HW01 - CS 189

1.

Who else did you work with on this homework? In case of course events, just describe the group. How did you work on this homework? Any comments about the homework?

I worked on this homework with Ehimare Okoyomon, Prashanth Ganeth, and Daniel Mockaitis. We worked by getting together throughout the week and communicating on facebook.

I certify that all solutions are entirely in my words and that I have not looked at another student's solutions. I have credited all external sources in this write up}

Nicholas Lorio, 26089160

2.

c. a

$p_i = \frac{(n-(i-1))}{n}$

b_i = time (# days) to collect i^{th} card after $i-1$ cards have already been collected

b_i has geometric distribution w/ exp of $1/p_i$

$B = b_1 + \ldots + b_n = \sum_{i=1}^{n} b_i$

B := total time to win competition

$E(B) = E(b_1) + \ldots + E(b_n)$ by linearity of expectation

$= 1 \cdot \frac{1}{p_1} + \ldots + 1 \cdot \frac{1}{p_n}$

$= \frac{n}{n} + \frac{n}{n-1} + \frac{1}{n}$ = ~~[illegible]~~ $\left(\frac{n}{n} + \frac{n}{n-1} + \frac{n}{n-2} + \ldots + \frac{n}{2} + \frac{n}{1}\right)$

$n\left(\frac{1}{n} + \frac{1}{n-1} + \frac{1}{n-2} + \ldots + 1\right) = n \cdot \left(1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}\right)$

$= n \cdot \underbrace{H_n}_{\text{Harmonic \#}}$

$H_n = \sum 1/k = \gamma + \psi_0(n+1) \sim \ln(n) + \gamma + \frac{1}{2n} - \frac{1}{12}n^{-2} + \text{higher order terms}$

$\gamma \approx 0.5772156$ by Euler-Mascheroni constant.

$n = 40$ $H_{40} =$

$E(T) = 40 \cdot (4.2786) = 171.14 \approx \underline{172 \text{ days}}$

prove risemans proof:

2.b. $P(\text{win}) = \frac{b}{d}$ let b be the # of distinct card types

$\frac{b}{d} \geq (1-\delta)$ $\quad b \geq (1-\delta)d$ $\quad \underline{b = (1-\delta)d}$

2.c. $P(\text{win @ draw after } \wedge \text{ days}) = \frac{d - E[Y]}{d}$

$Y :=$ the # of cards (unique) not selected (~~Unique card not selected all day~~

else $X_i := \begin{cases} 1 & : \text{if unique 'card } i\text{' is not in the } \wedge \text{ days} \\ 0 & : \text{o.w.} \end{cases}$

$Y = X_1 + \ldots + X_d$ $\quad E[Y] = E[X_1] + \ldots + E[X_d]$ by linearity of expectation

$E[X_i] = 1 \cdot P(X_i = 1) + 0 \cdot P(X_i = 0) = P(X_i = 1)$

$P(X_i = 1) = \left(1 - \frac{(d-1)!}{d!}\right)^{\wedge} = \left(1 - \frac{1}{d}\right)^{\wedge}$ $\quad = P(\text{unique card } d \text{ not in } \wedge \text{ days})$

$E[Y] = d\left(1 - \frac{1}{d}\right)^{\wedge}$ $\quad \therefore \begin{cases} \text{expected \# of filled bins} \\ = d - E[Y] = d - d\left(1 - \frac{1}{d}\right)^{\wedge} \end{cases}$

$P(\text{win @ draw after } \wedge \text{ day}) = \frac{d - d\left(1 - \frac{1}{d}\right)^{\wedge}}{d}$

$= 1 - \left(1 - \frac{1}{d}\right)^{\wedge}$

2.d. $\lim_{d \to \infty} \left(\quad \right) - \left(1 - \frac{1}{d}\right)^{\wedge d}$ $\quad \exp\left[\ln\left[\lim\left(1 - \left(1 - \frac{1}{d}\right)^{\wedge d}\right)\right]\right]$

$\Rightarrow 1 - \exp\left[\lim_{d \to \infty} \wedge d \ln\left(1 - \frac{1}{d}\right)\right] = 1 - \exp(-\wedge) = \boxed{1 - \frac{1}{e^{\wedge}}}$

i.e. analogous to what we did in part c

$d := (\#$ of distinct cards$)$ & $= ($the size of our domain $D)$

$\Lambda := (\#$ of days before the draw$)$ & $= ($our # of samples$) = ($the size of training set req.$)$

$\mathbb{P}($successfully estimate perms$) = 1 - (1 - \frac{1}{d})^{\Lambda}$

$\mathbb{P}(\text{success}) \geq 1 - \delta \qquad 1 - (1 - \frac{1}{d})^{\Lambda} \geq 1 - \delta$

$\ln(1 - \frac{1}{d})^{\Lambda} \leq \ln(+\delta)$

$\Lambda \cdot \ln(1 - 1/d) \leq \ln(+\delta)$

$\Lambda \geq \frac{\ln(+\delta)}{\ln(1 - \frac{1}{d})}$

$$\boxed{\Lambda = \frac{\ln(+\delta)}{\ln(1 - 1/d)}}$$

behavior as d gets large but does not approach ∞:
(but finite)

if d is a large finite # then Λ becomes a large finite # as well.

$\Lambda = \frac{\ln(\delta)}{}$ (very small #)

ℓ using Taylor based

as d gets large $\ln(1 - 1/d) \approx -1/d$

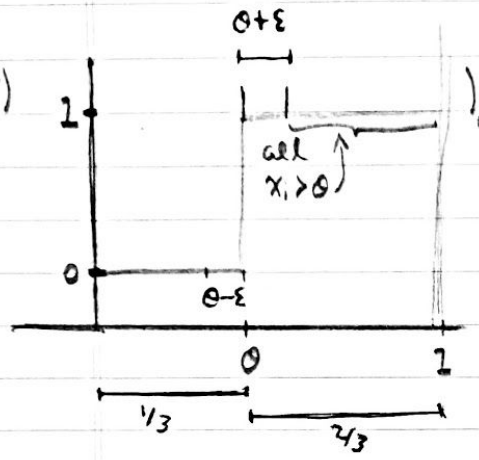
$\Lambda \approx \frac{\ln(\delta)}{\ln(-1/d)} = -d \ln(\delta) = d \ln((\delta)^{-1})$

$$\boxed{\Lambda = d \cdot \ln(1/\delta)}$$

3.

