998082159
Leo Woiceshyn

CSC 2515 A1 Q1

1. Since data is i.i.d., the likelihood function of the posterior is defined as follows:

$$L(\vec{w}) = \prod_{i=1}^{N} p(\vec{w} \mid t^{(i)}, \vec{x}^{(i)})$$

According to Bayes' Rule:

$$p(\vec{w} \mid \vec{t}, \vec{x}) = \frac{p(\vec{t} \mid \vec{w}, \vec{x}) \cdot p(\vec{w})}{p(\vec{t} \mid \vec{x})}$$

$\therefore$ $p(\vec{w} \mid \vec{t}, \vec{x}) \propto p(\vec{t} \mid \vec{w}, \vec{x}) \cdot p(\vec{w})$, since the denominator is independent of $\vec{w}$

Now, likelihood can be written as:

$$L(\vec{w}) = p(\vec{w}) \cdot \prod_{i=1}^{N} p(t^{(i)} \mid \vec{x}^{(i)}, \vec{w})$$

To find most likely parameters $\vec{w}$ given the training examples, maximize the likelihood function:

$$\max_{\vec{w}} L(\vec{w}) = p(\vec{w}) \cdot \prod_{i=1}^{N} p(t^{(i)} \mid \vec{x}^{(i)}, \vec{w})$$

Since the logarithm is a monotonically increasing function, the same $\vec{w}$ maximizes $L(\vec{w})$ and $\log L(\vec{w})$

For our loss function, taking the negative and minimizing is equivalent to maximizing the original likelihood

Thus, cost function $J(\vec{w}) = -\log L(\vec{w}) = -\log\left[p(\vec{w}) \cdot \prod_{i=1}^{N} p(t^{(i)} \mid \vec{x}^{(i)}, \vec{w})\right]$

Using logarithm rules:

$$-\log L(\vec{w}) = -\log p(\vec{w}) - \sum_{i=1}^{N} \log p(t^{(i)} | \vec{x}^{(i)}, \vec{w})$$

Since targets are binary, $p(t=0|\vec{x}^{(i)}, \vec{w}) = 1 - p(t=1|\vec{x}^{(i)}, \vec{w})$

and $p(t^{(i)}|\vec{x}^{(i)}; \vec{w}) = p(t=1|\vec{x}^{(i)}, \vec{w})^{t(i)} \cdot p(t=0|\vec{x}^{(i)}; \vec{w})^{1-t(i)}$

Thus, subbing in:

$$-\log L(\vec{w}) = -\log p(\vec{w}) - \sum_{i=1}^{N} \log\left[p(t^{(i)}=1 | \vec{x}^{(i)}, \vec{w})^{t(i)} \cdot p(t^{(i)}=0 | \vec{x}^{(i)}; \vec{w})^{1-t(i)}\right]$$

$$= -\log p(\vec{w}) - \sum_{i=1}^{N} t(i) \log p(t^{(i)}=1|\vec{x}^{(i)}, \vec{w}) - \sum_{i=1}^{N} (1-t(i)) \log(1-p(t^{(i)}=1|\vec{x}, \vec{w}))$$

Simplifying each term:

$$p(t^{(i)}=1|\vec{x}^{(i)}, \vec{w}, w_0) = \frac{1}{1+\exp(-z^{(i)})}, \text{ where } z^{(i)} = \sum_{d=1}^{D} (\vec{w}_d \vec{x}_d^{(i)} - w_0)$$

$$\log\left(\frac{1}{1+e^{-z}}\right) = -\log(1+e^{-z})$$

$$\log\left(1 - \frac{1}{1+e^{-z}}\right) = \log\left(\frac{1+e^{-z}-1}{1+e^{-z}}\right)$$

$$= \log(e^{-z}) - \log(1+e^{-z})$$

$$= -z - \log(1+e^{-z})$$

$$\sum_{i=1}^{N} t^{(i)} \log p(t^{(i)}=1|\vec{x}^{(i)}, \vec{w}) = \sum_{i=1}^{N} t^{(i)}(-\log(1+e^{-z^{(i)}}))$$

$$\sum_{i=1}^{N} (1-t(i)) \log(1-p(t^{(i)}=1|\vec{x}, \vec{w})) = \sum_{i=1}^{N}\left[(1-t^{(i)})(-z^{(i)} - \log(1+e^{-z^{(i)}}))\right]$$

Combining the sums:

$$J(\vec{w}) = -\log L(\vec{w}) = -\log p(\vec{w}) - \sum_{i=1}^{N} \left[t^{(i)}\left(-\log(1+e^{-z(i)})\right) + (1-t^{(i)})\left(-z(i) - \log(1+e^{z(i)}\right) \right]$$

$\downarrow$

Simplifying further:

$$-t^{(i)}\log(1+e^{-z(i)}) + (1-t^{(i)})(-z(i) - \log(1+e^{-z(i)}))$$

$$= \cancel{-t^{(i)}\log(1+e^{-z(i)})} - z(i) + z(i)t^{(i)} - \log(1+e^{-z(i)}) + t^{(i)}\log(1+e^{-z(i)})$$

First and last terms cancel out:

$$= z(i)t^{(i)} \underbrace{- z(i) - \log(1+e^{-z(i)})}$$

$$-z(i) - \log(1+e^{-z(i)}) = -\left[\log e^{z(i)} + \log(1+e^{-z(i)})\right]$$

$$= -\log\left(e^{z(i)}(1+e^{-z(i)})\right)$$

$$= -\log(e^{z(i)}+1)$$

Subbing back in:

$$J(\vec{w}) = -\log L(\vec{w}) = -\log p(\vec{w}) - \sum_{i=1}^{N}\left[z(i)t^{(i)} - \log(e^{z(i)}+1)\right]$$

Since prior of $\vec{w}$ is distributed normally w/ $\mu = 0$, $\sigma^2 = \frac{1}{\alpha}$:

$$p(\vec{w}) = \mathcal{N}(0, \alpha^{-1}I) \rightarrow p(w_j) = \frac{1}{\sqrt{2\pi/\alpha}} e^{-\frac{w_j^2\alpha}{2}}$$

$$\log p(\vec{w}) = \log\left(\frac{1}{\sqrt{2\pi/\alpha}}\right) - \frac{\alpha}{2}\sum_{j=1}^{M} w_j^2$$

$$= \log 1 - \log(\sqrt{2\pi/\alpha}) = -\log(\sqrt{2\pi/\alpha})$$

$$-\log p(\vec{w}) = \frac{\alpha}{2}\sum_{j=1}^{M} w_j - \log\left(\frac{1}{\sqrt{2\pi/\alpha}}\right) = \frac{\alpha}{2}\sum_{j=1}^{M} w_j + \log(\sqrt{2\pi/\alpha})$$

