CSC 411: Assignment #1

Farzad Niroui (999029910)

1. Logistic Regression

1.

$$L(w) = -\log\left[p(w)\prod_i p(t^{(i)}|x^{(i)},w)\right]$$

$$= -\log p(w) - \log\left[\prod_{i=1}^{N} p(t^{(i)}|x^{(i)},w)\right]$$

Side: $p(t^{(i)}|x^{(i)},w) = p(C=0|x^{(i)},w)^{1-t^{(i)}}\, p(C=1|x^{(i)})^{t^{(i)}}$

$$= \left(1-p(C=1|x^{(i)},w)\right)^{1-t^{(i)}}\, p(C=1|x^{(i)},w)^{t^{(i)}}$$

$$p(w) = N(w|0,\alpha^{-1}I) = \frac{1}{\sqrt{2\alpha^{-1}I\pi}}\exp\left[\left(-w^Tw\right)/(2\alpha^{-1}I)\right]$$

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

$$= -\log\left[\frac{1}{\sqrt{2\alpha^{-1}I\pi}}\exp\left[-w^Tw/(2\alpha^{-1}I)\right]\right] - \sum_{i=1}^{N}(1-t^{(i)})\log\left(1-p(C=1|x^{(i)},w)\right) - \sum_{i=1}^{N} t^{(i)}\log p(C=1|x^{(i)},w)$$

let $z^{(i)} = w^Tx^{(i)} + w_0 = \sum_{d=1}^{D} w_d x_d^{(i)} - w_0$

$$= \frac{w^Tw}{2\alpha^{-1}I} + \log\sqrt{2\alpha^{-1}I\pi} - \sum_{i=1}^{N}(1-t^{(i)})\log\left(1-\frac{1}{1+\exp(-z^{(i)})}\right) - \sum_{i=1}^{N} t^{(i)}\log\left(\frac{1}{1+\exp(-z^{(i)})}\right)$$

$$= \frac{w^Tw}{2\alpha^{-1}I} + \log\sqrt{2\alpha^{-1}I\pi} - \sum_{i=1}^{N}(1-t^{(i)})\log\left(\frac{\exp(-z^{(i)})}{1+\exp(-z^{(i)})}\right) - \sum_{i=1}^{N} t^{(i)}\log\left(\frac{1}{1+\exp(-z^{(i)})}\right)$$

$$= \frac{w^Tw}{2\alpha^{-1}I} + \log\sqrt{2\alpha^{-1}I\pi} - \sum_{i=1}^{N}\left[-z^{(i)} - \log\left(1+\exp(-z^{(i)})\right)\right] + \sum_{i=1}^{N} t^{(i)}\left[-z^{(i)} - \log\left(1+\exp(-z^{(i)})\right)\right] + \cdots$$

$$\cdots \sum_{i=1}^{N} t^{(i)}\log\left(1+\exp(-z^{(i)})\right)$$

$$= \frac{w^Tw}{2\alpha^{-1}I} + \log\sqrt{2\alpha^{-1}I\pi} + \sum_{i=1}^{N} z^{(i)} + \sum_{i=1}^{N}\log\left(1+\exp(-z^{(i)})\right) - \sum_{i=1}^{N} t^{(i)}z^{(i)} - \sum_{i=1}^{N} t^{(i)}\log\left(1+\exp(-z^{(i)})\right) + \cdots$$

$$\cdots \sum_{i=1}^{N} t^{(i)}\log\left(1+\exp(-z^{(i)})\right)$$

$$= \frac{w^Tw}{2\alpha^{-1}I} + \log\sqrt{2\alpha^{-1}I\pi} + \sum_{i=1}^{N}\left[w^Tx^{(i)} + w_0\right] + \sum_{i=1}^{N}\log\left(1+\exp(-w^Tx^{(i)} - w_0)\right) - \sum_{i=1}^{N} t^{(i)}(w^Tx^{(i)} + w_0)$$

$$= \frac{w^Tw\,\alpha}{2} + \log\sqrt{\frac{2\pi}{\alpha}} + \sum_{i=1}^{N}\left[(1-t^{(i)})(w^Tx^{(i)} + w_0)\right] + \sum_{i=1}^{N}\log\left(1+\exp(-w^Tx^{(i)} - w_0)\right)$$

