

Learning From Data - worked examples

Mac Radigan

Chapter 1 Notes

Symbol	Name
\mathcal{X}	Input Set
\mathcal{D}	Training Set
\mathcal{Y}	Target Set
f	Target Function
\mathcal{H}	Hypothesis Set
\mathcal{A}	Learning Algorithm
M	Cardinality of Hypothesis Set
g	Final Hypothesis
ϵ	Residual Error

Relations and Sets

$$f : \mathcal{X} \rightarrow \mathcal{Y}$$

$$\{h_1, h_2, \dots, h_M\} \in \mathcal{H}$$

Hoeffding Inequality

$$P[|\nu - \mu| > \epsilon] \leq 2e^{-2\epsilon^2 N} \forall \epsilon > 0 \quad (1)$$

$$P[|E_{in}(h) - E_{out}(h)| > \epsilon] \leq 2e^{-2\epsilon^2 N} \forall \epsilon > 0 \quad (2)$$

For the final hypothesis g selected from \mathcal{H} ,

$$\begin{aligned}
P[|E_{in}(g) - E_{out}(g)| > \epsilon] &\leq P[|E_{in}(h_1) - E_{out}(h_1)| > \epsilon] \\
&\text{or} \leq P[|E_{in}(h_1) - E_{out}(h_1)| > \epsilon] \\
&\dots \\
&\text{or} \leq P[|E_{in}(h_M) - E_{out}(h_M)| > \epsilon] \\
&\leq \sum_{m=1}^M P[|E_{in}(h_m) - E_{out}(h_m)| > \epsilon] \\
&\leq 2Me^{-1\epsilon^2 N}
\end{aligned} \tag{3}$$

In-sample Error

$$E_{in}(h) = \frac{1}{N} \sum_{n=1}^N P[h(x_n) \neq f(x_n)] \text{ fraction of D where f \& h disagree} \tag{4}$$

Out-of-sample Error

$$E_{out}(h) = P[h(x) \neq f(x)] \tag{5}$$

Chapter 1 Additional Notes

Markov's Inequality

$$P[x \geq \alpha] \leq \frac{E(x)}{\alpha} \text{ for } \alpha > 0$$

Proof:

$$\begin{aligned}
E(x) &= \int xP(x)dx \\
&= \int_0^\alpha xP(x)dx + \int_\alpha^\infty xP(x)dx \\
&\geq \int_\alpha^\infty xP(x)dx \\
&\geq \int_\alpha^\infty \alpha P(x)dx \\
&= \alpha \int_\alpha^\infty P(x)dx \\
&= \alpha P[x \geq \alpha]
\end{aligned} \tag{6}$$

Information Theory

In information theory, the analog of the law of large numbers is the Asymptotic Equipartition Property (AEP).

The Law Of Large Numbers states that for Independent, Identically Distributed (*i.i.d.*) random variables states:

$$\frac{1}{N} \sum_{k=1}^N X_k \rightarrow E(N) \text{ for sufficiently large } N$$

The Asymptotic Equipartition Property (AEP) states:

$$\frac{1}{N} \log_2 \frac{1}{p(X_1, X_2, \dots, X_N)} \rightarrow H(X) \text{ where } H \text{ is the entropy, } X_k \text{'s are } i.i.d. \text{ and } p(X_1, X_2, \dots, X_N) \text{ is the probability of observing the sequence } X_1, X_2, \dots, X_N.$$

This enables us to divide the set of all sequences into two sets, the *typical set*, where the sample entropy is close to the true entropy, and the nontypical set, which contains the other sequences.

Entropy

The entropy is a measure of the average uncertainty in a random variable.

The entropy of a random variable X with a probability mass function $p(x)$ is defined by

$$H(X) = - \sum_x p(x) \log_2 p(x) \quad (7)$$

Asymptotic Equipartition Property (AEP)

If X_1, X_2, \dots, X_N are *i.i.d* $\sim p(x)$, then

$$\frac{1}{N} \log_2 p(X_1, X_2, \dots, X_N) \rightarrow H(X) \text{ in probability} \quad (8)$$

Proof:

$$\begin{aligned} \frac{1}{N} \log_2 p(X_1, X_2, \dots, X_N) &= -\frac{1}{N} \sum_k \log_2 p(X_k) \\ &\rightarrow -E \log_2 p(X) \\ &\rightarrow H(X) \end{aligned}$$

Typical Set $A_\epsilon^{(n)}$

A *typical set* $A_\epsilon^{(n)}$ with respect to $p(x)$ is the set of sequences $(x_1, x_2, \dots, x_N) \in \mathcal{X}^N$ with the property

$$2^{-N(H(X)+\epsilon)} \leq p(x_1, x_2, \dots, x_N) \leq 2^{-N(H(X)-\epsilon)}$$

Problem 1.1

We have 2 opaque bags, each containing 2 balls. One bag has 2 black balls and the other has a black and a white ball. You pick a bag at random and then pick one of the balls in that bag at random. When you look at the ball it is black. You now pick the second ball from that same bag. What is the probability that this ball is also black? [Hint: Use Bayes' Theorem: $P[A \text{ and } B] = P[A|B]P[B] = P[B|A]P[A]$.]

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}$$

- B_A the event that bag A was selected
- B_B the event that bag B was selected
- k_A the event that a black ball from bag A was selected
- k_B the event that a black ball from bag B was selected
- k_1 the event that a black ball was selected on the first selection
- k_2 the event that a black ball was selected on the second selection

$$k_1 = B_A k_A + B_B k_B = \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2} \cdot \frac{1}{4} = \frac{3}{4}$$

$$P(B_A|k_1) = \frac{P(B_A \cap k_1)}{P(k_1)} = \frac{P(k_1|B_B)P(B_A)}{P(k_1)}$$

$$P(B_B|k_1) = \frac{P(B_B \cap k_1)}{P(k_1)} = \frac{P(k_1|B_B)P(B_B)}{P(k_1)}$$

$$P(k_1|B_A) = \frac{1}{2} \cdot 1 = \frac{1}{2}$$

$$P(k_1|B_B) = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$$

$$P(B_A) = \frac{1}{2}$$

$$P(B_B) = \frac{1}{2}$$

$$P(k_1) = \frac{3}{4}$$

$$P(B_A|k_1) = \frac{P(k_1|B_A)P(B_A)}{P(k_1)} = \frac{\frac{1}{2} \cdot \frac{1}{2}}{\frac{3}{4}} = \frac{\frac{1}{4}}{\frac{3}{4}} = \frac{1}{3}$$

$$P(B_B|k_1) = \frac{P(k_1|B_B)P(B_B)}{P(k_1)} = \frac{\frac{1}{4} \cdot \frac{1}{2}}{\frac{3}{4}} = \frac{\frac{1}{8}}{\frac{3}{4}} = \frac{1}{6}$$

$$P(k_2) = P(k_1|B_B)P(k_B) + P(B_B|k_1)P(k_B) = \frac{1}{3} \cdot 1 + \frac{1}{6} \cdot \frac{1}{2} = \frac{1}{3} + \frac{1}{12} + \frac{1}{12} = \frac{5}{12}$$

Problem 1.2

Consider the perceptron in two dimensions: $h(x) = \text{sign}(w^\top x)$ where $w = [w_0, w_1, w_2]$ and $x = [1, x_1, x_2]$. Technically, x has three coordinates, but we call this perceptron two-dimensional because the first coordinate is fixed at 1.

- (a) Show that the regions on the plane where $h(x) = +1$ and $h(x) = -1$ are separated by a line. If we express this line by the equation $x_2 = ax_1 + b$, what are the slope a and intercept b in terms of w_0, w_1, w_2 ?

$$w^\top x = 0$$

$$\implies w_0x_0 + w_1x_1 + w_2x_2 = 0$$

$$\implies w_2x_2 = -w_1x_1 - w_0x_0 = -w_1x_1 - x_0$$

$$\implies x_2 = -\frac{w_1}{w_2}x_1 - \frac{w_0}{w_2}$$

$$a = -\frac{w_1}{w_2}$$

$$b = -\frac{w_0}{w_2}$$

- (b) Draw a picture for the cases $w = [1, 2, 3]$ and $w = -[1, 2, 3]$.

In more than two dimensions, the $+1$ and -1 regions are separated by a hyperplane, the generalization of a line.

Problem 1.3

Prove that the PLA eventually converges to a linear separator for separable data. The following steps will guide you through the proof. Let w^* be an optimal set of weights (one which separates the data). The essential idea in this proof is to show that the PLA weights $w(t)$ get “more aligned” with w^* with every iteration. For simplicity, assume that $w(0) = 0$.

- (a) Let $\rho = \min_{1 \leq n \leq N} (w^{*\top} x_n)$. Show that $\rho > 0$.

We know $w^{*\top} x_n > 0 \forall n$, since x is linearly separable, and w^* separates x .

So by definition,

$$w^{*\top} x_n > 0 \quad \forall n \text{ s.t. } y_n = +1$$

and

$$w^{*\top} x_n < 0 \quad \forall n \text{ s.t. } y_n = -1$$

and thus

$$y_n (w^{\top} x_n) \forall n$$

that is, ρ is strictly positive.

(b) Show that $w^{\top}(t)w^* \geq w^{\top}(t-1)w^* + \rho$, and conclude that $w^{\top}(t) \geq t\rho$.

Let x' be the set of misclassified points, and let y' be the corresponding truth values

Further, let $\rho_m = \min_m y_m (w^{\top} x_m)$

Then $w_t = \sum_{m=0}^t y'_m x'_m$

But since $(y'_m x'_m) \leq (y'_t x'_t)^{\top} w^* \forall m, t$

then

$$\left(\sum_{m=0}^t y'_m x'_m \right) w^* \geq \left(\sum_{m=0}^{t-1} y'_m x'_m \right)^{\top} w^* + y_t (x_t^{\top} w^*)$$

and thus

$$w^{\top}(t) \geq t\rho$$

(c) Show that $\|w(t)\|^2 \leq \|w(t-1)\| + \|w(t-1)\|^2$.

[Hint: $y(t-1) \cdot (w^{\top}(t-1)x(t-1)) \leq 0$ because $x(t-1)$ was misclassified by $w(t-1)$.]

Note $w_t = w_{t-1} + y_{t-1}x_{t-1}$

so

$$\|w_{t-1} + y_{t-1}x_{t-1}\|^2 \leq \|w_{t-1}^2\|^2 + \|x_{t-1}^2\|^2$$

then

$$(w_{t-1} + y_{t-1}x_{t-1})^2 \triangleq w_{t-1}^2 + 2w_{t-1}^{\top}y_{t-1}x_{t-1} + y_{t-1}^2x_{t-1}^2$$

but $2w_{t-1}^{\top}y_{t-1}x_{t-1} < 0$

thus

$$\|w(t)\|^2 \leq \|w(t-1)\| + \|w(t-1)\|^2$$

(d) Show by induction that $\|w(t)\|^2 \leq tR^2$, where $R = \max_{1 \leq n \leq N} \|x_n\|$.

Base case $t = 1$:

$$w_1 = w_0 + x'_m y'_m = 0 + x'_m y'_m = y'_m x'_m, \text{ and } \|x_m\| \leq \|x_n\|$$

(e) Using (b) and (d), show that

$$\frac{w^\top}{\|w(t)\|^2} w^* \geq \sqrt{t} \cdot \frac{\rho}{R},$$

and hence prove that

$$t \leq \frac{R^2 \|w^*\|^2}{\rho^2}.$$

[Hint: $\frac{w^\top(t)w^*}{\|w(t)\|\|w^*\|} \leq 1$. Why?]

Note that $\frac{w_t^\top w^*}{\|w_t\|\|w^*\|} \leq 1$

$$\text{so } \frac{w_t^\top w^*}{\|w_t\|} \leq \|w^*\|$$

$$\text{Now } \|w_t\|^2 \leq tR^2 \Rightarrow \|w_t\| \leq \sqrt{t}R$$

$$\text{Then } \frac{1}{\|w_t\|} \geq \frac{1}{\sqrt{t}R}$$

$$\text{Thus } \frac{w_t^\top w^*}{\|w_t\|} \geq \frac{\sqrt{t}\rho}{R}$$

In practice, PLA converges more quickly than the bound $\frac{R^2 \|w^*\|^2}{\rho^2}$ suggests. Nevertheless, because we do not know ρ in advance, we can't determine the number of iterations to convergence, which does pose a problem if the data is non-separable.