Final Expression:

$$J(\vec{w}) = -\sum_{i=1}^{N}\left[t^{(i)}z(i) - \log\left(1+e^{z(i)}\right)\right] + \frac{\alpha}{2}\sum_{d=1}^{D} w_d^2 + \log\left(\sqrt{2\pi/\alpha}\right)$$

where $z(i) = \sum_{d=1}^{D} w_d \vec{x}_d^{(i)} - w_0$, and ↗ these don't include w_0

2. $$\frac{\partial J(\vec{w})}{\partial w_d} = -\sum_{i=1}^{N}\left[t^{(i)}\frac{\partial z(i)}{\partial w_d} - \frac{1}{1+e^{z(i)}}\frac{\partial z(i)}{\partial w_d}\right] + \alpha w_d$$

$$\frac{\partial\left(\frac{\alpha w_d^2}{2}\right)}{\partial w_d} = \alpha w_d$$

$$\frac{\partial z(i)}{\partial w_d} = x_d^{(i)}$$

$$\therefore \frac{\partial J(\vec{w})}{\partial w_d} = -\sum_{i=1}^{N}\left[t^{(i)}x_d^{(i)} - \frac{1}{1+e^{z(i)}}x_d^{(i)}\right] + \alpha w_d$$

$$\frac{\partial J(\vec{w})}{\partial w_d} = -\sum_{i=1}^{N}\left[\left(t^{(i)} - \frac{1}{1+e^{z(i)}}\right)x_d^{(i)}\right] + \alpha w_d$$

$$\frac{\partial J(\vec{w})}{\partial w_0} = -\sum_{i=1}^{N}\left[\left(t^{(i)} - \frac{1}{1+e^{z(i)}}\right)\overset{1}{\cancel{x_0^{(i)}}}\right]$$

→ since $\vec{x}_0$ is a column vector of 1s

$$\frac{\partial J(\vec{w})}{\partial w_d} = -\sum_{i=1}^{N}\left(t^{(i)} - \frac{1}{1+e^{z(i)}}\right)\vec{x}_d + \alpha w_d$$

$$= \sum_{i=1}^{N}\left(\frac{1}{1+e^{z(i)}} - t^{(i)}\right)\vec{x}_d + \alpha w_d$$

3. Gradient Descent Update Pseudocode

If change in cost function less than 0.1%, loop ends

```
while $(J(\vec{w})_i - J(\vec{w})_{i-1})/J(\vec{w})_i > 0.001$ ) {
    $w_0 := w_0 - \lambda \sum_{i=1}^{N} \left( \frac{1}{1+e^{z(i)}} - t^{(i)} \right)$
    $w_1 := w_1 - \lambda \sum_{i=1}^{N} \left( \frac{1}{1+e^{z(i)}} - t^{(i)} \right) \vec{x}_1 - \lambda(\alpha w_1)$
    ⋮
    $w_d := w_d - \lambda \sum_{i=1}^{N} \left( \frac{1}{1+e^{z(i)}} - t^{(i)} \right) \vec{x}_d - \lambda(\alpha w_d)$
    Store old $J$
    Calculate new $J$ for $\vec{w}$ }
```

Note: $z(i) = \sum_{d=1}^{D} w_d \vec{x}_d^{(i)} - w_0$

and definition of convergence is defined as a <0.1% change in loss function over an iteration.

CSC2515 Assignment 1 Q2

First Split:

Overcooked Pasta

$$\begin{tikzcd}[column sep=small] & \text{Overcooked Pasta?} \arrow[dl, "Y"'] \arrow[dr, "N"] & \\ \text{S? } 1Y2N & & 2Y \end{tikzcd}$$

OC: Overcooked Pasta
S: Satisfied
$\neg$: Not

Satisfied sample space: 3Y2N

$IG(S|OC) = H(S) - H(S|OC)$

	OC	$\neg$OC
S	1	2
$\neg$S	2	0

$$H(S) = -\sum_{x \in S} p(S)\log_2 p(S)$$

$$= -p(S=Y)\log_2 p(S=Y) - p(S=N)\log_2 p(S=N)$$

$$H(S) = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} \approx 0.971 \text{ bits}$$

$$H(S|OC) = -\sum_{y \in OC}\sum_{x \in S} p(S,OC)\log_2 p(S|OC)$$

$$H(S|OC) = H(S|OC)\,p(OC) + H(S|\neg OC)\,p(\neg OC)$$

$p(OC) = 3/5$ $\quad p(\neg OC) = 2/5$ $\quad p(S|OC) = \frac{1}{3}$ $\quad p(S|\neg OC) = 1$

$$H(S|OC) = -\sum_{x \in S} p(S|OC)\log_2 p(S|OC) = -\left(\frac{1}{3}\right)\log_2\left(\frac{1}{3}\right) - \left(\frac{2}{3}\right)\log_2\left(\frac{2}{3}\right) \approx 0.918 \text{ bits}$$

$$H(S|\neg OC) = -\sum_{x \in S} p(S|\neg OC)\log_2 p(S|\neg OC) = 0$$ since all samples are $S=Y$

Subbing in:

$$H(S|OC) = (0.918)(3/5) + (0)(2/5) = 0.551$$

Information Gained: $IG(S|OC) = (0.971) - (0.551) = 0.427$ bits

Waiting Time

WL: Long waiting time
WS: Short waiting time

	WL	WS
S	2	1
¬S	1	1

$IG(S|W) = H(S) - H(S|W)$

From before, $H(S) = 0.971$ bits

$H(S|W) = H(S|WL)\,p(WL) + H(S|WS)\,p(WS)$

From table,

$p(WL) = 3/5 \quad p(WS) = 2/5 \quad p(S|WL) = 2/3 \quad p(S|WS) = 1/2$

$H(S|WL) = -\sum_{x \in WL} p(S|WL)\log_2 p(S|WL) = -\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3} \approx 0.918$ bits

$H(S|WS) = -\sum_{x \in WS} p(S|WS)\log_2 p(S|WS) = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1$ bit

Subbing in:

$H(S|W) = (0.918)(3/5) + (1)(2/5) = 0.951$ bits

$IG(S|W) = (0.971) - (0.951) = 0.02$ bits

Rude Waiter

YR: Rude
NR: not rude

	YR	NR
S	2	1
¬S	2	0

$IG(S|R) = H(S) - H(S|R)$

From before, $H(S) = 0.971$ bits

$$H(S|R) = H(S|YR)\,p(YR) + H(S|NR)\,p(NR)$$

From table,

$p(NR) = 1/5 \quad P(YR) = 4/5 \quad p(S|YR) = 1/2 \quad p(S|NR) = 1$

$$H(S|YR) = -\sum_{x \in YR} p(S|YR)\log_2 p(S|YR) = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1 \text{ bit}$$

$$H(S|NR) = -\sum_{x \in NR} p(S|NR)\log_2 p(S|NR) = 0$$ since all samples are $S = Y$