2.

$$\frac{\partial L(w)}{\partial w_d} = \frac{2w_d}{2\alpha^{-1}} + \sum_{i=1}^{N}[x_d^{(i)}] + \sum_{i=1}^{N}\left[\frac{-1}{1+\exp(-w_d x_d^{(i)}-w_0)} \cdot x_d^{(i)} \cdot \exp(-w_d x_d^{(i)}-w_0)\right] - \sum_{i=1}^{N} t^{(i)} x_d^{(i)}$$

$$= \frac{w_d}{\alpha^{-1}} + \sum_{i=1}^{N}\left[\left(1 - t^{(i)} - \frac{\exp(-w_d x_d^{(i)}-w_0)}{1+\exp(-w_d x_d^{(i)}-w_0)}\right) x_d^{(i)}\right]$$

$$= w_d \alpha + \sum_{i=1}^{N}\left[\left(\frac{1}{1+\exp(-w_d x_d^{(i)}-w_0)} - t^{(i)}\right) x_d^{(i)}\right]$$

$$\frac{\partial L(w)}{\partial w_0} = \sum_{i=1}^{N} 1 + \sum_{i=1}^{N}\left[\frac{-1}{1+\exp(-w^T x^{(i)}-w_0)} \cdot \exp(-w^T x^{(i)}-w_0)\right] - \sum_{i=1}^{N} t^{(i)}$$

$$= \sum_{i=1}^{N}\left[\frac{1}{1+\exp(-w^T x^{(i)}-w_0)} - t^{(i)}\right]$$

3.

Pseudocode:

-Initialize weight (w_0,…,w_M)

-Choose number of iterations, learning rate and weight decay

-For x in range (0 to chosen iterations)

-Update weights:

$$w_0^{n+1} = w_0^{n} - \lambda\left(\sum_{i=1}^{N}\left[\frac{1}{1+\exp(-w^T x^{(i)}-w_0)} - t^{(i)}\right]\right)$$

$$w_1^{n+1} = w_1^{n} - \lambda\left(\sum_{i=1}^{N}\left[\left(\frac{1}{1+\exp(-w_d x_d^{(i)}-w_0)} - t^{(i)}\right) x_d^{(i)}\right] + w_d \alpha\right)$$

$$\vdots$$

$$w_M^{n+1} = w_n^{n} - \lambda(\ldots.)$$

2. Decision Trees

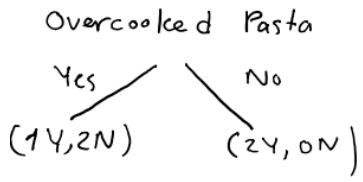
1.

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

$$= -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}$$

$$= 0.971$$

$IG(\text{Overcooked Pasta}) = H(Y) - H(Y|X)$



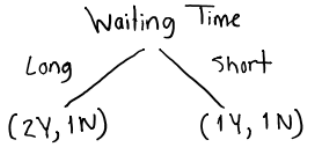
$= 0.971 - \left[\frac{3}{5}H(Y|\text{Yes}) + \frac{2}{5}H(Y|\text{No})\right]$

$= 0.971 - \frac{3}{5}\left(-\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3}\right) - \frac{2}{5}\left(-\log_2 1\right)$

$= 0.971 - 0.551$

$= 0.42$

$IG(\text{Waiting Time}) = H(Y) - H(Y|X)$



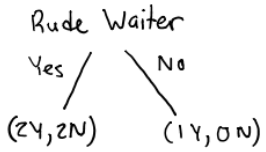
$= 0.971 - \left[\frac{3}{5}H(Y|\text{Long}) + \frac{2}{5}H(Y|\text{Short})\right]$

$= 0.971 - \frac{3}{5}\left(-\frac{2}{3}\log_2\left(\frac{2}{3}\right) - \frac{1}{3}\log_2\left(\frac{1}{3}\right)\right) - \frac{2}{5}\left(-\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right)\right)$