3.a $\mathbb{P}(T_{Max} - \theta > \varepsilon) = 1 - \mathbb{P}(T_{max} - \theta \leq \varepsilon)$

$= \mathbb{P}(T_{max} > \varepsilon + \theta) = 1 - \mathbb{P}(T_{max} \leq \theta + \varepsilon)$



$= 1 - \mathbb{P}(\theta + \varepsilon) = (1-\varepsilon)^n =$

$\mathbb{P}(T_{max} > \theta + \varepsilon) = \begin{cases} \text{Cases} \\ 1. \text{ all } x_i > 2/3 \\ 2. \text{ some } \theta < x_i < 2/3 \end{cases}$

$x_i > \theta + \varepsilon$

$T_{max} > \theta + \varepsilon$

For $\theta + \varepsilon < 2/3$ • $\mathbb{P}(\min\{[2/3] \cup \{x_i \mid F_\theta(x_i) = 1\}\} > \theta + \varepsilon) =$

Lets implies that

$\Rightarrow$ all $x_i > \theta$ is $\theta + \varepsilon$

IF $\theta + \varepsilon > 2/3$ $\mathbb{P} = 0$

$= \mathbb{P}(\min\{x_i \mid x_i > \theta\} > \theta + \varepsilon) = P(\text{No } x \in (\theta, \theta + \varepsilon)) = (1-\varepsilon)^n$ ($\theta \to \theta + \varepsilon$)

For • $\mathbb{P}(\theta - T_{min} > \varepsilon) = \mathbb{P}(T_{min} < \theta - \varepsilon)$

$\varepsilon - \theta < 1/3$ By Similar train of logic

$= (1-\varepsilon)^n$

IF $\varepsilon - \theta > 1/3$

$\mathbb{P} = 0$

3.b. $(\theta - T_{n,n}) + (T_{n,n} - \theta) < 2\varepsilon$

$P((\theta - T_{n,n}) + (T_{n,n} - \theta) < 2\varepsilon)$

$= P(\theta - T_{n,n} < \varepsilon) \cap P(T_{n,n} - \theta < \varepsilon)$

$\geq 2(1-\varepsilon)^n \geq \delta$

$\ln(2(1-\varepsilon)^n) \geq \ln(\delta)$

$n \cdot \ln(2(1-\varepsilon)) \geq \ln(\delta)$

$\therefore \boxed{n > \frac{\ln(\delta)}{\ln(2(1-\varepsilon))}}$

3.c. Choose where to sample. $\theta \in (1/3, 2/3)$

Sample between

Method proposed to estimate θ?

$\hat{\theta} = \frac{1}{3} + \text{Something} \quad \Lambda = 1, \ldots,$

(want Λ points between $1/3$, $2/3$)

let $k_i \in \{1, \ldots, \Lambda\}$, $\hat{\theta}_i = \frac{1}{3} + \frac{k_i}{3(\Lambda+2)}$

Our i^{th} estimate given that we have Λ samples

$\frac{1}{3(\Lambda+1)} < \varepsilon \Rightarrow$

$1 < (3\Lambda+1)\varepsilon$, $3\Lambda > -\varepsilon + 1$

$\Lambda > \frac{1}{3} -$

(our rate) - the step size / size of our partitions increments

$1 < \varepsilon(3\Lambda+3)$, $1 < 3\Lambda\varepsilon + 3\varepsilon$

$1 - 3\varepsilon < 3\Lambda\varepsilon$ $\quad \Lambda > \frac{1-3\varepsilon}{3\varepsilon}$

$$\boxed{\Lambda > \frac{1}{3\varepsilon} - 1}$$

3.d. Binary Search, want ε close

1/3 cut n times in a binary/halving manner until our estimate is ε close

0 1/3 is the space between 1/3 & 2/3

$$\frac{\frac{1}{3}}{2^n} < \varepsilon \Rightarrow \frac{1}{3 \cdot 2^n} < \varepsilon$$

$$\Rightarrow \frac{1}{3\varepsilon} < 2^n$$

$$\ln\left(\tfrac{1}{3\varepsilon}\right) < n \cdot \ln(2)$$

$$n > \frac{\ln\left(\frac{1}{3\varepsilon}\right)}{\ln(2)}$$

n is the min # of guesses required to get ε close.

3.e. Need to be specific

Methods ① ② ③

b. c. d.

	Random ①	Deterministic ②	adaptive ③
$\wedge >$	$\frac{\ln(\delta)}{\ln(2(1-\varepsilon))}$	$\frac{1}{3\varepsilon} - 2$	$\frac{\ln(1/\varepsilon)}{\ln(2)}$

① $\wedge, \varepsilon$ · as ε approaches 2, $\wedge$ approaches a large finite #

$\wedge, \delta$ · as δ approaches 2, $\wedge$ approaches 0.

- "$\wedge$ is proportional to $\ln(\delta)$ $\log(\delta)$
- $\wedge$ is inversely proportional to ε

② $\wedge, \varepsilon$ · . $\wedge$ is inverse to ε ? adjust case pat.

③ $\wedge, \varepsilon$ · $\wedge$ is proportional to $\ln(1/\varepsilon)$ $\log(\varepsilon)$

3.f. Write sorry up.

rand better than deterministic.

f. Different algorithms take different amount of training data and iterations to achieve similar estimates. Some may be more appropriate for different situations/cases. This problem exercised our probability and had an emphasis on how lots of data helps improve the accuracy of learning.

4.

4. Norms = 1 for this question

4.a.

i) $A = \begin{bmatrix} 2 & -4 \\ -1 & -1 \end{bmatrix}$

right eig.

$\begin{bmatrix} 2-\lambda & -4 \\ -1 & -1-\lambda \end{bmatrix}$ $(2-\lambda)(1-\lambda) - 4 = 0$

$\lambda = 3, \lambda = -2$

eigvector $(A-\lambda I)x = 0$ $\lambda_1 = 3$ $\left[\begin{array}{cc|c} -1 & -4 & 0 \\ -1 & -4 & 0 \end{array}\right]$

$-x_1 - 4x_2 = 0$ $x_1 = -4$ $x_2 = 1$ $\vec{x}_1 = \begin{bmatrix} -4 \\ 1 \end{bmatrix}$

$\lambda = -2$ $\left[\begin{array}{cc|c} +4 & -4 & 0 \\ -1 & 1 & 0 \end{array}\right]$

$-x_1 + x_2 = 0$ $x_1 = x_2$ $x_1 = 1$ $x_2 = 1$ $\vec{x}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

$x_1 - x_2 = 0$

left $x^T A = \lambda x^T \rightsquigarrow A^T x = x\lambda$

same eigenvalues, diff eigenvectors

$A^T = \begin{bmatrix} 2 & -1 \\ -4 & -1 \end{bmatrix}$ $(A^T - \lambda I)x = 0$ $\lambda_1 = 3$ $\left[\begin{array}{cc|c} -1 & -1 & 0 \\ -4 & -4 & 0 \end{array}\right]$

$-x_1 - x_2 = 0$

$-x_1 = x_2$

$x_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

$\lambda = -2$ $\left[\begin{array}{cc|c} -4 & 1 & 0 \\ -4 & 1 & 0 \end{array}\right]$

$-4x_1 + x_2 = 0$ $x_2 = 4x_1$ $x_1 = 1$ $x_2 = 4$

$x_2 = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$

ii) $B = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$ $|(A - \lambda I)|$

symmetric ∴ right eigval/vectors = left eigval/vectors

$$\left| \begin{bmatrix} 3-\lambda & 1 \\ 1 & 3-\lambda \end{bmatrix} \right| = 0 \qquad (3-\lambda)^2 - 1 = 0$$

$$\lambda = 4, \ \lambda = 2$$

$\lambda = 4$ $\begin{bmatrix} -1 & +1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\vec{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

$\lambda = 2$ $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\vec{x} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

- when A a matrix is squared its eigenvalues become squared also & its eigenvectors remain the same.

