Homework 2

Thursday, February 23, 2017 11:07 PM

Griffin Solimini(.1)
CSE 5523 Machine Learning

1a. $r = \ln \dfrac{N\left(\begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}\right)}{N\left(\begin{pmatrix}-1\\0\end{pmatrix}, \begin{pmatrix}1 & 1/2\\1/2 & 1\end{pmatrix}\right)}$

$$r = \ln \frac{\frac{1}{2\pi|C_+|^{1/2}} e^{-\frac{1}{2}(x-u_+)^T C_+^{-1}(x-u_+)}}{\frac{1}{2\pi|C_-|^{1/2}} e^{-\frac{1}{2}(x-u_-)C_-^{-1}(x-u_-)}}$$

$$r = \ln\left(\frac{1}{|C_+|^{1/2}}\right) - \frac{1}{2}(x-u_+)^T C_+^{-1}(x-u_+) - \ln\left(\frac{1}{|C_-|^{1/2}}\right) + \frac{1}{2}(x-u_-)C_-^{-1}(x-u_-)$$

$$|C_+| = \left|\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\right| = 1 \qquad C_+^{-1} = \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$$

$$|C_-| = \left|\begin{bmatrix}1 & 1/2\\1/2 & 1\end{bmatrix}\right| = \frac{3}{4} \qquad C_-^{-1} = \begin{bmatrix}4/3 & -2/3\\-2/3 & 4/3\end{bmatrix}$$

$$r = \cancel{\ln(1)}^{\,0} - \frac{1}{2}\begin{bmatrix}x_1-1 & x_2\end{bmatrix}\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}^{-1}\begin{bmatrix}x_1-1\\x_2\end{bmatrix}$$

$$- \ln\left(\frac{2}{\sqrt{3}}\right) + \frac{1}{2}\begin{bmatrix}x_1+1 & x_2\end{bmatrix}\begin{bmatrix}1 & 1/2\\1/2 & 1\end{bmatrix}^{-1}\begin{bmatrix}x_1+1\\x_2\end{bmatrix}$$

$$r = -\frac{1}{2}\left(\begin{bmatrix}x_1-1 & x_2\end{bmatrix}\begin{bmatrix}x_1-1\\x_2\end{bmatrix}\right) - \ln\left(\frac{2}{\sqrt{3}}\right) + \frac{1}{2}\left(\begin{bmatrix}\frac{4}{3}(x_1+1)-\frac{2}{3}x_2 & -\frac{2}{3}(x_1+1)+\frac{4}{3}x_2\end{bmatrix}\begin{bmatrix}x_1+1\\x_2\end{bmatrix}\right.$$

$$r = -\frac{1}{2}\left((x_1-1)^2 + x_2^2\right) - \ln\left(\frac{2}{\sqrt{3}}\right) + \frac{1}{2}\left((x_1+1)\left(\frac{4}{3}(x_1+1) - \frac{2}{3}x_2\right) + x_2\left(-\frac{2}{3}(x_1+1) + \frac{4}{3}x_2\right)\right)$$

$$r = -\frac{1}{2}\left(x_1^2 - 2x_1 + 1 + x_2^2\right) - \ln\left(\frac{2}{\sqrt{3}}\right) + \frac{1}{2}\left(\frac{4}{3}(x_1+1)^2 - \frac{4}{3}x_2(x_1+1) + \frac{4}{3}x_2^2\right)$$