$= 0.971 - 0.551 - 0.4$

$= 0.02$

$IG(\text{Rude Waiter}) = H(Y) - H(Y|X)$



$= 0.971 - \left[\frac{4}{5}H(Y|\text{Yes}) + \frac{1}{5}H(Y|\text{No})\right]$

$= 0.971 - \frac{4}{5}\left(-\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right)\right) - \frac{1}{5}\left(\log_2(1)\right)$

$= 0.971 - 0.8$

$= 0.171$ $\quad \therefore$ Choose (Overcooked Pasta)

Overcooked:

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

$$= 0.918$$

$$IG(\text{Waiting Time}) = 0.918 - \left[\frac{2}{3}H(Y|\text{Long}) + \frac{1}{3}H(Y|\text{Short})\right]$$

$$= 0.918 - \frac{2}{3}\left(-\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right)\right) - \frac{1}{3}(0)$$

$$= 0.918 - 0.667$$

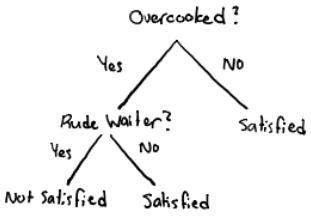
$$= 0.251$$

$$IG(\text{Rude Waiter}) = 0.918 - \left[\frac{2}{3}H(Y|\text{Yes}) + \frac{1}{3}H(Y|\text{No})\right]$$

$$= 0.918 - \frac{2}{3}(0) - \frac{1}{3}(0)$$

$$= 0.918$$

∴ Choose (Rude Waiter) in the overcooked branch.



Final Tree:

2.

Test Data

Person ID	Overcooked?	Waiting time	Rude Waiter?	Satisfied?
6	No	Short	No	Yes
7	Yes	Long	Yes	No
8	Yes	Short	No	Yes

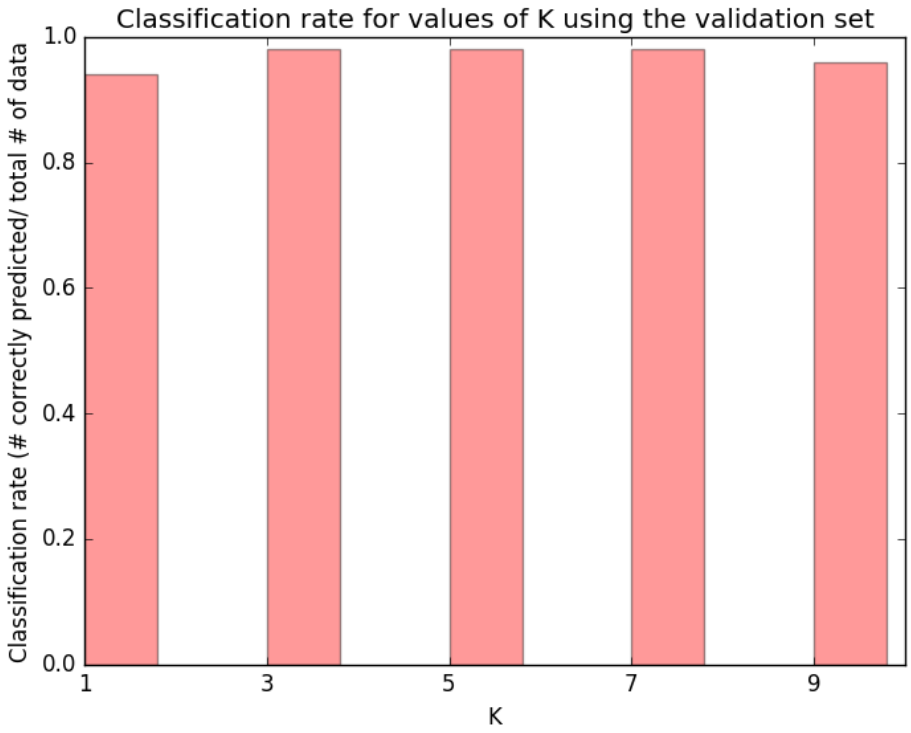
3. Logistic Regression vs KNN

3.1 k-Nearest Neighbors

1.

The kNN function was implemented inside run_knn.py and knearest.py was developed to run kNN for different values of k. kNN function was ran with k values of 1, 3, 5, 7, and 9. The results can be found in the table and plot below.