$\therefore H(S|R) = (1)(4/5) + (0)(1/5) = 4/5 = 0.8$

$IG(S|R) = (0.971) - (0.8) = 0.171$ bits

Comparing IG:

	IG (bits)
Overcooked?	0.427
Wait time	0.02
Rude?	0.171

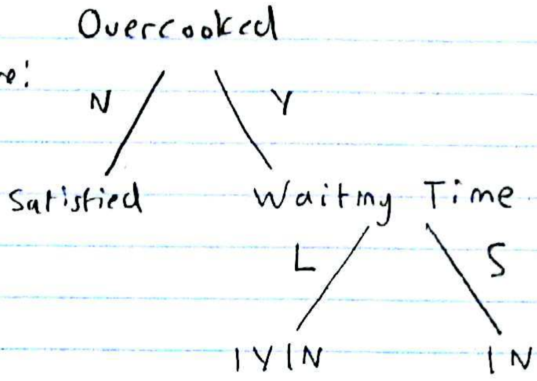
$\therefore$ choose overcooked as the first split

$$\begin{tikzcd} & \text{Overcooked} \arrow[dl, dash, "Y"'] \arrow[dr, dash, "N"] & \\ 1Y2N & & 2Y \end{tikzcd}$$

So if the pasta is not overcooked, the "satisfied" label is automatially assigned since all training examples in this bucket are Y

Second Split:

Try waiting time:



	WS	WL
S	0	1
7S	1	1

$$H(S) = -\sum_{x \in S} p(s)\log_2 p(s) = -p(S=Y)\log_2 p(S=Y) - p(S=N)\log_2 p(S=N)$$

$$H(S) = -\left(\frac{1}{3}\right)\log_2\left(\frac{1}{3}\right) - \left(\frac{2}{3}\right)\log_2\left(\frac{2}{3}\right)$$

$$H(S) = 0.918 \text{ bits}$$

From observation of the sample space, $H(S|WS) = 0$
$H(S|WL) = 1$

And from the table, $p(WS) = 1/3$, $p(WL) = 2/3$

$$H(S|W) = H(S|WS)\,p(WS) + H(S|WL)\,p(WL)$$

$$= (0)\,(1/3) + (1)\,(2/3)$$

$$H(S|W) = 2/3 \text{ bits}$$

$$IG(S|W) = H(S) - H(S|W)$$

$$= (0.918) - (2/3)$$

$$IG(S|W) = 0.251$$

Rude Waiter (Second Split):

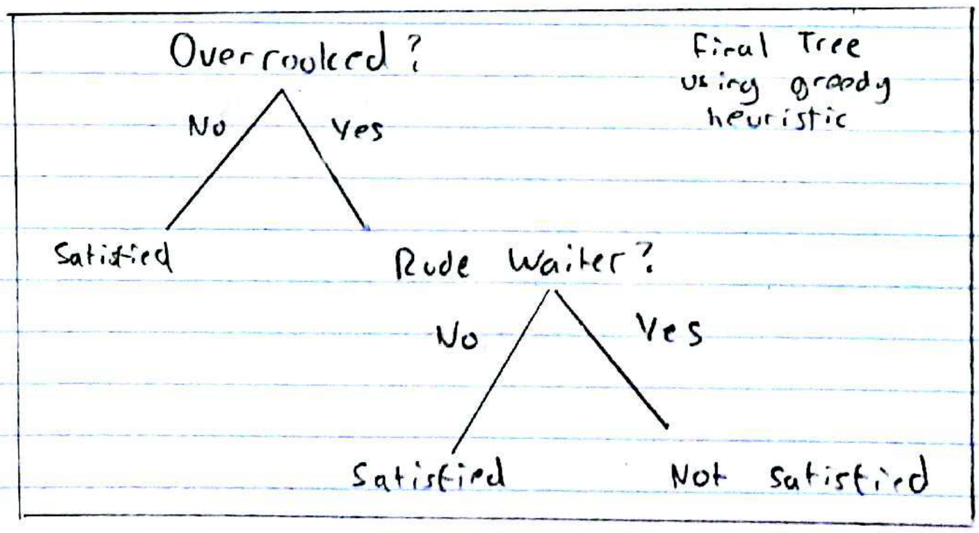
	YR	NR
S	0	1
¬S	2	0

By table observation, $H(S|YR) = 0$, $H(S|NR) = 0$

$\therefore$ $H(S|R) = 0$

$\therefore$ $IG(S|R) = H(S) = 0.918$ bits $> IG(S|W)$

$\therefore$ Choose Rude waiter for second split. Since all examples in each branch are of the same class, the tree is finalized:



2.

Person	Classification
6	Satisfied
7	Not satisfied
8	Satisfied

Easily observed from tree

CSC 2515 Question 3: k-NN vs Logistic Regression

Leo Woiceshyn (woicesh1, 998082159)

3.1 - 1)

Below are a table and plot of the classification results for the k-Nearest Neighbors algorithm on the validation set:

