HW 2

Lisa Yu
1005786366

1) $pr(t = spam) = 0.1$

a) $E[\mathcal{L}(y,t)] = \sum_{(y,t)} \mathcal{L}(y,t) Pr(y,t)$

$= \mathcal{L}(keep, Spam) Pr(y = keep, t = spam)$
$+ \mathcal{L}(keep, NonSpam) Pr(y = keep, t = NonSpam)$
$+ \mathcal{L}(Remove, Spam) Pr(y = Remove, t = Spam)$
$+ \mathcal{L}(Remove, NonSpam) Pr(y = Remove, t = NonSpam)$

keeping every email $\Rightarrow Pr(y = keep) = 1$:

$E[\mathcal{L}(y,t)] = \mathcal{L}(keep, Spam) Pr(y = keep) Pr(t = spam)$
$+ \mathcal{L}(keep, NonSpam) Pr(y = keep) Pr(t = NonSpam)$
$+ \mathcal{L}(Remove, Spam) Pr(y = Remove) Pr(t = Spam)$
$+ \mathcal{L}(Remove, NonSpam) Pr(y = Remove) Pr(t = NonSpam)$
$= (1)(1)(0.1)$
$+ (0)(1)(0.9)$
$+ (0)(0)(0.1)$
$+ (100)(0)(0.9)$
$= 0.1$

✯ y is independent of t since we choose $y = keep$ always regardless of t.

removing every email $\Rightarrow Pr(y = remove) = 1$:

$E[\mathcal{L}(y,t)] = \mathcal{L}(keep, Spam) Pr(y = keep) Pr(t = spam)$
$+ \mathcal{L}(keep, NonSpam) Pr(y = keep) Pr(t = NonSpam)$
$+ \mathcal{L}(Remove, Spam) Pr(y = Remove) Pr(t = Spam)$
$+ \mathcal{L}(Remove, NonSpam) Pr(y = Remove) Pr(t = NonSpam)$
$= (1)(0)(0.1)$
$+ (0)(0)(0.9)$
$+ (0)(1)(0.1)$
$+ (100)(1)(0.9)$
$= 90$

✯ y is independent of t since we choose $y = Remove$ always regardless of t.

$\therefore$ The expected loss $E[\mathcal{L}(y,t)]$ for the policy that keeps every email is 0.1 and the expected loss $E[\mathcal{L}(y,t)]$ for the policy that removes every email is 90.

b) Pick $y_* = Remove$ if

$$E[\mathcal{L}(y = Remove, t) \mid \vec{x}] < E[\mathcal{L}(y = keep, t) \mid \vec{x}]$$

$$\Leftrightarrow \sum_{t_i \in \{Spam, NonSpam\}} \mathcal{L}(y = Remove, t = t_i) pr(t = t_i \mid \vec{x}) < \sum_{t_i \in \{Spam, NonSpam\}} \mathcal{L}(y = keep, t = t_i) pr(t = t_i \mid \vec{x})$$

$\Leftrightarrow \mathcal{L}(y=\text{Remove}, t=\text{Spam})\, pr(t=\text{spam}|\vec{x}) + \mathcal{L}(y=\text{Remove}, t=\text{NonSpam})\, pr(t=\text{NonSpam}|\vec{x})$
$< \mathcal{L}(y=\text{keep}, t=\text{spam})\, pr(t=\text{Spam}|\vec{x}) + \mathcal{L}(y=\text{keep}, t=\text{NonSpam})\, pr(t=\text{NonSpam}|\vec{x})$

$\Leftrightarrow 0 \cdot pr(t=\text{Spam}|\vec{x}) + 100 \cdot (1-pr(t=\text{spam}|\vec{x})) < 1 \cdot pr(t=\text{spam}|\vec{x}) + 0 \cdot (1-pr(t=\text{spam}|\vec{x}))$

$\Leftrightarrow 100(1-pr(t=\text{spam}|\vec{x})) < pr(t=\text{Spam}|\vec{x})$

$\Leftrightarrow 100 - 100\,pr(t=\text{Spam}|\vec{x}) < pr(t=\text{Spam}|\vec{x})$

$\Leftrightarrow 100 < Pr(t=\text{Spam}|\vec{x})(1+100)$

$\Leftrightarrow \frac{100}{101} < pr(t=\text{spam}|\vec{x})$

$\therefore$ Pick $y_* =$ Remove if $Pr(t=\text{Spam}|\vec{x}) > \frac{100}{101} \approx 0.99$
and $y_* =$ Keep if otherwise.

c) want to find $pr(t=\text{Spam}|\vec{x}=(x_1,x_2))$ for all (x_1,x_2) pairs and then compare to the $\frac{100}{101}$ threshold.

$$Pr(t=\text{Spam}|\vec{x}=(x_1,x_2)) = \frac{Pr(t=\text{spam}, \vec{x}=(x_1,x_2))}{Pr(\vec{x}=(x_1,x_2))}$$

$$= \frac{Pr(\vec{x}=(x_1,x_2)|t=\text{Spam}) \cdot Pr(t=\text{Spam})}{Pr(\vec{x}=(x_1,x_2)|t=\text{Spam}) \cdot Pr(t=\text{Spam}) + Pr(\vec{x}=(x_1,x_2)|t=\text{Non Spam}) \cdot p(t=\text{Non Spam})}$$

for $\vec{x}=(x_1,x_2)=(0,0)$:

$$pr(t=\text{Spam}|\vec{x}=(0,0)) = \frac{Pr(\vec{x}=(0,0)|t=\text{Spam}) \cdot Pr(t=\text{Spam})}{Pr(\vec{x}=(0,0)|t=\text{Spam}) \cdot Pr(t=\text{Spam}) + Pr(\vec{x}=(0,0)|t=\text{Non Spam}) \cdot Pr(t=\text{NonSpam})}$$

$$= \frac{(0.4)(0.1)}{(0.4)(0.1)+(0.998)(0.9)} = \frac{0.04}{0.9382} \approx 0.0426 < \frac{100}{101}$$

