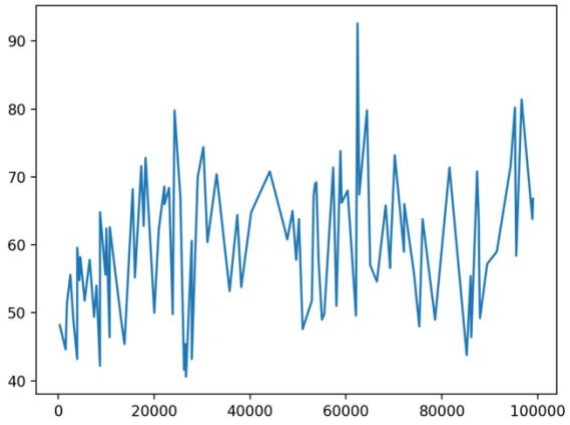
CS536 HW3: Perceptrons and SVMs

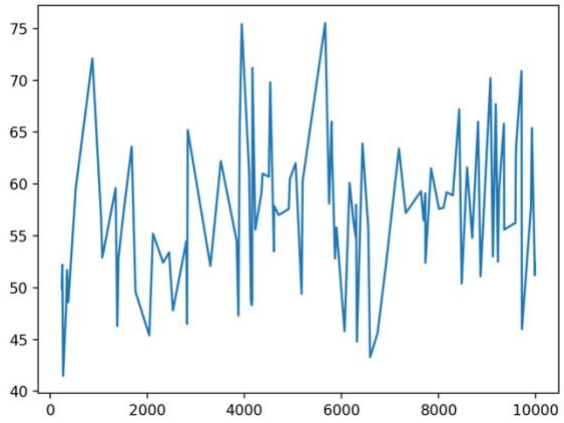
Student Name: Xuenan Wang
NetID: xw336

1. Perceptrons

1) For k = 5 and ε = 0.1, I generate 100 evenly distributed numbers range from (10, 100000) as value of m. For each combination of k, ε and m, I generate a training dataset and fit a perceptron model to it, count the steps it needed to converge. Repeat this for 10 times and get the average outcome. Plot for this result is as follow.

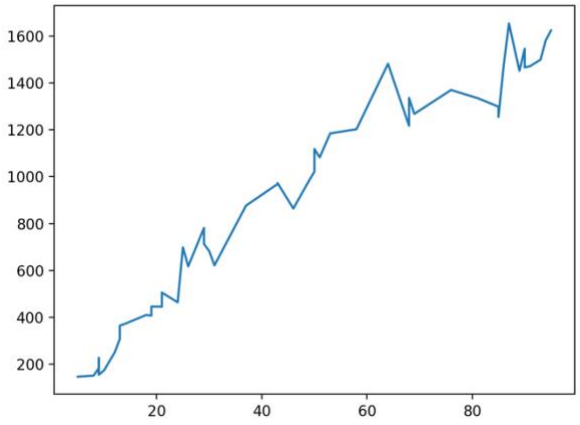


We can see from this figure that the convergence steps and m is not clearly related. Although we do can find a roughly positive relationship between these two parameters. I tried to repeat this experiment. For this time, I want to further eliminate the influence from extreme values, so I increased the repeated time for each parameter combinations to 20 times. However, based on consideration of getting result in reasonable time, I pick m from (10, 10000) this time. Result is as following.



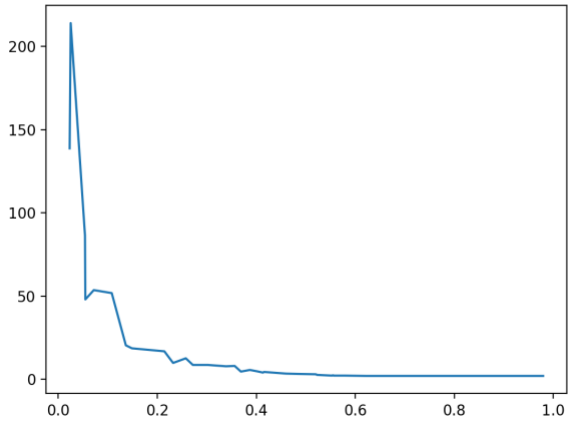
The result is basically same as the previous one.