k	1	3	5	7	9
Validation Set Classification Accuracy (%)	94.0	98.0	98.0	98.0	96.0

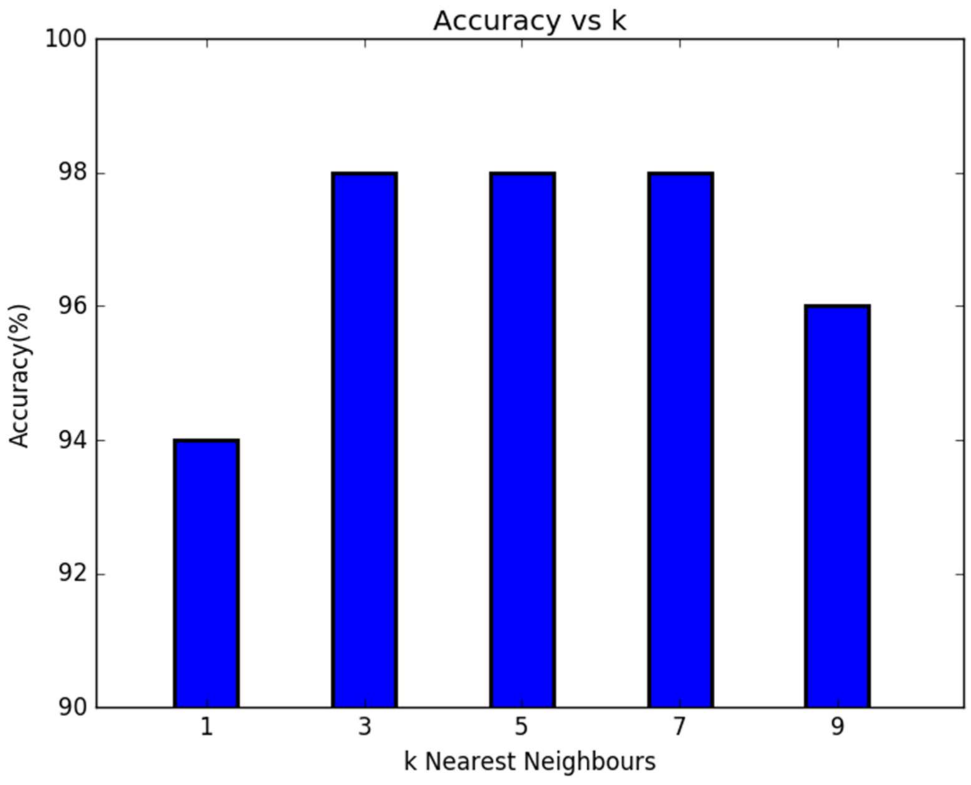


Figure 1 - Accuracy vs k for Validation Set

Based on the performance on the validation set, the value of k I would choose for the classifier would be 5. The justification for this is that the highest accuracy achieved for the validation set was 98%, which occurred for k = 3, 5, and 7, and the median value which achieved the highest accuracy was k = 5. The median value seems like the best choice; too low of a k value would lead to overfitting (high variance, and too high of a k value would lead to underfitting (high bias).

Test set results:

k	1	3	5	7	9
Test Set Classification Accuracy (%)	100.0	98.0	98.0	92.0	92.0

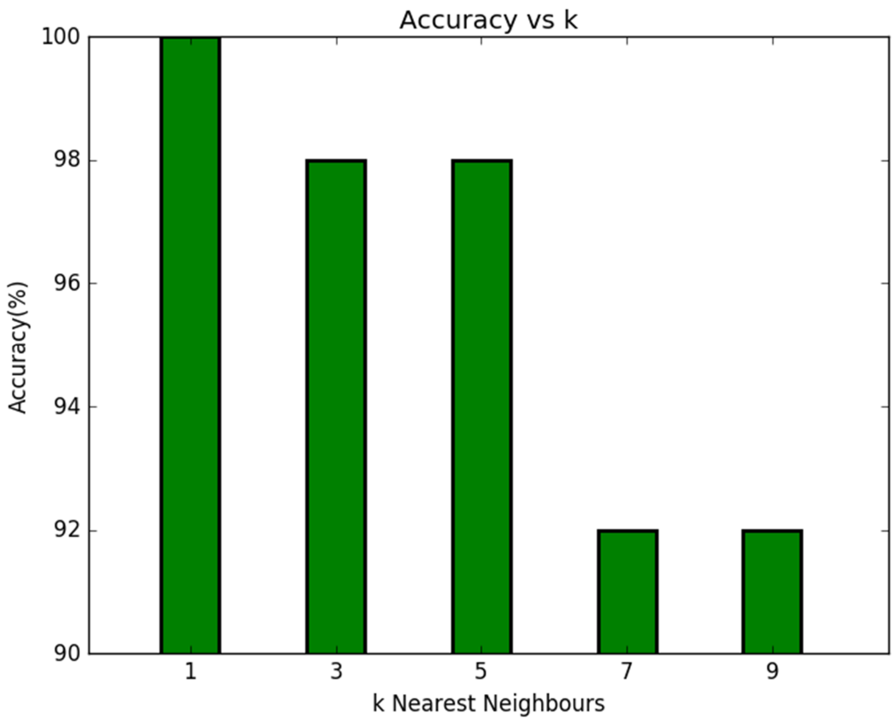


Figure 2- Accuracy vs k for Test Set

3.1 - 2)

The test performance of these values does not correspond perfectly to what would be typically expected. The test set achieves 100% accuracy for k=1, whereas I would have predicted k=1 to perform poorly compared to k > 1, as it did in the validation set, due to overfitting the training data. For k = 3 and k=5, the test set accuracy matches the validation set accuracy, meaning that our choice for k=5 was suitable. The accuracy is seen to get worse for the two higher k values, likely due to underfitting, as predicted. Assuming that the data is consistent and was split randomly into the training, validation, and test sets, the validation and test sets should produce similar results, which is the case here.

3.2 - 1)

For the unregularized model, I found that the learning rate that achieved the best results on the validation set was 0.07. Rather than choosing a set number of iterations, which caused my cross entropy to increase after reaching a minimum value in most cases, I set the number of iterations high (5000) and used the following metric to determine when to stop gradient descent:

```
if sum(abs(weights_old - weights)) > 0.05:
    pass
else:
    break
```

This code simply computes the sum of the absolute difference between the weights in each gradient descent iteration and stops when the sum of the differences reaches an acceptably low value, in which convergence has occurred. I also tuned this value to choose the one that produced the lowest validation cross entropy at convergence.

Large Training Set Results:

	Classification Accuracy(%)	**Final Cross-Entropy**
Training	98.0	15.02
Validation	92.0	11.87
Test	96.0	6.31

Small Training Set Results:

	Classification Accuracy(%)	**Final Cross-Entropy**
Training	100.0	0.181
Validation	80.0	30.04
Test	72.0	29.38

3.2 – 2)

The following are results from my chosen hyperparameters for minist_train and mnist_train_small:

Mnist_train:

Results at convergence:

Iterations	Train Cross-Entropy	Train Accuracy (%)	Validation Cross-Entropy	Validation Accuracy (%)
333	15.02	98.0	11.87	92.0

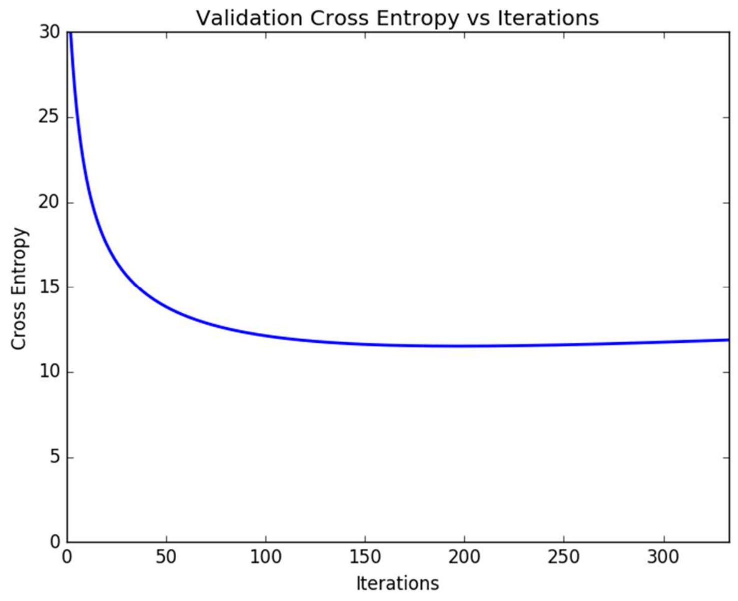


Figure 3 - Cross Entropy as a Function of Iterations on Validation Set for Mnist_train