$\Rightarrow$ since $pr(t=\text{Spam}|\vec{x}=(0,0)) < \frac{100}{101}$, by part b), $\boxed{y_* = \text{keep for } \vec{x}=(0,0)}$

for $\vec{x} = (x_1, x_2) = (0,1)$:

$$Pr(t=Spam \mid \vec{x}=(0,1)) = \frac{Pr(\vec{x}=(0,1) \mid t=Spam) \cdot Pr(t=Spam)}{Pr(\vec{x}=(0,1) \mid t=Spam) \cdot Pr(t=Spam) + Pr(\vec{x}=(0,1) \mid t=NonSpam) \cdot Pr(t=NonSpam)}$$

$$= \frac{(0.3)(0.1)}{(0.3)(0.1)+(0.001)(0.9)} = \frac{0.03}{0.0309} \approx 0.9709 < \frac{100}{101}$$

$\Rightarrow$ since $pr(t=spam \mid \vec{x}=(0,1)) < \frac{100}{101}$, by part b), $\boxed{y_* = keep \text{ for } \vec{x}=(0,1)}$

for $\vec{x} = (x_1, x_2) = (1,0)$:

$$pr(t=Spam \mid \vec{x}=(1,0)) = \frac{Pr(\vec{x}=(1,0) \mid t=Spam) \cdot Pr(t=Spam)}{Pr(\vec{x}=(1,0) \mid t=Spam) \cdot Pr(t=Spam) + Pr(\vec{x}=(1,0) \mid t=NonSpam) \cdot Pr(t=NonSpam)}$$

$$= \frac{(0.2)(0.1)}{(0.2)(0.1)+(0.001)(0.9)} = \frac{0.02}{0.0209} \approx 0.9569 < \frac{100}{101}$$

$\Rightarrow$ since $pr(t=spam \mid \vec{x}=(1,0)) < \frac{100}{101}$, by part b), $\boxed{y_* = keep \text{ for } \vec{x}=(0,0)}$

for $\vec{x} = (x_1, x_2) = (1,1)$:

$$pr(t=Spam \mid \vec{x}=(1,1)) = \frac{Pr(\vec{x}=(1,1) \mid t=Spam) \cdot Pr(t=Spam)}{Pr(\vec{x}=(1,1) \mid t=Spam) \cdot Pr(t=Spam) + Pr(\vec{x}=(1,1) \mid t=NonSpam) \cdot Pr(t=NonSpam)}$$

$$= \frac{(0.1)(0.1)}{(0.1)(0.1)+(0)(0.9)} = \frac{0.01}{0.01} = 1 > \frac{100}{101}$$

$\Rightarrow$ since $pr(t=spam \mid \vec{x}=(1,1)) < \frac{100}{101}$, by part b), $\boxed{y_* = Remove \text{ for } \vec{x}=(1,1)}$

d) $$E[\mathcal{L}(y_*, t)] = \sum_{(y_*,t)} \mathcal{L}(y_*, t)\, Pr(y_*, t)$$

$$\begin{aligned} = \; & \mathcal{L}(y_*=keep, t=Spam)\, Pr(y_*=keep, t=spam) \\ & + \mathcal{L}(y_*=keep, t=NonSpam)\, Pr(y_*=keep, t=NonSpam) \\ & + \mathcal{L}(y_*=Remove, t=Spam)\, Pr(y_*=Remove, t=Spam) \\ & + \mathcal{L}(y_*=Remove, t=NonSpam)\, Pr(y_*=Remove, t=NoneSpam) \end{aligned}$$

$$= \mathcal{L}(y_* = \text{keep}, t = \text{Spam}) \Pr(\vec{x} \neq (1,1), t = \text{Spam})$$
$$+ \mathcal{L}(y_* = \text{keep}, t = \text{NonSpam}) \Pr(\vec{x} \neq (1,1), t = \text{NonSpam})$$
$$+ \mathcal{L}(y_* = \text{Remove}, t = \text{Spam}) \Pr(\vec{x} = (1,1), t = \text{Spam})$$
$$+ \mathcal{L}(y_* = \text{Remove}, t = \text{NonSpam}) \Pr(\vec{x} = (1,1), t = \text{NonSpam})$$

$$= (1) \Pr(\vec{x} \neq (1,1) \mid t = \text{spam}) \Pr(t = \text{spam})$$
$$+ (0) \Pr(\vec{x} \neq (1,1) \mid t = \text{NonSpam}) \Pr(t = \text{NonSpam})$$
$$+ (0) \Pr(\vec{x} = (1,1) \mid t = \text{spam}) \Pr(t = \text{spam})$$
$$+ (100) \Pr(\vec{x} = (1,1) \mid t = \text{NonSpam}) \Pr(t = \text{NonSpam})$$

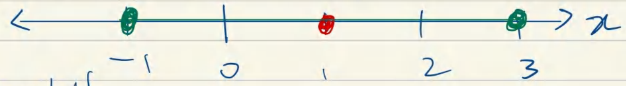
$$= (1)(0.4 + 0.2 + 0.3)(0.1) + (100)(0)(0.9)$$
$$= (1)(0.9)(0.1) = 0.09.$$

$\boxed{\therefore \text{ The expected loss } E[\mathcal{L}(y_*, t)] \text{ is } 0.09.}$

(Q2)

a) We can show that the data is not linearly separable by contradiction:
Assume that there were some hypothesis of feasible weights to divide the 1-D region linearly into two half-space. If two points lie in a half-space, then line segments connecting them also lie in the same halfspace. In this case, we have $x=-1$ and $x=3$ lying in the $t=1$ half space, so the line segment connecting -1 and 3 must also lie in the $t=1$ half space. However, $x=0$ is in the $t=0$ half space, but $x=0$ is also on the line segment connecting -1 and 3. The point cannot be in both half-spaces. So by contradiction, our assumption must be wrong and the data must not be linearly separable.

