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

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

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

Hoeffding Inequality

$$P\left[|\nu - \mu| > \epsilon\right] \leq 2e^{-2\epsilon^2 N} \forall \epsilon > 0 \tag{1}$$

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

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

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

In-sample Error

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

Out-of-sample Error

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

Chapter 1 Additional Notes

Markov's Inequality

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

Proof:

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

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 Indenpendent, Identically Distributed (*i.i.d.*) random variables states:

$\frac{1}{N}\Sigma_{k=1}^{N} X_k \to E\left(N\right)$ for sufficiently large N

The Asymptotic Equipartition Property (AEP) states:

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

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

$$H\left(X\right) = -\sum_{x} p\left(x\right)\log_2 p\left(x\right) \tag{7}$$

Asymptotic Equipartition Property (AEP)

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

$$\frac{1}{N}\log_2 p\left(X_1, X_2, \cdots, X_N\right) \to H\left(X\right) \text{ in probability} \tag{8}$$

Proof:

$$\begin{aligned}\frac{1}{N}\log_2 p\left(X_1, X_2, \cdots, X_N\right) &= -\frac{1}{N}\sum_{k}\log_2 p\left(X_k\right)\\ &\to -E\log_2 p\left(X\right)\\ &\to H\left(X\right)\end{aligned}$$

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

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

$$2^{-N(H(X)+\epsilon)} \leq p\left(x_1, x_2, \cdots, x_n\right) \leq 2^{-N(H(X)-\epsilon)}$$

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: $P[AandB] = P[A|B]P[B] = P[B|A]P[A]$.]

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

- 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

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

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

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

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

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

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

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

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

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

$P(B_B|k_1) = \frac{P(k_1|B_B)P(B_B)}{P(k_1)} = \frac{\frac{1}{4} \cdot \frac{1}{2}}{\frac{3}{4}} = \frac{\frac{1}{8}}{\frac{3}{4}} = \frac{4}{24} = \frac{1}{6}$

$P(k_2) = P(k_1|B_B)P(k_B) + P(B_B|k_1)P(k_B) = \frac{1}{3} \cdot 1 + \frac{1}{6} \cdot \frac{1}{2} = \frac{1}{3} + \frac{1}{12} + \frac{4}{12} + \frac{1}{12} = \frac{5}{12}$

Problem 1.2

Consider the perceptron in two dimensions: $h(x) = sign(w^\intercal x)$ 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?

$w^\intercal x = 0$

$\implies w_0x_0 + w_1 + x_1 + w_2x_2 = 0$

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

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

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

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

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

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

So by definition,

$w^{*\intercal} x_n > 0 \; \forall n$ s.t. $y_n = +1$

and

$w^{*\intercal} x_n < 0 \; \forall n$ s.t. $y_n = -1$

and thus

$y_n\left(w{*}^{\intercal}x_n\right)\,\forall n$

that is, ρ is strictly positive.

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

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

Further, let $\rho_m=\underset{m}{min}\;y_m\left(w^{*\intercal}x_m\right)$

Then $w_t=\Sigma_{m=0}^{t}y_m^{'}x_m^{'}$

But since $\left(y_m^{'}x_m^{'}\right)\leq\left(y_t^{'}x_t^{'}\right)^{\intercal}w^{*}\;\forall m,t$

then

$\left(\Sigma_{m=0}^{t}y_m^{'}x_m^{'}\right)w^{*}\geq\left(\Sigma_{m=0}^{t-1}y_m^{'}x_m^{'}\right)^{\intercal}w^{*}+y_m\left(x_m^{\intercal}w^{*}\right)$

and thus

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

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

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

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

so

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

then

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

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

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 $t=1$:

$w_1=w_0+x_m^{'}y_m^{'}=0+x_m^{'}y_m^{'}=y_m^{'}x_m^{'}$, and $\|x_m\|\leq\|x_n\|$