Problem 1.4

In Exercise 1.4, we use an artificial data set to study the perceptron learning algorithm. This problem leads you to explore the algorithm further with data sets of different sizes and dimensions.

```

1  ## pla
2  ## Mac Radigan
3
4  """pla.py
5
6  Perceptron Learning Algorithm (PLA)
7
8
9  initialization:
10
11     N data dimensionality
12     K number of samples in dataset
13
14     W [Nx1] initial weights
15
16
17 random dataset generation:
18
19     X      [NxK] ~ U[a,b] random training samples

```

```

20     W_star [Nx1] ~ U[a,b]  random target function
21
22
23     update step:
24
25     y_hat  = signum( X * W' )
26
27     y_err  = |y - y_hat|
28
29     rhos   = x * y
30     k      = argmax y_err
31
32     err    = exist y" in y_err s.t. |y"| > 0
33
34     W      = W + Y_k * X_k  if err
35
36
37     training:
38
39     update while err
40
41
42     """
43
44     import numpy as np
45     import matplotlib.pyplot as plt
46
47     class PLA:
48
49     def __init__(self, n):
50         """initialize a new PLA:
51             n - dimension
52         """
53         self.new(n)
54
55     def new(self, n):
56         """initialize a new PLA:
57             n - dimension
58         """
59         self.n = n                # data dimensionality
60         self.t = 0                # total iterations
61         self.w = np.random.rand(1,n+1) # initial weight
62
63     def __str__(self):
64         """print insternal state"""
65         return str(self.w)
66
67     def rand_data(self, k):
68         """create random data:
69             returns X, a Kx(N+1) real matrix of samples augmented as
        ⇨ homogeneous coordinates
70         """
71         a = -100                  # randomization lower bound
72         b = +100                  # randomization upper bound
73         return np.concatenate((
74             np.ones((k, 1)),

```



```

75         a + (b-a) * np.random.rand(k, self.n)
76     ), axis=1)
77
78     def rand_target(self):
79         """create a random target function:
80             returns a random target function
81         """
82         a = -100                                # randomization lower bound
83         b = +100                                # randomization upper bound
84         return a + (b-a) * np.random.rand(1, self.n+1)
85
86     def target(self, x, w_star):
87         """create a training set:
88             x - data set, an 1xK matrix with {-1,+1} entries
89             returns truth training set
90         """
91         return np.sign(x.dot(w_star.T))
92
93     def train(self, x, y):
94         """run the learning algorithm
95             x - training data set, an NxK real matrix, augmented as homogenous
96             ↪ coordinates
97             returns the perceptron weights
98         """
99         while not self.update(x, y):
100             pass
101         return self.w
102
103     def select(self, x, y):
104         """choose a misclassified point
105             x - data set
106             returns a misclassified point
107         """
108         y_hat = np.sign(x.dot(self.w.T))          # classify training set
109         ↪ using current weights
110         y_err = np.abs(y - y_hat)                # compute error from
111         ↪ truth
112         ks = y_err.nonzero()[0]
113         err = np.any(ks)                          # did the algorithm
114         ↪ converge?
115         if err:
116             k = ks[0]
117         else:
118             k = None
119         return k
120
121     def update(self, x, y):
122         """PLA iterative update step
123             x - data set
124             returns a boolean indicating convergence
125         """
126         self.t = self.t + 1                        # increment iteration
127         y_hat = np.sign(x.dot(self.w.T))          # classify training set
128         ↪ using current weights
129         y_err = np.abs(y - y_hat)                # compute error from
130         ↪ truth

```

```

125     #ks = y_err.nonzero()[0]
126     #err = np.any(ks)                                # did the algorithm
    ↪ converge?
127     #if err:
128     #     k = ks[0]
129     #     self.w = self.w + y[k] * x[k]                # update weights
130     #     print('t: %d' % self.t)
131     #return not err
132     k = self.select(x, y)
133     if k is not None:
134         self.w = self.w + y[k] * x[k]                # update weights
135         print('t: %d' % self.t)
136         return False
137     else:
138         return True
139
140 def classify(self, x):
141     """classify a dataset using the internal PLA weights
142         x - data set
143         returns the classification
144     """
145     y = np.sign(x.dot(self.w.T))
146     return y
147
148 def plot(self, x, w, w_star, y, title=None, outfile=None):
149     """plot classification results
150         x - data set
151         w_star - target function
152     """
153     #plt.rc('text', usetex=True)
154     #plt.rc('font', family='serif')
155     fig = plt.figure()
156     ax = fig.add_subplot(1, 1, 1)
157     k_a = (y > 0).nonzero()[0]
158     x1_a = x[k_a, 1]
159     x2_a = x[k_a, 2]
160     k_b = (y < 0).nonzero()[0]
161     x1_b = x[k_b, 1]
162     x2_b = x[k_b, 2]
163     ax.scatter(x1_a, x2_a, color='b', marker='o')
164     ax.scatter(x1_b, x2_b, color='r', marker='x')
165     dx1 = np.linspace(-150, 150, 2)
166     ## target function
167     w0 = w_star.T[0]
168     w1 = w_star.T[1]
169     w2 = w_star.T[2]
170     m = -w1/w2
171     b = -w0/w2
172     dx2 = m * dx1 + b
173     ax.plot(dx1, dx2, 'g')
174     ## PLA results
175     w0 = w.T[0]
176     w1 = w.T[1]
177     w2 = w.T[2]
178     m = -w1/w2
179     b = -w0/w2

```

```

180     dx2 = m * dx1 + b
181     ax.plot(dx1, dx2, 'm--')
182     ## decorations
183     ax.set_xlim(-200.0, 200.0)
184     ax.set_ylim(-200.0, 200.0)
185     plt.xlabel(r'x1')
186     plt.ylabel(r'x2')
187     if title is not None:
188         ax.set_title(title)
189     if outfile is not None:
190         print('save: %s' % (outfile))
191         plt.savefig(outfile)
192     else:
193         plt.show()
194     return ax
195
196 def plot1(self, x, w_star, y, title=None, outfile=None):
197     """plot classification results
198         x - data set
199         w_star - target function
200     """
201     #plt.rc('text', usetex=True)
202     #plt.rc('font', family='serif')
203     fig = plt.figure()
204     ax = fig.add_subplot(1, 1, 1)
205     k_a = (y > 0).nonzero()[0]
206     x1_a = x[k_a, 1]
207     x2_a = x[k_a, 2]
208     k_b = (y < 0).nonzero()[0]
209     x1_b = x[k_b, 1]
210     x2_b = x[k_b, 2]
211     ax.scatter(x1_a, x2_a, color='b', marker='o')
212     ax.scatter(x1_b, x2_b, color='r', marker='x')
213     w0 = w_star.T[0]
214     w1 = w_star.T[1]
215     w2 = w_star.T[2]
216     m = -w1/w2
217     b = -w0/w2
218     dx1 = np.linspace(-150, 150, 2)
219     dx2 = m * dx1 + b
220     ax.plot(dx1, dx2, color='m')
221     ax.set_xlim(-200.0, 200.0)
222     ax.set_ylim(-200.0, 200.0)
223     plt.xlabel(r'x1')
224     plt.ylabel(r'x2')
225     if title is not None:
226         ax.set_title(title)
227     if outfile is not None:
228         print('save: %s' % (outfile))
229         plt.savefig(outfile)
230     else:
231         plt.show()
232     return ax
233
234 def plot_misclassified(self, x, w_star, y, k, title=None, outfile=None):
235     """plot a misclassified point

```

```

236         x - data set
237         w_star - target function
238         """
239         #plt.rc('text', usetex=True)
240         #plt.rc('font', family='serif')
241         fig = plt.figure()
242         ax = fig.add_subplot(1, 1, 1)
243         k_a = (y > 0).nonzero()[0]
244         x1_a = x[k_a, 1]
245         x2_a = x[k_a, 2]
246         k_b = (y < 0).nonzero()[0]
247         x1_b = x[k_b, 1]
248         x2_b = x[k_b, 2]
249         ax.scatter(x1_a, x2_a, color='b', marker='o')
250         ax.scatter(x1_b, x2_b, color='r', marker='x')
251         x1_m = k[1]
252         x2_m = k[2]
253         ax.scatter(x1_m, x2_m, color='b', marker='*')
254         w0 = w_star.T[0]
255         w1 = w_star.T[1]
256         w2 = w_star.T[2]
257         m = -w1/w2
258         b = -w0/w2
259         dx1 = np.linspace(-150, 150, 2)
260         dx2 = m * dx1 + b
261         ax.plot(dx1, dx2, color='m')
262         ax.set_xlim(-200.0, 200.0)
263         ax.set_ylim(-200.0, 200.0)
264         plt.xlabel(r'x1')
265         plt.ylabel(r'x2')
266         if title is not None:
267             ax.set_title(title)
268         if outfile is not None:
269             print('save: %s' % (outfile))
270             plt.savefig(outfile)
271         else:
272             plt.show()
273         return ax
274
275     def training_movie(self, x, w_star, y, title=None, outfile=None):
276         """save a movie of the PLA training process
277         returns the perceptron weights
278         """
279         t = 0
280         err = True
281         base = outfile
282         #while err:
283         # k = self.select(x, y)
284         # outfile = "%s_%03.3d.png" % (base, self.t)
285         # title = "%s, t=%d" % (title, self.t)
286         # if k is not None:
287         #     self.plot_misclassified(x, w_star, y, k, title, outfile)
288         #     self.w = self.w + y[k] * x[k] # update weights
289         #     print('t: %d' % self.t)
290         #     err = True
291         # else:

```

```

292 # self.plot1(x, w_star, y, title, outfile)
293 # print('t: %d' % self.t)
294 # err = False
295 # t = t + 1
296 #return self.w
297 #####
298 while err:
299     k = self.select(x, y)
300     self.t = self.t + 1 # increment iteration
301     outfile = "%s_%03.3d.png" % (base, self.t)
302     title = "%s, t=%d" % (title, self.t)
303     y_hat = np.sign(x.dot(self.w.T)) # classify training
    ↪ set using current weights
304     y_err = np.abs(y - y_hat) # compute error from
    ↪ truth
305     ks = y_err.nonzero()[0]
306     err = np.any(ks) # did the algorithm
    ↪ converge?
307     if err:
308         k = ks[0]
309         self.w = self.w + y[k] * x[k] # update weights
310         self.plot_misclassified(x, w_star, y, k, title, outfile)
311         print('t: %d' % self.t)
312     else:
313         self.plot1(x, w_star, y, title, outfile)
314         print('t: %d' % self.t)
315     return not err
316
317 ## *EOF*

```

- (a) Generate a linearly separable data set of size 20 as indicated in Exercise 1.4. Plot the examples (x_n, y_n) as well as the target function f on a plane. Be sure to mark the examples from different classes differently, and add labels to the axes of the plot.

```

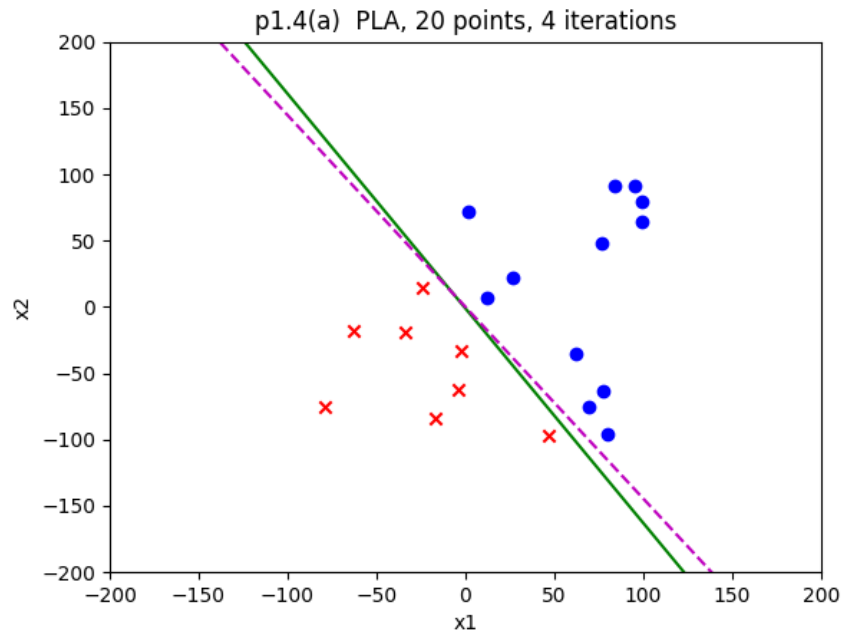
1  #!/usr/bin/env python
2  ## problem_1_4_a.py
3  ## Mac Radigan
4
5  """problem_1_4_a.py
6  """
7
8  import argparse
9  from sys import stdout
10 from perceptron import pla
11 import numpy as np
12
13 def main(verbose):
14     'problem 1.4'
15     dim = 20
16     test(dim)
17
18 def case(dim):
19     'problem 1.4'