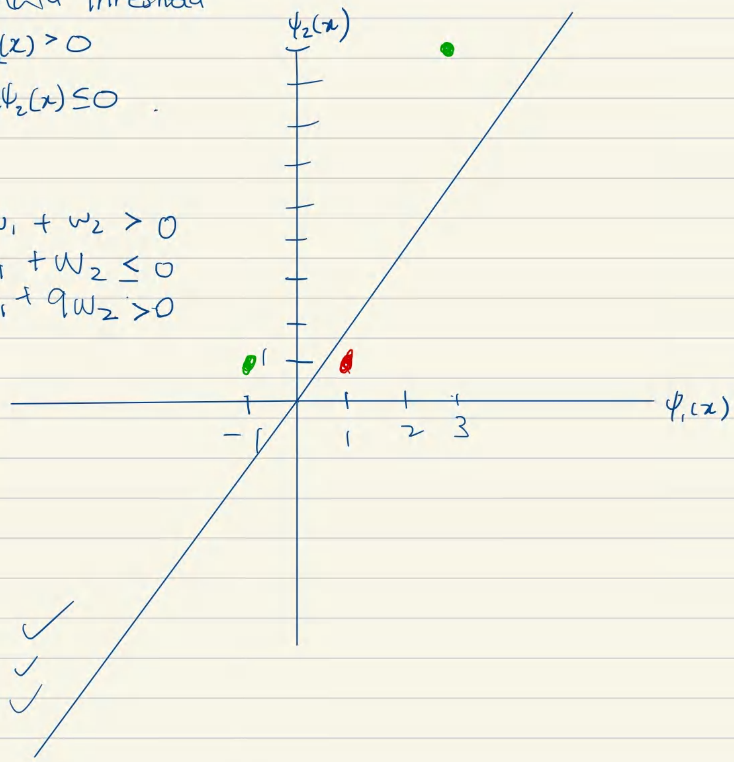
(or can think of 1-D data, so can only have a point as a decision boundary.) But any point that splits $x=-1$ and $x=1$ will also split $x=-1$ from $x=3$.



b) assume we are using hard threshold
where
$$y=\begin{cases}1 & \text{if } w_1\psi_1(x)+w_2\psi_2(x)>0\\ 0 & \text{if } w_1\psi_1(x)+w_2\psi_2(x)\le 0\end{cases}.$$

	x	$\psi_1(x)$	$\psi_2(x)$	t	
1	-1	-1	1	1	$-w_1+w_2>0$
2	1	1	1	0	$w_1+w_2\le 0$
3	3	3	9	1	$3w_1+9w_2>0$

$$\boxed{\text{pick } w_1=-2 \quad w_2=1}$$



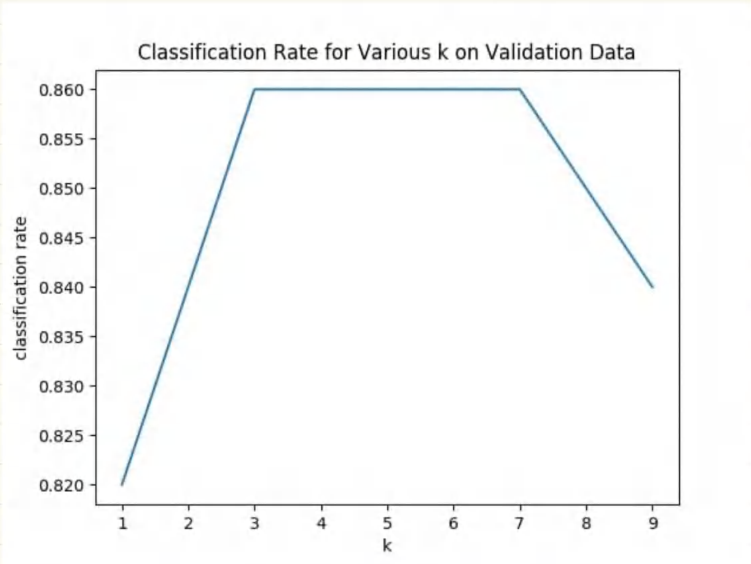
This satisfies

1: $-(-2)+(1)=3>0$ ✓
2: $(-2)+(1)=-1\le 0$ ✓
3: $3(-2)+9(1)=3>0$ ✓

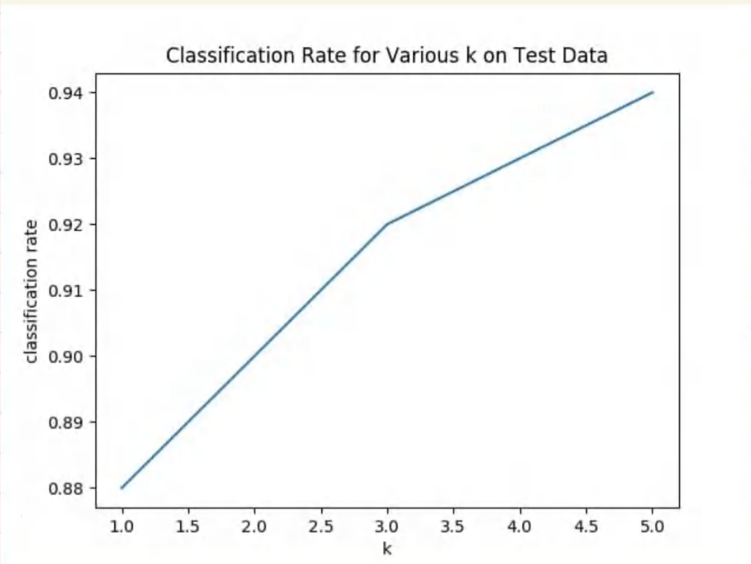
Q3)

3.1)

a)



b) On the validation set, the classifier had the highest classification rate of 86% for $k = 3, 5,$ and 7.
For $k=1$, the classifier achieved a classification rate of 82%, and 84% for $k=9$.
We can see from the graph above that the performance of the classifier is increasing as k increases until $k=3$.
Since the performance held constant for $k=3,5,7$ and drops for $k=9$ on this particular validation set, we pick $k^* = 3$ since it would result in the most efficient algorithm out of the 3 optimal k values. $k=9$'s drop in performance signals over fitting of the model.