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

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

and hence prove that

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

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

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

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

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

Then $\frac{1}{\|w_t\|} \geq \frac{1}{\sqrt[2]{t}R}$

Thus $\frac{w^\intercal w^*}{\geq} \frac{\sqrt[2]{t}\rho}{R}$

In practice, PLA converges more quickly than the bound $\frac{R^2\|w^*\|^2}{\rho^2}$ suggests. Nevertheless, because we do not know p 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.

```
## pla
## Mac Radigan

"""pla.py

  Perceptron Learning Algorithm (PLA)


    initialization:

      N data dimensionality
      K number of samples in dataset

      W [Nx1] initial weights


    random dataset generation:

      X      [NxK] ~ U[a,b]  random training samples
```

```
      W_star [Nx1] ~ U[a,b]  random target function


    update step:

      y_hat   = signum( X * W' )

      y_err   = |y - y_hat|

      rhos    = x * y
      k       = argmax y_err

      err     = exist y" in y_err s.t. |y"| > 0

      W       = W + Y_k*X_k  if err


    training:

      update while err


"""

import numpy as np
import matplotlib.pyplot as plt

class PLA:

  def __init__(self, n):
    """initialize a new PLA:
         n - dimension
    """
    self.new(n)

  def new(self, n):
    """initialize a new PLA:
         n - dimension
    """
    self.n = n                          # data dimensionality
    self.t = 0                          # total iterations
    self.w = np.random.rand(1,n+1)     # initial weight

  def __str__(self):
    """print insternal state"""
    return str(self.w)

  def rand_data(self, k):
    """create random data:
         returns X, a Kx(N+1) real matrix of samples augmented as
 ↪  homogoneous coordinates
    """
    a = -100                           # randomization lower bound
    b = +100                           # randomization upper bound
    return np.concatenate((
        np.ones((k, 1)),
```

```
        a + (b-a) * np.random.rand(k, self.n)
      ), axis=1)

  def rand_target(self):
    """create a random target function:
         returns a random target function
    """
    a = -100                        # randomization lower bound
    b = +100                        # randomization upper bound
    return a + (b-a) * np.random.rand(1, self.n+1)

  def target(self, x, w_star):
    """create a training set:
         x - data set, an 1xK matrix with {-1,+1} entries
         returns truth training set
    """
    return np.sign(x.dot(w_star.T))

  def train(self, x, y):
    """run the learning algorithm
         x - training data set, an NxK real matrix, augmented as homogenous coordinates
         returns the perceptron weights
    """
    while not self.update(x, y):
      pass
    return self.w

  def select(self, x, y):
    """choose a miclassified point
         x - data set
         returns a misclassified point
    """
    y_hat = np.sign(x.dot(self.w.T))               # classify training set using current weights
    y_err = np.abs(y - y_hat)                      # compute error from truth
    ks = y_err.nonzero()[0]
    err = np.any(ks)                               # did the algorihtm converge?
    if err:
      k = ks[0]
    else:
      k = None
    return k

  def update(self, x, y):
    """PLA iterative update step
         x - data set
         returns a boolean indicating convergence
    """
    self.t = self.t + 1                            # increment iteration
   #y_hat = np.sign(x.dot(self.w.T))               # classify training set using current weights
   #y_err = np.abs(y - y_hat)                      # compute error from truth
```

```
    #ks = y_err.nonzero()[0]
    #err = np.any(ks)                              # did the algorihtm converge?
    #if err:
    #  k = ks[0]
    #  self.w = self.w + y[k] * x[k]               # update weights
    #  print('t: %d' % self.t)
    #return not err
    k = self.select(x, y)
    if k is not None:
      self.w = self.w + y[k] * x[k]                # update weights
      print('t: %d' % self.t)
      return False
    else:
      return True

  def classify(self, x):
    """classify a dataset using the internal PLA weights
        x - data set
        returns the classifification
    """
    y = np.sign(x.dot(self.w.T))
    return y

  def plot(self, x, w, w_star, y, title=None, outfile=None):
    """plot classification results
        x - data set
        w_star - target function
    """
    #plt.rc('text', usetex=True)
    #plt.rc('font', family='serif')
    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    k_a  = (y > 0).nonzero()[0]
    x1_a = x[k_a, 1]
    x2_a = x[k_a, 2]
    k_b  = (y < 0).nonzero()[0]
    x1_b = x[k_b, 1]
    x2_b = x[k_b, 2]
    ax.scatter(x1_a, x2_a, color='b', marker='o')
    ax.scatter(x1_b, x2_b, color='r', marker='x')
    dx1 = np.linspace(-150, 150, 2)
    ## target function
    w0 = w_star.T[0]
    w1 = w_star.T[1]
    w2 = w_star.T[2]
    m = -w1/w2
    b = -w0/w2
    dx2 = m * dx1 + b
    ax.plot(dx1, dx2, 'g')
    ## PLA results
    w0 = w.T[0]
    w1 = w.T[1]
    w2 = w.T[2]
    m = -w1/w2
    b = -w0/w2
```

```python
    dx2 = m * dx1 + b
    ax.plot(dx1, dx2, 'm--')
    ## decorations
    ax.set_xlim(-200.0, 200.0)
    ax.set_ylim(-200.0, 200.0)
    plt.xlabel(r'x1')
    plt.ylabel(r'x2')
    if title is not None:
      ax.set_title(title)
    if outfile is not None:
      print('save: %s' % (outfile))
      plt.savefig(outfile)
    else:
      plt.show()
    return ax

  def plot1(self, x, w_star, y, title=None, outfile=None):
    """plot classification results
        x - data set
        w_star - target function
    """
   #plt.rc('text', usetex=True)
   #plt.rc('font', family='serif')
    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    k_a  = (y > 0).nonzero()[0]
    x1_a = x[k_a, 1]
    x2_a = x[k_a, 2]
    k_b  = (y < 0).nonzero()[0]
    x1_b = x[k_b, 1]
    x2_b = x[k_b, 2]
    ax.scatter(x1_a, x2_a, color='b', marker='o')
    ax.scatter(x1_b, x2_b, color='r', marker='x')
    w0 = w_star.T[0]
    w1 = w_star.T[1]
    w2 = w_star.T[2]
    m = -w1/w2
    b = -w0/w2
    dx1 = np.linspace(-150, 150, 2)
    dx2 = m * dx1 + b
    ax.plot(dx1, dx2, color='m')
    ax.set_xlim(-200.0, 200.0)
    ax.set_ylim(-200.0, 200.0)
    plt.xlabel(r'x1')
    plt.ylabel(r'x2')
    if title is not None:
      ax.set_title(title)
    if outfile is not None:
      print('save: %s' % (outfile))
      plt.savefig(outfile)
    else:
      plt.show()
    return ax

  def plot_misclassified(self, x, w_star, y, k, title=None, outfile=None):
    """plot a misclassified point
```