```

```

20     outdir = 'figures/'
21     p = pla.PLA(2)
22     x = p.rand_data(dim)
23     w_star = p.rand_target()
24     y = p.target(x, w_star)
25     #outfile = '%s/%s' % (outdir, 'p1.4a_target.png')
26     #title = 'p1.4(a) Target Function, %d points' % (dim)
27     #ax = p.plot1(x, w_star, y, title, outfile)
28     w = p.train(x, y)
29     y_hat = p.classify(x)
30     y_err = y_hat - y;
31     err = 0 != np.sum(y_err)
32     #outfile = '%s/%s' % (outdir, 'p1.4a_classify.png')
33     #title = 'p1.4(a) PLA Results, %d iterations' % (p.t)
34     #ax = p.plot1(x, w, y_hat, title, outfile)
35     outfile = '%s/%s' % (outdir, 'p1.4a.png')
36     title = 'p1.4(a) PLA, %d points, %d iterations' % (dim, p.t)
37     ax = p.plot(x, w, w_star, y_hat, title, outfile)
38     return (p, x, w_star, y, w, y_hat, y_err, err)
39
40 def prn(name, value):
41     print('%s:\n%s\n' % (name, value))
42
43 def test(dim):
44     'test'
45     (p, x, w_star, y, w, y_hat, y_err, err) = case(dim)
46     prn("x", x)
47     prn("w_star", w_star)
48     prn("y", y.T)
49     prn("w", w)
50     prn("y_hat", y_hat.T)
51     #prn("y_err", y_err.T)
52     prn("err", err)
53
54 if __name__ == '__main__':
55     parser = argparse.ArgumentParser(description='problem 1.4 (a)')
56     parser.add_argument('-v', '--verbose', action='store_true',
57         ↪ dest='verbose', default=False, help='verbose output to stdout')
57     args = parser.parse_args()
58     main(args.verbose)
59
60 ## *EOF*

```



- (b) Run the perceptron learning algorithm on the data set above. Report the number of updates that the algorithm takes before converging. Plot the examples (x_n, y_n) , the target function f , and the final hypothesis g in the same figure. Comment on whether f is close to g .

```

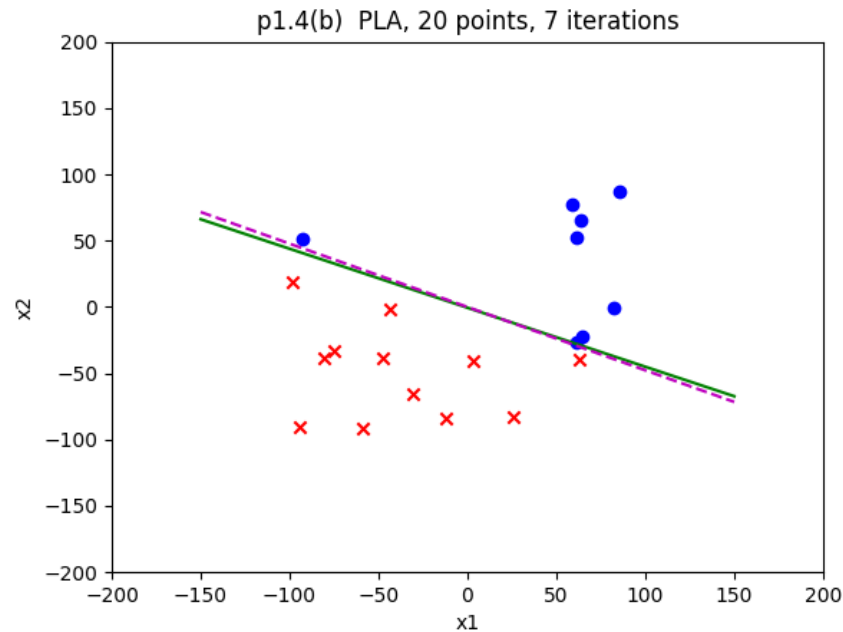
1  #!/usr/bin/env python
2  ## problem_1_4_b.py
3  ## Mac Radigan
4
5  """problem_1_4_b.py
6  """
7
8  import argparse
9  from sys import stdout
10 from perceptron import pla
11 import numpy as np
12
13 def main(verbose):
14     'problem 1.4'
15     dim = 20
16     test(dim)
17
18 def case(dim):
19     'problem 1.4'
20     outdir = 'figures/'
21     p = pla.PLA(2)

```

```

22 x = p.rand_data(dim)
23 w_star = p.rand_target()
24 y = p.target(x, w_star)
25 w = p.train(x, y)
26 y_hat = p.classify(x)
27 y_err = y_hat - y;
28 err = 0 != np.sum(y_err)
29 outfile = '%s/%s' % (outdir, 'p1.4b.png')
30 title = 'p1.4(b) PLA, %d points, %d iterations' % (dim, p.t)
31 ax = p.plot(x, w, w_star, y_hat, title, outfile)
32 return (p, x, w_star, y, w, y_hat, y_err, err)
33
34 def prn(name, value):
35     print('%s:\n%s\n' % (name, value))
36
37 def test(dim):
38     'test'
39     (p, x, w_star, y, w, y_hat, y_err, err) = case(dim)
40     prn("x", x)
41     prn("w_star", w_star)
42     prn("y", y.T)
43     prn("w", w)
44     prn("y_hat", y_hat.T)
45     #prn("y_err", y_err.T)
46     prn("err", err)
47
48 if __name__ == '__main__':
49     parser = argparse.ArgumentParser(description='problem 1.4 (a)')
50     parser.add_argument('-v', '--verbose', action='store_true',
51         ↪ dest='verbose', default=False, help='verbose output to stdout')
52     args = parser.parse_args()
53     main(args.verbose)
54     ## *EOF*

```

- (c) Repeat everything in (b) with another randomly generated data set of size 20. Compare your results with (b).

```

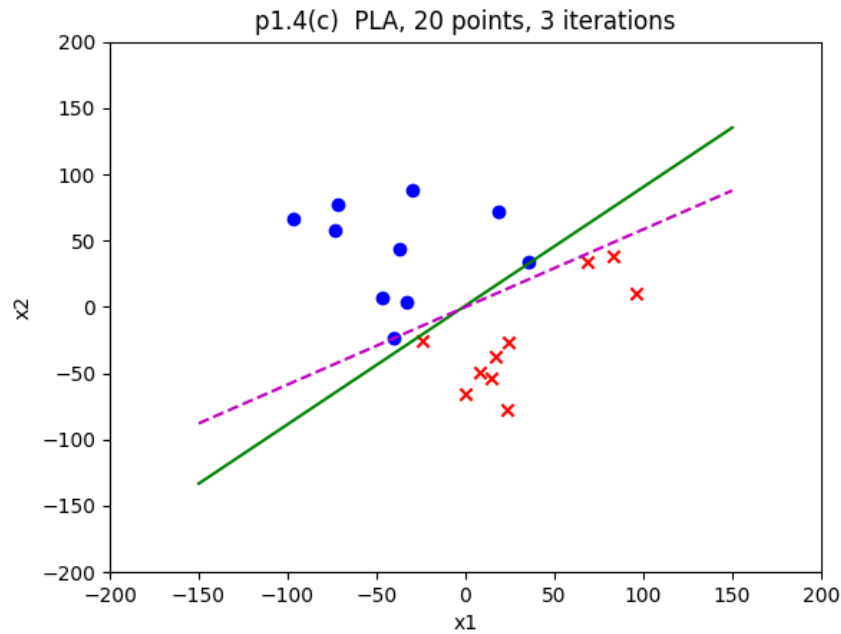
1  #!/usr/bin/env python
2  ## problem_1_4_c.py
3  ## Mac Radigan
4
5  """problem_1_4_c.py
6  """
7
8  import argparse
9  from sys import stdout
10 from perceptron import pla
11 import numpy as np
12
13 def main(verbose):
14     'problem 1.4'
15     dim = 20
16     test(dim)
17
18 def case(dim):
19     'problem 1.4'
20     outdir = 'figures/'
21     p = pla.PLA(2)
22     x = p.rand_data(dim)
23     w_star = p.rand_target()

```

```

24     y = p.target(x, w_star)
25     w = p.train(x, y)
26     y_hat = p.classify(x)
27     y_err = y_hat - y;
28     err = 0 != np.sum(y_err)
29     outfile = '%s/%s' % (outdir, 'p1.4c.png')
30     title = 'p1.4(c) PLA, %d points, %d iterations' % (dim, p.t)
31     ax = p.plot(x, w, w_star, y_hat, title, outfile)
32     return (p, x, w_star, y, w, y_hat, y_err, err)
33
34 def prn(name, value):
35     print('%s:\n%s\n' % (name, value))
36
37 def test(dim):
38     'test'
39     (p, x, w_star, y, w, y_hat, y_err, err) = case(dim)
40     prn("x", x)
41     prn("w_star", w_star)
42     prn("y'", y.T)
43     prn("w", w)
44     prn("y_hat'", y_hat.T)
45     #prn("y_err'", y_err.T)
46     prn("err'", err)
47
48 if __name__ == '__main__':
49     parser = argparse.ArgumentParser(description='problem 1.4 (a)')
50     parser.add_argument('-v', '--verbose', action='store_true',
51                         dest='verbose', default=False, help='verbose output to stdout')
52     args = parser.parse_args()
53     main(args.verbose)
54     ## *EOF*

```



- (d) Repeat everything in (b) with another randomly generated data set of size 100. Compare your results with (b).

```

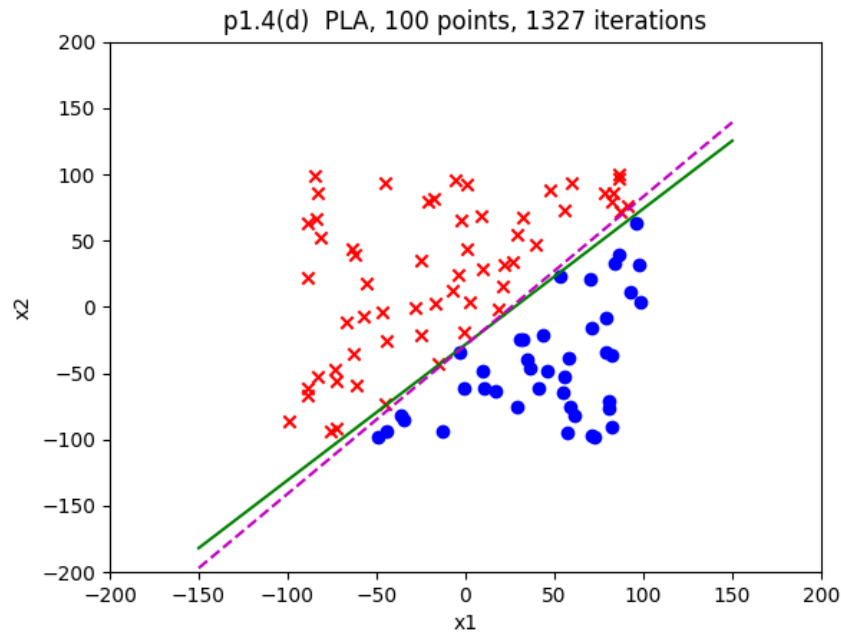
1  #!/usr/bin/env python
2  ## problem_1_4_d.py
3  ## Mac Radigan
4
5  """problem_1_4_d.py
6  """
7
8  import argparse
9  from sys import stdout
10 from perceptron import pla
11 import numpy as np
12
13 def main(verbose):
14     'problem 1.4'
15     dim = 100
16     test(dim)
17
18 def case(dim):
19     'problem 1.4'
20     outdir = 'figures/'
21     p = pla.PLA(2)
22     x = p.rand_data(dim)
23     w_star = p.rand_target()