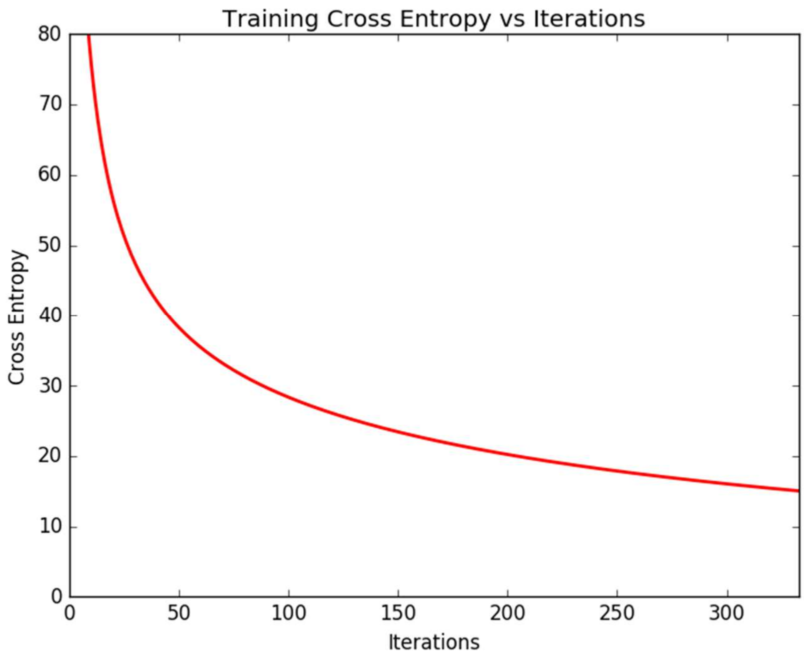


Figure 4 – Cross Entropy as a Function of Iterations on Training Set for Mnist_train

Mnist_train_small:

Results at convergence:

Iterations	Train Cross-Entropy	Train Accuracy (%)	Validation Cross-Entropy	Validation Accuracy (%)
110	0.181167	100.0	30.04	80.0

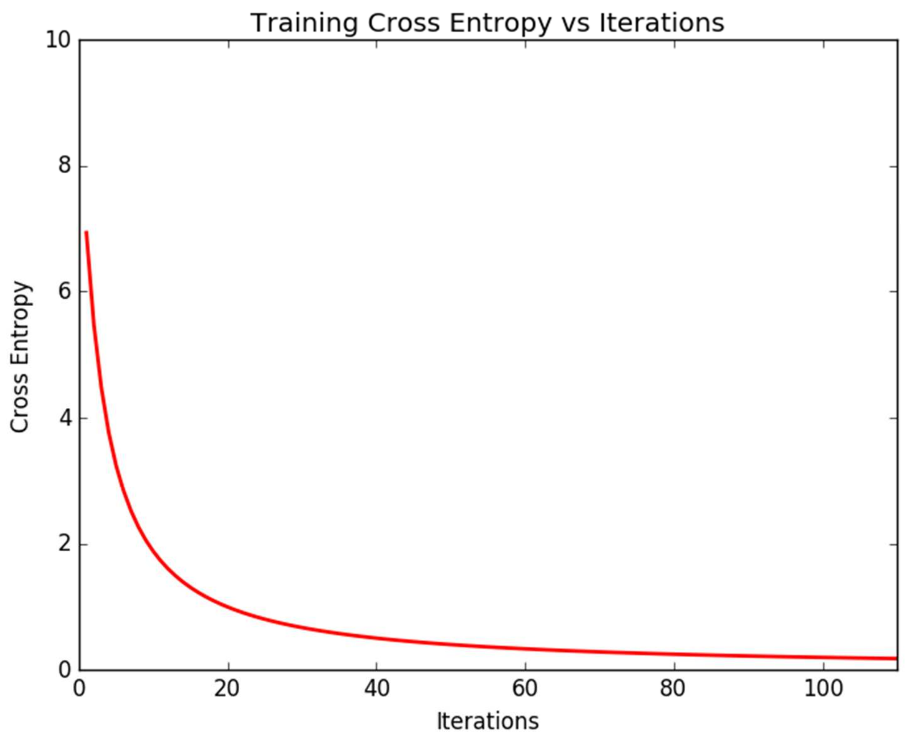


Figure 5 - Cross Entropy as a Function of Iterations on Training Set for Mnist_train_small

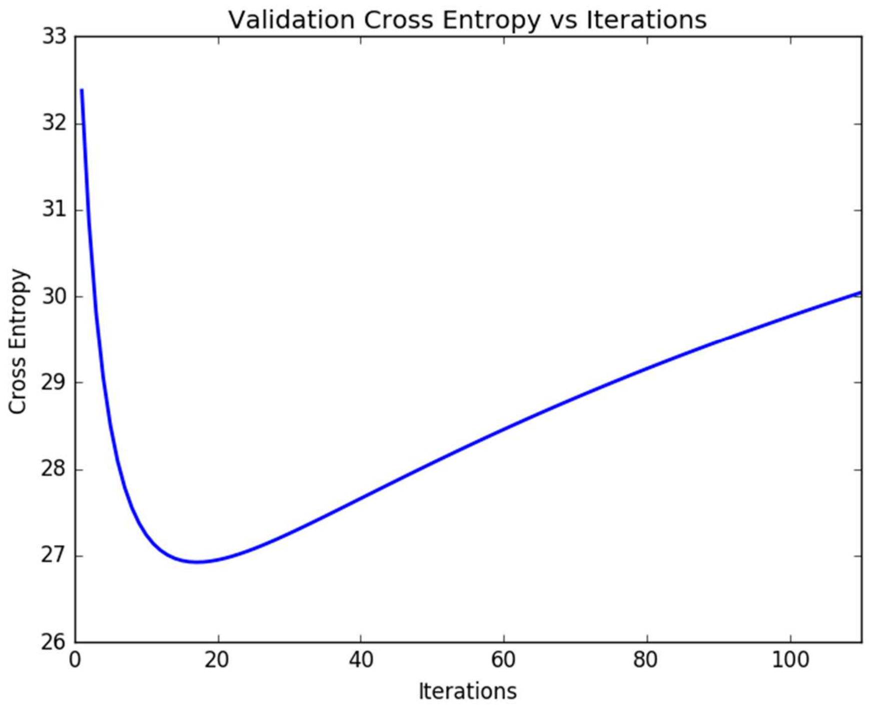


Figure 6 - Cross Entropy as a Function of Iterations on Validation Set for Mnist_train_small

For the larger training set, the results are quite good. The highest classification accuracy achieved is 92%, which is close to on-par with a human who would be classifying these digits. When running the code for a set number of iterations, the cross entropy would reach a minimum value and then begin to increase after. By setting a condition to stop gradient descent on convergence, the cross entropy stays around its minimum achieved value.