The classification rate when classifying the test data for $k^*=3$ is 92%.
the classification rate for $k^*-2=1$ is 88%.
The classification rate for $k^*+2=5$ is 94%.

The test performance of these k values are all better than their validation performance.
This may be due to an overall easier to classify group of test data.

3.2)

a) The implementation of all of the functions are straightforward.

note: to implement the function logistic, we must find how to calculate df.

$$df = \begin{bmatrix} \frac{df}{dw_1} \\ \frac{df}{dw_2} \\ \vdots \\ \frac{df}{dw_{M+1}} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} (y^{(i)} - t^{(i)}) x_{(1)}^{(i)} \\ \sum_{i=1}^{N} (y^{(i)} - t^{(i)}) x_{(2)}^{(i)} \\ \vdots \\ \sum_{i=1}^{N} (y^{(i)} - t^{(i)}) x_{(M+1)}^{(i)} \end{bmatrix} = \begin{bmatrix} (y^{(1)} - t^{(1)}) x_{(1)}^{(1)} + (y^{(2)} - t^{(2)}) x_{(1)}^{(2)} + \cdots + (y^{(N)} - t^{(N)}) x_{(1)}^{(N)} \\ \vdots \\ (y^{(1)} - t^{(1)}) x_{(M+1)}^{(1)} + (y^{(2)} - t^{(2)}) x_{(M+1)}^{(2)} + \cdots + (y^{(N)} - t^{(N)}) x_{(M+1)}^{(N)} \end{bmatrix}$$

$$= \begin{bmatrix} x_1^{(1)} & \cdots & x_1^{(N)} \\ x_2 & & x_2^{(N)} \\ \vdots & & \vdots \\ x_{M+1} & & x_{M+1}^{(N)} \end{bmatrix} \cdot \begin{bmatrix} y^{(1)} - t^{(1)} \\ y^{(2)} - t^{(2)} \\ \vdots \\ y^{(N)} - t^{(N)} \end{bmatrix}$$

$$= X^T (y - t)$$

$\uparrow$
data.

b) diff $\approx 2.41375 \times 10^{-8}$

For dataset mnist-train:

Initializing weights = all zeros:

```
{'learning_rate': 0.2, 'weight_regularization': 0.0, 'num_iterations': 10}
Validation_CE: 0.4048194403486778
Validation Fraction Correct: 0.84
```

```
{'learning_rate': 0.2, 'weight_regularization': 0.0, 'num_iterations': 100}
Validation_CE: 0.21845131776026266
Validation Fraction Correct: 0.88
```

```
{'learning_rate': 0.1, 'weight_regularization': 0.0, 'num_iterations': 500}
Validation_CE: 0.196701747604367
Validation Fraction Correct: 0.88
```

```
{'learning_rate': 0.2, 'weight_regularization': 0.0, 'num_iterations': 500}
Validation_CE: 0.1918608574377248
Validation Fraction Correct: 0.88
```

```
{'learning_rate': 0.3, 'weight_regularization': 0.0, 'num_iterations': 500}
Validation_CE: 0.1921441638209888
Validation Fraction Correct: 0.88
```

```
{'learning_rate': 0.1, 'weight_regularization': 0.0, 'num_iterations': 600}
Validation_CE: 0.19464489522397382
Validation Fraction Correct: 0.88
```

→
```
{'learning_rate': 0.2, 'weight_regularization': 0.0, 'num_iterations': 600}
Validation_CE: 0.1917174996292446
Validation Fraction Correct: 0.88
```

← Best hyper parameters

- → initialize weights as all 0's
- → learning rate = 0.2
- → number of iterations = 600

```
{'learning_rate': 0.3, 'weight_regularization': 0.0, 'num_iterations': 600}
Validation_CE: 0.19278170063794195
Validation Fraction Correct: 0.88
```

```
{'learning_rate': 0.1, 'weight_regularization': 0.0, 'num_iterations': 700}
Validation_CE: 0.19336382085567916
Validation Fraction Correct: 0.88
```

```
{'learning_rate': 0.2, 'weight_regularization': 0.0, 'num_iterations': 700}
Validation_CE: 0.1918926597507856
Validation Fraction Correct: 0.88
```

Randomly initializing weights:

```
{'learning_rate': 0.2, 'weight_regularization': 0.0, 'num_iterations': 600}
Validation_CE: 0.4415146325146413
Validation Fraction Correct: 0.86
```

For dataset mnist_train_small:

initializing weights as all zeros:

```
{'learning_rate': 0.2, 'weight_regularization': 0.0, 'num_iterations': 600}
Validation_CE: 0.8626894187295057
Validation Fraction Correct: 0.72
```

```
{'learning_rate': 0.2, 'weight_regularization': 0.0, 'num_iterations': 100}
Validation_CE: 0.7515261131067335
Validation Fraction Correct: 0.7
```

```
{'learning_rate': 0.2, 'weight_regularization': 0.0, 'num_iterations': 50}
Validation_CE: 0.7176562554197128
Validation Fraction Correct: 0.66
```

```
{'learning_rate': 0.2, 'weight_regularization': 0.0, 'num_iterations': 10}
Validation_CE: 0.6776555299449193
Validation Fraction Correct: 0.64
```

```
{'learning_rate': 0.2, 'weight_regularization': 0.0, 'num_iterations': 5}
Validation_CE: 0.6805286706328936
Validation Fraction Correct: 0.6
```

```
{'learning_rate': 0.3, 'weight_regularization': 0.0, 'num_iterations': 10}
Validation_CE: 0.691480094884881
Validation Fraction Correct: 0.64
```

```
{'learning_rate': 0.1, 'weight_regularization': 0.0, 'num_iterations': 10}
Validation_CE: 0.6745201807400079
Validation Fraction Correct: 0.6
```