```

```

24     y = p.target(x, w_star)
25     w = p.train(x, y)
26     y_hat = p.classify(x)
27     y_err = y_hat - y;
28     err = 0 != np.sum(y_err)
29     outfile = '%s/%s' % (outdir, 'p1.4d.png')
30     title = 'p1.4(d) PLA, %d points, %d iterations' % (dim, p.t)
31     ax = p.plot(x, w, w_star, y_hat, title, outfile)
32     return (p, x, w_star, y, w, y_hat, y_err, err)
33
34 def prn(name, value):
35     print('%s:\n%s\n' % (name, value))
36
37 def test(dim):
38     'test'
39     (p, x, w_star, y, w, y_hat, y_err, err) = case(dim)
40     prn("x", x)
41     prn("w_star", w_star)
42     prn("y'", y.T)
43     prn("w", w)
44     prn("y_hat'", y_hat.T)
45     #prn("y_err'", y_err.T)
46     prn("err'", err)
47
48 if __name__ == '__main__':
49     parser = argparse.ArgumentParser(description='problem 1.4 (a)')
50     parser.add_argument('-v', '--verbose', action='store_true',
51                         dest='verbose', default=False, help='verbose output to stdout')
52     args = parser.parse_args()
53     main(args.verbose)
54     ## *EOF*

```



- (e) Repeat everything in (b) with another randomly generated data set of size 1,000. Compare your results with (b).

```

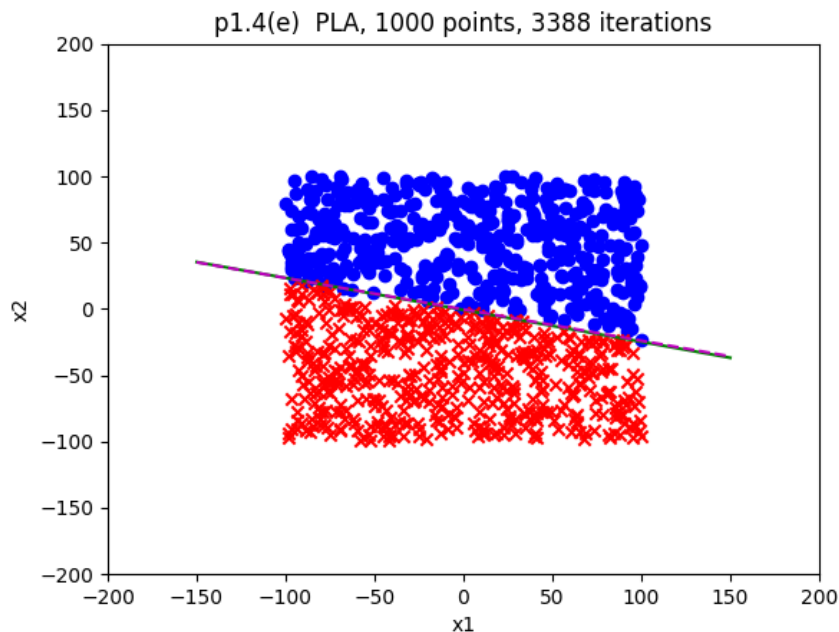
1  #!/usr/bin/env python
2  ## problem_1_4_d.py
3  ## Mac Radigan
4
5  """problem_1_4_d.py
6  """
7
8  import argparse
9  from sys import stdout
10 from perceptron import pla
11 import numpy as np
12
13 def main(verbose):
14     'problem 1.4'
15     dim = 100
16     test(dim)
17
18 def case(dim):
19     'problem 1.4'
20     outdir = 'figures/'
21     p = pla.PLA(2)
22     x = p.rand_data(dim)
23     w_star = p.rand_target()

```

```

24     y = p.target(x, w_star)
25     w = p.train(x, y)
26     y_hat = p.classify(x)
27     y_err = y_hat - y;
28     err = 0 != np.sum(y_err)
29     outfile = '%s/%s' % (outdir, 'p1.4d.png')
30     title = 'p1.4(d) PLA, %d points, %d iterations' % (dim, p.t)
31     ax = p.plot(x, w, w_star, y_hat, title, outfile)
32     return (p, x, w_star, y, w, y_hat, y_err, err)
33
34 def prn(name, value):
35     print('%s:\n%s\n' % (name, value))
36
37 def test(dim):
38     'test'
39     (p, x, w_star, y, w, y_hat, y_err, err) = case(dim)
40     prn("x", x)
41     prn("w_star", w_star)
42     prn("y'", y.T)
43     prn("w", w)
44     prn("y_hat'", y_hat.T)
45     #prn("y_err'", y_err.T)
46     prn("err'", err)
47
48 if __name__ == '__main__':
49     parser = argparse.ArgumentParser(description='problem 1.4 (a)')
50     parser.add_argument('-v', '--verbose', action='store_true',
51                         dest='verbose', default=False, help='verbose output to stdout')
52     args = parser.parse_args()
53     main(args.verbose)
54     ## *EOF*

```



- (f) Modify the algorithm such that it takes $x_n \in \mathbb{R}^n$ instead of \mathbb{R}^2 . Randomly generate a linearly separable data set of size 1,000 with $x_n \in \mathbb{R}^{10}$ and feed the data set to the algorithm. How many updates does the algorithm take to converge?
- (g) Repeat the algorithm on the same data set as (f) for 100 experiments. In the iterations of each experiment, pick $x(t)$ randomly instead of deterministically. Plot a histogram of the number of updates that the algorithm takes to converge.
- (h) Summarize your conclusions with respect to accuracy and running time as a function of N and d .

Problem 1.8

The Hoeffding Inequality is one form of the law of large numbers. One of the simplest forms of that law is the Chebyshev Inequality, which you will prove here.

- (a) If t is a non negative random variable, prove that for any $\alpha > 0$, $P[t \geq \alpha] \leq \frac{E(t)}{\alpha}$

By definition,

$$P[t \geq \alpha] \triangleq \int_{\alpha}^{\infty} p(x)dx$$

and

$$E(t) \triangleq \int_{-\infty}^{\infty} xp(x)dx$$

so evaluate

$$\alpha \int_{\alpha}^{\infty} p(x)dx = \alpha P[t \geq \alpha] \leq E(t) = \int_{-\infty}^{\infty} xp(x)dx$$

since t is strictly positive, this can be written as

$$\int_{\alpha}^{\infty} \alpha p(x)dx \geq \int_0^{\alpha} xp(x)dx + \int_{\alpha}^{\infty} xp(x)dx$$

$$\text{note that } \int_{\alpha}^{\infty} \alpha p(x)dx \geq \int_{\alpha}^{\infty} xp(x)dx$$

$$\text{and because } t > 0, \int_0^{\alpha} xp(x)dx \geq 0$$

$$\text{so it holds that } \alpha \int_{\alpha}^{\infty} p(x)dx = \int_{-\infty}^{\infty} xp(x)dx$$

$$\text{and thus } P[t \geq \alpha] \leq \frac{E(t)}{\alpha}$$

- (b) If u is any random variable with mean μ and variance σ^2 , prove that for any $\alpha > 0$, $P[(u - \mu)^2 \geq \alpha] \leq \frac{\sigma^2}{\alpha}$. [Hint: Use (a)]

By definition,

$$\sigma^2 \triangleq E[(u - \mu)^2]$$

and thus with substitution of $t := (u - \mu)^2$ and using (a), we have

$$P[(u - \mu)^2 \geq \alpha] \leq \frac{\sigma^2}{\alpha}$$

- (c) If u_1, \dots, u_N are iid random variables, each with mean μ and variance σ^2 , and $u = \frac{1}{N} \sum_{n=1}^N u_n$, prove that for any $\alpha > 0$, $P[(u - \mu)^2 \geq \alpha] \leq \frac{\sigma^2}{N\alpha}$.

Notice that the RHS of this Chebyshev Inequality goes down linearly in N , while the counterpart in Hoeffding's Inequality goes down exponentially. In Problem 1.9, we develop an exponential bound using a similar approach.

Chapter 2 Notes

	Symbol	Name	
	$h(x_k) \exists x_k \in (X)$	Dichotomy	
	N	Number of Data Points	
	$m_{\mathcal{H}}(N)$	Growth Function	
	$m_{\mathcal{H}}(N)$	Growth Function	
	$B(N, k)$	Growth Bound with Breakpoint k	
v	d_{vc}	Vapnik-Chervonenkis Dimension	24

Generalization Error

$$E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{1}{2N} \ln \frac{2M}{\delta}} \quad (9)$$

Growth Function

$$m_{\mathcal{H}}(N) = \max_{\vec{x} \in \mathcal{X}} |\mathcal{H}(\vec{x})| \quad (10)$$

Bounding the Growth Function

$B(N, k)$ is the maximum number of dichotomies on N points such that no subset of size k of the N points can be shattered by these dichotomies.

Vapnik-Chervonenkis Dimension

$$d_{VC}(\mathcal{H}) = \begin{cases} \max N \text{ s.t. } m_{\mathcal{H}}(N) = 2^N & \text{if } m_{\mathcal{H}}(N) < 2^N \exists N \\ \infty & \text{if } m_{\mathcal{H}}(N) = 2^N \forall N \end{cases} \quad (11)$$

$k = d_{VC} + 1$ is a breakpoint for $m_{\mathcal{H}}(N)$

$$m_{\mathcal{H}}(N) \leq \sum_{i=0}^{d_{VC}} \binom{N}{i}$$

$$m_{\mathcal{H}}(N) \leq N^{d_{VC}} + 1$$

Sauer's Lemma

$$B(N, k) \leq \sum_{i=0}^{k-1} \binom{N}{i} \quad (12)$$

Problem 2.1

In Equation (2.1), set $\delta = 0.03$ and let

$$\sqrt{\frac{1}{2N} \ln \frac{2M}{\delta}} \leq \epsilon \implies 2N \geq \frac{\ln \frac{2M}{\delta}}{\epsilon^2} \implies N \geq \frac{1}{2\epsilon^2} \ln \frac{2M}{\delta}$$

(a) For $M = 1$, how many examples do we need to make $\epsilon \leq 0.05$?

$$N \geq \frac{1}{2(0.05)^2} \ln \frac{2 \cdot 1}{0.03}$$

(b) For $M = 100$, how many examples do we need to make $\epsilon \leq 0.05$?

$$N \geq \frac{1}{2(0.05)^2} \ln \frac{2 \cdot 100}{0.03}$$

(c) For $M = 10,000$, how many examples do we need to make $\epsilon \leq 0.05$?

$$N \geq \frac{1}{2(0.05)^2} \ln \frac{2 \cdot 10,000}{0.03}$$

Problem 2.3

Compute the maximum number of dichotomies, $m_{\mathcal{H}}(N)$, for these learning models, and consequently compute d_{VC} , the VC dimension.

