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 $\mathcal{H}$,

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

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:

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

The Asymptotic Equipartition Property (AEP) states:

$\frac{1}{N}\log_2\frac{1}{p(X_1,X_2,\cdots,X_N)} \rightarrow H\left(X\right)$ 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* ~ p(x), then

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

Proof:

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

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

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

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?

$w^\intercal x = 0$

$\implies w_0 x_0 + w_1 + x_1 + w_2 x_2 = 0$

$\implies w_2 x_2 = -w_1 x_1 - w_0 x_0 = -w_1 x_1 - x_0$

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

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

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

(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 $\rho = min_{1 \le n \le N}(w^{*\intercal} x_n)$. 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

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

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
```

```
    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'
```

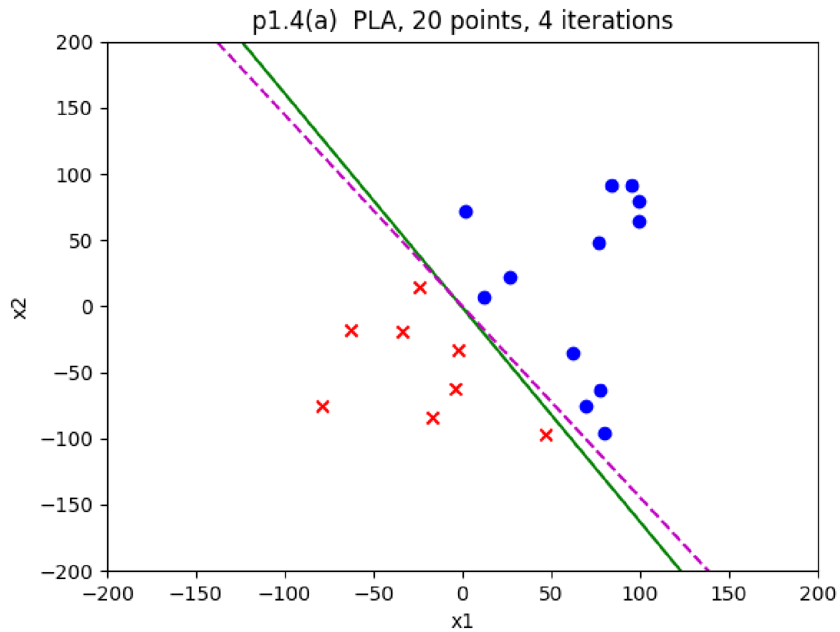
```
  outdir = 'figures/'
  p = pla.PLA(2)
  x = p.rand_data(dim)
  w_star = p.rand_target()
  y = p.target(x, w_star)
 #outfile = '%s/%s' % (outdir, 'p1.4a_target.png')
 #title = 'p1.4(a)  Target Function, %d points' % (dim)
 #ax = p.plot1(x, w_star, y, title, outfile)
  w = p.train(x, y)
  y_hat = 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)
```

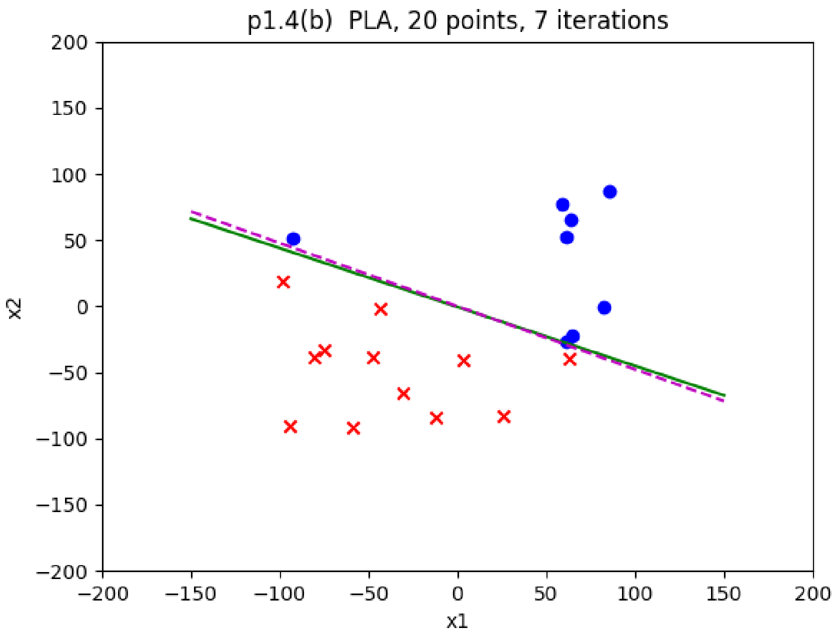
```python
  x = p.rand_data(dim)
  w_star = p.rand_target()
  y = p.target(x, w_star)
  w = p.train(x, y)
  y_hat = p.classify(x)
  y_err = y_hat - y;
  err = 0 != np.sum(y_err)
  outfile = '%s/%s' % (outdir, 'p1.4b.png')
  title = 'p1.4(b)  PLA, %d points, %d iterations' % (dim, p.t)
  ax = p.plot(x,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