← Best hyperparameters

→ initialize weights as all 0's
→ learning rate = 0.1
→ number of iterations = 10

```
{'learning_rate': 0.05, 'weight_regularization': 0.0, 'num_iterations': 10}
Validation_CE: 0.6783353497100102
Validation Fraction Correct: 0.54
```

initializing weights randomly:

```
{'learning_rate': 0.1, 'weight_regularization': 0.0, 'num_iterations': 10}
Validation_CE: 3.3513504356853185
Validation Fraction Correct: 0.52
```

Results for dataset mnist_train using optimal hyperparameters:

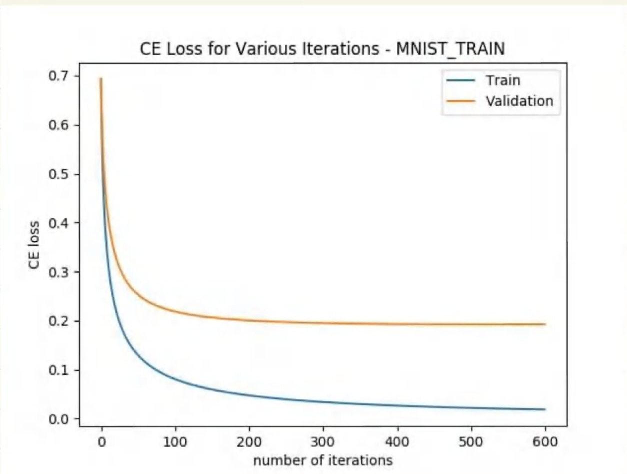
```
hyperparameters:  {'learning_rate': 0.2, 'weight_regularization': 0.0, 'num_iterations': 600}

Train_CE: 0.018233295380989868
Train Classification Error: 0.0

Validation_CE: 0.1917174996292446
Validation Classification Error: 0.12

Test_CE: 0.21883979812252144
Test Classification Error: 0.07999999999999996
```

Results for dataset mnist_train_small using optimal hyperparameters:

```
hyperparameters:  {'learning_rate': 0.1, 'weight_regularization': 0.0, 'num_iterations': 10}

Train_CE: 0.194123486558535
Train Classification Error: 0.0

Validation_CE: 0.6745201807400079
Validation Classification Error: 0.4

Test_CE: 0.5477831683036107
Test Classification Error: 0.24
```

c)

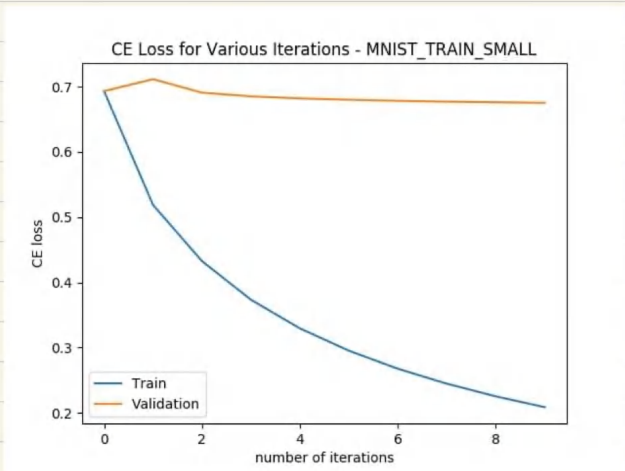
optimal ↓




```
{'learning_rate': 0.2, 'weight_regularization': 0.0, 'num_iterations': 600}
Validation_CE: 0.1917174996292446
Validation Fraction Correct: 0.88
```

weight initialization = all zeros

optimal ↓

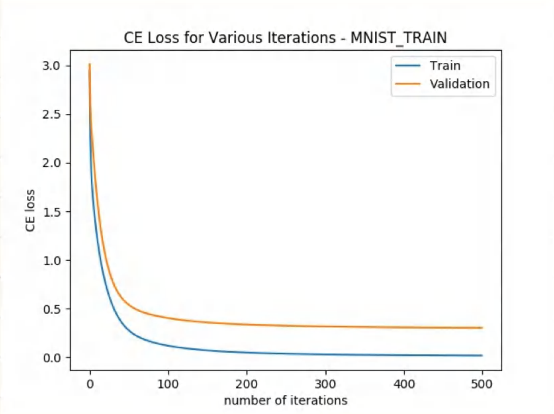



```
{'learning_rate': 0.1, 'weight_regularization': 0.0, 'num_iterations': 10}
Validation_CE: 0.6745201807400079
Validation Fraction Correct: 0.6
```

weight initialization = all zeros.

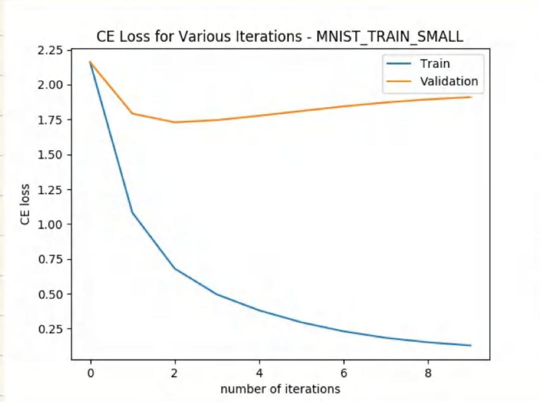
for the above two graphs, the weight initialization is constant (not random), so repeating the code generates the same graph.

for the graphs below, we try initializing the weights at random. this results in various different local CE loss minimums.