(a) Positive or negative ray: \mathcal{H} contains the functions which are +1 on $[a, \infty)$ (for some a), together with those that are +1 on $(-\infty, a]$ (for some a).

$$m_{\mathcal{H}}(N) = 2N + 1 \text{ dichotomies}$$

(b) Positive or negative interval. \mathcal{H} contains the functions which are +1 on an interval $[a, b]$ and -1 elsewhere, or -1 on an interval $[a, b]$ and +1 elsewhere.

$$m_{\mathcal{H}}(N) = \binom{N+1}{2} + 1 = N^2 + N + 1 \text{ dichotomies}$$

Problem 2.8

Which of the following are possible growth functions $m_{\mathcal{H}}(N)$ for some hypothesis set:

(a) $1 + N$

yes, example positive ray

(b) $1 + N + \frac{N(N-1)}{2}$

no, must be integer-valued (cannot have a fractional number of dichotomies)

(c) 2^N

yes

(d) $2^{\lfloor N \rfloor}$

yes

(e) $2^{\lfloor \sqrt{N} \rfloor}$

yes

(f) $2^{\lfloor \frac{N}{2} \rfloor}$

yes

(g) $1 + N + \frac{N(N-1)(N-2)}{6}$

$$1 + N + \frac{N(N-1)(N-2)}{6} = \frac{N^3}{6} + \frac{N}{2} + \frac{4}{3}$$

no, must be integer-valued (cannot have a fractional number of dichotomies)

Problem 2.13

- (a) Let $\mathcal{H} = \{h_1, h_2, \dots, h_M\}$. Prove that $d_{VC}(\mathcal{H}) \leq \log_2 M$.

Finite \mathcal{H} can represent at most $|\mathcal{H}| = M$ dichotomies. Thus, $2^{v_{DC}} \leq M$, or $v_{DC} \leq \log_2 M$.

- (b) For hypothesis sets $\mathcal{H}_1, \mathcal{H}_2, \dots, \mathcal{H}_K$ with finite dimensions $d_{VC}(\mathcal{H}_K)$, derive and prove the tightest upper and lower bound that you can get on $d_{VC}\left(\bigcap_{k=1}^K \mathcal{H}_k\right)$.
- (c) For hypothesis sets $\mathcal{H}_1, \mathcal{H}_2, \dots, \mathcal{H}_K$ with finite dimensions $d_{VC}(\mathcal{H}_K)$, derive and prove the tightest upper and lower bound that you can get on $d_{VC}\left(\bigcup_{k=1}^K \mathcal{H}_k\right)$.

Chapter 3 Notes

Problem 3.1

Consider the double semi-circle “toy” learning task below (see figure). There are two semi-circles of width thk with inner radius rad , separated by sep as shown (red is -1, blue is +1). The center of the top semi-circle is linearly separable when $sep \geq 0$, and not so for $sep < 0$. Set $rad = 10$, $thk = 5$, and $sep = 5$. Then, generate 2e3 examples uniformly, which means you will have approximately 1e3 examples for each class.

- (a) Run the PLA starting from $w = 0$ until it converges. Plot the data and final hypothesis.

Each iteration in dashed magenta, final classification in solid green:

```

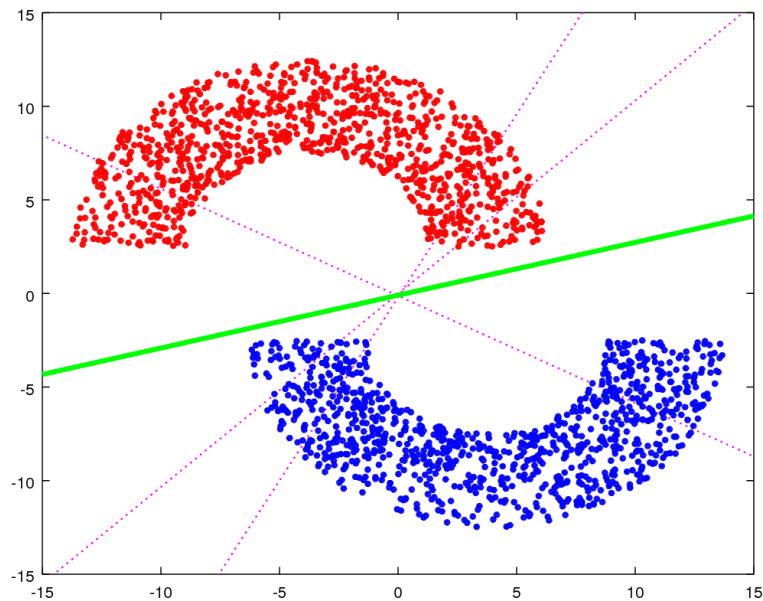
1  #!/usr/bin/env octave
2  ## problem_3_1_a_fast.m
3  ## Mac Radigan
4
5  FORCES__SCRIPT_FILE=1;
6  ux = ~isempty(getenv('LFD'));
7
8  function ax = do_plot(ux, rad, thk, c1, c2, x1, x2, w, k_err)
9
10     if ux
11         show = 'on';
12     else
13         show = 'off';
14     end
15
16     % plot input dataset
17     ax = figure(30111);
18     set(ax, 'visible', show);
19     plot(x1(c1), x2(c1), 'LineWidth', 1, 'r. ');
20     hold on;
21     plot(x1(c2), x2(c2), 'LineWidth', 1, 'b. ');
22     hold on;
23     axis(15 * [-1 +1 -1 +1]);
24
25     % plot final hypothesis, g(x)
26     ext = rad + thk;
27     m = -w(2)/w(3);
28     b = -w(1)/w(3);
29     dx1 = linspace(-ext, ext, 2);
30     dx2 = m * dx1 + b;
31     ax = figure(30111);
32     set(ax, 'visible', show);
33     if k_err > 0
34         plot(dx1, dx2, 'Color', 'magenta', 'LineWidth', 1, 'LineStyle',
35             ↪ ': ');
36     else
37         plot(dx1, dx2, 'Color', 'green', 'LineWidth', 3);
38     end
39     hold on;
40     axis(15 * [-1 +1 -1 +1]);
41
42     drawnow();
43
44 end % function do_plot
45
46 N = 2e3; % number of training samples
47 rad = 10; % radius of semi-circle
48 thk = 5; % thickness of semi-circle
49 sep = 5; % separation between semi-circles

```

```

50  %% uniformly distributed data
51  z = (rad - thk * (1 - rand(1,N))) .* exp(-j*2*pi*rand(1,N));
52
53  %% target function
54  f = @(z) sign(angle(z));
55
56  %% target set
57  y = f(z);
58
59  %% cartesian basis representation
60  x1 = -f(z)*(rad/2-thk/4) + abs(z) .* cos(angle(z));
61  x2 = f(z)*sep/2 + abs(z) .* sin(angle(z));
62
63  %% categorical index
64  c1 = y > 0;
65  c2 = y < 0;
66
67  %% Perceptron Learning Algorithm (PLA)
68  t = 0; % training step counter
69  w = zeros(1,3); % initial weights
70  %w = randn(1,3); % initial weights
71  X = [ones(size(y)); x1; x2]; % training data
72  k_err = inf; % convergence criteria
73  while k_err > 0
74      t = t + 1; % update training step counter
75      ax = do_plot(ux, rad, thk, c1, c2, x1, x2, w, k_err);
76      if ux
77          fprintf(1, 'error: %f\n', k_err);
78          t
79          w
80      end
81      y_hat = sign(w*X); % classify
82      y_err = 0.5*abs(y - y_hat); % residual
83      [k_err, k] = max(y_err); % select misclassified point
84      if k_err <= 0, continue, end
85      x_k = X(:,k); % in the training data
86      y_k = y(k); % in the target set
87      w = w + y_k*x_k'; % update weights
88      if ux
89          disp('...')
90          input('...')
91      end
92  end % training
93
94  ax = do_plot(ux, rad, thk, c1, c2, x1, x2, w, k_err);
95  if ux
96      disp('training done.')
97  else
98      saveas(ax, 'figures/p3.1a_fast.png');
99  end
100
101  ## *EOF*

```



- (b) Repeat part (a) using the linear regression (for classification) to obtain w . Explain your observations.

```

1  #!/usr/bin/env octave
2  ## problem_3_1_b.m
3  ## Mac Radigan
4
5  FORCES__SCRIPT_FILE=1;
6  ux = ~isempty(getenv('LFD'));
7  if ux, graphics_toolkit('qt'); end
8
9  function ax = do_plot(ux, rad, thk, c1, c2, x1, x2, w)
10
11     if ux
12         show = 'on';
13     else
14         show = 'off';
15     end
16
17     %% plot input dataset
18     ax = figure(30112);
19     set(ax, 'visible', show);
20     plot(x1(c1), x2(c1), 'LineWidth', 1, 'r. ');
21     hold on;
22     plot(x1(c2), x2(c2), 'LineWidth', 1, 'b. ');
23     hold on;
24

```

```

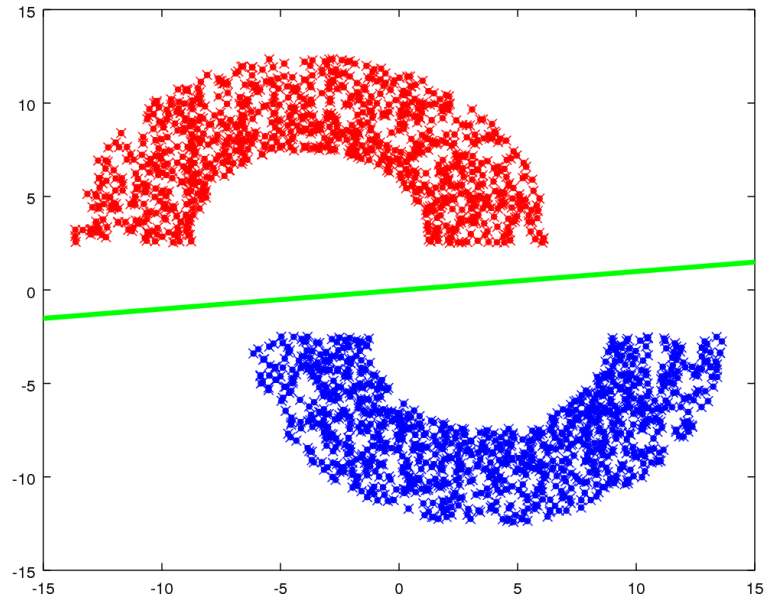
25     %% plot input dataset
26     ax = figure(30112);
27     set(ax, 'visible', show);
28     plot(x1(c1), x2(c1), 'rx');
29     hold on;
30     plot(x1(c2), x2(c2), 'bx');
31     hold on;
32
33     %% plot final hypothesis, g(x)
34     ext = rad + thk;
35     m = -w(2)/w(3);
36     b = -w(1)/w(3);
37     dx1 = linspace(-ext, ext, 2);
38     dx2 = m * dx1 + b;
39     ax = figure(30112);
40     set(ax, 'visible', show);
41     plot(dx1, dx2, 'Color', 'green', 'LineWidth', 3);
42     hold off;
43     axis(15 * [-1 +1 -1 +1]);
44
45     drawnow();
46
47 end % function do_plot
48
49 N = 2e3; % number of training samples
50 rad = 10; % radius of semi-circle
51 thk = 5; % thickness of semi-circle
52 sep = 5; % separation between semi-circles
53
54 %% uniformly distributed data
55 z = (rad - thk * (1 - rand(1,N))) .* exp(-j*2*pi*rand(1,N));
56
57 %% target function
58 f = @(z) sign(angle(z));
59
60 %% target set
61 y = f(z);
62
63 %% cartesian basis representation
64 x1 = -f(z)*(rad/2-thk/4) + abs(z) .* cos(angle(z));
65 x2 = f(z)*sep/2 + abs(z) .* sin(angle(z));
66
67 %% categorical index
68 c1 = y > 0;
69 c2 = y < 0;
70
71 %% Linear Regression
72 X = [ones(size(y)); x1; x2]; % training data
73 w = y * pinv(X);
74
75 %% plot results
76 ax = do_plot(ux, rad, thk, c1, c2, x1, x2, w);
77
78 if ux
79     disp('training done.')
80     input('...')

```

```

81     else
82         saveas(ax, 'figures/p3.1b.png');
83     end
84
85     ## *EOF*

```



Problem 3.2

For the double-semi-circle task in Problem 3.1, vary sep in the range $\{0.2, 0.4, \dots, 0.5\}$. Generate 2,000 examples and run the PLA starting with $\mathbf{w} = \mathbf{0}$. Record the number of iterations PLA takes to converge.