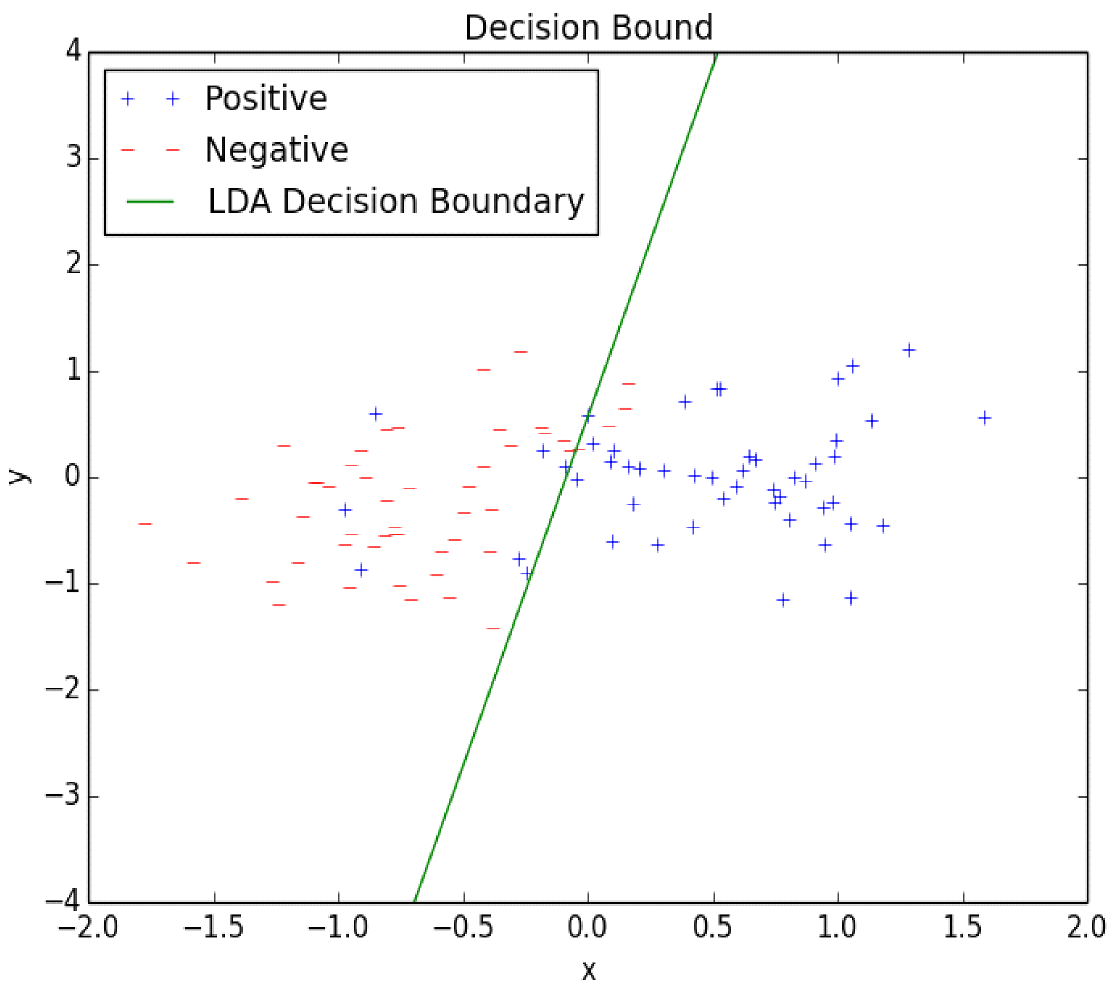
$$r = -\frac{1}{2}x_1^2 + x_1 - \frac{1}{2} - \frac{1}{2}x_2^2 + \frac{1}{2}\left(\frac{4}{3}\left(x_1^2 + 2x_1 + 1\right) - \frac{4}{3}x_2x_1 - \frac{4}{3}x_2 + \frac{4}{3}x_2^2\right) - \ln\left(\frac{2}{\sqrt{3}}\right)$$

$$r = \cancel{-\frac{1}{2}x_1^2} + \cancel{x_1} - \cancel{\frac{1}{2}} - \cancel{\frac{1}{2}x_2^2} + \cancel{\frac{2}{3}x_1^2} + \cancel{\frac{4}{3}x_1} + \cancel{\frac{2}{3}} - \cancel{\frac{2}{3}}x_2x_1 - \cancel{\frac{2}{3}x_2} + \cancel{\frac{2}{3}x_2^2} - \ln\left(\frac{2}{\sqrt{3}}\right)$$