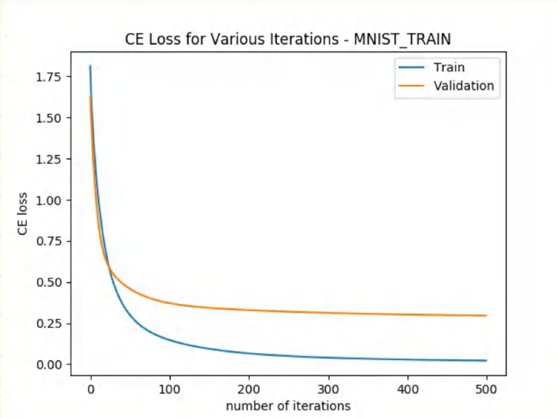

```
hyperparameters:  {'learning_rate': 0.2, 'weight_regularization': 0.0, 'num_iterations': 500}

Train_CE: 0.018015842836679487
Train Classification Error: 0.0

Validation_CE: 0.3040788463949122
Validation Classification Error: 0.09999999999999998
```

weight initialization: np.random.randn.



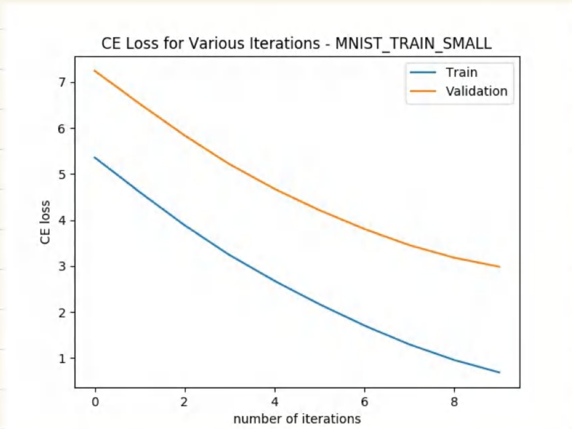

```
hyperparameters:  {'learning_rate': 0.1, 'weight_regularization': 0.0, 'num_iterations': 10}

Train_CE: 0.11307106226915262
Train Classification Error: 0.0

Validation_CE: 1.92219943654731
Validation Classification Error: 0.36
```

weight initialization: np.random.randn.




```
hyperparameters:  {'learning_rate': 0.2, 'weight_regularization': 0.0, 'num_iterations': 500}

Train_CE: 0.021807718187323643
Train Classification Error: 0.0

Validation_CE: 0.29499323938375294
Validation Classification Error: 0.14
```

weight initialization: np.random.randn.




```
hyperparameters:  {'learning_rate': 0.1, 'weight_regularization': 0.0, 'num_iterations': 10}

Train_CE: 0.5011071790860167
Train Classification Error: 0.30000000000000004

Validation_CE: 2.8608491957804243
Validation Classification Error: 0.48
```

weight initialization: np.random.randn.

★ After running the code multiple times, including initializing weights at random, we see that the best parameter settings is still given by the first set of hyperparameters where we initialized the weights as all zeros. However, suppose we did get a better result (lower CE loss local minimum) with one of the other random weights, we could record the initial weights and hyper/parameters for each run, and based on the results, go back and pick the weights/hyper parameters that resulted in the lowest local minimum of CE loss. We can also fine-tune our hyperparameters based on the results (for example, the second graph of the MNIST_TRAIN_SMALL dataset above shows potential overfitting. we can lower the # iterations instead) and repeat the process.

4) a) call $J = \left(\frac{1}{2} \sum_{i=1}^{N} a^{(i)} (y^{(i)} - \vec{w}^T \vec{x}^{(i)})^2 + \frac{\lambda}{2} \|\vec{w}\|^2 \right)$

$$\frac{\partial}{\partial w_j} \left(\frac{1}{2} \sum_{i=1}^{N} a^{(i)} (y^{(i)} - \vec{w}^T \vec{x}^{(i)})^2 + \frac{\lambda}{2} \|\vec{w}\|^2 \right)$$

$$= \frac{1}{2} \sum_{i=1}^{N} \frac{\partial}{\partial w_j} \left[a^{(i)} (y^{(i)} - \vec{w}^T \vec{x}^{(i)})^2 \right] + \frac{\lambda}{2} \frac{\partial}{\partial w_j} \sum_i w_i^2$$

$$= \cancel{\frac{1}{2}} \sum_{i=1}^{N} \cancel{2} a^{(i)} (y^{(i)} - \vec{w}^T \vec{x}^{(i)}) (-x_j^{(i)}) + \cancel{\frac{\lambda}{2}} \cancel{2} w_j$$

$$= \sum_{i=1}^{N} a^{(i)} (y^{(i)} - \vec{w}^T \vec{x}^{(i)}) (-x_j^{(i)}) + \lambda w_j$$