2) For m = 100 and ε = 0.05, I generate 50 evenly distributed numbers range from (5, 100) as value of k. Similar as previous one, I repeated each parameter combination for 10 times and calculate the average value. The result is as follow.

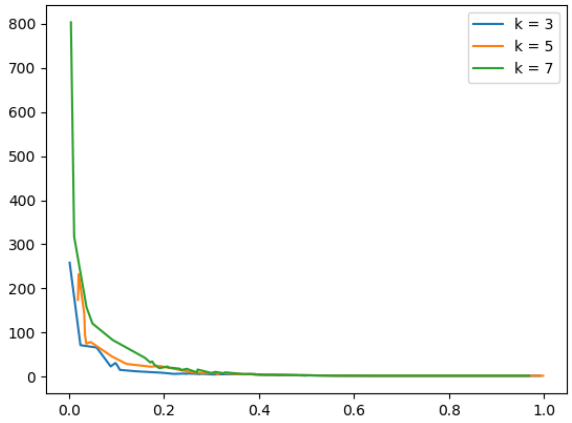


We can see that with k increase, convergence steps increase. Although there are shaking process, but the trend is pretty clear. We can understand this as, with the increasing number of dimensions, it's going to be harder to find a plane to separate the data. However, despite this doesn't show in my plot, I assume the convergence steps may decrease after certain k dimensions. Since the data size is stable, so with the increment of dimensions, the data is going to be sparser and, in this case, the convergence steps is going to decrease. Again, it's just a guess. My experiment doesn't show valid this assumption.

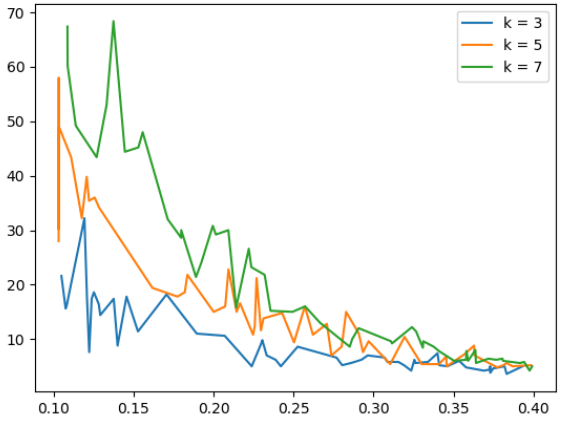
3) For k = k and m = 100, I generate 50 evenly distributed numbers range from (0, 1) as value of ε. Similar as previous one, I repeated each parameter combination for 10 times and calculate the average value. The result is as follow.



We can see that with ε increase, convergence steps decrease dramatically. At about ε = 0.5, the convergence steps becomes super small.
For different k, I set K as 3, 5 and 7, repeat the previous process. The result is as follow.



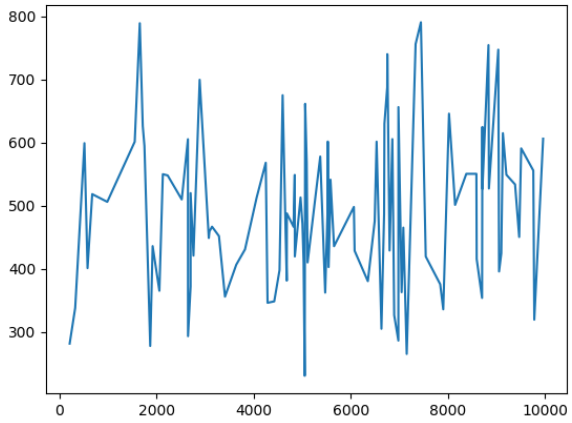
We can clearly see that for a bigger k, it's going to take more steps to converge. However, also at around ε = 0.5, the convergence step become really small.
To clearly see the process, I repeated this process, but for this time, I only focus on ε in range of (0.1, 0.4). The result is as follow.