$$\boxed{r = \frac{1}{6}x_1^2 + \frac{7}{3}x_1 + \frac{1}{6}x_2^2 - \frac{2}{3}x_2 - \frac{2}{3}x_2x_1 + \frac{1}{6} - \ln\left(\frac{2}{\sqrt{3}}\right)}$$

```python
import numpy
import matplotlib.pyplot as plt
import math

# sample properties
mean_pos = [1, 0]
mean_neg = [-1, 0]
cov_pos = [[1, 0], [0, 1]]
cov_neg = [[1, .5], [.5, 1]]

# bayesian decision bound equation
def bayesianDecisionBound((x,y)):
    return (1/6 * x**2) + (7/3 * x) + (1/6 * y**2) - (2/3 * y) - (2/3 * x * y) + (1/6) - math.log(2 / math.sqrt(3)) >= 0

# data generation
x, y = .5 * numpy.random.multivariate_normal(mean_pos, cov_pos, 50).T
plt.plot(x, y, '+', label='Positive')

pos_pts = zip(x, y)

x, y = .5 * numpy.random.multivariate_normal(mean_neg, cov_neg, 50).T
plt.plot(x, y, 'r_', label='Negative')

neg_pts = zip(x, y)

# calculate bayesian decision bound test error

# calculate LDA
# Calculate sample mean and sample covariance for positive class
x, y = zip(*pos_pts)
sample_mean_pos = numpy.matrix([numpy.mean(x), numpy.mean(y)]).T

sample_cov_pos = numpy.matrix([[0.0, 0.0],[0.0, 0.0]])
for i, j in pos_pts:
    tmp = numpy.matrix([i, j]).T
    sample_cov_pos += (tmp - sample_mean_pos) * (tmp - sample_mean_pos).T
sample_cov_pos /= 50.0

# Calculate sample mean and sample covariance for negative class
x, y = zip(*neg_pts)
sample_mean_neg = numpy.matrix([numpy.mean(x), numpy.mean(y)]).T

sample_cov_neg = numpy.matrix([[0.0, 0.0],[0.0, 0.0]])
for i, j in zip(x, y):
    tmp = numpy.matrix([i, j]).T
    sample_cov_neg += (tmp - sample_mean_neg) * (tmp - sample_mean_neg).T
sample_cov_neg /= 50.0

Cw = sample_cov_pos + sample_cov_neg
direction = Cw.I * (sample_mean_pos - sample_mean_neg)
const = -.5 * sample_mean_pos.T * Cw.I * sample_mean_pos + 0.5 * sample_mean_neg.T * Cw.I * sample_mean_neg

# equation for lda line
def lda_bound(x):
    return (float(direction[0]) * x + float(const)) / -float(direction[1])


# lda decision bound given a point
def ldaDecision((x,y)):
   return float(direction[0]) * x + float(direction[1]) * y + float(const) >= 0

# calcualte bayesian test error
bayesian_correct = 0
for pt in pos_pts:
    if bayesianDecisionBound(pt):
        bayesian_correct += 1

for pt in neg_pts:
    if not bayesianDecisionBound(pt):
        bayesian_correct += 1

print "bayesian test error: " + str(1 - (bayesian_correct / 100.0))

# calculate lda test error
lda_correct = 0
for pt in pos_pts:
    if ldaDecision(pt):
        lda_correct += 1

for pt in neg_pts:
    if not ldaDecision(pt):
        lda_correct += 1

print "lda test error: " + str(1 - (lda_correct / 100.0))
```

```
    if not ldaDecision(pt):
        lda_correct += 1

print "lda test error: " + str(1 - (lda_correct / 100.0))

# plot lda line
x = numpy.arange(-3, 4, 1)
bound = lda_bound(x)

plt.plot(x, bound, '-', label='LDA Decision Boundary')

plt.xlabel('x')
plt.ylabel('y')
plt.title('Decision Bound')
plt.legend(loc='upper left')
plt.xlim(-2, 2)
plt.ylim(-4, 4)
plt.show()
```



1b. Test error of decision bound: 0.13

1d. Test error of LDA band: 0.12

Yes, the errors of the two bands were close

2a. With probability at least $1 - \delta$, 2a. With probability at least $1 - \delta$,

we know $R(f_s) \leq \epsilon$

So can redefine $\epsilon = R(f_s)$ as upper band on $R(f_s)$

So we can use $R(f_s) \leq \frac{1}{N}\left(\log |H| + \log \frac{1}{\delta}\right)$

to write $\epsilon \leq \frac{1}{N}\left(\log |H| + \log \frac{1}{\delta}\right)$

so $N \leq \frac{1}{\epsilon}\left(\log |H| + \log \frac{1}{\delta}\right)$

and $|S| \leq N$ so

$$\boxed{|S| \leq \frac{1}{\epsilon}\left(\log |H| + \log \frac{1}{\delta}\right)}$$

2b. We know with probability at least $1-\delta$,

$$R(f) - R_s(f) \leq \sqrt{\frac{1}{2N}\left(\log|H| + \log\frac{2}{\delta}\right)}$$

and because we want $R(f) - R_s(f)$

to be less than ϵ

we redefine $\epsilon = R(f) - R_s(f)$

so with probability at least $1-\delta$

$$\epsilon \leq \sqrt{\frac{1}{2N}\left(\log|H| + \log\frac{2}{\delta}\right)}$$

$$\epsilon^2 \leq \frac{1}{2N}\left(\log|H| + \log\frac{2}{\delta}\right.$$

$$N \leq \frac{1}{2\epsilon^2}\left(\log|H| + \log\frac{2}{\delta}\right)$$

and because $|S| \leq N$

$$\boxed{|S| \leq \frac{1}{2\epsilon^2}\left(\log|H| + \log\frac{2}{\delta}\right)}$$

3a. $P_r[x \geq tE[x]] = \int_{tE[x]}^{\infty} p(x)\,dx$ $\quad ?: \underline{tE[x]}$

$$\leq \int_{tE[x]}^{\infty} p(x)\frac{x}{tE[x]}\,dx \quad ?: \underline{tE[x]}$$

$$\leq \int_{0}^{\infty} p(x)\frac{x}{tE[x]}\,dx$$

$$= E\left[\frac{x}{tE[x]}\right] \quad ?: \underline{\frac{x}{tE[x]}}$$

$$= \frac{1}{tE[x]}E[x] \quad ?: \underline{x}$$

$$= \frac{1}{t}$$

3b. We need to specify $\boxed{z_i = 1_{p(x_i) \neq y_i}}$ for

$S = \{(x_i, y_i\}$ where $1 \leq i \leq N$

using the range $\boxed{0 \leq z_i \leq 1}$ for classification

and the average $S_N = \frac{1}{N} \sum_{i=1}^{N} 1_{p(x_i) \neq y_i} \boxed{= R_S(f)}$

so $E[S_N] = E[R_S(f)] = \frac{1}{N} \sum E[1_{p(x_i) \neq y_i}] = \frac{N}{N} R(f) \boxed{= R(f)}$