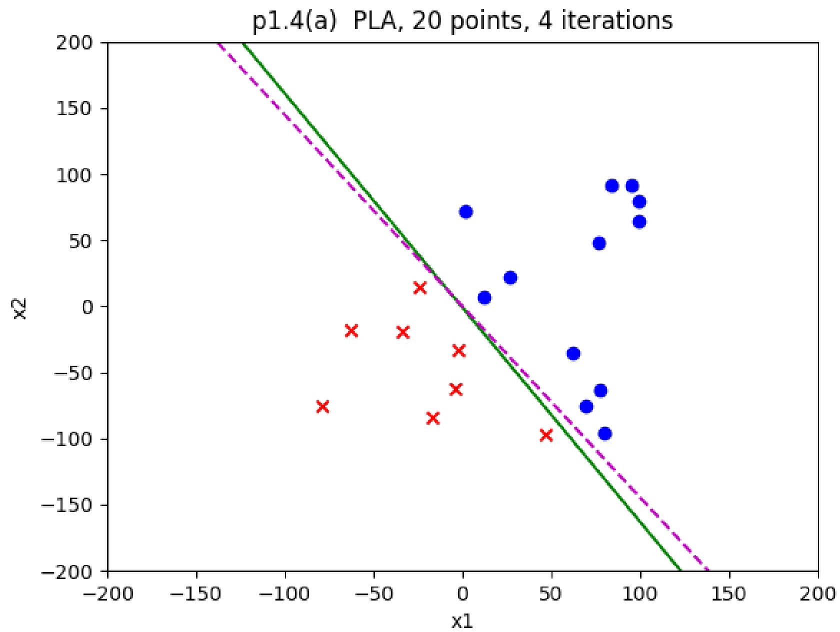
```
        x - data set
        w_star - target function
    """
   #plt.rc('text', usetex=True)
   #plt.rc('font', family='serif')
    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    k_a  = (y > 0).nonzero()[0]
    x1_a = x[k_a, 1]
    x2_a = x[k_a, 2]
    k_b  = (y < 0).nonzero()[0]
    x1_b = x[k_b, 1]
    x2_b = x[k_b, 2]
    ax.scatter(x1_a, x2_a, color='b', marker='o')
    ax.scatter(x1_b, x2_b, color='r', marker='x')
    x1_m = k[1]
    x2_m = k[2]
    ax.scatter(x1_m, x2_m, color='b', marker='*')
    w0 = w_star.T[0]
    w1 = w_star.T[1]
    w2 = w_star.T[2]
    m = -w1/w2
    b = -w0/w2
    dx1 = np.linspace(-150, 150, 2)
    dx2 = m * dx1 + b
    ax.plot(dx1, dx2, color='m')
    ax.set_xlim(-200.0, 200.0)
    ax.set_ylim(-200.0, 200.0)
    plt.xlabel(r'x1')
    plt.ylabel(r'x2')
    if title is not None:
      ax.set_title(title)
    if outfile is not None:
      print('save: %s' % (outfile))
      plt.savefig(outfile)
    else:
      plt.show()
    return ax

  def training_movie(self, x, w_star, y, title=None, outfile=None):
    """save a movie of the PLA training process
        returns the perceptron weights
    """
    t = 0
    err = True
    base = outfile
   #while err:
   #  k = self.select(x, y)
   #  outfile = "%s_%03.3d.png" % (base, self.t)
   #  title = "%s, t=%d" % (title, self.t)
   #  if k is not None:
   #    self.plot_misclassified(x, w_star, y, k, title, outfile)
   #    self.w = self.w + y[k] * x[k]                # update weights
   #    print('t: %d' % self.t)
   #    err = True
   #  else:
```

```
    #    self.plot1(x, w_star, y, title, outfile)
    #    print('t: %d' % self.t)
    #    err = False
    #  t = t + 1
    #return self.w
    ########
    while err:
      k = self.select(x, y)
      self.t = self.t + 1                                 # increment iteration
      outfile = "%s_%03.3d.png" % (base, self.t)
      title = "%s, t=%d" % (title, self.t)
      y_hat = np.sign(x.dot(self.w.T))                    # classify training set using current weights
      y_err = np.abs(y - y_hat)                           # compute error from truth
      ks = y_err.nonzero()[0]
      err = np.any(ks)                                    # did the algorihtm converge?
      if err:
        k = ks[0]
        self.w = self.w + y[k] * x[k]                     # update weights
        self.plot_misclassified(x, w_star, y, k, title, outfile)
        print('t: %d' % self.t)
      else:
        self.plot1(x, w_star, y, title, outfile)
        print('t: %d' % self.t)
      return not err

## *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.

```
#!/usr/bin/env python
## problem_1_4_a.py
## Mac Radigan

"""problem_1_4_a.py
"""

import argparse
from sys import stdout
from perceptron import pla
import numpy as np

def main(verbose):
  'problem 1.4'
  dim = 20
  test(dim)

def case(dim):
  'problem 1.4'
```

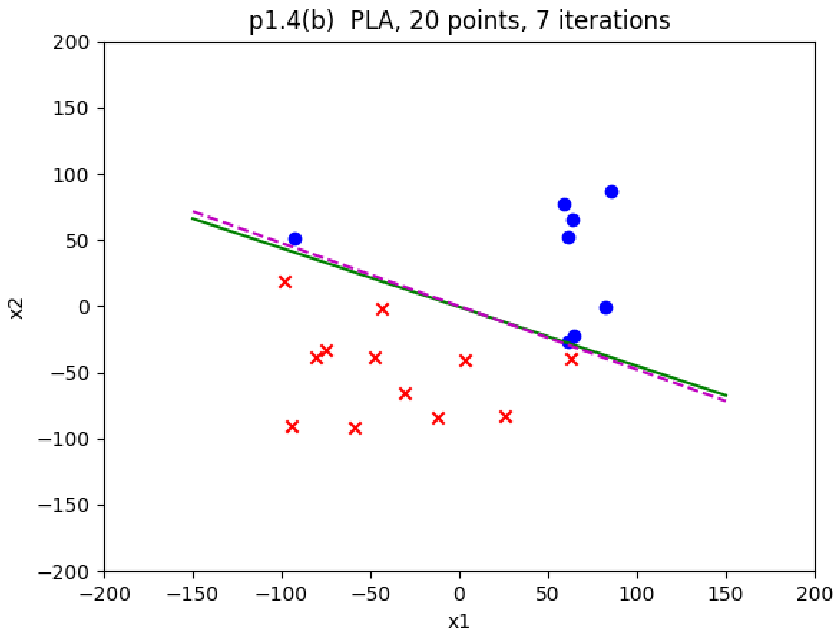
```
  outdir = 'figures/'
  p = pla.PLA(2)
  x = p.rand_data(dim)
  w_star = p.rand_target()
  y = p.target(x, w_star)
 #outfile = '%s/%s' % (outdir, 'p1.4a_target.png')
 #title = 'p1.4(a)  Target Function, %d points' % (dim)
 #ax = p.plot1(x, w_star, y, title, outfile)
  w = p.train(x, y)
  y_hat = p.classify(x)
  y_err = y_hat - y;
  err = 0 != np.sum(y_err)
 #outfile = '%s/%s' % (outdir, 'p1.4a_classify.png')
 #title = 'p1.4(a)  PLA Results, %d iterations' % (p.t)
 #ax = p.plot1(x, w, y_hat, title, outfile)
  outfile = '%s/%s' % (outdir, 'p1.4a.png')
  title = 'p1.4(a)  PLA, %d points, %d iterations' % (dim, p.t)
  ax = p.plot(x, w, w_star, y_hat, title, outfile)
  return (p, x, w_star, y, w, y_hat, y_err, err)

def prn(name, value):
  print('%s:\n%s\n' % (name, value))

def test(dim):
  'test'
  (p, x, w_star, y, w, y_hat, y_err, err) = case(dim)
  prn("x", x)
  prn("w_star", w_star)
  prn("y'", y.T)
  prn("w", w)
  prn("y_hat'", y_hat.T)
 #prn("y_err'", y_err.T)
  prn("err'", err)

if __name__ == '__main__':
 parser = argparse.ArgumentParser(description='problem 1.4 (a)')
 parser.add_argument('-v', '--verbose', action='store_true', dest='verbose',   default=False,   help='verbose output to stdout')
 args = parser.parse_args()
 main(args.verbose)