Plot sep versus the number of iterations PLA takes to converge. Explain your observations [Hint: Problem 1.3.]

```

1  #!/usr/bin/env octave
2  ## problem_3_2.m
3  ## Mac Radigan
4
5  FORCES__SCRIPT_FILE=1;
6  ux = ~isempty(getenv('LFD'));
7  if ux, graphics_toolkit('qt'); end
8

```



```

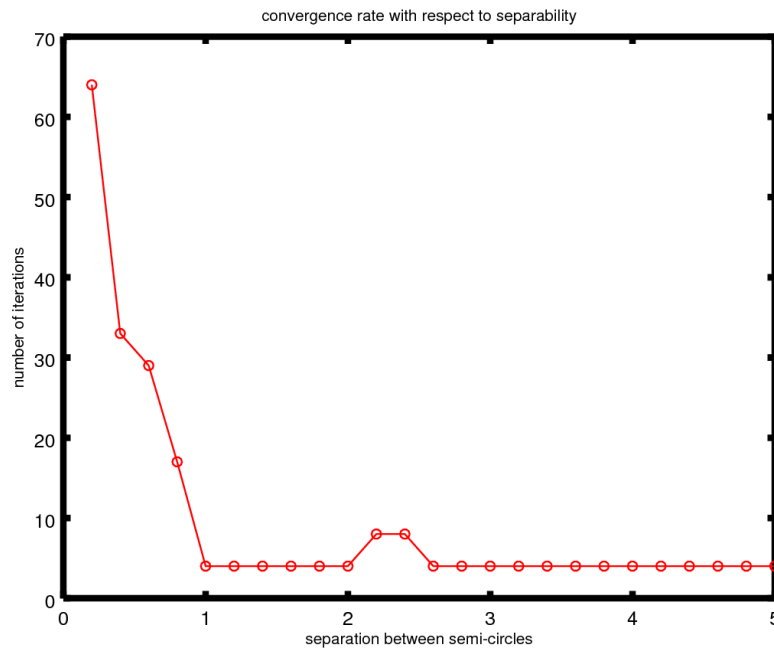
9  function ax = do_plot(ux, seps, ts)
10
11      if ux
12          show = 'on';
13      else
14          show = 'off';
15      end
16
17      %% plot input dataset
18      ax = figure(30211);
19      set(ax, 'visible', show);
20      plot(seps, ts, 'LineWidth', 1, 'ro-');
21      xlabel('separation between semi-circles');
22      ylabel('number of iterations');
23      title('convergence rate with respect to separability');
24      set(gca, 'linewidth', 4, 'fontsize', 12)
25      hold on;
26
27      drawnow();
28
29  end % function do_plot
30
31  function t = do_pla(sep, N, rs, thetas)
32
33      rad = 10;    % radius of semi-circle
34      thk = 5;    % thickness of semi-circle
35
36      %% uniformly distributed data
37      z = (rad - thk * (1 - rs)) .* exp(-j*2*pi*thetas);
38
39      %% target function
40      f = @(z) sign(angle(z));
41
42      %% target set
43      y = f(z);
44
45      %% cartesian basis representation
46      x1 = -f(z)*(rad/2-thk/4) + abs(z) .* cos(angle(z));
47      x2 = f(z)*sep/2 + abs(z) .* sin(angle(z));
48
49      %% categorical index
50      c1 = y > 0;
51      c2 = y < 0;
52
53      %% Perceptron Learning Algorithm (PLA)
54      t = 0; % training step counter
55      w = zeros(1,3); % initial weights
56      %w = randn(1,3); % initial weights
57      X = [ones(size(y)); x1; x2]; % training data
58      k_err = inf; % convergence criteria
59      for n = 1:N
60          x_n = X(:,n);
61          t = t + 1; % update training step counter
62          y_hat = sign(w*X); % classify
63          y_err = 0.5*abs(y - y_hat); % residual
64          [k_err, k] = max(y_err); % select misclassified point

```

```

65     if k_err <= 0
66         break
67     else
68         wg = w;
69     end
70     x_k = X(:,k);           % in the training data
71     y_k = y(k);            % in the target set
72     w = w + y_k*x_k';      % update weights
73 end % training
74
75 end % function do_pla
76
77 N = 2e3;                  % number of training samples
78 rs = rand(1,N);          % radii
79 thetas = rand(1,N);      % angles
80
81 seps = 0.2 : 0.2 : 5; % separation between semi-circles
82
83 f = @(seps) do_pla(seps, N, rs, thetas);
84
85 ts = arrayfun(f, seps, 'UniformOutput', false);
86
87 ax = do_plot(ux, seps, cell2mat(ts));
88 if ux
89     input('...')
90 else
91     saveas(ax, 'figures/p3.2.png');
92 end
93
94 ## *EOF*

```



Problem 3.3

For the double-semi-circle task in Problem 3.1, set $sep = -5$ and generate 2,000 examples.

- (a) What will happen if you run PLA on those examples?

The algorithm will not converge, and will not terminate.

```

1  #!/usr/bin/env octave
2  ## problem_3_3_a.m
3  ## Mac Radigan
4
5  FORCES__SCRIPT_FILE=1;
6  ux = ~isempty(getenv('LFD'));
7  if ux, graphics_toolkit('qt'); end
8
9  function ax = do_plot(ux, rad, thk, c1, c2, x1, x2, w, k_err)
10
11     if ux
12         show = 'on';
13     else
14         show = 'off';
15     end

```

```

16
17     %% plot input dataset
18     ax = figure(30311);
19     set(ax, 'visible', show);
20     plot(x1(c1), x2(c1), 'LineWidth', 1, 'r. ');
21     hold on;
22     plot(x1(c2), x2(c2), 'LineWidth', 1, 'b. ');
23     axis(15 * [-1 +1 -1 +1]);
24     hold on;
25
26     %% plot final hypothesis, g(x)
27     ext = rad + thk;
28     m = -w(2)/max(w(3),eps);
29     b = -w(1)/max(w(3),eps);
30     dx1 = linspace(-ext, ext, 2);
31     dx2 = m * dx1 + b;
32     ax = figure(30111);
33     set(ax, 'visible', show);
34     if k_err > 0
35         plot(dx1, dx2, 'Color', 'magenta', 'LineWidth', 1, 'LineStyle',
↪      'r. ');
36     else
37         plot(dx1, dx2, 'Color', 'green', 'LineWidth', 3);
38     end
39     axis(15 * [-1 +1 -1 +1]);
40     hold on;
41
42     drawnow();
43
44     end % function do_plot
45
46     N = 2e3; % number of training samples
47     rad = 10; % radius of semi-circle
48     thk = 5; % thickness of semi-circle
49     sep = -5; % separation between semi-circles
50
51     %% uniformly distributed data
52     z = (rad - thk * (1 - rand(1,N))) .* exp(-j*2*pi*rand(1,N));
53
54     %% target function
55     f = @(z) sign(angle(z));
56
57     %% target set
58     y = f(z);
59
60     %% cartesian basis representation
61     x1 = -f(z)*(rad/2-thk/4) + abs(z) .* cos(angle(z));
62     x2 = f(z)*sep/2 + abs(z) .* sin(angle(z));
63
64     %% categorical index
65     c1 = y > 0;
66     c2 = y < 0;
67
68     %% Perceptron Learning Algorithm (PLA)
69     t = 0; % training step counter
70     %w = zeros(1,3); % initial weights

```

```

71 w = randn(1,3); % initial weights
72 X = [ones(size(y)); x1; x2]; % training data
73 k_err = inf; % convergence criteria
74 while k_err > 0
75     t = t + 1; % update training step counter
76     y_hat = sign(w*X); % classify
77     y_err = 0.5*abs(y - y_hat); % residual
78     [k_err, k] = max(y_err); % select misclassified point
79     if ux
80         fprintf(1, 'error: %f\n', k_err);
81         t
82         w
83     end
84     if k_err <= 0
85         break
86     else
87         wg = w;
88     end
89     if 0 == mod(t,1)
90         ax = do_plot(ux, rad, thk, c1, c2, x1, x2, w, k_err);
91     endif
92     x_k = X(:,k); % in the training data
93     y_k = y(k); % in the target set
94     w = w + y_k*x_k'; % update weights
95     if ux
96         input('...')
97     end
98 end % training
99 %ax = do_plot(ux, rad, thk, c1, c2, x1, x2, wg, k_err);
100
101 ax = do_plot(ux, rad, thk, c1, c2, x1, x2, w, k_err);
102 if ux
103     disp('training done.')
104     input('...')
105 else
106     saveas(ax, 'figures/p3.3a.png');
107 end
108
109 ## *EOF*

```

- (b) Run the pocket algorithm for 10,000 iterations and plot E_{in} versus the iteration number t .

```

1  #!/usr/bin/env octave
2  ## problem_3_3_b.m
3  ## Mac Radigan
4
5  FORCES_SCRIPT_FILE=1;
6  ux = ~isempty(getenv('LFD'));
7  if ux, graphics_toolkit('qt'); end
8
9  function ax = do_plot(ux, E_in, ts)
10
11     if ux
12         show = 'on';

```

```

13     else
14         show = 'off';
15     end
16
17     %% plot input dataset
18     ax = figure(30321);
19     set(ax, 'visible', show);
20     plot(E_in, ts, 'LineWidth', 1, 'ro-');
21     hold on;
22     set(gca, 'linewidth', 4, 'fontsize', 12)
23     xlabel('in-sample error (E_in)');
24     ylabel('iteration number (t)');
25     title('convergence of pocket algorithm on non-seperable data');
26
27     drawnow();
28
29 end % function do_plot
30
31 N = 10e3; % number of training samples
32 rad = 10; % radius of semi-circle
33 thk = 5; % thickness of semi-circle
34 sep = -5; % separation between semi-circles
35
36 %% uniformly distributed data
37 z = (rad - thk * (1 - rand(1,N))) .* exp(-j*2*pi*rand(1,N));
38
39 %% target function
40 f = @(z) sign(angle(z));
41
42 %% target set
43 y = f(z);
44
45 %% cartesian basis representation
46 x1 = -f(z)*(rad/2-thk/4) + abs(z) .* cos(angle(z));
47 x2 = f(z)*sep/2 + abs(z) .* sin(angle(z));
48
49 %% categorical index
50 c1 = y > 0;
51 c2 = y < 0;
52
53 %% Pocket Algorithm
54 t = 0; % training step counter
55 %w = zeros(1,3); % initial weights
56 w = randn(1,3); % initial weights
57 X = [ones(size(y)); x1; x2]; % training data
58 k_err = inf; % convergence criteria
59 ts = 0:(N-1);
60 for t = ts
61     n = t+1;
62     %% Perceptron Learning Algorithm (PLA)
63     y_hat = sign(w*X(:,n)); % classify
64     y_err = 0.5*abs(y(:,n) - y_hat); % residual
65     [k_err, k] = max(y_err); % select misclassified point
66     x_k = X(:,k); % in the training data
67     y_k = y(k); % in the target set
68     w_g = w + y_k*x_k'; % guess weight

```

```

69     %% check pocket
70     y_hat_g = sign(w_g*X(:,n));           % classify
71     y_err_g = 0.5*abs(y(:,n) - y_hat_g);   % residual
72     [k_err_g, k_g] = max(y_err_g);        % select misclassified point
73     if k_err > k_err_g                     % is better than the pocket weights?
74         w = w_g;                          % ... yes, replace pocket
75     end
76     E_in(t+1) = min(k_err, k_err_g);
77 end
78
79 ax = do_plot(ux, E_in, ts);
80 if ux
81     disp('training done.')
82     input('...')
83 else
84     saveas(ax, 'figures/p3.3b.png');
85 end
86
87 ## *EOF*