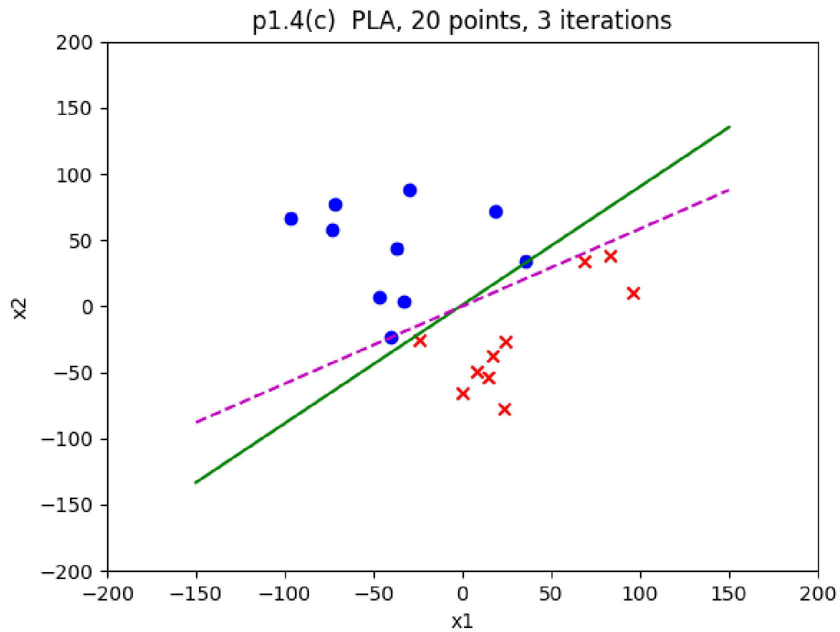
```
  y = p.target(x, w_star)
  w = p.train(x, y)
  y_hat = p.classify(x)
  y_err = y_hat - y;
  err = 0 != np.sum(y_err)
  outfile = '%s/%s' % (outdir, 'p1.4c.png')
  title = 'p1.4(c)  PLA, %d points, %d iterations' % (dim, p.t)
  ax = p.plot(x, w, w_star, y_hat, title, outfile)
  return (p, x, w_star,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

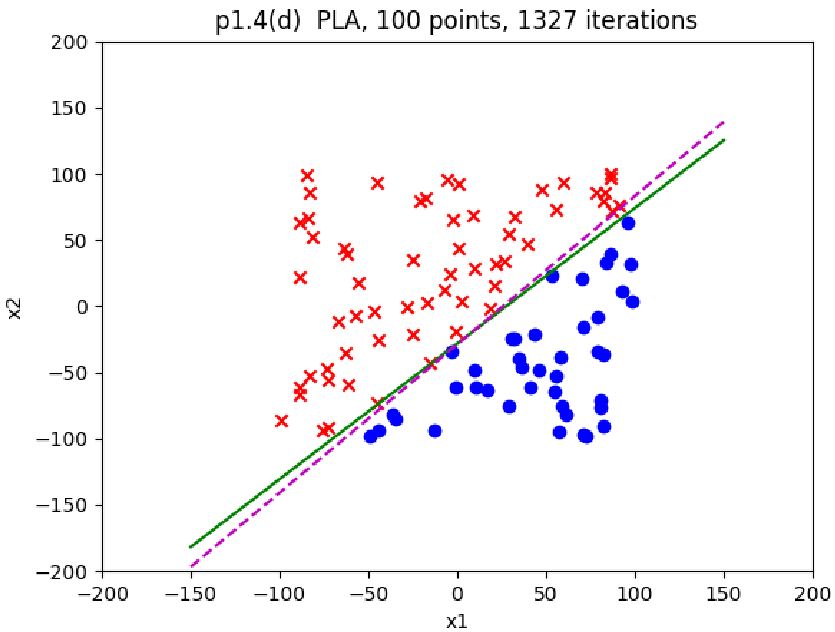
```
  y = p.target(x, w_star)
  w = p.train(x, y)
  y_hat = p.classify(x)
  y_err = y_hat - y;
  err = 0 != np.sum(y_err)
  outfile = '%s/%s' % (outdir, 'p1.4d.png')
  title = 'p1.4(d)  PLA, %d points, %d iterations' % (dim, p.t)
  ax = p.plot(x, w, w_star, y_hat, title, outfile)
  return (p, x, w_star,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