## *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.

```
#!/usr/bin/env python
## problem_1_4_b.py
## Mac Radigan

"""problem_1_4_b.py
"""

import argparse
from sys import stdout
from perceptron import pla
import numpy as np

def main(verbose):
  'problem 1.4'
  dim = 20
  test(dim)

def case(dim):
  'problem 1.4'
  outdir = 'figures/'
  p = pla.PLA(2)
```

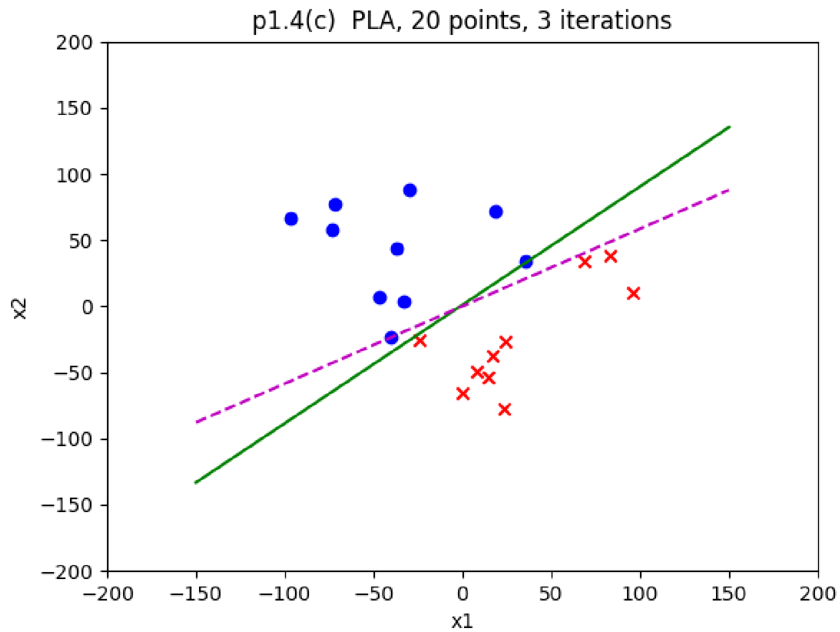
```
  x = p.rand_data(dim)
  w_star = p.rand_target()
  y = p.target(x, w_star)
  w = p.train(x, y)
  y_hat = p.classify(x)
  y_err = y_hat - y;
  err = 0 != np.sum(y_err)
  outfile = '%s/%s' % (outdir, 'p1.4b.png')
  title = 'p1.4(b)  PLA, %d points, %d iterations' % (dim, p.t)
  ax = p.plot(x, w, w_star, y_hat, title, outfile)
  return (p, x, w_star, y, w, y_hat, y_err, err)

def prn(name, value):
  print('%s:\n%s\n' % (name, value))

def test(dim):
  'test'
  (p, x, w_star, y, w, y_hat, y_err, err) = case(dim)
  prn("x", x)
  prn("w_star", w_star)
  prn("y'", y.T)
  prn("w", w)
  prn("y_hat'", y_hat.T)
 #prn("y_err'", y_err.T)
  prn("err'", err)

if __name__ == '__main__':
 parser = argparse.ArgumentParser(description='problem 1.4 (a)')
 parser.add_argument('-v', '--verbose', action='store_true',   dest='verbose',   default=False,   help='verbose output to stdout')
 args = parser.parse_args()
 main(args.verbose)

## *EOF*
```



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

```
#!/usr/bin/env python
## problem_1_4_c.py
## Mac Radigan

"""problem_1_4_c.py
"""

import argparse
from sys import stdout
from perceptron import pla
import numpy as np

def main(verbose):
  'problem 1.4'
  dim = 20
  test(dim)

def case(dim):
  'problem 1.4'
  outdir = 'figures/'
  p = pla.PLA(2)
  x = p.rand_data(dim)
  w_star = p.rand_target()
```

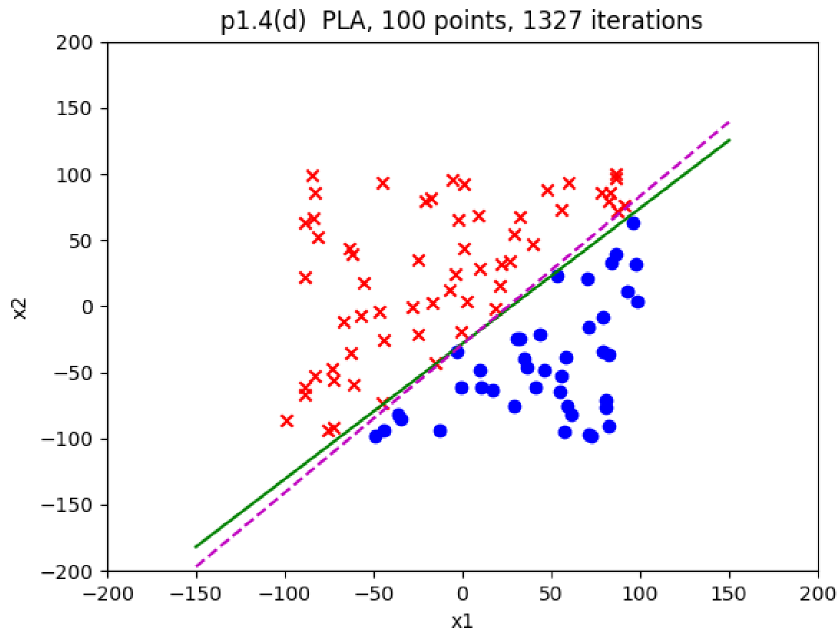
```python
  y = p.target(x, w_star)
  w = p.train(x, y)
  y_hat = p.classify(x)
  y_err = y_hat - y;
  err = 0 != np.sum(y_err)
  outfile = '%s/%s' % (outdir, 'p1.4c.png')
  title = 'p1.4(c)  PLA, %d points, %d iterations' % (dim, p.t)
  ax = p.plot(x, w, w_star, y_hat, title, outfile)
  return (p, x, w_star, y, w, y_hat, y_err, err)

def prn(name, value):
  print('%s:\n%s\n' % (name, value))

def test(dim):
  'test'
  (p, x, w_star, y, w, y_hat, y_err, err) = case(dim)
  prn("x", x)
  prn("w_star", w_star)
  prn("y'", y.T)
  prn("w", w)
  prn("y_hat'", y_hat.T)
 #prn("y_err'", y_err.T)
  prn("err'", err)

if __name__ == '__main__':
 parser = argparse.ArgumentParser(description='problem 1.4 (a)')
 parser.add_argument('-v', '--verbose', action='store_true',   dest='verbose',   default=False,   help='verbose output to stdout')
 args = parser.parse_args()
 main(args.verbose)

## *EOF*
```



