. Перемножение двух матриц $k \times k$ имеет ассимптотику $O(k^3)$ или, если делать по оптимальному алгоритму, то имеет примерно $O(k^{2.373})$, что приводит к тому, что масштабизирование проблемы даёт всплекс по сложности вычислений из-за минимизации в том числе по параметрам, которые уже присутствуют в самой матрице Q, т.е. асимптотика, даже используя оптимальный алгоритм перемножения матриц, имеет асимптотику $O(k^{4.373})$. Что касается решаемой проектной задачи, её разрешимость за адекватное конечное время обуславливается очень редкой сеткой связей между узлами и списком запретов на те или иные виды перемещения.

1.2.2 Аналогия с алгоритмом SCF

В квантовой химии алгоритм SCF (Self-Consistent-Field) используется для нахождения минимизирующего функционал энергии $\frac{\langle\psi|\hat{H}|psi\rangle}{\langle\psi|\psi\rangle}$ путем варьирования коэффициентов орбиталей, т.е. коэффициентов в линейной комбинации, последовательными итерациями. Поэтому для данной задачи мы решили применить похожую методику: выбираем начальное приближение в виде распределения групп по автобусам – $\sum_i x_{v,p,i}$ (суммирование по всем достопримечательностям).

Решаем задачу $QUBO$ для такого начального распределения, т.е. находим E_{min} в терминах Хартри-Фока и минимальную стоимость в терминах нашей задачи. При этом пока ограничения не учитываем. То есть, примерный вид гамильтониана($J = 0$):

$$H = \sum_i \sum_p Path_{ip}$$

Добавляем в систему ограничение на пути автобусов. На основании этого ограничения формируем новый вид матрицы QUBO, которая будет зависеть от распределения людей. Оставляя стоимость такой же, подбираем распределение людей, которое будет соответствовать такой энергии. Добавляем следующее ограничение. Вновь для распределения из предыдущего пункта ищем минимум стоимости при данном ограничении. Повторяем процесс до последнего ограничения включительно.

1.3 Задача 3. Семантический анализ отзывов о продукте

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from qiskit.circuit.library import ZZFeatureMap, TwoLocal
from qiskit_machine_learning.algorithms import VQC
from qiskit.primitives import StatevectorSampler
from qiskit_machine_learning.optimizers import COBYLA
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report, log_loss

# Загрузка и подготовка данных
file_path = 'task-3-dataset.csv'
data = pd.read_csv(file_path)

# Преобразование меток в бинарные значения
# (1 для положительных, 0 для отрицательных)
data['разметка'] = data['разметка'].apply(lambda x: 1 if x == '+' else 0)

# Выделение признаков и меток
reviews = data['отзывы'].values
labels = data['разметка'].values

# Векторизация отзывов
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(reviews).toarray()
```

```
y = np.array(labels)

# Разделение данных на тренировочные и тестовые
X_train, X_test, y_train, y_test = train_test_split(X,
        y, test_size=0.2, random_state=42)

# Настройка квантовой модели
feature_dim = X_train.shape[1]
feature_map = ZZFeatureMap(feature_dimension=feature_dim)
ansatz = TwoLocal(feature_dim, ['ry', 'rz'], 'cz', reps=3)
sampler = StatevectorSampler()

# Кастомный оптимизатор для отслеживания потерь
class LossTrackingCOBYLA(COBYLA):
    def __init__(self, maxiter=100):
        super().__init__(maxiter=maxiter)
        self.loss_history = []

    def step(self, loss):
        self.loss_history.append(loss)
        return super().step(loss)

# Используем кастомный оптимизатор
optimizer = LossTrackingCOBYLA(maxiter=20)
# Количество итераций, которое хотим контролировать

# Инициализация VQC с использованием кастомного оптимизатора
vqc = VQC(feature_map=feature_map,
        ansatz=ansatz, optimizer=optimizer, sampler=sampler)