Using Hoeffding's Inequality $Pr[S_N - E[S_N] \geq \epsilon] \, e^{-\frac{2N\epsilon^2}{(a-b)^2}}$

and
$$a = 0$$
$$b = 1$$
$$S_N = R_s(f)$$
$$E[S_N] = R(f)$$

we get $P[|R(f) - R_s(f)| \geq \epsilon] \leq 2e^{-2N\epsilon^2}$

4a. Let $\{v_1, v_2, v_3, v_4, v_5\} \subseteq \mathbb{R}$ be distinct

Let R be a rectangle that contains all the maximum and minimum x values, and maximum and minimum y values, which may or may not be distinct.

Let this set of points, S, fulfill:

$$S \subset \{v_1, v_2, v_3, v_4, v_5\}$$

Any axis aligned rectangle containing S must also contain $\{v_1, v_2, v_3, v_4, v_5\}$

There must be a $v \in \{v_1, v_2, v_3, v_4, v_5\}$

not in S that is in the rectangle.

So labeling each point in S with '+' and v with '-' is impossible with any axis aligned rectangle

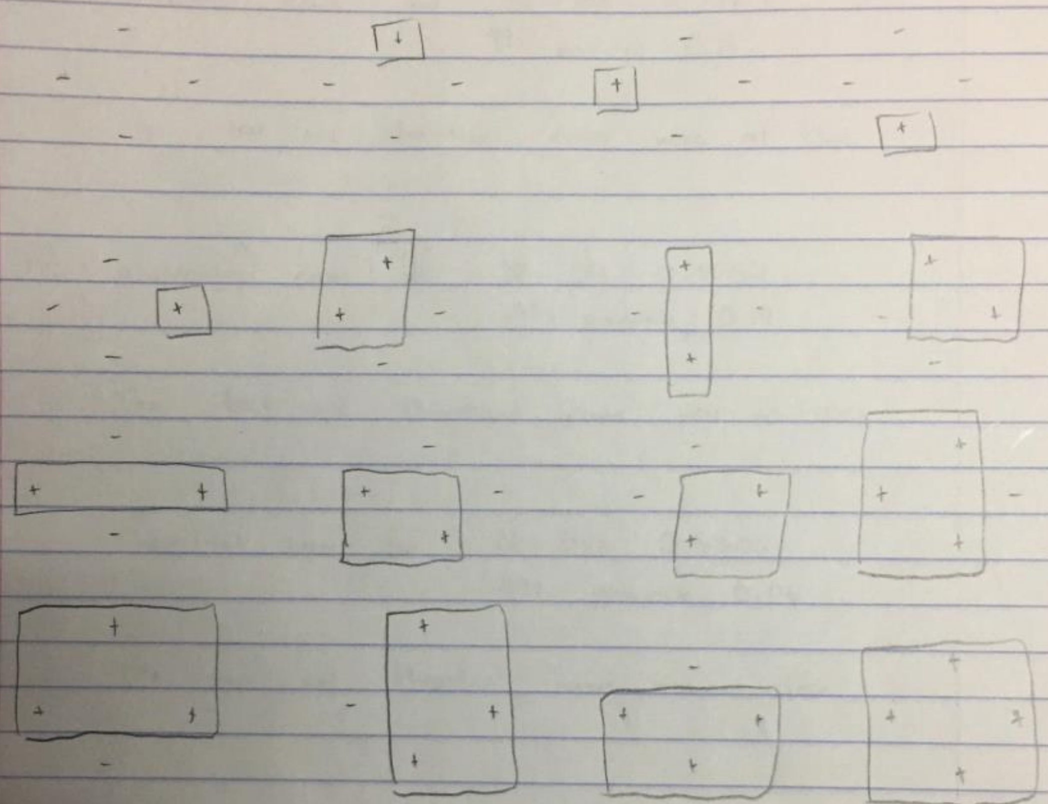
Therefore there is no shattered set of size 5.