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

```
#!/usr/bin/env python
## problem_1_4_d.py
## Mac Radigan

"""problem_1_4_d.py
"""

import argparse
from sys import stdout
from perceptron import pla
import numpy as np

def main(verbose):
  'problem 1.4'
  dim = 100
  test(dim)

def case(dim):
  'problem 1.4'
  outdir = 'figures/'
  p = pla.PLA(2)
  x = p.rand_data(dim)
  w_star = p.rand_target()
```

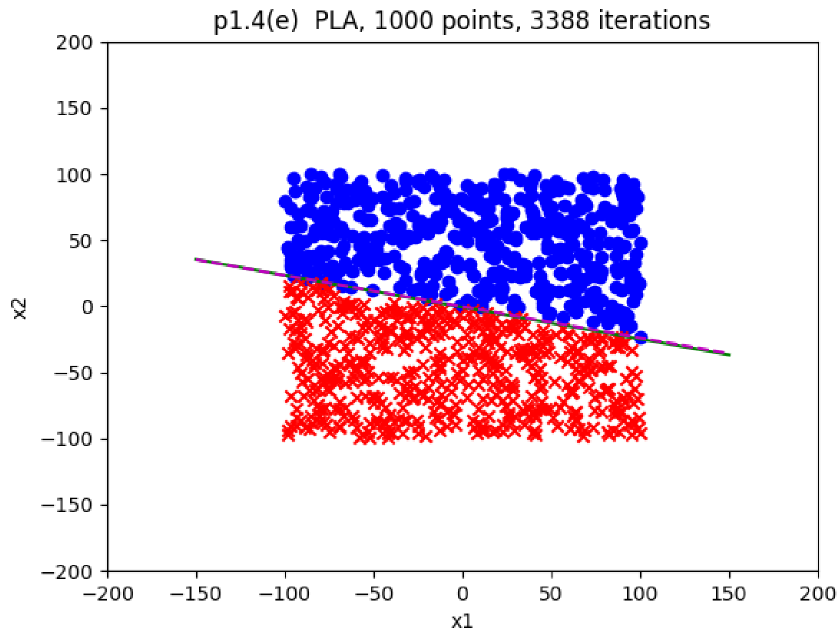
```
  y = p.target(x, w_star)
  w = p.train(x, y)
  y_hat = p.classify(x)
  y_err = y_hat - y;
  err = 0 != np.sum(y_err)
  outfile = '%s/%s' % (outdir, 'p1.4d.png')
  title = 'p1.4(d)  PLA, %d points, %d iterations' % (dim, p.t)
  ax = p.plot(x, w, w_star, y_hat, title, outfile)
  return (p, x, w_star, y, w, y_hat, y_err, err)

def prn(name, value):
  print('%s:\n%s\n' % (name, value))

def test(dim):
  'test'
  (p, x, w_star, y, w, y_hat, y_err, err) = case(dim)
  prn("x", x)
  prn("w_star", w_star)
  prn("y'", y.T)
  prn("w", w)
  prn("y_hat'", y_hat.T)
 #prn("y_err'", y_err.T)
  prn("err'", err)

if __name__ == '__main__':
 parser = argparse.ArgumentParser(description='problem 1.4 (a)')
 parser.add_argument('-v', '--verbose', action='store_true',
 ↪  dest='verbose',   default=False,   help='verbose output to stdout')
 args = parser.parse_args()
 main(args.verbose)

## *EOF*
```