# Обучение модели
vqc.fit(X_train, y_train)

# Кривая потерь на тренировочных данных
train_losses = optimizer.loss_history

# Оценка модели на тестовых данных
y_test_pred = vqc.predict(X_test)
print(classification_report(y_test, y_test_pred))

# Визуализация кривой функции потерь
```

```
plt.figure(figsize=(10, 5))
plt.plot(range(1, len(train_losses) + 1),
        train_losses, marker='o', label='Training Loss')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.title('Training Loss Curve')
plt.legend()
plt.grid(True)
plt.show()
```

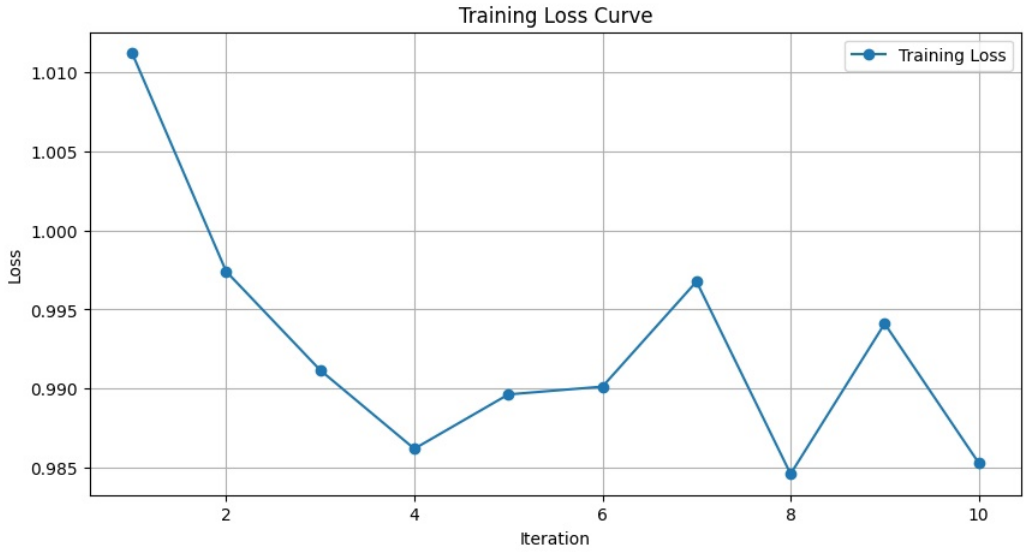

Рис. 1: Кривая обучения

Версия 2

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from qiskit.circuit.library import ZZFeatureMap, TwoLocal
from qiskit_machine_learning.algorithms import VQC
from qiskit.primitives import StatevectorSampler
from qiskit_machine_learning.optimizers import SPSA
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report
```

```
# Загрузка данных
file_path = 'task-3-dataset.csv'
data = pd.read_csv(file_path)
data['разметка'] = data['разметка'].apply(lambda x: 1 if x == '+' else 0)

# Преобразование данных
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(data['отзывы']).toarray()
y = np.array(data['разметка'])

# Снижение размерности с использованием PCA
pca = PCA(n_components=3)  # Уменьшили размерность до 3 признаков
X_reduced = pca.fit_transform(X)

# Разделение данных
X_train, X_test, y_train, y_test = train_test_split(X_reduced,
y, test_size=0.2, random_state=42)

# Настройка квантовой модели с уменьшенной размерностью
feature_dim = X_train.shape[1]
feature_map = ZZFeatureMap(feature_dimension=feature_dim)
ansatz = TwoLocal(feature_dim, ['ry', 'rz'], 'cz', reps=5)
# Увеличили сложность анзаца
sampler = StatevectorSampler()

# Массив для хранения потерь
train_losses = []

# Callback функция для отслеживания потерь
def callback(objective_weights, objective_value):
    train_losses.append(objective_value)
    print(f"Loss = {objective_value}")

# Настройка оптимизатора
optimizer = SPSA(maxiter=100)  # Используем SPSA и увеличили maxiter

# Инициализация VQC с использованием callback
vqc = VQC(feature_map=feature_map,
ansatz=ansatz, optimizer=optimizer, sampler=sampler, callback=callback)
```

```
# Обучение модели
vqc.fit(X_train, y_train)

# Оценка модели на тестовых данных
y_test_pred = vqc.predict(X_test)
print(classification_report(y_test, y_test_pred))

# Визуализация кривой функции потерь
plt.figure(figsize=(10, 5))
plt.plot(range(1, len(train_losses) + 1), train_losses, marker='o',
label='Training Loss')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.title('Training Loss Curve')
plt.legend()
plt.grid(True)
plt.show()
```

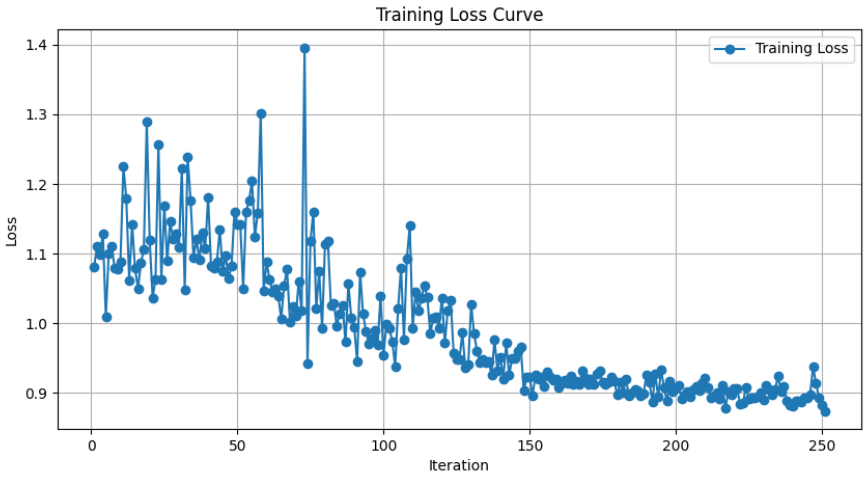

Рис. 2: Кривая обучения