Proof:

$$A^2 x = A(Ax) = A(\lambda x) = \lambda(Ax) = \lambda(\lambda x) = \lambda^2 x$$

$\begin{bmatrix} A x_1 = x_1 \\ \forall \wedge \end{bmatrix}$

eigenvectors stay in own direction & never get mixed.

- eigenvalues are put to the same power as the matrix.

iii) A^2 right $\begin{cases} \text{eigenvalues: } \lambda_1 = 9, \ \lambda_2 = 4 \\ \text{eigenvectors: } x_1 = \begin{bmatrix} -4 \\ 1 \end{bmatrix} \quad x_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \end{cases}$

left $\begin{cases} \text{eigenval: } \lambda_1 = 9, \ \lambda_2 = 4 \\ \text{eigenvectors } \vec{x}_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad \vec{x}_2 = \begin{bmatrix} 1 \\ 4 \end{bmatrix} \end{cases}$

iv) B^2 Right eigenvectors = left eigenvectors

eigenvalues: $\lambda_1 = 16 \qquad \lambda_2 = 4$

eigenvectors: $x_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad x_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

v) & vi) in python jupyter script.

4.5. SVD.

$$G = U\Sigma V^T \begin{cases} G^TG = V\Sigma^T\Sigma V^T & (1) \\ GV = U\Sigma & (2) \end{cases}$$

SVD for B

$$B^TB = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}\begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} 10 & 6 \\ 6 & 10 \end{bmatrix} \qquad \left|\begin{bmatrix} 10-\lambda & 6 \\ 6 & 10-\lambda \end{bmatrix}\right| = 0$$

$$V = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$\lambda_1 = 16 \quad \lambda_2 = 4$

$\vec{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \vec{x}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

$$\Sigma = \begin{bmatrix} \sqrt{16} & 0 \\ 0 & \sqrt{4} \end{bmatrix} = \begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix}$$

$$BV = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 4 & 2 \\ 4 & -2 \end{bmatrix} = U\Sigma$$

Need normalized

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = U$$

$$B = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$ Nom of 2.

Now by similar logic for symmetric Square Matrix

$$B^2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 16 & 0 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

SVD for others to be found in the python script in Jupyter.

C. Prove/show both directions

$Ax = \lambda x$

$$\underset{n\times n}{\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \\ a_{n1} & & a_{nn} \end{bmatrix}} \underset{n\times 1}{\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}} = \lambda \underset{n\times 1}{\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}}$$

$$\underset{n\times 1 \text{ matrix}}{\begin{bmatrix} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ \vdots \\ a_{n1}x_1 + \cdots + a_{nn}x_n \end{bmatrix}} = \begin{bmatrix} \lambda x_1 \\ \lambda x_2 \\ \vdots \\ \lambda x_n \end{bmatrix}$$

row i of ↑ system of linear eqns above.

For $i \in n$, $i = 1, \ldots, n$:

$$\sum_{j=1}^{n} A_{ij} = \lambda x_i \quad \Rightarrow \quad \lambda x_i = \sum_{j \neq i} A_{ij} x_j + a_{ii} x_i$$

$$= \lambda x_i - a_{ii} x_i = \sum_{j \neq i} A_{ij} x_j$$

$$x_i(\lambda - a_{ii}) = \sum_{j \neq i} A_{ij} x_j$$

Represents each row of the syst. of lin eqns which composes $Ax = \lambda x$, the definition of a right eigenvalue/eigenvector

U.D.

i chosen s.t $|x_i| \geq |x_j|$ for all $j \neq i \Rightarrow |x_i| > 0$

$\therefore x_i \neq 0$ & also $\frac{|x_j|}{|x_i|} \leq 1$ for $j \neq i$

$$|\lambda - a_{ii}| = \left| \frac{\sum_{j \neq i} a_{ij} x_j}{x_i} \right|$$

(x_i constant)

$$\leq \sum_{j \neq i} \left| \frac{a_{ij} x_j}{x_i} \right| \leq \sum_{j \neq i} |a_{ij}|$$

less than or equal as we need the absolute val s.t.s.s to encompass more

reason for inequality here: triangle inequality $|a+b| \leq |a| + |b|$

as stated earlier $\frac{|x_j|}{|x_i|} \leq 1$ for $j \neq i$

Python Calculated Results:

```
A = [[2, -4], [-1, -1]]
B = [[3, 1], [1,3]]
```

```
G = np.multiply(A, B)
```

```
LA.eig(G)
```

```
(array([ 6.4244289, -3.4244289]), array([[ 0.99441772,  0.39069532],
        [-0.10551491,  0.92052005]]))
```

```
D = np.multiply(B, A)
```

```
LA.eig(D)
```

```
(array([ 6.4244289, -3.4244289]), array([[ 0.99441772,  0.39069532],
        [-0.10551491,  0.92052005]]))
```

```
np.linalg.svd(A)
```

```
(array([[ 0.99402894,  0.10911677],
        [ 0.10911677, -0.99402894]]),
 array([ 4.49661478,  1.33433712]),
 array([[ 0.41785681, -0.9085129 ],
        [ 0.9085129 ,  0.41785681]]))
```

```
np.linalg.svd(np.multiply(A,A))
```

```
(array([[-0.99729721, -0.07347291],
        [-0.07347291,  0.99729721]]),
 array([ 16.5370324,   0.7256441]),
 array([[-0.24567055, -0.96935338],
        [ 0.96935338, -0.24567055]]))
```

```
#G = AB
np.linalg.svd(G)
```

```
(array([[-0.99033427, -0.13870121],
        [-0.13870121,  0.99033427]]),
 array([ 7.26913541,  3.02649473]),
 array([[-0.79834865,  0.60219551],
        [-0.60219551, -0.79834865]]))
```

```
#D = BA
np.linalg.svd(D)
```

```
(array([[-0.99033427, -0.13870121],
        [-0.13870121,  0.99033427]]),
 array([ 7.26913541,  3.02649473]),
 array([[-0.79834865,  0.60219551],
        [-0.60219551, -0.79834865]]))
```

```
C = [[3, 1], [1, 3], [2, -4], [-1, -1]]
np.linalg.svd(C)
```

```
(array([[-0.14395373,  0.79545124, -0.53046828,  0.25522159],
        [-0.5586541 ,  0.32209809,  0.73516449,  0.20902533],
        [ 0.79770252,  0.4306424 ,  0.42187759, -0.01539876],
        [ 0.17565196, -0.27938733, -0.01248516,  0.94389259]]),
 array([ 5.20411016,  3.86228397]),
 array([[ 0.08248053, -0.99659268],
        [ 0.99659268,  0.08248053]]))
```

By Hand Results:

5.

a. *Supervised Learning.* Our machine learning algorithm is given n samples (x_i, y_i) from which to develop the model. This is called the training set. We are inferring a function / developing a model from training data hence this is supervised learning.