So VCdim (H = axis aligned rectangles) < 5

(Next page)

However, 4 points can be shattered as

However, 4 points can be shattered as shown:



So $VCdim(H = \text{axis aligned rectangles}) \geq 4$

Using both proofs we know

$$VCdim(H = \text{axis aligned rectangles}) = 4$$

```python
import numpy
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import math
import sys

# equation that returns the theoretical error
def g_bound(N, d, delta):
    return math.sqrt(2*d*math.log(2.7182*N/d)/N)+math.sqrt(math.log(1/delta)/(2*N))

# function that performs algorithm 1 with N points
def algorithm1(N, plot):

    # Randomly generate rectangle
    rect_x = float(numpy.random.rand(1, 1))
    rect_y = float(numpy.random.rand(1, 1))

    rect_width = 1.0
    while rect_width + rect_x > 1.0:
        rect_width = float(numpy.random.rand(1, 1))

    rect_height = 1.0
    while rect_height + rect_y > 1.0:
        rect_height = float(numpy.random.rand(1, 1))

    # Randomly generate training points
    pts = numpy.random.rand(N, 2)

    # Classify points
    pos_pts = []
    neg_pts = []
    for pt in pts:
        x, y = pt
        if x >= rect_x and x <= rect_x + rect_width and y >= rect_y and y <= rect_y + rect_height:
            pos_pts.append((x, y))
        else:
            neg_pts.append((x, y))

    # If no positive points, exit
    if len(pos_pts) == 0:
        return -1

    # Generate smallest possible rectangle
    min_x = float("inf")
    max_x = 0.0
    min_y = float("inf")
    max_y = 0.0

    for pt in pos_pts:
        x, y = pt
        if x > max_x:
            max_x = x
        if x < min_x:
            min_x = x
        if y > max_y:
            max_y = y
        if y < min_y:
        if y < min_y:
            min_y = y

    # Classify test points based on algorithm 1 rectangle
    correct = 0
    test_pts = numpy.random.rand(N, 2)
    for pt in test_pts:
        x, y = pt
        if x >= rect_x and x <= rect_x+rect_width and y >= rect_y and y <= rect_y+rect_height:
            if x >= min_x and x <= max_x and y >= min_y and y <= max_y:
                correct += 1
        else:
            if x < min_x or x > max_x or y < min_y or y > max_y:
                correct += 1

    # Plot if flag set
    if plot:
        currentAxis = plt.gca()
        currentAxis.add_patch(patches.Rectangle((rect_x, rect_y),
                                                rect_width,
```

```
    if plot:
        currentAxis = plt.gca()
        currentAxis.add_patch(patches.Rectangle((rect_x, rect_y),
                                                rect_width,
                                                rect_height,
                                                alpha=1,
                                                facecolor='none'))

        currentAxis.add_patch(patches.Rectangle((min_x, min_y),
                                            max_x - min_x,
                                            max_y - min_y,
                                            alpha=1,
                                            facecolor='none',
                                            edgecolor="magenta"))

        x, y = zip(*pos_pts)
        plt.plot(x, y,'+', label="Positive")

        x, y = zip(*neg_pts)
        plt.plot(x, y, 'r_', label="Negative")

        plt.plot([],[], 'm-', label="Algorithm 1")

        plt.plot([],[], 'k-', label="Concept")

        plt.xlim(-0.25, 1.25)
        plt.ylim(-0.25, 1.25)

        plt.xlabel('x')
        plt.ylabel('y')
        plt.title('Algorithm 1')
        plt.legend(loc="upper left")
        plt.show()
```

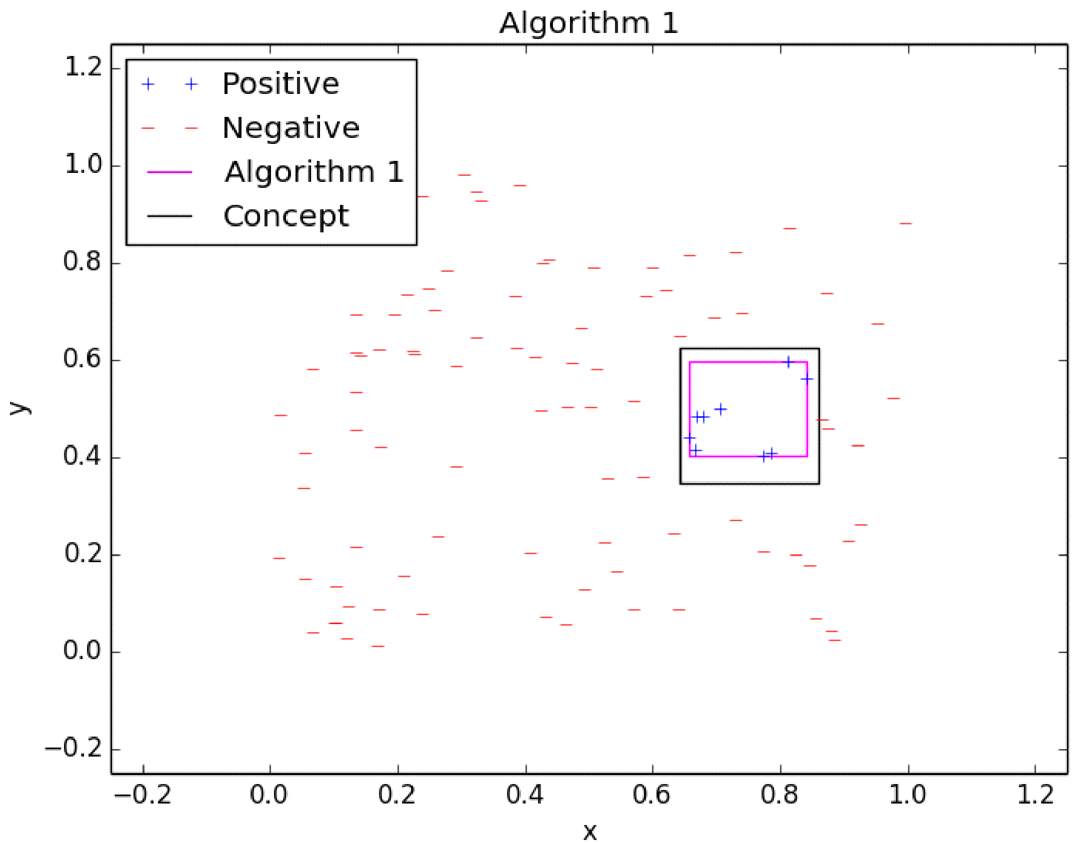

```
        plt.title('Algorithm 1')
        plt.legend(loc="upper left")
        plt.show()
```