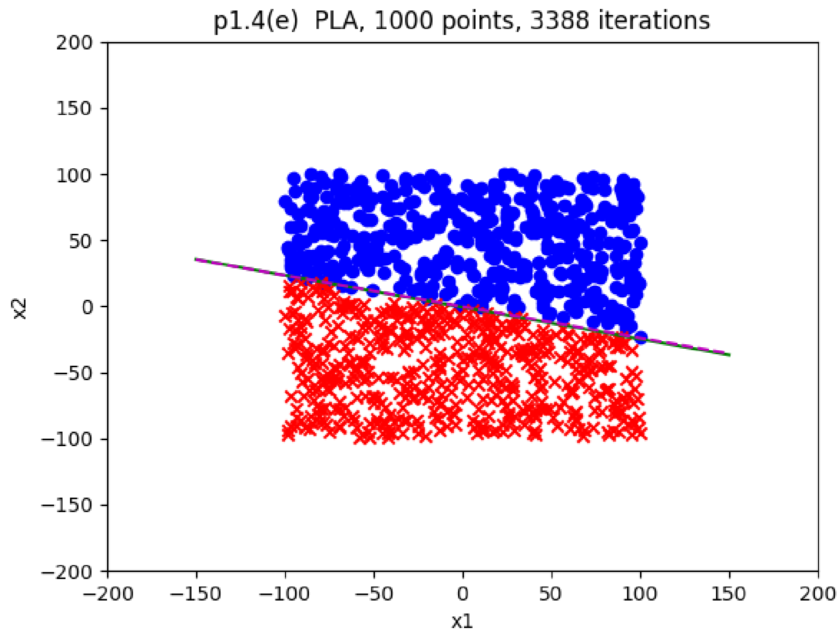
```
  y = p.target(x, w_star)
  w = p.train(x, y)
  y_hat = p.classify(x)
  y_err = y_hat - y;
  err = 0 != np.sum(y_err)
  outfile = '%s/%s' % (outdir, 'p1.4d.png')
  title = 'p1.4(d)  PLA, %d points, %d iterations' % (dim, p.t)
  ax = p.plot(x, w, w_star, y_hat, title, outfile)
  return (p, x, w_star, 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 $x_n \in \mathbb{R}^10$ 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 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

$B\,(N, k)$ 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

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

$k = d_{VC} + 1$ 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

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

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

Chapter 3 Notes