```

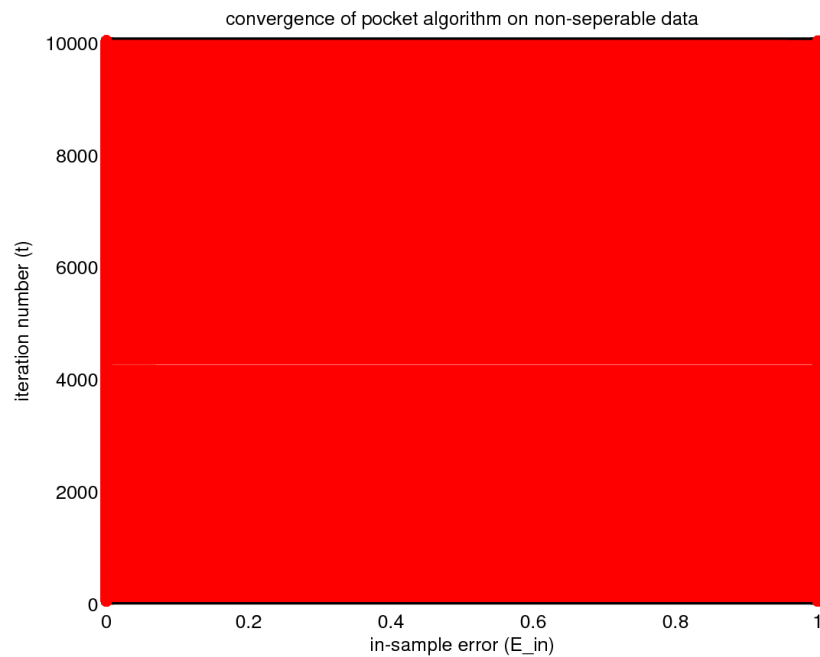


Figure 1: Target

(c) Plot the data and final hypothesis in part (b).

```

1  #!/usr/bin/env octave
2  ## problem_3_3_c.m
3  ## Mac Radigan
4
5  FORCES__SCRIPT_FILE=1;
6  ux = ~isempty(getenv('LFD'));
7  if ux, graphics_toolkit('qt'); end
8
9  function ax = do_plot(ux, rad, thk, c1, c2, x1, x2, w, k_err)
10
11     if ux
12         show = 'on';
13     else
14         show = 'off';
15     end
16
17     %% plot input dataset
18     ax = figure(30331);
19     set(ax, 'visible', show);
20     plot(x1(c1), x2(c1), 'LineWidth', 1, 'r. ');
21     hold on;
22     plot(x1(c2), x2(c2), 'LineWidth', 1, 'b. ');
23     axis(15 * [-1 +1 -1 +1]);
24     hold on;
25
26     %% plot final hypothesis, g(x)
27     ext = rad + thk;
28     m = -w(2)/max(w(3),eps);
29     b = -w(1)/max(w(3),eps);
30     dx1 = linspace(-ext, ext, 2);
31     dx2 = m * dx1 + b;
32     ax = figure(30331);
33     set(ax, 'visible', show);
34     plot(dx1, dx2, 'Color', 'green', 'LineWidth', 3);
35     axis(15 * [-1 +1 -1 +1]);
36     hold on;
37
38     drawnow();
39
40 end % function do_plot
41
42 N = 10e3; % number of training samples
43 rad = 10; % radius of semi-circle
44 thk = 5; % thickness of semi-circle
45 sep = -5; % separation between semi-circles
46
47 %% uniformly distributed data
48 z = (rad - thk * (1 - rand(1,N))) .* exp(-j*2*pi*rand(1,N));
49
50 %% target function
51 f = @(z) sign(angle(z));
52
53 %% target set
54 y = f(z);
55
56 %% cartesian basis representation

```



```

57 x1 = -f(z)*(rad/2-thk/4) + abs(z) .* cos(angle(z));
58 x2 = f(z)*sep/2 + abs(z) .* sin(angle(z));
59
60 %% categorical index
61 c1 = y > 0;
62 c2 = y < 0;
63
64 %% Pocket Algorithm
65 t = 0; % training step counter
66 %w = zeros(1,3); % initial weights
67 w = randn(1,3); % initial weights
68 X = [ones(size(y)); x1; x2]; % training data
69 k_err = inf; % convergence criteria
70 for t = 0:(N-1)
71     %% Perceptron Learning Algorithm (PLA)
72     y_hat = sign(w*X); % classify
73     y_err = 0.5*abs(y - y_hat); % residual
74     [k_err, k] = max(y_err); % select misclassified point
75     x_k = X(:,k); % in the training data
76     y_k = y(k); % in the target set
77     w_g = w + y_k*x_k'; % guess weight
78     %% check pocket
79     y_hat_g = sign(w_g*X); % classify
80     y_err_g = 0.5*abs(y - y_hat_g); % residual
81     [k_err_g, k_g] = max(y_err_g); % select misclassified point
82     if k_err_g < k_err % is better than the pocket weights?
83         w = w_g; % ... yes, replace pocket
84     end
85 end
86
87 ax = do_plot(ux, rad, thk, c1, c2, x1, x2, w, k_err);
88 if ux
89     disp('training done.')
90     input('...')
91 else
92     saveas(ax, 'figures/p3.3c.png');
93 end
94
95 ## *EOF*

```

Problem 3.10

- (b) Show that the trace of a symmetric matrix equals the sum of its eigen-values. [Hint: Use the spectral theorem and the cyclic property of the trace. Note that the same result holds for non-symmetric matrices, but is a little harder to prove.]

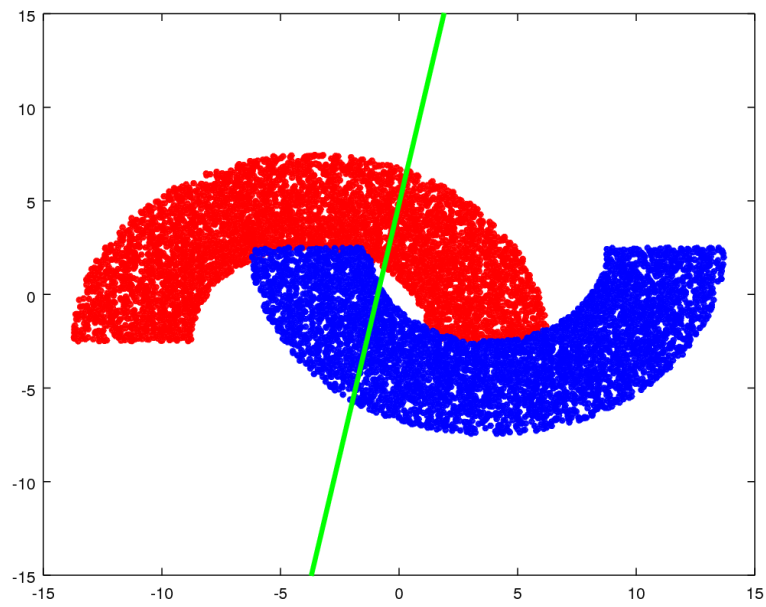


Figure 2: Target

Appendix A: Cayley-Hamilton Theorem

The following excerpt is from **Matrix Theory, From Generalized Inverses to Jordan Forms**, by Robert Piziak and P. L. Odell.

In 1949, **J. Sutherland Frame** (24 December 1907 - 27 February 1997) published an abstract in the *Bulletin of the American Mathematical Society* indicating a recursive algorithm for computing the inverse of a matrix and, as a by-product, getting additional information, including the famous Cayley-Hamilton theorem. (Hamilton is the Irish mathematician **William Rowan Hamilton** (4 August 1805 - 2 September 1895), and Cayley is **Aurthur Cayley** (16 August 1821 - 26 January 1895).) We have not been able to find an actual paper with a detailed account of these claims. Perhaps the author thought the abstract sufficient and went on with his work in group representations. Perhaps he was told this algorithm had been rediscovered many times (see [House, 1964, p. 72]). Whatever the case, in this section, we will expand on and expose the details of Frame's algorithm. Suppose $A \in \mathbf{C}^{n \times n}$. The *characteristic matrix* of A is $xI - A \in \mathbf{C}[x]^{n \times n}$, the collection of n -by- n matrices with polynomial entries. We must open our minds to accepting matrices with polynomial entries. For example, $\begin{bmatrix} x^2 + 1 & x - 3 \\ 4x + 2 & x^3 - 7 \end{bmatrix} \in \mathbf{C}[x]^{2 \times 2}$. Determinants work just fine for these kinds of matrices. The determinant of $xI - A$, $\det(xI - A) \in \mathbf{C}[x]$, the polynomials in x , and is what we call the *characteristic polynomial* of A :

$$\mathcal{X}_A(x) = \det(xI - A) = x^n + c_1x^{n-1} + \cdots + c_{n-1}x + c_n$$

For example, if $A = \begin{bmatrix} 1 & 2 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} \in \mathbf{C}[x]^{3 \times 3}$, then $xI_3 - A =$

$$\begin{bmatrix} x-1 & -2 & -2 \\ -3 & x-4 & -5 \\ -6 & -7 & x-8 \end{bmatrix} \in \mathbf{C}[x]^{3 \times 3}.$$

Thus $\mathcal{X}_A(x) = \begin{vmatrix} x-1 & -2 & -2 \\ -3 & x-4 & -5 \\ -6 & -7 & x-8 \end{vmatrix} = x^3 - 13x^2 - 9x - 3$. This is computed using the usual familiar rules for expanding a determinant.

You may recall that the roots of the characteristic polynomial are quite important, being the *eigenvalues* of the matrix. We will turn to this topic later. For now, we focus on the coefficients of the characteristic polynomial.

First, we consider the constant term c_n . You may already know the answer here, but let's make an argument. Now $\det(A) = (-1)^n \det(-1) = (-1)^n \det(0I - A) = (-1)^n (X)_A(0) = (-1)^n c_n$. Therefore,

$$\det(A) = (-1)^n c_n.$$

As a consequence, we see immediately that A is invertible iff $c_n \neq 0$, in which case

$$A^{-1} = \frac{(-1)^n}{c_n} \text{adj}(A),$$

where $\text{adj}(A)$ is the adjugate matrix of A introduced previously. Also recall the important relationship, $\text{Badj}(B) = \det(B)I$. We conclude that

$$(xI - A) \text{adj}(xI - A) = \chi_A(x)I$$

To illustrate with the example above,

$$\begin{aligned} (xI - A) &= \begin{bmatrix} x-1 & -2 & -2 \\ -3 & x-4 & -5 \\ -6 & -7 & x-8 \end{bmatrix} \cdot \begin{bmatrix} x^2-12x-3 & 2x-2 & 2x+2 \\ 3x+6 & x^2-9x-4 & 5x+1 \\ 6x-3 & 7x+5 & x^2-5x-2 \end{bmatrix} \\ &= \begin{bmatrix} x^3-13x^2-9x-3 & 0 & 0 \\ 0 & x^3-13x^2-9x-3 & 0 \\ 0 & 0 & x^3-13x^2-9x-3 \end{bmatrix} \\ &= x^3-13x^2-9x-3 \cdots \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

Next, let $C(x) = \text{adj}(xI - A) \in \mathbf{C}[x]^{n \times n}$. We note that the elements of $\text{adj}(xI - A)$ are computed as $(n-1) \times (n-1)$ subdeterminants of $(xI - A)$, so the highest power that can occur in $C(x)$ is x^{n-1} . Also, note that we can identify $\mathbf{C}[x]^{n \times n}$, the n -by- n matrices with polynomial entries with $\mathbf{C}^{n \times n}[x]$, the polynomials with matrix coefficients, so we can view $C(x)$ as a polynomial in x with scalar matrices as coefficients. For example,

$$\begin{aligned} &\begin{bmatrix} x^2+1 & x-3 \\ 4x+2 & x^3-7 \end{bmatrix} = \\ &\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} x^3 + \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} x^2 + \begin{bmatrix} 0 & 1 \\ 4 & 0 \end{bmatrix} x + \begin{bmatrix} 1 & -3 \\ 2 & -7 \end{bmatrix}. \end{aligned}$$