$x_i = [$ observation i w/ d Features $]$

5. b.

$\hat{y}^t$, $t = 1, \ldots, d$ $\quad \hat{y} = X^T \hat{w}$

$(x)_j$, $j = 1, \ldots, t$ • each observation x_i w/ t Features
• optimal given t

Model: $\hat{y}^t = X^T \hat{w}^t$ $\quad X \in \mathbb{R}^{t \times 2}$ $\quad \hat{w} \in \mathbb{R}^{t \times 2}$

• use OLS w/ first t to get our measures.

$$\hat{w}^t = (X^T X)^{-1} X^T \vec{y}$$

Use polynomial model $X = \begin{bmatrix} (x_1^T)_1 \ldots (x_1^T)_t \\ (x_n)_1^T \ldots (x_n^T)_t \end{bmatrix}$

$\hat{y}^t = w_1 x_1 + \ldots + w_t x_t$ where each x_i is an observation @ time i For velocity

S.C. Stage 1: $x_1, \ldots, x^{t-1}$ linearly to x^t using least squares for each t → $\Delta x_1, \ldots, \Delta x_t$ w/ $\Delta x_i = (x)_t - (\hat{x})_t$

"More more data"

∴ Stage 2: OLS But w/ the Δx_i $i = 1, \ldots, t$ to predict y^t w/ $\hat{y}^t$

Stage 1 predict x^t as $\hat{x}^t$ from $x_1, \ldots, x_{t-1}$

$$A = \begin{bmatrix} \vdots & & \vdots \\ x^1 & \cdots & x^{t-1} \\ \vdots & & \vdots \end{bmatrix} \qquad A\hat{w}^{t-1} - x^t$$

Opt Stage 1 $\min_w \| A\hat{w}^{t-1} - \vec{x}^t \|_2^2$

Sol^n Stage 1 $\hat{w}^{t-1} = (A^T A)^{-1} A^T \vec{x}^t$

Stage 2

$$\tilde{X}^t = \begin{bmatrix} \vdots & & \vdots \\ x^1 & \cdots & x^t \\ \vdots & & \vdots \end{bmatrix} - \begin{bmatrix} \vdots & & \vdots \\ \hat{x}^1 & \cdots & \hat{x}^t \\ \vdots & & \vdots \end{bmatrix} = \begin{bmatrix} \Delta x^1 & \cdots & \Delta x^t \end{bmatrix}$$

$\tilde{X}^t w = \tilde{y}$ Opt Stage 2 $\min_w \| \tilde{X}^t w^t - \vec{y}^t \|_2^2$

Sol^n Stage 1 $\hat{w}^t = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T \vec{y}^t$

Explain $\hat{y}^t$ $\vec{y}^t$ the same?

No. Each predicted $\hat{x}^t$ is a function of the preceding $x_1, \ldots x_{t-1}$, Δx^t is a function of $\hat{x}^t$ & x^t ∴ it is a function of all preceding $x_1, \ldots, x_{t-1}$ & the true x^t. (Cont. Next page)

S.C. Explain (cont.) Thus Δx^i for $i=1,\dots,t$ is simply a linear combination of x^i for $i=1,\dots,t$.

ex. For self $\hat{x}^3$ is =

$$\hat{x}^3 = \alpha_1 x_1 + x_2 x_2$$

$$\Delta x^3 = x^3 - \hat{x}^3$$

∴ our solution for OLS w/ $\hat{x}^t$ will have different weights, & thus $\hat{\vec{y}}^t$ is different than $\vec{y}^t$.

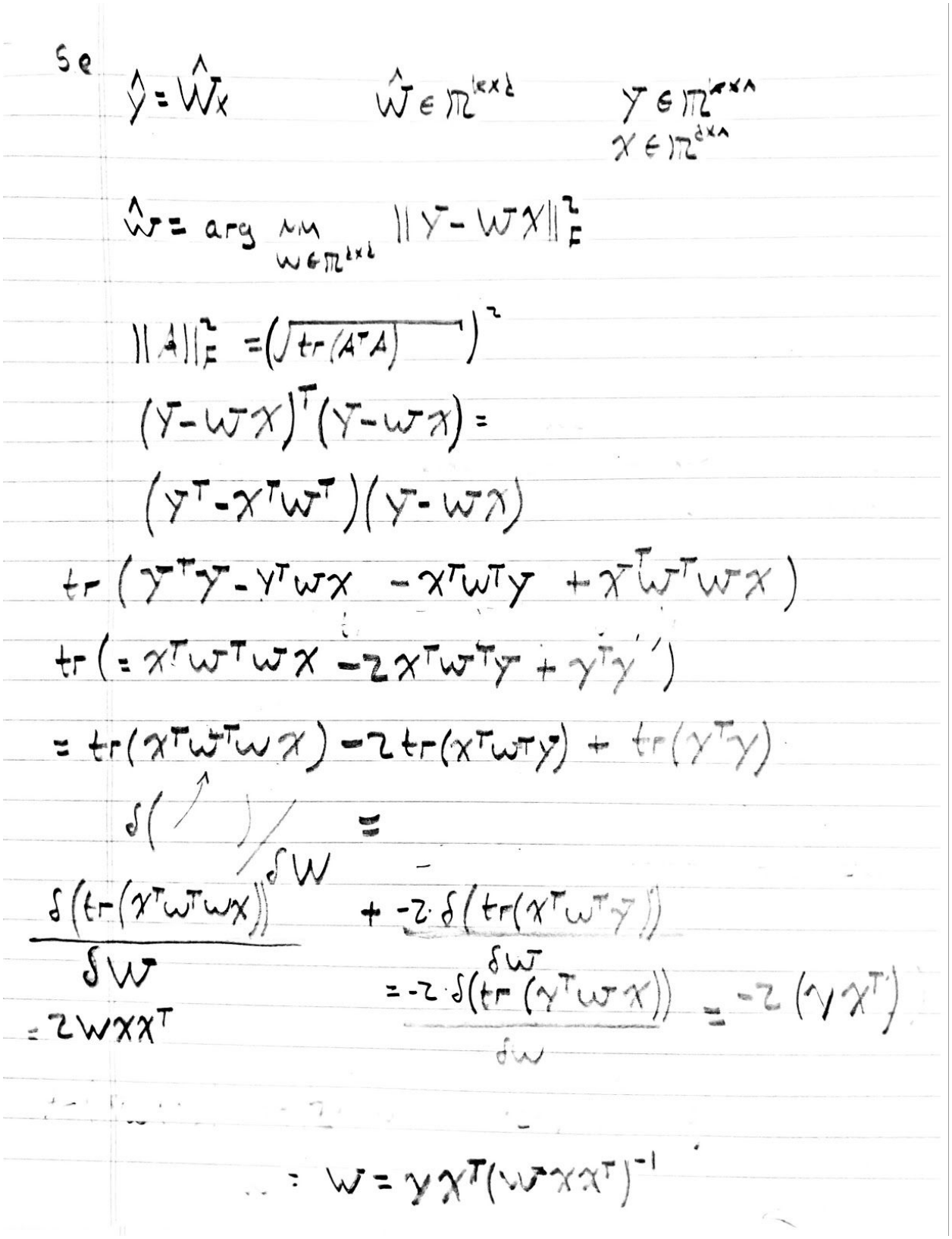
$$A = \begin{bmatrix} d & e & f \end{bmatrix}, \quad A = \begin{bmatrix} d & 2e+f & 3f+2d+e \end{bmatrix}$$