$$\begin{bmatrix} \frac{\partial J}{\partial w_1} \\ \frac{\partial J}{\partial w_2} \\ \vdots \\ \frac{\partial J}{\partial w_M} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} a^{(i)} (y^{(i)} - \vec{w}^T \vec{x}^{(i)}) (-x_1^{(i)}) + \lambda w_1 \\ \sum_{i=1}^{N} a^{(i)} (y^{(i)} - \vec{w}^T \vec{x}^{(i)}) (-x_2^{(i)}) + \lambda w_2 \\ \vdots \\ \sum_{i=1}^{N} a^{(i)} (y^{(i)} - \vec{w}^T \vec{x}^{(i)}) (-x_M^{(i)}) + \lambda w_M \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} a^{(i)} (y^{(i)} - \vec{w}^T \vec{x}^{(i)}) (-x_1^{(i)}) \\ \sum_{i=1}^{N} a^{(i)} (y^{(i)} - \vec{w}^T \vec{x}^{(i)}) (-x_2^{(i)}) \\ \vdots \\ \sum_{i=1}^{N} a^{(i)} (y^{(i)} - \vec{w}^T \vec{x}^{(i)}) (-x_M^{(i)}) \end{bmatrix} + \lambda \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_M \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{i=1}^{N} a^{(i)} (y^{(i)}) (-x_1^{(i)}) + \sum_{i=1}^{N} a^{(i)} (\vec{w}^T \vec{x}^{(i)}) (x_1^{(i)}) \\ \sum_{i=1}^{N} a^{(i)} (y^{(i)}) (-x_2^{(i)}) + \sum_{i=1}^{N} a^{(i)} (\vec{w}^T \vec{x}^{(i)}) (x_2^{(i)}) \\ \vdots \\ \sum_{i=1}^{N} a^{(i)} (y^{(i)}) (-x_M^{(i)}) + \sum_{i=1}^{N} a^{(i)} (\vec{w}^T \vec{x}^{(i)}) (x_M^{(i)}) \end{bmatrix} + \lambda \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_M \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{i=1}^{N} a^{(i)} (\vec{w}^T \vec{x}^{(i)}) (x_1^{(i)}) \\ \sum_{i=1}^{N} a^{(i)} (\vec{w}^T \vec{x}^{(i)}) (x_2^{(i)}) \\ \vdots \\ \sum_{i=1}^{N} a^{(i)} (\vec{w}^T \vec{x}^{(i)}) (x_M^{(i)}) \end{bmatrix} - \begin{bmatrix} \sum_{i=1}^{N} a^{(i)} (y^{(i)}) (-x_1^{(i)}) \\ \sum_{i=1}^{N} a^{(i)} (y^{(i)}) (-x_2^{(i)}) \\ \vdots \\ \sum_{i=1}^{N} a^{(i)} (y^{(i)}) (-x_M^{(i)}) \end{bmatrix} + \lambda \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_M \end{bmatrix}$$

we can expand the first row for visualization:

$$a^{(1)} (\vec{w}^T \vec{x}^{(1)}) (x_1^{(1)}) + a^{(2)} (\vec{w}^T \vec{x}^{(2)}) (x_1^{(2)}) + \ldots + a^{(N)} (\vec{w}^T \vec{x}^{(N)}) (x_1^{(N)})$$

$$a^{(1)} y^{(1)} x_1^{(1)} + a^{(2)} y^{(2)} x_1^{(2)} + \ldots + a^{(N)} y^{(N)} x_1^{(N)}$$

$$= \underset{M \times N}{\begin{bmatrix} x_1^{(1)} & \cdots & x_1^{(N)} \\ x_2^{(1)} & \cdots & x_2^{(N)} \\ \vdots & & \\ x_M^{(1)} & \cdots & x_M^{(N)} \end{bmatrix}} \underset{N \times 1}{\begin{bmatrix} a^{(1)} (\vec{w}^T x^{(1)}) \\ a^{(2)} (\vec{w}^T x^{(2)}) \\ \vdots \\ a^{N} (\vec{w}^T x^{(N)}) \end{bmatrix}} - \underset{M \times N}{\begin{bmatrix} x_1^{(1)} & \cdots & x_1^{(N)} \\ x_2^{(1)} & \cdots & x_2^{(N)} \\ \vdots & & \\ x_M^{(1)} & \cdots & x_M^{(N)} \end{bmatrix}} \underset{N \times 1}{\begin{bmatrix} a^{(1)} y^{(1)} \\ a^{(2)} y^{(2)} \\ \vdots \\ a^{(N)} y^{(N)} \end{bmatrix}} + \lambda \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_M \end{bmatrix}$$

$$= \underbrace{\begin{bmatrix} x_1^{(1)} & \cdots & x_1^{(N)} \\ x_2^{(1)} & \cdots & x_2^{(N)} \\ \vdots & & \\ x_M^{(1)} & \cdots & x_M^{(N)} \end{bmatrix}}_{M\times N} \underbrace{\begin{bmatrix} a^{(1)} & & \\ & a^{(2)} & \\ & & \ddots & \\ & & & a^{(N)} \end{bmatrix}}_{N\times N} \underbrace{\begin{bmatrix} x_1^{(1)} & \cdots & x_M^{(1)} \\ x_1^{(2)} & \cdots & x_M^{(2)} \\ \vdots & & \\ x_1^{(N)} & \cdots & x_M^{(N)} \end{bmatrix}}_{N\times M} \underbrace{\begin{bmatrix} w_1 \\ \vdots \\ w_M \end{bmatrix}}_{M\times 1} - \underbrace{\begin{bmatrix} x_1^{(1)} & \cdots & x_1^{(N)} \\ x_2^{(1)} & \cdots & x_2^{(N)} \\ \vdots & & \\ x_M^{(1)} & \cdots & x_M^{(N)} \end{bmatrix}}_{M\times N} \underbrace{\begin{bmatrix} a^{(1)} & & \\ & a^{(2)} & \\ & & \ddots & \\ & & & a^{(N)} \end{bmatrix}}_{N\times N} \underbrace{\begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{N} \end{bmatrix}}_{N\times 1} + \lambda \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_M \end{bmatrix}$$

$$= X^T A X \vec{w} - X^T A \vec{y} + \lambda \vec{w} \overset{\text{set}}{=} 0$$

$$\Rightarrow X^T A X \vec{w}^* + \lambda \vec{w}^* = X^T A \vec{y}$$

$$\Rightarrow (X^T A X + \lambda I) \vec{w}^* = X^T A \vec{y}$$

$$\Rightarrow (X^T A X + \lambda I)^{-1} (X^T A X + \lambda I) \vec{w}^* = (X^T A X + \lambda I)^{-1} X^T A \vec{y}$$

$$\Rightarrow I \vec{w}^* = (X^T A X + \lambda I)^{-1} X^T A \vec{y}$$

$$\Rightarrow \vec{w}^* = (X^T A X + \lambda I)^{-1} X^T A \vec{y} \quad \text{as wanted //.}$$