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

```
#!/usr/bin/env python
## problem_1_4_d.py
## Mac Radigan

"""problem_1_4_d.py
"""

import argparse
from sys import stdout
from perceptron import pla
import numpy as np

def main(verbose):
  'problem 1.4'
  dim = 100
  test(dim)

def case(dim):
  'problem 1.4'
  outdir = 'figures/'
  p = pla.PLA(2)
  x = p.rand_data(dim)
  w_star = p.rand_target()
```

```
  y = p.target(x, w_star)
  w = p.train(x, y)
  y_hat = p.classify(x)
  y_err = y_hat - y;
  err = 0 != np.sum(y_err)
  outfile = '%s/%s' % (outdir, 'p1.4d.png')
  title = 'p1.4(d)  PLA, %d points, %d iterations' % (dim, p.t)
  ax = p.plot(x, w, w_star, y_hat, title, outfile)
  return (p, x, w_star, y, w, y_hat, y_err, err)

def prn(name, value):
  print('%s:\n%s\n' % (name, value))

def test(dim):
  'test'
  (p, x, w_star, y, w, y_hat, y_err, err) = case(dim)
  prn("x", x)
  prn("w_star", w_star)
  prn("y'", y.T)
  prn("w", w)
  prn("y_hat'", y_hat.T)
 #prn("y_err'", y_err.T)
  prn("err'", err)

if __name__ == '__main__':
 parser = argparse.ArgumentParser(description='problem 1.4 (a)')
 parser.add_argument('-v', '--verbose', action='store_true',    dest='verbose',   default=False,   help='verbose output to stdout')
 args = parser.parse_args()
 main(args.verbose)

## *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 $x_n \in \mathbb{R}^1 0$ 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 r 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

$\int_{\alpha}^{\infty} \alpha p(x)dx \geq \int_{0}^{\alpha} xp(x)dx + \int_{\alpha}^{\infty} xp(x)dx$

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

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

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

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 $u_1, \ldots, u_N$ are iid random variables, each with mean μ and variance σ^2, and $u = \frac{1}{N}\Sigma_{n=1}^{N} u_n$, 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
d_{vc}	Vapnik-Chervonenkis Dimension

Generalization Error

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

Growth Function

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

Bounding the Growth Function

$B\left(N,k\right)$ 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

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

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

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

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

Sauer's Lemma

$$B\left(N,k\right) \leq \Sigma_{i=0}^{k-1} \binom{N}{i} \tag{12}$$

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 exapmles 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 exapmles 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 exapmles 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 thise 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).

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

$m_{\mathcal{H}}(N) = \binom{N+1}{2} + 1 = N^2 + N + 1$ 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 $\mathcal{H} = \{h_1, h_2, \cdots, h_M\}$. 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 $\mathcal{H}_1, \mathcal{H}_2, \cdots, \mathcal{H}_K$ wiht 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 $\mathcal{H}_1, \mathcal{H}_2, \cdots, \mathcal{H}_K$ wiht 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

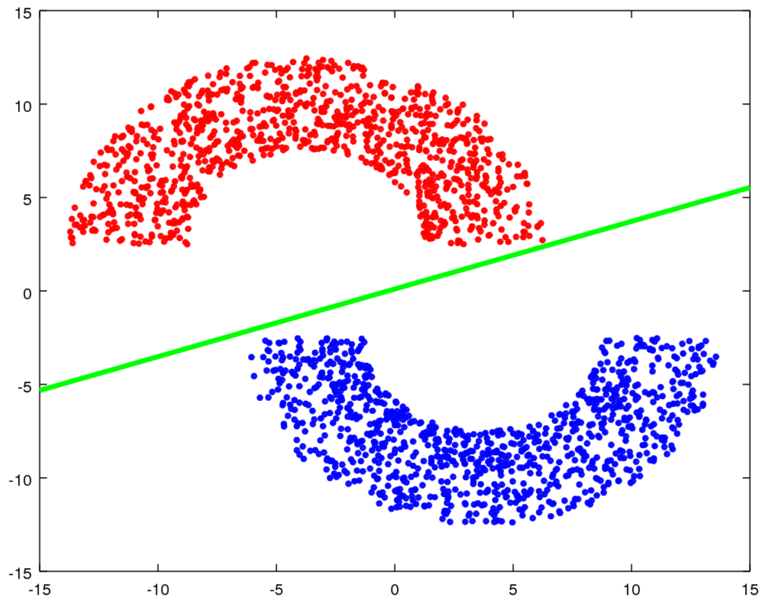
Problem 3.1

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.

```
#!/usr/bin/env octave
## problem_3_1_a.m
## Mac Radigan

FORCES__SCRIPT_FILE=1;
ux=false;

function ax = do_plot(ux, rad, thk, c1, c2, x1, x2, w, k_err)

  if ux
    show = 'on';
  else
    show = 'off';
  end

  %% plot input dataset
  ax = figure(30111);
  set(ax, 'visible', show);
    plot(x1(c1), x2(c1), 'LineWidth', 1, 'r.');
    hold on;
    plot(x1(c2), x2(c2), 'LineWidth', 1, 'b.');
    hold on;

  %% plot final hypothesis, g(x)
  ext = rad + thk;
  m = -w(2)/w(3);
  b = -w(1)/w(3);
  dx1 = linspace(-ext, ext, 2);
  dx2 = m * dx1 + b;
  ax = figure(30111);
  set(ax, 'visible', show);
    if k_err > 0
      plot(dx1, dx2, 'Color', 'magenta', 'LineWidth', 1, 'LineStyle',
↪  ':');
    else
      plot(dx1, dx2, 'Color', 'green', 'LineWidth', 3);
    end
    hold on;

  drawnow();

end % function do_plot

N   = 2e3;  % number of training samples
rad = 10;   % radius of semi-circle
thk =  5;   % thickness of semi-circle
sep =  5;   % separation between semi-circles

%% uniformly distributed data
z  = (rad - thk * (1 - rand(1,N))) .* exp(-j*2*pi*rand(1,N));

%% target function
```

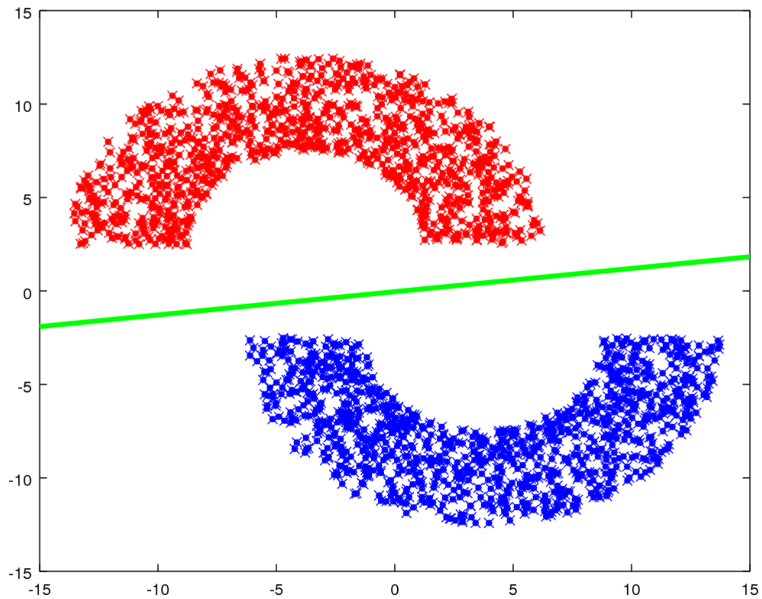
```
  f = @(z) sign(angle(z));