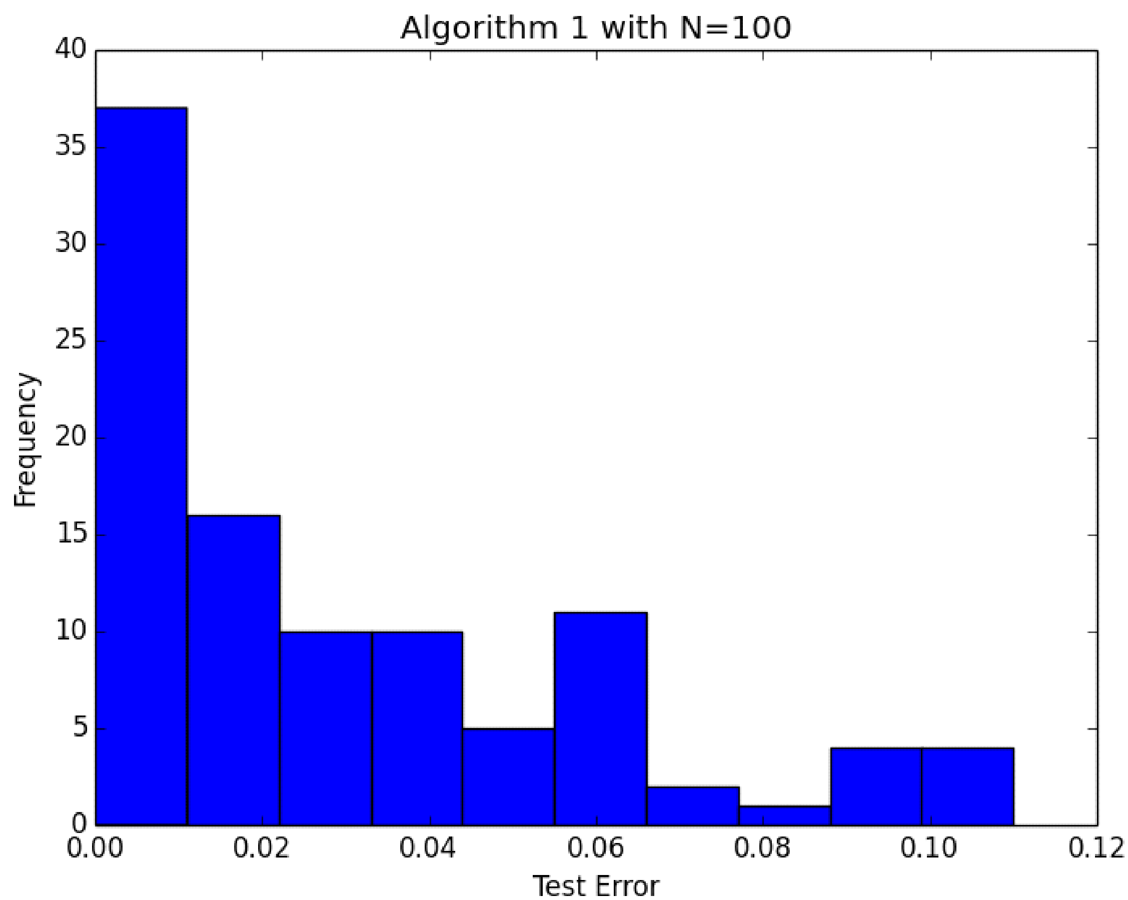

```

    # return test error
    return (1 - correct / float(N))


# Perform algorithm once
result = -1
while result == -1:
    result = algorithm1(100, True)

print "test error from one trial, N=100: " + str(result)

# build histogram for T=100 N=100
results = []
i = 0
while i < 100:
   result = algorithm1(100, False)
   if result != -1:
       results.append(result)
       i += 1

plt.hist(results, 10)
plt.xlabel('Test Error')
plt.ylabel('Frequency')
plt.title('Algorithm 1 with N=100')
plt.show()

print "theoretical error for T=100 N=100: " + str(g_bound(100, 4.0, 0.01))

# build histogram for T=100 N=50
results = []
i = 0
while i < 100:
   result = algorithm1(50, False)
   if result != -1:
       results.append(result)
```