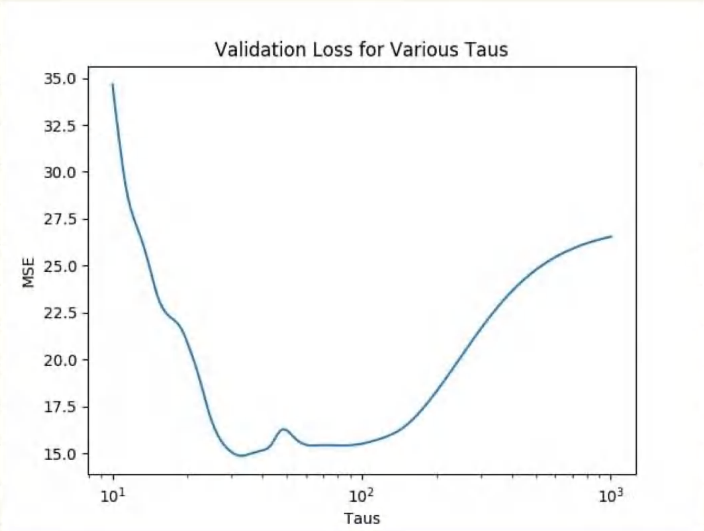
b) see code.

NOTE: $$\log\left(\frac{\exp(A_i)}{\sum_j \exp(A_j)}\right) = \log(\exp(A_i)) - \log\left(\sum_j \exp(A_j)\right)$$

$$= A_i - \log\left(\sum_j \exp(A_j)\right)$$

$$\Rightarrow \frac{\exp(A_i)}{\sum_j \exp(A_j)} = \exp\Big(A_i - \underbrace{\log \sum_j \exp(A_j)}_{\text{use logsumexp function.}}\Big)$$

c)



d) $\boxed{\text{as } \tau \to \infty}$, $\quad -\|\vec{x}-\vec{x}^{(i)}\|^2/2\tau^2 \to 0$

$$\Rightarrow \exp(-\|\vec{x}-\vec{x}^{(i)}\|^2/2\tau^2) \to 1$$

$$\Rightarrow a^{(i)} = \frac{\exp(-\|\vec{x}-\vec{x}^{(i)}\|^2/2\tau^2)}{\sum_j^N \exp(-\|\vec{x}-\vec{x}^{(j)}\|^2/2\tau^2)} \to \frac{1}{N}$$

, (equal weighting on the $(y^{(i)}-\vec{w}^T\vec{x}^{(i)})$ loss terms)

$$\Rightarrow w^* = \arg\min \frac{1}{2N}\sum_{i=1}^N (y^{(i)}-\vec{w}^T\vec{x}^{(i)})^2 + \frac{\lambda}{2}\|\vec{w}\|^2$$

⇒ $\boxed{\text{So the algorithm will be a ridge regression algorithm as } \tau \to \infty.}$

$\boxed{\text{as } \tau \to 0}$, $\quad -\|\vec{x}-\vec{x}^{(i)}\|^2/2\tau^2 \to -\infty$

However, the closer $\vec{x}$ and $\vec{x}^{(i)}$ is, the closer $\|\vec{x}-\vec{x}^{(i)}\|^2$ is to zero, and the closer $\frac{-\|\vec{x}-\vec{x}^{(i)}\|^2}{2\tau^2}$ is to 0 as opposed to $-\infty$.

$$\Rightarrow \exp(-\|\vec{x}-\vec{x}^{(i)}\|^2/2\tau^2) \to 0$$

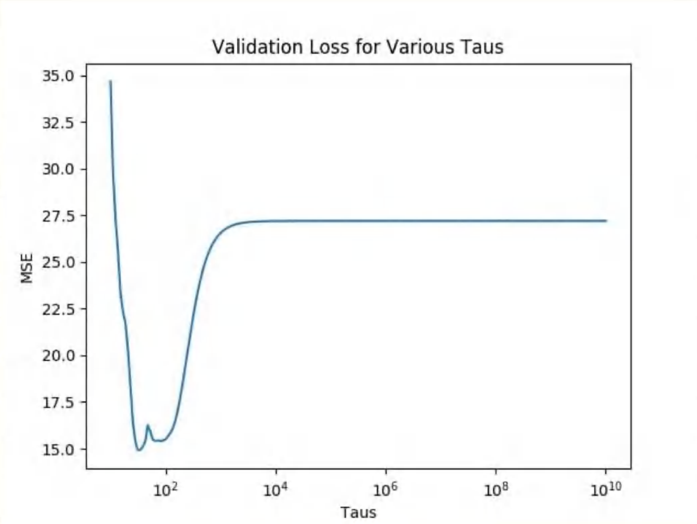
The closer $\frac{-\|\vec{x}-\vec{x}^{(i)}\|^2}{2\tau^2}$ is to 0, the closer $\exp(-\|\vec{x}-\vec{x}^{(i)}\|^2/2\tau^2)$ is to 1 as opposed to 0.

$$\Rightarrow a^{(i)} = \frac{\exp(-\|\vec{x}-\vec{x}^{(i)}\|^2/2\tau^2)}{\sum_j^N \exp(-\|\vec{x}-\vec{x}^{(j)}\|^2/2\tau^2)} \to \frac{0}{N\cdot 0}$$

This is tricky to visualize, so we analyze it by holding the denominator constant.

Holding the denominator constant, the closer the numerator is to 1 as opposed to 0, the bigger $a^{(i)}$ is.

This means that given a set of training data, and a test data, the loss between the test prediction and the closest training data's labels will be weighted the most heavily (penalized the most) when optimizing the weights. So w^* will be chosen to minimize the loss between the test prediction and those from the training set closest to it. $\boxed{\text{Hence the algorithm becomes a k-NN algorithm as } \tau \to 0.}$



By running the algorithm on larger τ values, we see that the above claim is correct.

as $\tau \to \infty$, the Loss stabalizes and approaches the MSE loss for a linear regression (ridge).

As $\tau \to 0$, the MSE loss is slightly bigger, which makes sense as it is more like a kNN algorithm, minimizing only the loss for the closest data.

The dip it MSE at around $\tau = 10^2$ displays a global minimum for the algorithm's MSE.
This shows the benefit of combining both KNN and and linear regression