For the smaller training set, the highest validation accuracy achieved using the same hyperparameters is only 80.0%, which is to be expected with less data. The validation cross entropy reaches a minimum value and begins to increase. Overall, the validation cross entropy on the smaller data set is a significant amount larger than for the larger training set, especially considering the differences in training set size.

The best performance I had was achieved through using the initial weights set to zero, rather than initial weights set to a random numbers drawn from a normal distribution. Thus, my results do not change when I run my code several times. If I change my initial weights to randomly assigned numbers, the results change slightly each time I run the code, and the classification results are consistently worse than with zero-initialized weights.

3.3 - 1)

Similarly, to the unregularized case, the learning rate which performed the best for me was 0.07. The difference in results using regularization seemed very negligible. After trying multiple values of weight decay, the only value which had any positive impact on the results was a weight decay value of 0.1, which slightly reduced the validation cross entropy.

Using a learning rate of 0.07 and an alpha value of 0.1, here are the results for regularized logistic regression:

	Classification Accuracy(%)	**Final Cross-Entropy**
Training	97.5	15.06
Validation	92.0	11.86
Test	96.0	6.32

Small Training Set Results:

	Classification Accuracy(%)	**Final Cross-Entropy**
Training	100.0	0.231
Validation	80.0	29.35
Test	72.0	28.77

Mnist_train:

Results at convergence:

Iterations	Train Cross-Entropy	Train Accuracy (%)	Validation Cross-Entropy	Validation Accuracy (%)
332	15.06	97.5	11.86	92.0

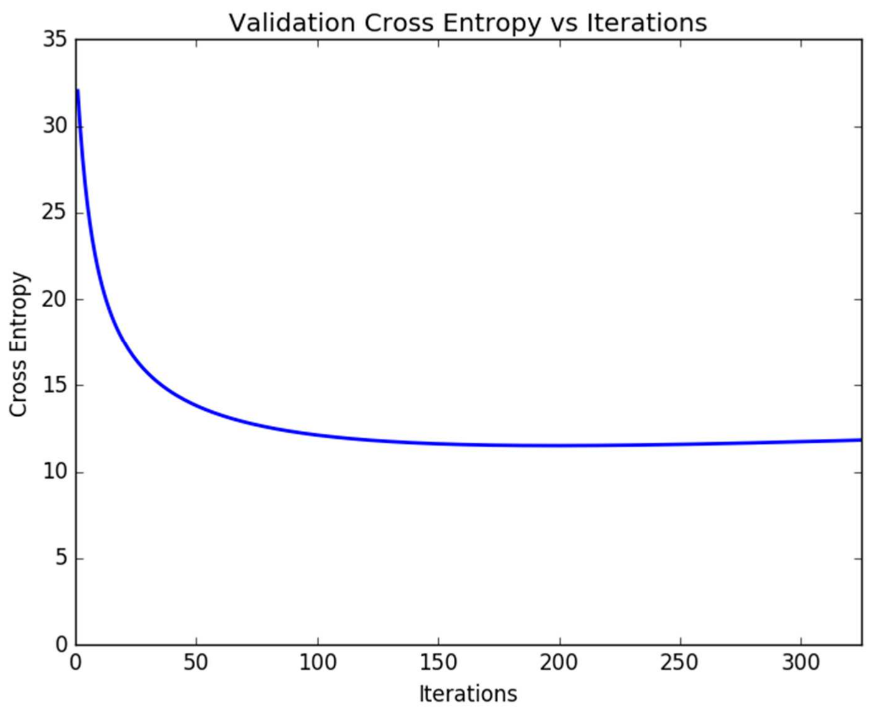


Figure 7 - Cross Entropy as a Function of Iterations on Validation Set for Mnist_train for Regularization

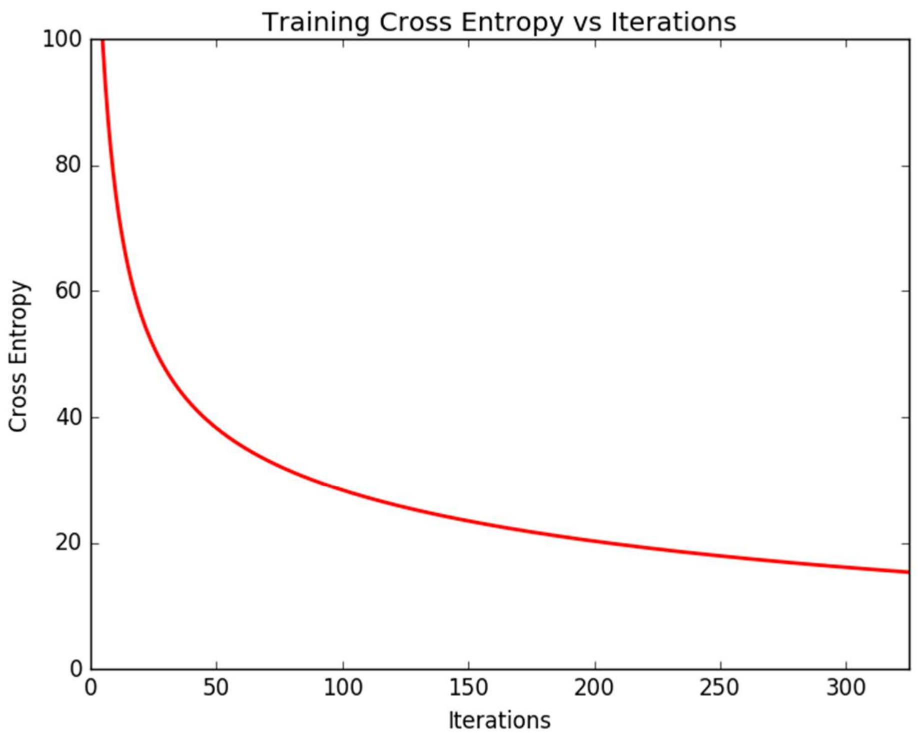


Figure 8 – Cross Entropy as a Function of Iterations on Training Set for Mnist_train for Regularization