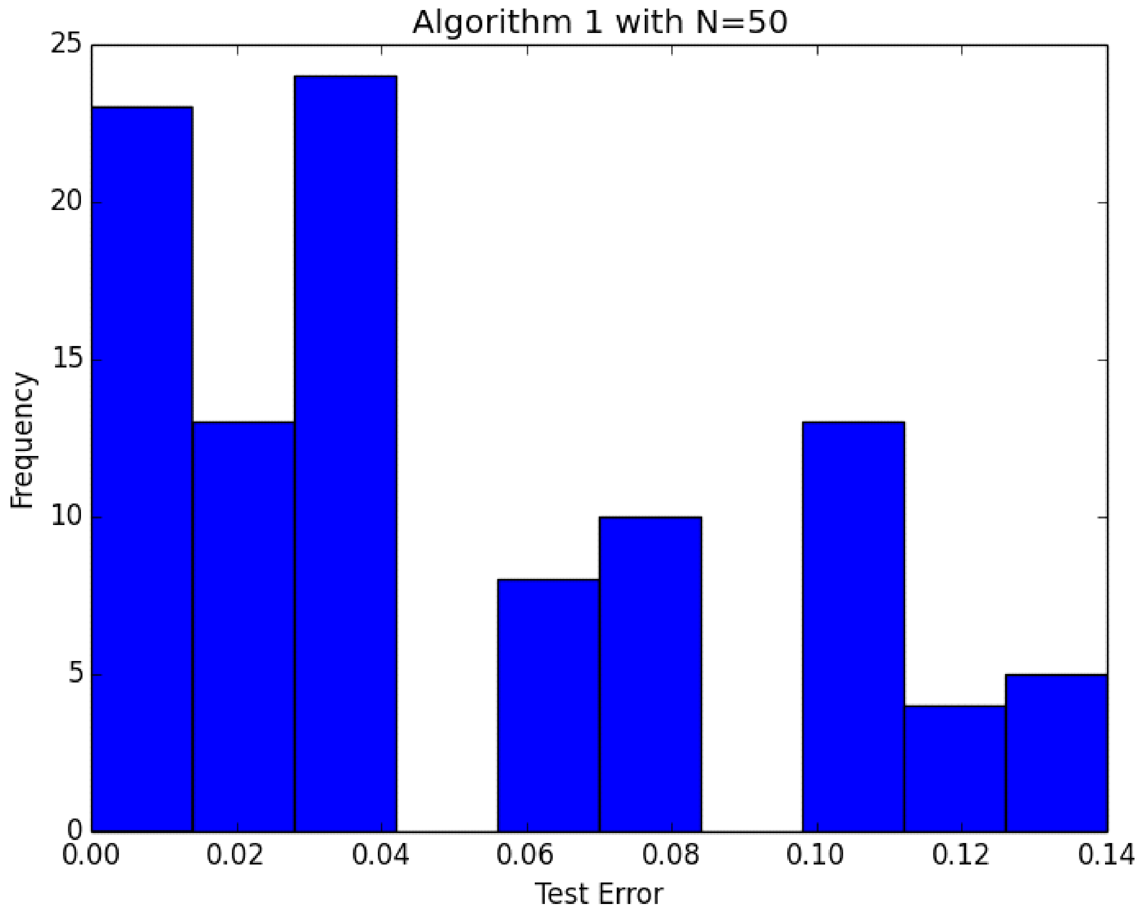
```
while i < 100:
    result = algorithm1(50, False)
    if result != -1:
        results.append(result)
        i += 1

plt.hist(results, 10)
plt.xlabel('Test Error')
plt.ylabel('Frequency')
plt.title('Algorithm 1 with N=50')
plt.show()

print "theoretical error for T=100 N=50: " + str(g_bound(50, 4.0, 0.01))

# build histogram for T=100 N=200
results = []
i = 0
while i < 100:
    result = algorithm1(200, False)
    if result != -1:
        results.append(result)
        i += 1