  %% target set
  y = f(z);

  %% cartesian basis representation
  x1 = -f(z)*(rad/2-thk/4) + abs(z) .* cos(angle(z));
  x2 = f(z)*sep/2 + abs(z) .* sin(angle(z));

  %% categorical index
  c1 = y > 0;
  c2 = y < 0;

  %% Perceptron Learning Algorithm (PLA)
  t = 0;                           % training step counter
  w = zeros(1,3);                  % initial weights
 %w = randn(1,3);                  % initial weights
  X = [ones(size(y)); x1; x2];     % training data
  k_err = inf;                     % convergence criteria
  for n = 1:N
    x_n = X(:,n);
    t = t + 1;                        % update training step counter
    if ~mod(n,500)
     %ax = do_plot(ux, rad, thk, c1, c2, x1, x2, w, k_err);
    end
    if ux
      fprintf(1, 'error: %f\n', k_err);
      t
      w
    end
    y_hat = sign(w*x_n);              % classify
    y_err = 0.5*abs(y(n) - y_hat);    % residual
    [k_err, k] = max(y_err);          % select misclassified point
    if k_err <= 0
      continue
    else
      wg = w;
    end
    x_k = x_n(:,k);                   %   in the training data
    y_k = y(k);                       %   in the target set
    w   = w + y_k*x_k';               % update weights
    if ux
      disp('...')
      input('...')
    end
  end % training
  ax = do_plot(ux, rad, thk, c1, c2, x1, x2, wg, k_err);

  ax = do_plot(ux, rad, thk, c1, c2, x1, x2, w, k_err);
  if ux
    disp('training done.')
  else
    saveas(ax, 'figures/p3.1a.png');
  end

## *EOF*
```



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

```
#!/usr/bin/env octave
## problem_3_1_b.m
## Mac Radigan

  FORCES__SCRIPT_FILE=1;
  ux=false;

  function ax = do_plot(ux, rad, thk, c1, c2, x1, x2, w)

    if ux
      show = 'on';
    else
      show = 'off';
    end

    %% plot input dataset
    ax = figure(30112);
    set(ax, 'visible', show);
      plot(x1(c1), x2(c1), 'LineWidth', 1, 'r.');
      hold on;
      plot(x1(c2), x2(c2), 'LineWidth', 1, 'b.');
      hold on;

    %% plot input dataset
```

```
    ax = figure(30112);
    set(ax, 'visible', show);
      plot(x1(c1), x2(c1), 'rx');
      hold on;
      plot(x1(c2), x2(c2), 'bx');
      hold on;

    %% plot final hypothesis, g(x)
    ext = rad + thk;
    m = -w(2)/w(3);
    b = -w(1)/w(3);
    dx1 = linspace(-ext, ext, 2);
    dx2 = m * dx1 + b;
    ax = figure(30112);
    set(ax, 'visible', show);
      plot(dx1, dx2, 'Color', 'green', 'LineWidth', 3);
      hold off;

    drawnow();

  end % function do_plot

  N   = 2e3;  % number of training samples
  rad = 10;   % radius of semi-circle
  thk =  5;   % thickness of semi-circle
  sep =  5;   % separation between semi-circles

  %% uniformly distributed data
  z = (rad - thk * (1 - rand(1,N))) .* exp(-j*2*pi*rand(1,N));

  %% target function
  f = @(z) sign(angle(z));

  %% target set
  y = f(z);

  %% cartesian basis representation
  x1 = -f(z)*(rad/2-thk/4) + abs(z) .* cos(angle(z));
  x2 = f(z)*sep/2 + abs(z) .* sin(angle(z));

  %% categorical index
  c1 = y > 0;
  c2 = y < 0;

  %% Linear Regression
  X = [ones(size(y)); x1; x2];   % training data
  w = y * pinv(X);

  %% plot results
  ax = do_plot(ux, rad, thk, c1, c2, x1, x2, w);