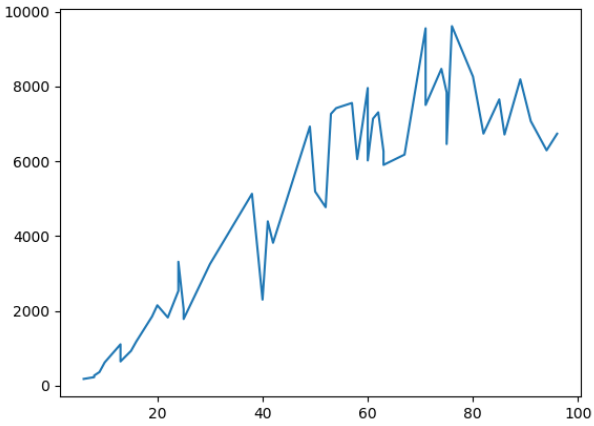
This time, it's more clear that with higher k, for same ε, the convergence steps are higher.

Bonus:

For different m: I pick k = 5, ε = 0.05 and 100 evenly distributed m in range of (10, 10000). Repeat each combination for 10 times. Result is as follow.

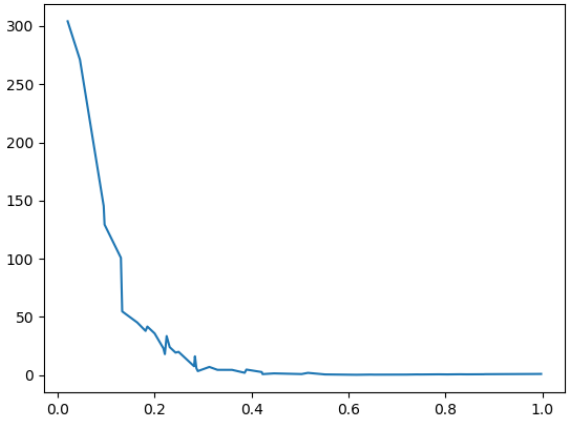


Same as problem 1), there are no clear relationship between the indicated parameter and m values.
For different k: I pick m = 100, ε = 0.1 and 50 evenly distributed k in range of (5, 100). Repeat each combination for 10 times. Result is as follow.



The indicated parameter increase as k increase. However, after about k = 80, the increasing trend is slowed. I don't have further data to indicate what's really going on at this point, but the basically idea is sound.

For different ε: I pick k = 5, k = 100 and 50 evenly distributed ε in range of (0, 1). Repeat each combination for 10 times. Result is as follow.



Similar as problem 3), the indicate parameter decrease as ε increase. At about k = 0.4, the indicated parameter become really small.

2. SVMs

1) As described for generating the data, it's obvious that there exists a separator as a circle. We think this circle is locate at position (a, b) and with radius of r. We can represent this circle as:

$$(x-a)^2+(y-b)^2=r^2$$

And we can further represent this circle as:

$$x^2+y^2-2ax-2bx=r^2-a^2-b^2$$

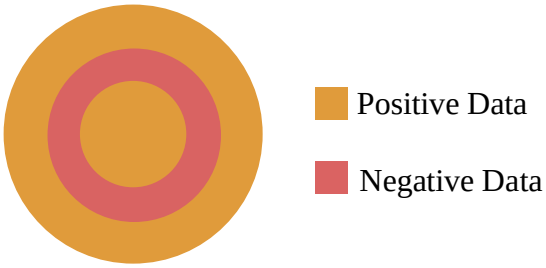
It's clear that we can represent this separator using $x, y, x^2 and\ y^2$. We notice that the radius and location of this circle doesn't matter here.

Ellipsoidal separator is a 3-dimentional separator and quadratic features are in a 2-dimentional space. Since in 3-dimentional space, a 2-dimentional space is just a plane, we can always separate a 2-dimentional data with higher dimensional separators.

2) To find a linear separator in 3-dimentional space, we need to map the data into a 4-dimentional space, where all 3-dimentional data are all just in a plane. Therefore, to find a ellipsoidal separator, we can use a 4-dimentional kernel, and in this case:

$$K(x,y)=(1+xy)^4$$

3) The suggested data is like this:



Without the outer space for positive data, we can easily find a non-linear separator in 2-dimentional space, and a linear separator in 3-dimentional space. However, in this case, it's like XOR data in 2-dimentional space. To find a linear separator, we need to search in higher dimensional space and for this case, a 4-dimentional space. The kernel should be like this:

$$K(x,y)=(1+xy)^4$$

4) The Dual SVM model is as following:

$$\max_{\alpha}\sum_{i=1}^{m}\alpha_i-\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i y^i(x^i y^j)y^j\alpha_j \quad (s.t.)\sum_{i=1}^{m}\alpha_i y^i=0 \quad \forall i:\alpha_i\geq 0.$$