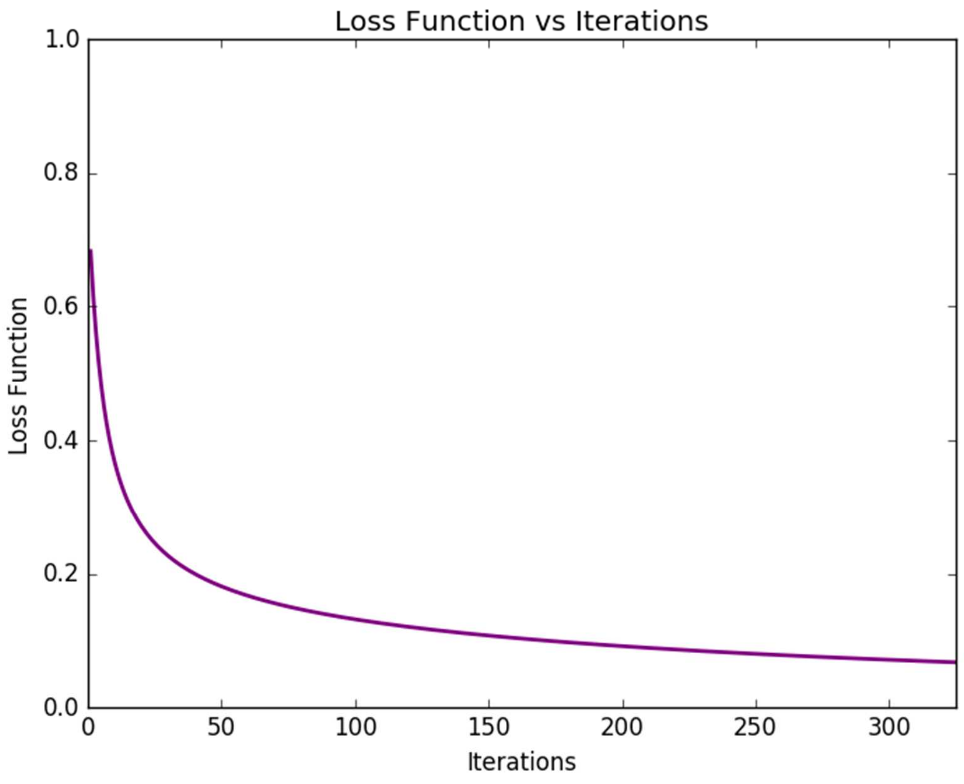


Figure 9 – Loss Function(f/N) as a Function of Iterations on Training Set for Mnist_train for Regularization

Mnist_train_small:

Results at convergence:

Iterations	Train Cross-Entropy	Train Accuracy (%)	Validation Cross-Entropy	Validation Accuracy (%)
108	0.187	100.0	29.95	80.0

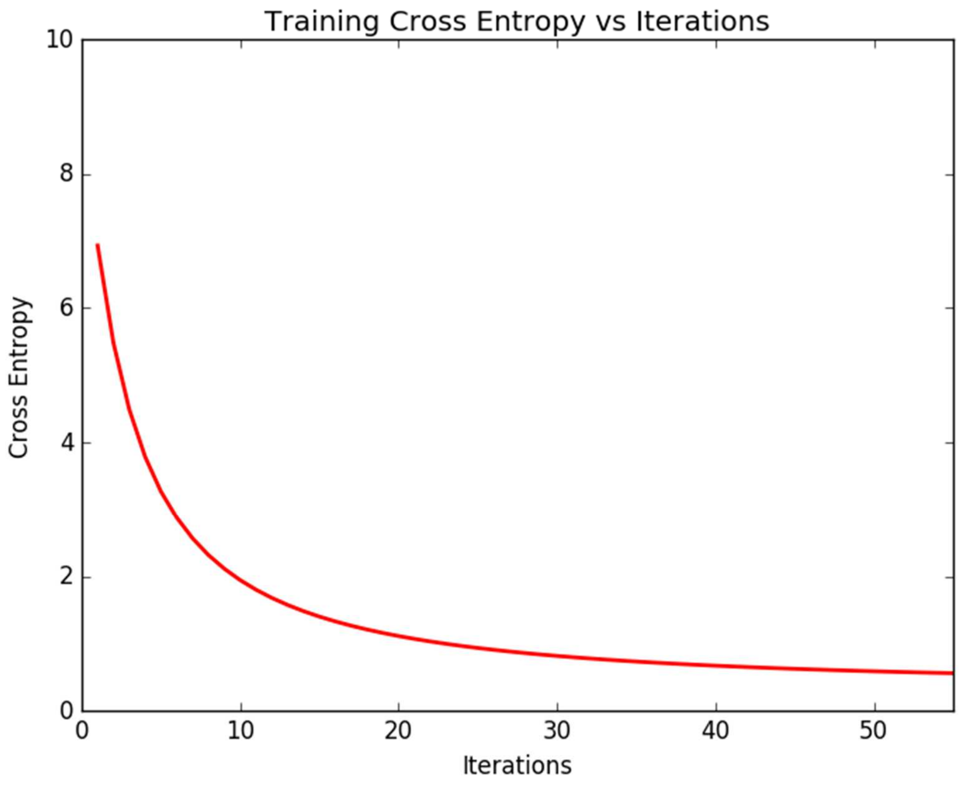


Figure 10 - Cross Entropy as a Function of Iterations on Training Set for Mnist_train_small for Regularization