```

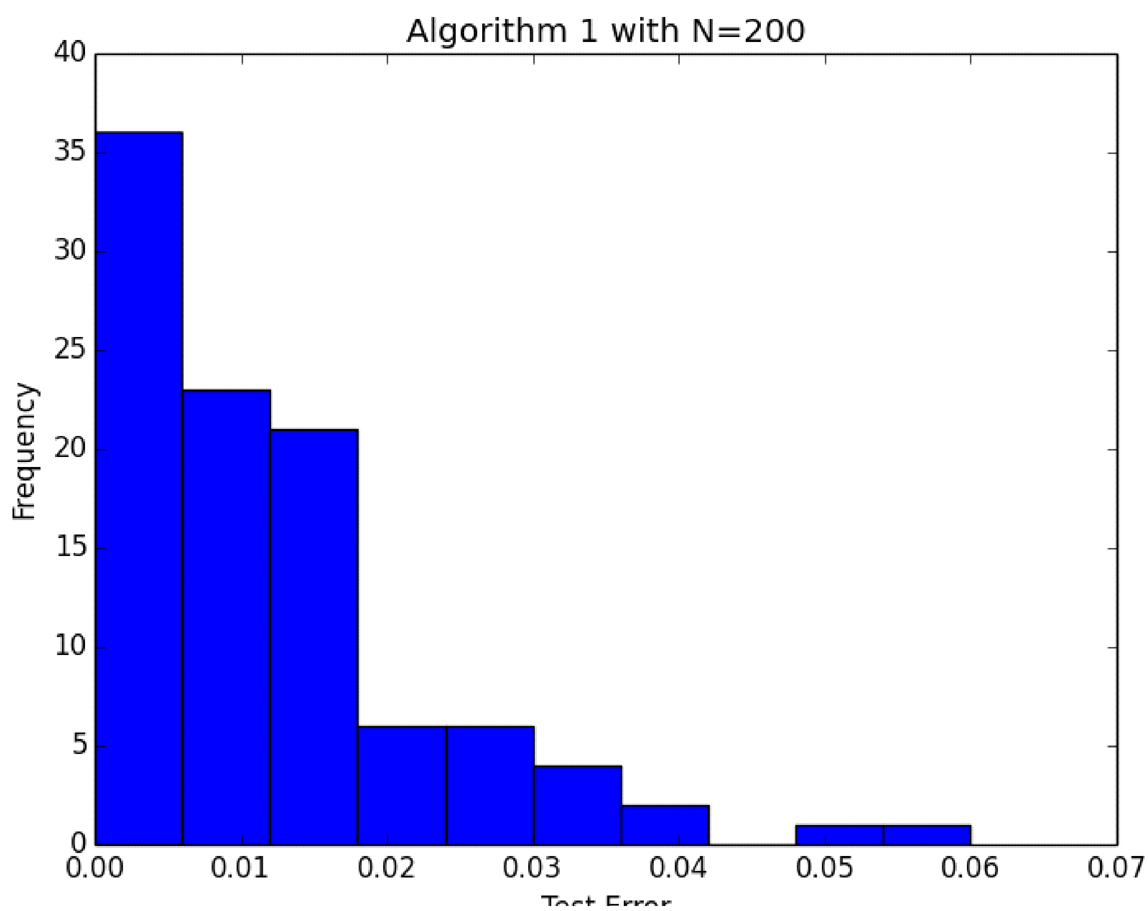
```
    if result != -1:
        results.append(result)
        i += 1

```

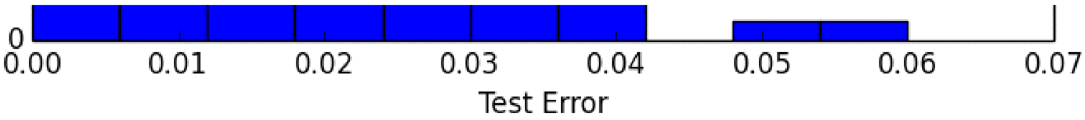

```
plt.hist(results, 10)
plt.xlabel('Test Error')
plt.ylabel('Frequency')
plt.title('Algorithm 1 with N=200')
plt.show()

print "theoretical error for T=100 N=200: " + str(g_bound(200, 4.0, 0.01))
```


Algorithm 1
Positive
Negative
Algorithm 1
Concept
y
x
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2


Algorithm 1 with N=100
Frequency
Test Error
40
35
30
25
20
15
10
5
0
0.00
0.02
0.04
0.06
0.08
0.10
0.12


Algorithm 1 with N=50
Frequency
Test Error
0
5
10
15
20
25
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14


Algorithm 1 with N=200
Frequency
0
5
10
15
20
25
30
35
40
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07



4c. The test error was 0.02

4e. Theoretical error for N = 100 : 0.7327
99% percentile : 0.11

The test and theoretical errors were not close

4f. Theoretical error for N = 50 : 0.9657
99% percentile : 0.14

The test and thoretical errors were not close

Theoretical error for N = 200 : 0.5506
99% percentile : 0.06

The test and theoretical errors were not close