We have two kernels, for the first one, it's a polynomial kernel:

$$K(x,y)=(1+xy)^2$$

For this kernel, we can change the model as:

$$f(x) = sign(\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i y^i y^j \alpha_j (1 + xx^i)^2 (1 + xy^j)^2 + b)$$

The second kernel is Gaussian kernel:

$$K(x, y) = \exp(-\| x - y \|^2)$$

For this kernel, we can change the model as:

$$f(x) = sign(\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i y^i y^j \alpha_j \exp(-\| x - x^i \|^2) \exp(-\| x - y^j \|^2) + b)$$

Source Code (Python 3)

```
import numpy as np
import matplotlib
matplotlib.use('Agg')
from matplotlib import pyplot as plt

class Data:

    def __init__(self, k, m, epsilon):
        self.k = k
        self.m = m
        self.epsilon = epsilon
        self.data = []
        self.Y = []
        self.generateData()

    def generateDataPoint(self):
        rawData = np.random.normal(0, 1, self.k)
        secondNorm = np.linalg.norm(rawData, 2)
        rst = []
        for data in rawData:
            rst.append(data/secondNorm)
        return rst

    def generateData(self):
        while len(self.data) < self.m:
            tmp = self.generateDataPoint()
            if abs(tmp[-1]) >= self.epsilon:
                self.data.append(tmp)
                if tmp[-1] > 0:
                    self.Y.append(1)
                else:
                    self.Y.append(-1)
```

```
class Perceptron:


    def __init__(self):
        self.w = 0
        self.b = 0
        self.trainingStep = 0

    def perceptronLearningAlg(self, data, Y):
        self.w = [0.000000001 for _ in range(len(data[0]))]
        self.b = 0.000000001
        while True:
            # print('Training Perceptron, step No.', self.trainingStep)
            misData, misY = self.identifyMisclassified(data, Y)
            if not misData:
                # print('Perceptron trained successfully!')
                return
            self.trainingStep += 1
            # deltaW = sum([d * misY for d in misData])
            # for i in range(len(self.w)):
            #     self.w[i] += deltaW
            for i in range(len(self.w)):
                self.w[i] += misY * misData[i]
            self.b += misY

    def identifyMisclassified(self, data, Y):
        for i in range(len(data)):
            if sum([self.w[j] * data[i][j] + self.b for j in range(len(data[i]))]) * Y[i] < 0:
            # if data[i] * Y[i] < 0:
                return data[i], Y[i]
        return None, None


if __name__ == "__main__":
    # # Problem 1.1
    # k = 5
```

```
# mUniform = np.random.uniform(10, 100000, 100)
# epsilon = 0.1
# repeatTimes = 10
# trainSteps = []
# mList = []
# for m in sorted(mUniform):
#     print('training with m =', int(m))
#     tmp = 0
#     for _ in range(repeatTimes):
#         trainData = Data(k, int(m), epsilon)
#         # print(trainData.data)
#         # print(trainData.Y)
#         model = Perceptron()
#         model.perceptronLearningAlg(trainData.data, trainData.Y)
#         tmp += model.trainingStep
#     mList.append(int(m))
#     trainSteps.append(tmp/repeatTimes)
# plt.plot(mList, trainSteps)
# plt.show()

# # Problem 1.2
# m = 100
# kUniform = np.random.uniform(5, 100, 50)
# epsilon = 0.05
# repeatTimes = 10
# trainSteps = []
# kList = []
# calculated = set()
# for k in sorted(kUniform):
#     if int(k) in calculated:
#         continue
#     calculated.add(int(k))
#     print('training with k =', int(k))
#     tmp = 0
#     for _ in range(repeatTimes):
#         trainData = Data(int(k), m, epsilon)
```

```