All you do is gather the coefficients of each power of x and make a matrix of scalar coefficients of that power of x . Note that what we have thusly created is an element of $\mathbf{C}[x]^{n \times n}$, the polynomials in x whose coefficients come form the n -by- n matrices over \mathbf{C} . Also note, $xB = Bx$ for all $B \in \mathbf{C}[x]^{n \times n}$, so it does not matter which side we put the x on. We now view $C(x)$ as such an expression in $\mathbf{C}^{n \times n}[x]$:

$$C(x) = B_0x^{n-1} + B_1x^{n-2} + \cdots + B_{n-2}x + B_{n-1}.$$

These coefficient matrices turn out to be of interest. For example, $\text{adj}(A) = (-1)^{n-1} \text{adj}(-A) = (-1)^{n-1} C(0) = (-1)^{n-1} B_{n-1}$, so

$$\text{adj}(A) = (-1)^{n-1} B_{n-1}.$$

Thus, if $c_n \neq 0$, A is invertible and we have

$$A^{-1} = \frac{-1}{c_n} B_{n-1}.$$

But now we compute

$$\begin{aligned} (xI - A) \text{adj}(xI - A) &= (xI - A) C(x) \\ &= (xI - A) (B_0x^{n-1} + B_1x^{n-2} + \cdots + B_{n-2}x + B_{n-1}) \\ &= x^n B_0 + x^{n-1} (B_1 - AB_0) + x^{n-2} (B_2 - AB_1) + \cdots \\ &\quad + x (B_{n-1} - AB_{n-2}) - AB_{n-1} \\ &= x^n I + x^{n-1} c_1 I + \cdots + x c_{n-1} I + c_n I \end{aligned}$$

and we compare coefficients using the following table:

Compare Coefficients	Multiply by	on the Left	on the Right
$B_0 = I$	A^n	$A^n B_0$	A^n
$B_1 - AB_0 = C_1 I$	A^{n-1}	$A^{n-1} B_1 - A^n B_0$	$c_1 A^{n-1}$
$B_2 - AB_1 = C_2 I$	A^{n-2}	$A^{n-2} B_2 - A^{n-1} B_1$	$c_2 A^{n-2}$
\vdots			
$B_k - AB_{k-1} = c_k I$			
\vdots			
$B_{n-2} - AB_{n-3} = C_{n-2} I$	A^2	$A^2 B_{n-2} - A^3 B_{n-3}$	$c_{n-2} A^2$
$B_{n-1} - AB_{n-2} = C_{n-1} I$	A	$AB_{n-1} - A^2 B_{n-2}$	$c_{n-1} A$
$-AB_{n-1} = C_n I$		$-AB_{n-1}$	$c_n I$
column	sum =	$\mathbf{O} =$	$\mathcal{X}_A(A)$

So, the first consequence we get from these observations is that the Cayley-Hamilton theorem just falls out as an easy consequence. (Actually, Liebler [2003] reports that Cayley and Hamilton only established the result for matrices up to size 4-by-4. He says it was Frobenius (**Ferdinand Georg Frobenius** [26 October 1849 - 3 August 1917] who gave the first complete proof in 1878.)

THEOREM 2.18 (Cayley-Hamilton theorem)

For any n -by- n matrix A over \mathbf{C} , $\mathcal{X}_A(A) = \mathbf{O}$.

What we are doing in the Cayley-Hamilton theorem is plugging a matrix into a polynomial. Plugging numbers into a polynomial seems reasonable, almost inevitable, but matrices? Given a polynomial

$$p(A) = 4I + 3A - 9A^3 = 4 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + 3 \begin{bmatrix} 1 & 2 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} - 9 \begin{bmatrix} 1 & 2 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}^3 = \begin{bmatrix} -2324 & -2964 & -3432 \\ -5499 & -7004 & -8112 \\ -9243 & -11778 & -13634 \end{bmatrix},$$

The Cayley-Hamilton theorem says that any square matrix is a “root” of its characteristic polynomial.

But there is much more information packed in those equations on the left of the table, so let’s push a little harder. Notice we can rewrite these equations as

$$\begin{aligned} B_0 &= I \\ B_1 &= AB_0 + c_1 I \\ B_2 &= AB_1 + c_2 I \\ &\vdots \\ B_{n-1} &= AB_{n-2} + c_{n-1} I \\ \mathbf{O} &= AB_{n-1} + c_n I. \end{aligned}$$

By setting $B_n := \mathbf{O}$, we have the following recursive scheme clear from above: for $k = 1, 2, \dots, n$,

$$\begin{aligned} B_0 &= I \\ B_k &= AB_{k-1} + c_k I. \end{aligned}$$

In other words, the matrix coefficients, the B_k s are given recursively in terms of the B_{k-1} s and c_k s. If we can get a formula for the B_k and c_k in terms of B_{k-1} , we will get a complete set of recurrence formulas for the B_k and c_k . In particular, if we know B_{n-1} and c_n , we have A^{-1} , provided, of course, A^{-1} exists (i.e. provided $c_n \neq 0$). For this, let’s exploit the recursion given below:

$$\begin{aligned} B_0 &= I \\ B_1 &= AB_0 + c_1 I = AI + c_1 I = A + c_1 I \\ B_2 &= AB_1 + c_2 I = A(A + c_1 I) + c_2 I = A^2 + c_1 A + c_2 I \\ B_3 &= A^3 + c_1 A^2 + c_2 A + c_3 I \\ &\vdots \end{aligned}$$

Inductively, we see for $k = 1, 2, \dots, n$,

$$B_k = A^k + c_1 A^{k-1} + \dots + c_{k-1} A + c_k I.$$

Indeed, when $k = n$, this is just the Cayley-Hamilton theorem all over again. Now we have for $k = 2, 3, \dots, n+1$,

$$B_{k-1} = A^{k-1} + c_1 A^{k-2} + \dots + c_{k-2} A + c_{k-1} I.$$

If we multiply through by A , we get for $k = 2, 3, \dots, n+1$,

$$AB_{k-1} = A^k + c_1 A^{k-1} + \dots + c_{k-2} A^2 + c_{k-1} A.$$

Now we pull a trick out of the mathematician's hat. Take the trace of both sides of the equation using linearity of the trace functional.

$$\text{tr}(AB_{k-1}) = \text{tr}(A^k) + \text{tr}(c_1 A^{k-1}) + \dots + \text{tr}(c_{k-2} A^2) + \text{tr}(c_{k-1} A).$$

for $k = 2, 3, \dots, n+1$.

Why would anybody think to do such a thing? Well, the appearance of the coefficients of the characteristic polynomial on the right is very suggestive. Those who know a little matrix theory realize that the trace of A^r is the sum of the r^{th} powers of the roots of the characteristic polynomial and so Newton's identities leap to mind. Let s_r denote the sum of the r^{th} powers of roots of the characteristic polynomial. Thus, for $k = 2, 3, \dots, n+1$,

$$\text{tr}(AB_{k-1}) = s_k + c_1 s_{k-1} + \dots + c_{k-2} s_2 + c_{k-1} s_1.$$

Digression on Newton's Identities

Newton's identities go back away. They relate the sums of powers of the roots of a polynomial recursively to the coefficients of the polynomial. Many proofs are available. Some involve the algebra of symmetric functions, but we do not want to take the time to go there. Instead, we will use a calculus-based argument following the ideas of [Eidswick 1968]. First, we need to recall some facts about polynomials. Let $p(x) = a_0 + a_1 x + \dots + a_n x^n$. Then the coefficients of p can be expressed in terms of derivatives of p evaluated at zero (remember Taylor polynomials?):

$$p(x) = p(0) + p'(0)x + \frac{p''(0)}{2!}x^2 + \dots + \frac{p^{(n)}(0)}{n!}x^n.$$

Now here is something really slick. Let's illustrate a general fact. Suppose $p(x) = (x-1)(x-2)(x-3) = -6 + 11x - 6x^2 + x^3$. Do a wild and crazy thing. Reverse the rolls [*sic roles*] of the coefficients and form the new reversed polynomial $q(x) = -6x^3 + 11x^2 - 6x + 1$. Clearly $q(1) = 0$ but, more amazingly, $q(\frac{1}{2}) = -\frac{6}{8} + \frac{11}{4} - \frac{6}{2} + 1 = \frac{-6+22-24+8}{8} = 0$. You can also check $q(\frac{1}{3}) = 0$. So the reversed polynomial has as roots the reciprocals of the roots of the original polynomial. Of course, the roots are not zero for this to work. This fact is generally true. Suppose $p(x) = a_0 + a_1x + \dots + a_nx^n$ and the reversed polynomial is $q(x) = a_n + a_{n-1}x + \dots + a_0x^n$. Note

$$q(0) = a_n, q'(0) = a_{n-1}, \dots, \frac{q^{(n)}(0)}{n!} = a_0.$$

Then $r \neq 0$ is a root of p iff $\frac{1}{r}$ is a root of q .

Suppose $p(x) = a_0 + a_1x + \dots + a_nx^n = a_n(x-r_1)(x-r_2)\dots(x-r_n)$. The r_i s are, of course, the roots of p , which we assume to be nonzero but not necessarily distinct. Then the reversed polynomial $q(x) = a_n + a_{n-1}x + \dots + a_0x^n = a_0\left(x - \frac{1}{r_1}\right)\left(x - \frac{1}{r_2}\right)\dots\left(x - \frac{1}{r_n}\right)$. For the sake of illustration, suppose $n = 3$.

Then form

$$\begin{aligned} f(x) &= \frac{q'(x)}{q(x)} = \frac{(x-r_1^{-1})[(x-r_2^{-1}) + (x-r_3^{-1})][(x-r_2^{-1}) + (x-r_3^{-1})]}{(r-r_1)(r-r_2)(r-r_3)} \\ &= \frac{1}{x-r_1^{-1}} + \frac{1}{x-r_2^{-1}} + \frac{1}{x-r_3^{-1}} \end{aligned}$$

. Generally then,

$$f(x) = \sum_{k=1}^n \frac{1}{(x-r_k^{-1})}.$$

Let's introduce more notation. Let $s_m = \sum_{k=1}^m$ for $m = 1, 2, 3, \dots$. Thus, s_m is the sum of the m^{th} powers of roots of p . The derivatives of f are intimately related to the s s. Basi differentiation yields

$$\begin{aligned} f(0) &= -s_1 \\ f'(x) &= \sum_{k=1}^n \frac{-1}{(x-r_k^{-1})^2} & f'(0) &= -s_2 \\ f''(x) &= \sum_{k=1}^n \frac{-2}{(x-r_k^{-1})^3} & f''(0) &= -2s_3 \\ &\vdots \\ f^{(k)}(x) &= \sum_{k=1}^n \frac{-k!}{(x-r_k^{-1})^{k+1}} & f^{(k)}(0) &= -k!s_{k+1} \end{aligned}$$

The last piece of the puzzle is the rule of taking the derivative of a product; this is the so-called *Leibnitz rule* for differentiating a product:

$$D^n(F(x)G(x)) = \sum_{j=0}^n \binom{n}{j} F^{(j)}(x) G^{(n-j)}(x).$$

All right, let's do the argument. We have $f(x) = \frac{q'(x)}{q(x)}$, so $q'(x) = f(x)q(x)$.

Therefore, using Leibnitz rule

$$q^{(m)}(x) = [f(x)q(x)]^{(m-1)} = \sum_{k=0}^{m-1} \binom{m-1}{k} f^{(k)}(x) q^{(m-1-k)}(x).$$

Plugging in zero, we get

$$\begin{aligned} q^{(m)}(0) &= \sum_{k=0}^{m-1} \binom{m-1}{k} f^{(k)}(0) q^{(m-1-k)}(0) \\ &= \sum_{k=0}^{m-1} \frac{(m-1)!}{k! (m-1-k)!} (-k!) s_{k+1} q^{(m-1-k)}(0). \end{aligned}$$

Therefore,

$$\frac{q^{(m)}(0)}{m!} = -\frac{1}{m} \sum_{k=0}^{m-1} \frac{q^{(m-1-k)}(0)}{(m-1-k)!} s_{k+1}.$$

One more substitution and we have the *Newton identities*

$$\begin{aligned} 0 &= m a_{n-m} = \sum_{k=0}^{m-1} a_{n-m+k+1} s_{k+1} \text{ if } 1 \leq m \leq n \\ 0 &= \sum_{k=m-n-1}^{m-1} a_{n-m+k+1} s_{k+1} \text{ if } m > n. \end{aligned}$$