$Ax = b$

these two A would have different weights.

S.D. Graham

D. **First Part:** Gram-Schmidt. **Second Part:** Yes: Neural Net weight back Propagation.



S.F. Each is independent from another

Using connects $(Y)_j$ For $j \neq i$ $\therefore$ connects $(Y)_i$

$\therefore$ $\ln(Y)_i$, the minimizing equation, are independent from one another.

$\therefore$ can be solved separately to find best linear prediction. For each $(Y)_j$ of target vector $\vec{Y}$.

6.

A.

```
A is   0.977552135184
B is   -0.0877532187735
```

B.

```
A is  [array([ 0.15207406,  0.03893567, -0.52552959]), array([ 0.93480864,  0.30958727,  0.0540906 ]), array([-0.0011
0243,  0.87436511, -0.47026217])]

B is  [array([ 0.04894161, -0.04524735,  0.91096923]), array([ 0.20568264, -0.92861546, -0.47124981]), array([-0.3709
0438,  0.12756569, -0.84222314])]
```

C.

```
Fitted dynamical system:
xdd_i = -0.012 x_i + -0.318 xd_i + 0.011 x_i-1 + 0.275 xd_i-1 + -0.883

array([-0.01152121, -0.31779341,  0.01128933,  0.27535824, -0.88293502])
```

7.

Output:

e. Report what percentage of digits are correctly classified using regression targets 0/1 and −1/1 with bias on the training set and test set. Try to explain the results!

The biased values give us the same results. Our values are the same due to the binary nature of our data. The column of ones simply shifts the weights.

8.

Question:

Is there a topic that machine learning currently cannot classify well?

Sources

Jupyter Notebook

http://localhost:8888/notebooks/School/Spring%202018/CS189/HW01%20-%20Code.ipynb

Coupon Collector Wiki
https://en.wikipedia.org/wiki/Coupon_collector%27s_problem

https://victor-bernal.weebly.com/uploads/5/3/6/9/53696137/gershgorin_circle_theorem.pdf

http://prob140.org/sp17/textbook/ch5/TheMatchingProblem.html

https://math.stackexchange.com/questions/28930/another-balls-and-bins-question

http://cal.cs.illinois.edu/~johannes/research/matrix%20calculus.pdf

http://snasiriany.me/files/ml-book.pdf

In [295]:
```
import numpy as np
import math
import scipy.io
import random
import sklearn
import numpy.linalg as LA
import time
import matplotlib.pyplot as plt
```

4.A V, VI

In [99]:
```
A = [[2, -4], [-1, -1]]
B = [[3, 1], [1,3]]
```

In [53]:
```
LA.eig(A)
```

Out[53]:
```
(array([ 3., -2.]), array([[ 0.9701425 ,  0.70710678],
        [-0.24253563,  0.70710678]]))
```

In [ ]:
```
# Do we have to adjust the scale of the eigenvectors based off of the fa
ct that the norm is mentioned to be = 1 ??
```

In [48]:
```
G = np.multiply(A, B)
```

In [49]:
```
LA.eig(G)
```

Out[49]:
```
(array([ 6.4244289, -3.4244289]), array([[ 0.99441772,  0.39069532],
        [-0.10551491,  0.92052005]]))
```

In [29]:
```
D = np.multiply(B, A)
```

In [30]:
```
LA.eig(D)
```

Out[30]:
```
(array([ 6.4244289, -3.4244289]), array([[ 0.99441772,  0.39069532],
        [-0.10551491,  0.92052005]]))
```

4.B

$C = U\Sigma V^T$

Therefore:

$C^T C = V\Sigma^T \Sigma V^T$

$CV = U\Sigma$

In [46]:
```
np.linalg.svd(A)
```

Out[46]:
```
(array([[ 0.99402894,  0.10911677],
        [ 0.10911677, -0.99402894]]),
 array([ 4.49661478,  1.33433712]),
 array([[ 0.41785681, -0.9085129 ],
        [ 0.9085129 ,  0.41785681]]))
```

In [42]:
```
np.linalg.svd(np.multiply(A,A))
```

Out[42]:
```
(array([[-0.99729721, -0.07347291],
        [-0.07347291,  0.99729721]]),
 array([ 16.5370324,   0.7256441]),
 array([[-0.24567055, -0.96935338],
        [ 0.96935338, -0.24567055]]))
```

In [50]:
```
#G = AB
np.linalg.svd(G)
```

Out[50]:
```
(array([[-0.99033427, -0.13870121],
        [-0.13870121,  0.99033427]]),
 array([ 7.26913541,  3.02649473]),
 array([[-0.79834865,  0.60219551],
        [-0.60219551, -0.79834865]]))
```

In [43]:
```
#D = BA
np.linalg.svd(D)
```

Out[43]:
```
(array([[-0.99033427, -0.13870121],
        [-0.13870121,  0.99033427]]),
 array([ 7.26913541,  3.02649473]),
 array([[-0.79834865,  0.60219551],
        [-0.60219551, -0.79834865]]))
```

In [52]:
```
C = [[3, 1], [1, 3], [2, -4], [-1, -1]]
np.linalg.svd(C)
```

Out[52]:
```
(array([[-0.14395373,  0.79545124, -0.53046828,  0.25522159],
        [-0.5586541 ,  0.32209809,  0.73516449,  0.20902533],
        [ 0.79770252,  0.4306424 ,  0.42187759, -0.01539876],
        [ 0.17565196, -0.27938733, -0.01248516,  0.94389259]]),
 array([ 5.20411016,  3.86228397]),
 array([[ 0.08248053, -0.99659268],
        [ 0.99659268,  0.08248053]]))
```