#         model = Perceptron()
#         model.perceptronLearningAlg(trainData.data, trainData.Y)
#         tmp += model.trainingStep
#     kList.append(int(k))
#     trainSteps.append(tmp/repeatTimes)
# plt.plot(kList, trainSteps)
# plt.show()

# # Problem 1.3
# k = 3
# m = 100
# epsilonUniform = np.random.uniform(0, 1, 50)
# repeatTimes = 10
# trainSteps = []
# epsilonList = []
# calculated = set()
# for epsilon in sorted(epsilonUniform):
#     if epsilon in calculated:
#         continue
#     calculated.add(epsilon)
#     print('training with epsilon =', epsilon)
#     tmp = 0
#     for _ in range(repeatTimes):
#         trainData = Data(k, m, epsilon)
#         model = Perceptron()
#         model.perceptronLearningAlg(trainData.data, trainData.Y)
#         tmp += model.trainingStep
#     epsilonList.append(epsilon)
#     trainSteps.append(tmp/repeatTimes)
# plt.plot(epsilonList, trainSteps, label='k = 3')
# k = 5
# m = 100
# epsilonUniform = np.random.uniform(0, 1, 50)
# repeatTimes = 10
# trainSteps = []
# epsilonList = []
```

```
# calculated = set()
# for epsilon in sorted(epsilonUniform):
#     if epsilon in calculated:
#         continue
#     calculated.add(epsilon)
#     print('training with epsilon =', epsilon)
#     tmp = 0
#     for _ in range(repeatTimes):
#         trainData = Data(k, m, epsilon)
#         model = Perceptron()
#         model.perceptronLearningAlg(trainData.data, trainData.Y)
#         tmp += model.trainingStep
#     epsilonList.append(epsilon)
#     trainSteps.append(tmp/repeatTimes)
# plt.plot(epsilonList, trainSteps, label='k = 5')
# k = 7
# m = 100
# epsilonUniform = np.random.uniform(0, 1, 50)
# repeatTimes = 10
# trainSteps = []
# epsilonList = []
# calculated = set()
# for epsilon in sorted(epsilonUniform):
#     if epsilon in calculated:
#         continue
#     calculated.add(epsilon)
#     print('training with epsilon =', epsilon)
#     tmp = 0
#     for _ in range(repeatTimes):
#         trainData = Data(k, m, epsilon)
#         model = Perceptron()
#         model.perceptronLearningAlg(trainData.data, trainData.Y)
#         tmp += model.trainingStep
#     epsilonList.append(epsilon)
#     trainSteps.append(tmp/repeatTimes)
# plt.plot(epsilonList, trainSteps, label='k = 7')
```

```
# plt.legend()
# plt.savefig('./problem1-3.png')


# # Problem 1.3 - 2
# k = 3
# m = 100
# epsilonUniform = np.random.uniform(0.1, 0.4, 50)
# repeatTimes = 10
# trainSteps = []
# epsilonList = []
# calculated = set()
# for epsilon in sorted(epsilonUniform):
#     if epsilon in calculated:
#         continue
#     calculated.add(epsilon)
#     print('training with epsilon =', epsilon)
#     tmp = 0
#     for _ in range(repeatTimes):
#         trainData = Data(k, m, epsilon)
#         model = Perceptron()
#         model.perceptronLearningAlg(trainData.data, trainData.Y)
#         tmp += model.trainingStep
#     epsilonList.append(epsilon)
#     trainSteps.append(tmp/repeatTimes)
# plt.plot(epsilonList, trainSteps, label='k = 3')
# k = 5
# m = 100
# epsilonUniform = np.random.uniform(0.1, 0.4, 50)
# repeatTimes = 10
# trainSteps = []
# epsilonList = []
# calculated = set()
# for epsilon in sorted(epsilonUniform):
#     if epsilon in calculated:
#         continue
#     calculated.add(epsilon)
```

```