  if ~ux
    saveas(ax, 'figures/p3.1b.png');
  end

## *EOF*
```



Exercise 3.3

Consider the hat matrix $H = X\left(X^{\intercal}X\right)^{-1}X^{\intercal}$, where X is an N by $d+1$ matrix, and $X^{\intercal}X$ is invertible.

(a) Show that H is symmetric.

$X^{\intercal}X$ is symmetric, so $\left(X^{\intercal}X\right)^{-1}$ is symmetric.

Now,

$$\begin{aligned}
H &= X\left(X^{\intercal}X\right)^{-1}X^{\intercal} \\
&= X\left(\left(X^{\intercal}X\right)^{-1}\right)^{\intercal}X^{\intercal} \\
&= \left(X\left(X^{\intercal}X\right)^{-1}X^{\intercal}\right)^{\intercal} \\
&= X\left(\left(X^{\intercal}X\right)^{-1}\right)^{\intercal}X^{\intercal} \\
&= X\left(X^{\intercal}X\right)^{-1}X^{\intercal} \\
&= H
\end{aligned}$$

Problem 3.10

(b) Show that the trace of a symmetic 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.]

$c_k = \frac{-1}{k} TR\left(AB_{k-1}\right)$ and $B_0 = I$, so for $k = 1$ $c_1 = TR\left(AB_0\right) = TR\left(AI\right) = TR\left(A\right)$.

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. Sutherlang 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 informaiton, 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 beeen 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 $xI - A \in \mathbf{C}\left[x\right]^{n\times n}$, the collection of n-by$n$ matrices with polynomial entries. We must open our minds to accepting matrices with polynomial entries. For example, $\begin{bmatrix} x^2+1 & x-3 \\ 4x+2 & x^3-7 \end{bmatrix} \in \mathbf{C}\left[x\right]^{2\times 2}$. Determinants work just fine for these kinds of matrices. The determinant of $xI - A$, $det\left(xI - A\right) \in \mathbf{C}\left[x\right]$, the polynomials in x, and is what we call the *characteristic polynomial* of A:

$$\mathcal{X}_A\left(x\right) = det\left(xI - A\right) = x^n + c_1 x^{n-1} + \cdots + c_{n-1}x + c_n$$

For example, if $A = \begin{bmatrix} 1 & 2 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} \in \mathbf{C}\left[x\right]^{3x3}$, then $xI_3 - A =$

$\begin{bmatrix} x-1 & -2 & -2 \\ -3 & x-4 & -5 \\ -6 & -7 & x-8 \end{bmatrix} \in \mathbf{C}\left[x\right]^{3x3}$.

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

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

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

$$(xI - A)\, adj(xI - A) = \mathcal{X}_A(x) I$$

To illustrate with the example above,

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

$$= \begin{bmatrix} x^3 - 13x^2 - 9x - 3 & 0 & 0 \\ 0 & x^3 - 13x^2 - 9x - 3 & 0 \\ 0 & 0 & x^3 - 13x^2 - 9x - 3 \end{bmatrix}$$

$$= x^3 - 13x^2 - 9x - 3 \cdots \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Next, let $C(x) = adj(xI - A) \in \mathbf{C}[x]^{n \times n}$. We note that the elements of $adj(xI - A)$ are computed as $(n-1)-by-(n-1)$ 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,

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

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

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, $adj(A) = (-1)^{n-1} adj(-A) = (-1)^{n-1} C(0) = (-1)^{n-1} B_{n-1}$, so

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

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

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

But now we compute

$$\begin{aligned}(xI-A)\,adj(xI-A) &= (xI-A)\,C(x) \\ &= (xI-A)\left(B_0x^{n-1} + B_1x^{n-2} + \cdots + B_{n-2}x + B_{n-1}\right) \\ &= x^nI + x^{n-1}c_1I + \cdots + xc_{n-1}I + c_nI\end{aligned}$$

and we compare coefficients using the following table:

Compare Coefficients	Multiply by	on the Left	on the Right
$B_0 = I$	A^n	A^nB_0	A^n
$B_1 - AB_0 = C_1I$	A^{n-1}	$A^{n-1}B_1 - A^nB_0$	c_1A^{n-1}
$B_2 - AB_1 = C_2I$	A^{n-2}	$A^{n-2}B_2 - A^{n-1}B_1$	c_2A^{n-2}
$\vdots$			
$B_k - AB_{k-1} = c_kI$			

Compare Coefficients	Multiply by	on the Left	on the Right
$\vdots$			
$B_{n-2} - AB_{n-3} = C_{n-2}I$	A^2	$A^2B_{n-2} - A^3B_{n-3}$	$c_{n-2}A^2$
$B_{n-1} - AB_{n-2} = C_{n-1}I$	A	$AB_{n-1} - A^2B_{n-2}$	$c_{n-1}A$
$-AB_{n-1} = C_nI$		$-AB_{n-1}$	c_nI
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 theroem 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 euqations on the left of the table, so let's push a little harder. Notice we can rewrite these equations as

$$\begin{aligned} B_0 &= I \\ B_1 &= AB_0 + c_1I \\ B_2 &= AB_1 + c_2I \\ &\vdots \\ B_{n-1} &= AB_{n-2} + c_{n-1}I \\ \mathbf{O} &= AB_{n-1} + c_nI. \end{aligned}$$

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

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

In other words, the matrix coefficients, the B_ks are given recursively in terms of the B_{k-1}s and c_ks. If we can get a formula for the B_k and c_k in terms of B_{k-1}, we will get acomplete 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:

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

Inductively, we see for k = 1, 2, . . . , n,

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

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

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

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

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

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

$$tr\left(AB_{k-1}\right) = tr\left(A^k\right) + tr\left(c_1 A^{k-1}\right) + \cdots + tr\left(c_{k-2} A^2\right) + tr\left(c_{k-1} A\right).$$

for k = 2, 3, ..., 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, . . . , n+1,

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

Digression on Newton's Identities

Newton's identities go back aways. 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-baed argument following the ideas of [Eidswick 1968]. First, we need to recall some facts about polynomials. Let $p(x) = a_0 + a_1 x + \cdots + a_n x^n$. Then the coefficients of p can be expressed in terms of derivatives of p evaluated at zero (remember Taylor polynomials?):

$$p(x) = p(0) + p^{'}(0)\, x + \tfrac{p^{''}(0)}{2!} x^2 + \cdots + \tfrac{p^{(n)} x^n}{n!} x^n.$$

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, $q\left(\frac{1}{2}\right) = -\frac{6}{8} + \frac{11}{4} - \frac{6}{2} + 1 = \frac{-6+22-24+8}{8} = 0$. You can also check $q\left(\frac{1}{3}\right) = 0$. 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 $p(x) = a_0 + a_1 x + \cdots + a_n x^n$ and the reversed polynomial is $q(x) = a_n + a_{n-1} x + \cdots + a_0 x^n$. Note

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

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

Suppose $p(x) = a_0 + a_1 x + \cdots + a_n x^n = a_n (x - r_1)(x - r_2) \cdots (x - r_n)$. The r_is are, of course, the roots of p, which we assume to be nonzero but not necessarily distinct. Then the reversed polynomial $q(x) = a_n + a_{n-1} x + \cdots + a_0 x^n = a_0 \left(x - \frac{1}{r_1}\right)\left(x - \frac{1}{r_2}\right) \cdots \left(x - \frac{1}{r_n}\right)$. For the sake of illustration, suppose $n = 3$.

Then form

$$f(x) = \frac{q^{'}(x)}{q(x)} = \frac{\left(x - r_1^{-1}\right)\left[\left(x - r_2^{-1}\right) + \left(x - r_3^{-1}\right)\right]\left[\left(x - r_2^{-1}\right) + \left(x - r_3^{-1}\right)\right]}{(r - r_1)(r - r_2)(r - r_3)}$$

$$= \frac{1}{x - r_1^{-1}} + \frac{1}{x - r_2^{-1}} + \frac{1}{x - r_3^{-1}}$$

. Generally then,

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

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

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

.

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:

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

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

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

Plugging in zero, we get

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

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*

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