6.a

In [283]:
```
mdict = scipy.io.loadmat("a.mat")

x = mdict['x']
u = mdict['u']

X = np.transpose([x[0][:-1], u[0][:-1]]) # Features Matrix, 2 features
 x, u

a, b = np.linalg.lstsq(X, x[0][1:])[0]
print("A is ", a)
print("B is ", b)
```

```
A is  0.977552135184
B is  -0.0877532187735
```

In [ ]:
```

```

6.b

In [291]:
```
mdict = scipy.io.loadmat("b.mat")

x = mdict['x']
u = mdict['u']

x1 = []
x2 = []
x3 = []
u1 = []
u2 = []
u3 = []


for i in range(len(x)):
    x1.append(x[i][0][0])
    x2.append(x[i][1][0])
    x3.append(x[i][2][0])
    u1.append(u[i][0][0])
    u2.append(u[i][1][0])
    u3.append(u[i][2][0])

X = np.transpose([x1[:-1], x2[:-1], x3[:-1], u1[:-1], u2[:-1], u3[:-1]])

truex = np.transpose([x1[1:], x2[1:], x3[1:]])

D = np.linalg.lstsq(X, truex)[0]
A = [D[0], D[1], D[2]]
B = [D[3], D[4], D[5]]
print("A is ", A)
print("                ")
print("B is ", B)
```

```
A is  [array([ 0.15207406,  0.03893567, -0.52552959]), array([ 0.934808
64,  0.30958727,  0.0540906 ]), array([-0.00110243,  0.87436511, -0.470
26217])]

B is  [array([ 0.04894161, -0.04524735,  0.91096923]), array([ 0.205682
64, -0.92861546, -0.47124981]), array([-0.37090438,  0.12756569, -0.842
22314])]
```

In [ ]:

In [ ]:

6.c

In [292]:
```
mdict = scipy.io.loadmat("train.mat")

# Assemble xu matrix
x = mdict["x"]   # position of a car
v = mdict["xd"]  # velocity of the car
xprev = mdict["xp"]   # position of the car ahead
vprev = mdict["xdp"]  # velocity of the car ahead

acc = mdict["xdd"]  # acceleration of the car

one = []
for i in range(len(x[0])):
    one.append(1)

a, b, c, d, e = 0, 0, 0, 0, 0

X = np.transpose([x[0], v[0], xprev[0], vprev[0], one])

values = np.linalg.lstsq(X, acc[0])[0]
a = values[0]
b = values[1]
c = values[2]
d = values[3]
e = values[4]



print("Fitted dynamical system:")
print("xdd_i = {:.3f} x_i + {:.3f} xd_i + {:.3f} x_i-1 + {:.3f} xd_i-1 +
 {:.3f}".format(a, b, c, d, e))
values
```

```
Fitted dynamical system:
xdd_i = -0.012 x_i + -0.318 xd_i + 0.011 x_i-1 + 0.275 xd_i-1 + -0.883
```

Out[292]:
```
array([-0.01152121, -0.31779341,  0.01128933,  0.27535824, -0.8829350
2])
```

6.d

Why is this reasonable:

Our results are physically reasonable. Assume the i-1 car is in front of the ith car.

$+h(x_{i-1} - x_i)$

The car in the i-1 place in line will be ahead of car i. If not the positive sign means that the acceleration will be positively influenced.

$+f(\dot{x}_{i-1} - \dot{x}_i)$

The velocity of the car in the i position is kept in check by the velocity of the next car in line, the i - 1 th position. This has a positive coefficient as a positive difference in velocity between the two cars will contribute to the ith car increasing its velocity.

$-g(\dot{x}_i - L)$

Speed is kept in check by the speed limit. If the ith car is above the speed limit than the acceleration of the car will be decrease due to the negative sign of this function.

$+W_i$

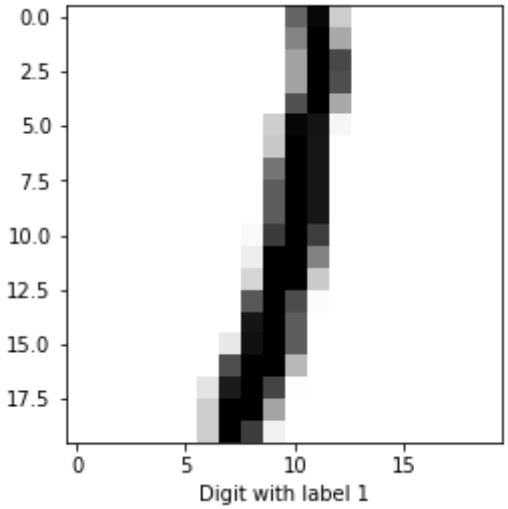
Extra factor which accounts for situations not covered by the previous functions. These factors are not explicitly stated our modeled but can have an affect on our acceleration.

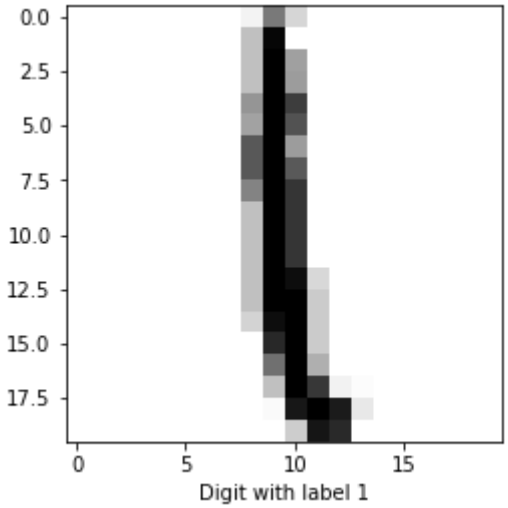
7

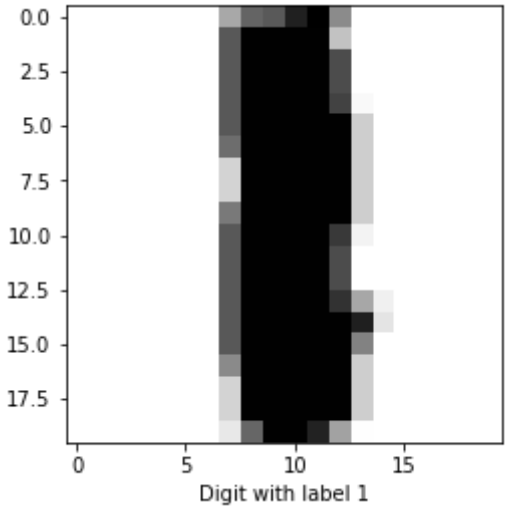
In [479]:
```
# Load the training dataset
train_features = np.load("train_features.npy")
train_labels = np.load("train_labels.npy").astype("int8")

n_train = train_labels.shape[0]

def visualize_digit(features, label):
    # Digits are stored as a vector of 400 pixel values. Here we
    # reshape it to a 20x20 image so we can display it.
    plt.imshow(features.reshape(20, 20), cmap="binary")
    plt.xlabel("Digit with label " + str(label))
    plt.show()

# Visualize a digit
# visualize_digit(train_features[0,:], train_labels[0])

# TODO: Plot three images with label 0 and three images with label 1
visualize_digit(train_features[0,:], train_labels[0])
visualize_digit(train_features[7,:], train_labels[7])
visualize_digit(train_features[400,:], train_labels[400])
visualize_digit(train_features[5,:], train_labels[5])
visualize_digit(train_features[6,:], train_labels[6])
```