#     print('training with epsilon =', epsilon)
#     tmp = 0
#     for _ in range(repeatTimes):
#         trainData = Data(k, m, epsilon)
#         model = Perceptron()
#         model.perceptronLearningAlg(trainData.data, trainData.Y)
#         tmp += model.trainingStep
#     epsilonList.append(epsilon)
#     trainSteps.append(tmp/repeatTimes)
# plt.plot(epsilonList, trainSteps, label='k = 5')
# k = 7
# m = 100
# epsilonUniform = np.random.uniform(0.1, 0.4, 50)
# repeatTimes = 10
# trainSteps = []
# epsilonList = []
# calculated = set()
# for epsilon in sorted(epsilonUniform):
#     if epsilon in calculated:
#         continue
#     calculated.add(epsilon)
#     print('training with epsilon =', epsilon)
#     tmp = 0
#     for _ in range(repeatTimes):
#         trainData = Data(k, m, epsilon)
#         model = Perceptron()
#         model.perceptronLearningAlg(trainData.data, trainData.Y)
#         tmp += model.trainingStep
#     epsilonList.append(epsilon)
#     trainSteps.append(tmp/repeatTimes)
# plt.plot(epsilonList, trainSteps, label='k = 7')
# plt.legend()
# plt.savefig('./problem1-3(0.1-0.4).png')

# # Problem 1 Bonus
# k = 5
```

```
# mUniform = np.random.uniform(10, 10000, 100) #10,10000,100
# epsilon = 0.05
# repeatTimes = 10
# idealW = [0 for _ in range(k-1)] + [1]
# idealB = 0
# mList = []
# bias = []
# for m in sorted(mUniform):
#     print('training with m =', int(m))
#     totalW = [0 for _ in range(k)]
#     totalB = 0
#     for _ in range(repeatTimes):
#         trainData = Data(k, int(m), epsilon)
#         model = Perceptron()
#         model.perceptronLearningAlg(trainData.data, trainData.Y)
#         for i in range(k):
#             totalW[i] += model.w[i]
#         totalB += model.b
#     mList.append(int(m))
#     w = []
#     for i in range(k):
#         w.append(totalW[i] / repeatTimes)
#     b = totalB / repeatTimes
#     deltaW = [idealW[i] - w[i] for i in range(k)]
#     deltaB = idealB - b
#     bias.append(np.linalg.norm(deltaW, 2)**2 + deltaB**2)
# print('bias:', bias)
# plt.plot(mList, bias)
# plt.savefig('./bonusM.png')
# plt.clf()

# m = 100
# kUniform = np.random.uniform(5, 100, 50) #5,100,50
# epsilon = 0.1
# repeatTimes = 10
# idealB = 0
```

```
# kList = []
# bias = []
# for k in sorted(kUniform):
#     k = int(k)
#     idealW = [0 for _ in range(k-1)] + [1]
#     print('training with k =', int(k))
#     totalW = [0 for _ in range(k)]
#     totalB = 0
#     for _ in range(repeatTimes):
#         trainData = Data(k, m, epsilon)
#         model = Perceptron()
#         model.perceptronLearningAlg(trainData.data, trainData.Y)
#         for i in range(k):
#             totalW[i] += model.w[i]
#         totalB += model.b
#     kList.append(k)
#     w = []
#     for i in range(k):
#         w.append(totalW[i] / repeatTimes)
#     b = totalB / repeatTimes
#     deltaW = [idealW[i] - w[i] for i in range(k)]
#     deltaB = idealB - b
#     bias.append(np.linalg.norm(deltaW, 2)**2 + deltaB**2)
# print('K:', kList)
# print('bias:', bias)
# plt.plot(kList, bias)
# plt.savefig('./bonusK.png')
# plt.clf()

# k = 5
# m = 100
# epsilonUniform = np.random.uniform(0, 1, 50) # 0,1,50
# repeatTimes = 10
# idealW = [0 for _ in range(k-1)] + [1]
# idealB = 0
# epsilonList = []
```

```
# bias = []
# for epsilon in sorted(epsilonUniform):
#     print('training with epsilon =', epsilonUniform)
#     totalW = [0 for _ in range(k)]
#     totalB = 0
#     for _ in range(repeatTimes):
#         trainData = Data(k, m, epsilon)
#         model = Perceptron()
#         model.perceptronLearningAlg(trainData.data, trainData.Y)
#         for i in range(k):
#             totalW[i] += model.w[i]
#         totalB += model.b
#     epsilonList.append(epsilon)
#     w = []
#     for i in range(k):
#         w.append(totalW[i] / repeatTimes)
#     b = totalB / repeatTimes
#     deltaW = [idealW[i] - w[i] for i in range(k)]
#     deltaB = idealB - b
#     bias.append(np.linalg.norm(deltaW, 2)**2 + deltaB**2)
# print('Epsilon:', epsilonList)
# print('bias:', bias)
# plt.plot(epsilonList, bias)
# plt.savefig('./bonusEpsilon.png')
```