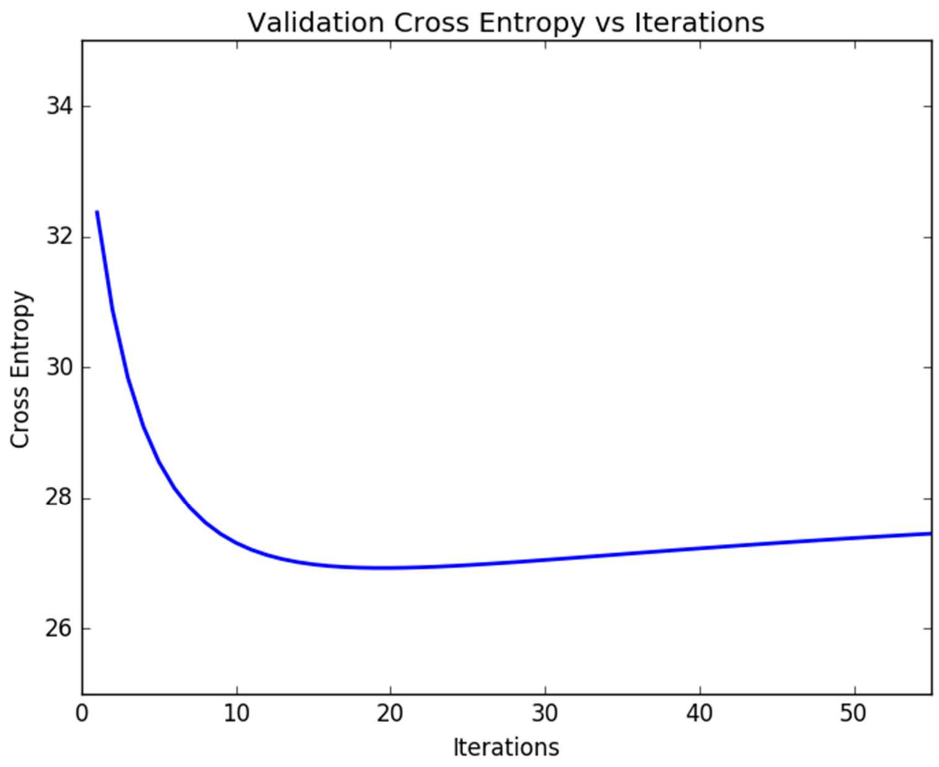


Figure 11 - Cross Entropy as a Function of Iterations on Validation Set for Mnist_train_small for Regularization

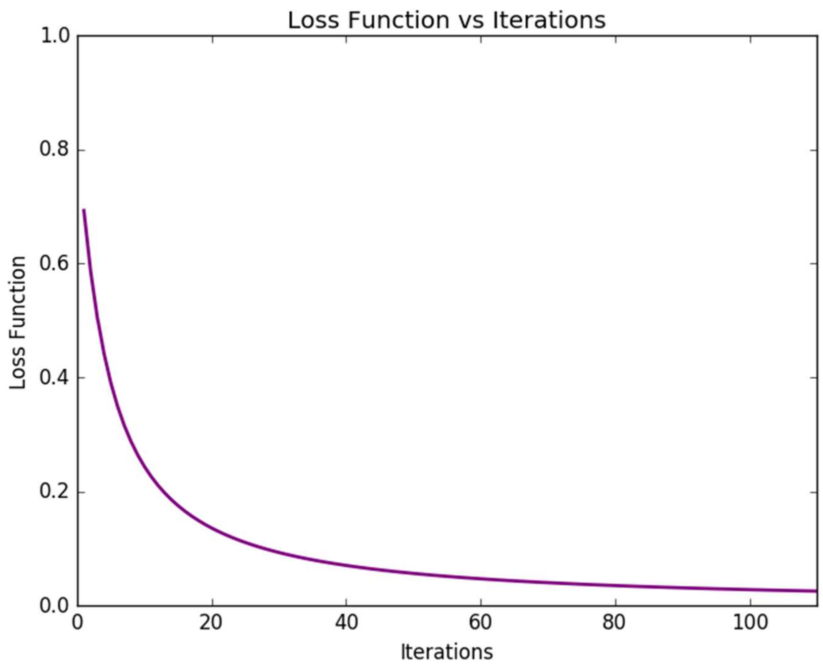


Figure 12 – Loss Function(f/N) vs Iterations on Validation Set for Mnist_train_small for Regularization

3.3 - 2)

With all other hyperparameters fixed, increasing α increases the loss, but reduces the cross entropy, and slightly reduces the rate at which the model converges to the best classification accuracy. To illustrate this, I set the iterations to a constant value of 1000, the learning rate to 0.7, and trained the model for the four different values of alpha. Here are the results:

α	f/N	Validation CE	Iterations to 92% Accuracy
0.001	0.01	14.43	201
0.01	0.02	14.42	201
0.1	0.03	14.31	201
1	0.06	13.49	207

Although there is not a significant change between values of α, the trends are consistent. The reason that the loss function increases with alpha is simply due to the fact that the regularization term being added to the unregularized loss is a positive quantity that is scaled with alpha, meaning the higher the value of alpha, the higher the magnitude of the loss function. The constant term I included in my loss function which is being subtracted also decreases with increased α , since α is in the denominator.

Based on my experiments, it is extremely hard, with this data set, to objectively determine the best value of α, since the difference in results between the regularized and non-regularized models is extremely negligible. Based on the theory, when the weight decay is very low, it will hardly have an effect on the model, and overfitting is more likely to occur. If the weight decay is too high, all of the model parameters will be heavily penalized for large values, meaning underfitting is likely to occur. Given this knowledge and the results, the best value for α of the four values would be 0.1, since this is the most likely to provide a good balance between overfitting and underfitting.

3.3 - 3)

On the larger training set, the effect of applying regularization was almost negligible. The change in values between the two was so small that it seems as if regularization made no difference. The performance on the test set using the smaller training set had a slightly more noticeable difference, with the regularized version having a better performance.

The smaller training set is much more prone to overfitting, since it has less training examples to learn from, and thus regularization is more likely to have a beneficial effect, whereas for the larger data set, the model is already less prone to overfitting due to the greater variety and number of training examples, meaning I would expect regularization to have a lesser effect. This is seen to a degree from my results, since the difference with regularization is more noticeable for the small training set compared to the large training set.

3.3 - 4)

Comparing the results between the kNN and logistic regression algorithms, both algorithms performed very well on the larger data set. For our test set comparison, the kNN, using a k value of 5, achieved a classification accuracy of 98%, whereas the logistic regression model achieved a classification accuracy of 96%. Both of these results are very good, so just based on my results, neither seems to provide a significant advantage in terms of classification accuracy. The kNN seems to perform slightly better due to having a higher validation set classification accuracy. Typically, the test set accuracy would be lower than the validation set accuracy, but for logistic regression, the test set accuracy is actually higher, meaning that this might just be a fortunate result with this given test set.

Comparing the two algorithms in general: kNN does not require any learning of the model, but at test time is computationally expensive, as the distance to every training example is calculated. Logistic regression requires training for the model, but does not have much computational expense at test time. kNN is able to form complex, non-linear decision boundaries, whereas logistic regression can only form linear decision boundaries. This means that for datasets that require nonlinear decision boundaries, kNN is likely to perform much better than logistic regression. kNN is a nonparametric algorithm, whereas logistic regression has a set of weight parameters, which can in some cases lead to intuition about feature performance. kNN in general is sensitive to noise in classes, attribute scaling, and distance tends to mean less in high dimensional space. Logistic regression, in general, is fast at classifying, resistant to overfitting, is quick to train, and is easily extended to multiple classes.