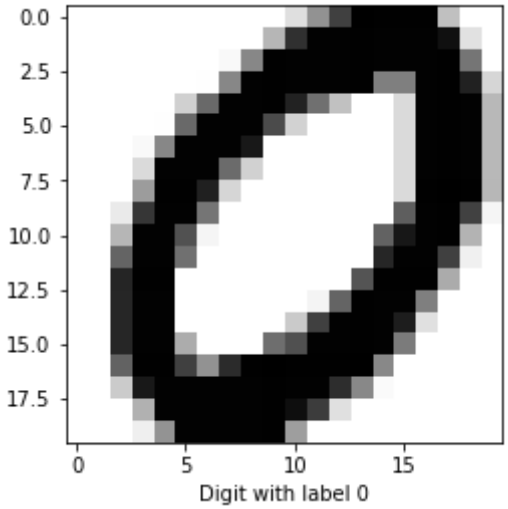
```
visualize_digit(train_features[3,:], train_labels[3])

# Linear regression

# TODO: Solve the linear regression problem, regressing
# X = train_features against y = 2 * train_labels - 1

X = train_features
y = 2 * train_labels - 1

W = np.linalg.lstsq(X, y)[0]

# TODO: Report the residual error and the weight vector

A = np.dot(X, W) - y
# ||A||_2^2 = A^TA
error = np.dot(A.T, A)

print("Error is: ", error)
print()
print()
print("Weight Vectors for W shown below")
print()
for i in range(200):
    print(W[i])

print()
print()

# Load the test dataset
# It is good practice to do this after the training has been
# completed to make sure that no training happens on the test
# set!
test_features = np.load("test_features.npy")
test_labels = np.load("test_labels.npy").astype("int8")

n_test = test_labels.shape[0]

# TODO: Implement the classification rule and evaluate it
# on the training and test set
success_training = 0
for i in range(len(train_features)):
    if np.dot(train_features[i], W) <= 0:
        if train_labels[i] == 0:
            success_training = success_training + 1
    else:
        if train_labels[i] == 1:
            success_training = success_training + 1

print("Training Success Percentage: ", success_training/len(train_featur
es))

success = 0
for i in range(len(test_features)):
    if np.dot(test_features[i], W) <= 0:
        if test_labels[i] == 0:
            success = success + 1
```

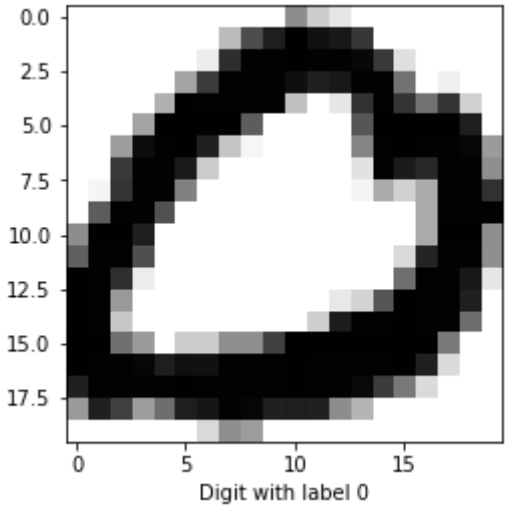
```
    else:
        if test_labels[i] == 1:
            success = success + 1

print("Test Success Percentage: ", success/len(test_features))

print()
print()

print("7.d Why is the performance typically evaluated on a separate test
 set (instead of the training \
set) and why is the performance on the training and test set similar in
 our case? \
\ If we evaluate our performance only on our training set than there is
 the potential to make our model overly \
complicated, specific. We could build a super complicated model which fi
ts our training data exactly. \
- Not overly deep, complicated. We want our model to be generalizable to
 more than just the training data \
- Our results are similar because our model is gooood.")

print()
print()

# TODO: Try regressing against a vector with 0 for class 0
# and 1 for class 1

W = np.linalg.lstsq(train_features, train_labels)[0]

success_training = 0
for i in range(len(train_features)):
    if np.dot(train_features[i], W) <= 0.5:
        if train_labels[i] == 0:
            success_training = success_training + 1
    else:
        if train_labels[i] == 1:
            success_training = success_training + 1

print("01 Training Success Percentage: ", success_training/len(train_fea
tures))

success = 0
for i in range(len(test_features)):
    if np.dot(test_features[i], W) <= 0.5:
        if test_labels[i] == 0:
            success = success + 1
    else:
        if test_labels[i] == 1:
            success = success + 1

print("01 Test Success Percentage: ", success/len(test_features))
```

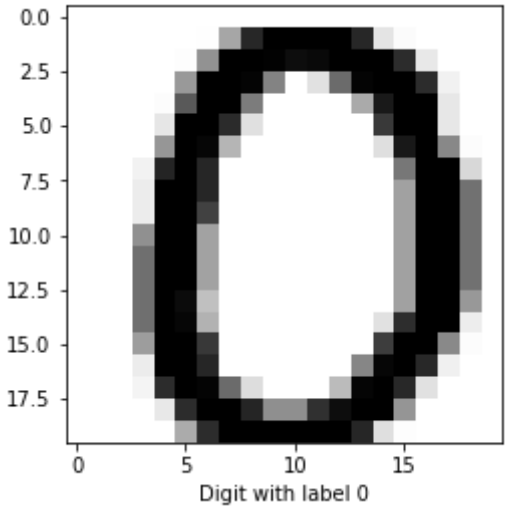

0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
5
10
15
Digit with label 1


0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
5
10
15
Digit with label 1


0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
5
10
15
Digit with label 1


0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
5
10
15
Digit with label 0


0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
5
10
15
Digit with label 0


0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
5
10
15
Digit with label 0

```
Error is:  422.750808692


Weight Vectors for W shown below

-0.330801139067
0.391726635637
0.148153633633
-0.160602318237
0.103277283927
-0.0196941103461
-0.127705113522
0.00945889276034
-0.01714938929
-0.00567518078931
-0.00469069785683
-0.0112641776158
-0.00571029120205
0.00459008400838
0.0178762857174
-0.0300976493609
0.0100373536309
-0.0645811914383
-0.0230413878641
-0.0263012485407
-0.163458897988
0.467076721165
-0.00282694075885
-0.105642710097
-0.180703185337
0.134413186668
-0.00509450682459
-0.0307269165653
-0.065922517274
0.0176021545422
-0.0306512636621
-0.00622918717754
0.0154059878049
-0.0313653554151
-0.00252826818938
-0.00617331537108
-0.00102623406197
0.0519405642678
0.0395632027087
0.0629594619743
-0.388408602963
-0.216922477513
-0.0980431113102
0.16833019762
-0.0570891161332
0.0150769252225
0.00243554088004
0.027297489253
0.0630415689521
-0.0269231967308
0.00773327626591
-0.0171158562057
```

```
-0.0502592486225
0.0080624049159
-0.00638849770176
-0.0126750593869
0.01550503834
-0.0106068843015
-0.00769542916973
-0.0418694971486
0.526520950665
0.231919517894
-0.0841796993503
0.0990934563876
-0.0396223757309
-0.0231646770501
0.00678329096398
-0.0528093878894
0.0226991410345
-0.018407990049
0.00287996076144
-0.0106359104441
0.0226940905982
-0.0303847987549
-0.0173679041618
0.0212025184736
0.0333768908626
-0.0348220335491
-0.0271485105544
-0.0614572380678
-0.208387559994
-0.0703671783003
-0.0645086499583
-0.0542103612209
0.0214511580527
-0.028340506626
-0.0276361408433
0.0108086806081
-0.0384239992985
0.0286567441237
-0.0290154300274
0.0059313685302
-0.00747531063565
-0.00910032759537
-0.0119273841164
-0.0155744010839
-0.0203919740993
-0.00655518086765
0.0485632937648
-0.0395774272441
-0.0330413742074
0.0184931206369
0.0831143892965
0.00948879906862
0.0104639995334
0.0205530104406
-0.0289796769491
0.0120898555634
0.0113589550196
```

```
-0.0136114833539
0.0148536638427
0.00462730991953
0.00368167942365
0.00735567039115
0.0230061097222
-0.0149416999955
-0.0577568488483
0.025588117549
-0.0123854511459
-0.0355654725958
0.0560248487911
-0.0786040799355
-0.0550626530272
-0.00380891487315
0.0205816647634
-0.00804723683711
0.000783590240448
0.00585386506775
-0.0378688889988
0.000250237080805
-0.00485700059929
0.00712657738386
-0.00608274230773
-0.010657075584
-0.0451331886972
0.00735493612382
-0.00057868705943
-0.0372889916617
0.022406679157
-0.0355815656456
0.12521177345
0.0323686421374
0.0273657228593
-0.0378581831144
-0.0284901406238
-0.00255289913368
-0.0319249773517
0.031373897536
0.0419972228545
0.00749288536592
0.0251217726307
0.00645677307361
0.00620250982717
-0.00207480713648
0.00590324803465
-0.00857605436429
0.00290282976325
0.0202997888272
-0.085950583538
-0.0089748803241
-0.0560652860416
0.054265720819 8
-0.0196536504658
0.0573922648744
-0.0216765555599
0.00790975732468
```

```
-0.00528758319515
-0.0776409307042
-0.0507018915314
-0.0274831121117
0.0607874106215
0.0488354240651
-0.0240432307702
-0.0498371694512
-0.0049346861192
-0.0480952809601
-0.041449690557
-0.060052651498
-0.0163114576278
-0.0578013508098
-0.104833519767
-0.0549018705863
0.0114533924154
-0.046290080335
0.00994957314128
-0.00229688096279
-0.0504550302303
0.063338708048
0.00883330138495
0.0128803088872
0.0598684110336
0.0508413486622
0.055565171045
0.0324115536361
0.0018571420473
-0.0065132956433
0.0207802344264
0.0416188717442
0.0511904721232
0.00870418105576


Training Success Percentage:  0.9975909833949926
Test Success Percentage:  0.9981087470449173
```

7.d Why is the performance typically evaluated on a separate test set (instead of the training set) and why is the performance on the trainin g and test set similar in our case? \ If we evaluate our performance on ly on our training set than there is the potential to make our model ov erly complicated, specific. We could build a super complicated model wh ich fits our training data exactly. - Not overly deep, complicated. We want our model to be generalizable to more than just the training data - Our results are similar because our model is gooood.

```
01 Training Success Percentage:  0.9896756431213972
01 Test Success Percentage:  0.9914893617021276
```

In [470]:
```
# TODO: Form a new feature matrix with a column of ones added
# and do both regressions with that matrix
```

```
# Linear regression 2 (with new feature matrix)

# TODO: Solve the linear regression problem, regressing
# X = train_features against y = 2 * train_labels - 1


ones = np.ones((train_features.shape[0], 1))
ones2 = np.ones((test_features.shape[0], 1))
X = np.hstack((train_features, ones))
X2 = np.hstack((test_features, ones2))

y = 2 * train_labels - 1

W = np.linalg.lstsq(X, y)[0]


# Load the test dataset
# It is good practice to do this after the training has been
# completed to make sure that no training happens on the test
# set!
test_features = np.load("test_features.npy")
test_labels = np.load("test_labels.npy").astype("int8")

n_test = test_labels.shape[0]

# TODO: Implement the classification rule and evaluate it
# on the training and test set
success_training = 0
for i in range(len(X)):
    if np.dot(X[i], W) <= 0:
        if train_labels[i] == 0:
            success_training = success_training + 1
    else:
        if train_labels[i] == 1:
            success_training = success_training + 1

print("Xprime Training Success Percentage: ", success_training/len(X))

success = 0
for i in range(len(X2)):
    if np.dot(X2[i], W) <= 0:
        if test_labels[i] == 0:
            success = success + 1
    else:
        if test_labels[i] == 1:
            success = success + 1

print("Xprime Test Success Percentage: ", success/len(X2))

print()
print()

# TODO: Try regressing against a vector with 0 for class 0
# and 1 for class 1

W = np.linalg.lstsq(X, train_labels)[0]
```

```
success_training = 0
for i in range(len(X)):
    if np.dot(X[i], W) <= 0.5:
        if train_labels[i] == 0:
            success_training = success_training + 1
    else:
        if train_labels[i] == 1:
            success_training = success_training + 1

print("01 Xprime Training Success Percentage: ", success_training/len(X
))

success = 0
for i in range(len(X2)):
    if np.dot(X2[i], W) <= 0.5:
        if test_labels[i] == 0:
            success = success + 1
    else:
        if test_labels[i] == 1:
            success = success + 1

print("01 Xprime Test Success Percentage: ", success/len(X2))




# Logistic Regression

# You can also compare against how well logistic regression is doing.
# We will learn more about logistic regression later in the course.

#import sklearn.linear_model

#lr = sklearn.linear_model.LogisticRegression()
#lr.fit(X, train_labels)

#test_error_lr = 1.0 * sum(lr.predict(test_features) != test_labels) / n
_test
```

```
Xprime Training Success Percentage:  0.9941495311021251
Xprime Test Success Percentage:  0.9962174940898345


01 Xprime Training Success Percentage:  0.9941495311021251
01 Xprime Test Success Percentage:  0.9962174940898345
```

In [ ]:

In [ ]:

In [ ]:

2 min to Spreed