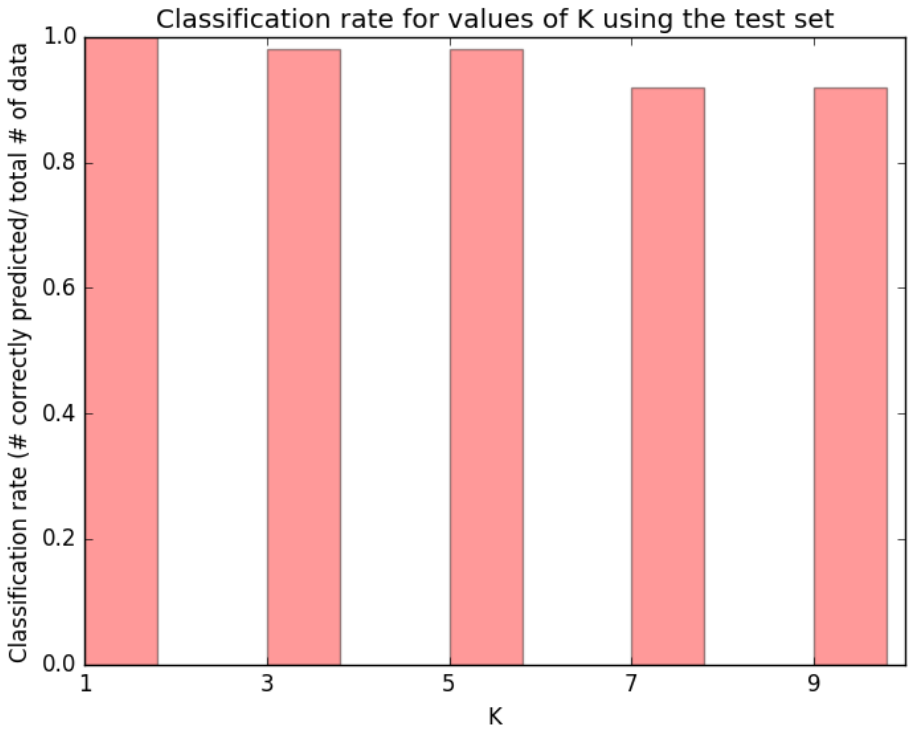
kNN using validation set					
K	**1**	**3**	**5**	**7**	**9**
Classification Rate	0.94	0.98	0.98	0.98	0.96



Overall, the performance of kNN is very good for all the 5 k values which were tested. k values of 3, 5, and 7 produced the best classification rate of 98%. Based on this any of the three can be chosen as the optimal k values. All three values are still valid if we follow the rule thumb that k must be less that the square root of the total number of training data (sqrt(200) = 14.14…). Considering all of the classification rate, it seems that the rate peaks somewhere between k=3 and k7. It can be estimated that k=5 is the peak, hence will produce the best classification rate.

Below, the results of running kNN with k values of 1, 3, 5, 7, 9 on the test set is demonstrated.

kNN using test set					
K	**1**	**3**	**5**	**7**	**9**
Classification Rate	1.00	0.98	0.98	0.92	0.92



2.

The chosen k value of 5 had the second highest classification rate. K=7 had the lowest rate and k=3 had the same rate as k=5. The performance of k=3 and k=5 stayed consistent but the classification rate for k=7 dropped significantly. This means that the optimal value of k likely a value less than 5. Another interesting finding is that k=1 had the best classification rate when using the test set and the worst rate when using validation set. This can be caused by the data being spread out causing the distances for large ks to be large, leading to misclassifications.

3.2 Logistic regression

1.

The files logistic_regression_template.py and logistic.py were completed. logistic_regression_template.py was used to experiment with the hyperparameter settings. All of the weights were initialized to 0.

To find the best hyperparameter setting, at first the number of iterations was set to 1000. This is because in all of the tested cases the classification rate converged within 1000 iterations. Keeping the number of iterations constant, different values for the learning rate were tried on the training, validation and test sets. It was found that the learning rate of 0.1 effectively sped up convergence in all three sets without causing the algorithm to crash or cause more classification errors. Moreover, higher learning rates cause the cross entropy to fluctuate at times and it made it increase after very few iterations. As a comparison, the training, validation, and test sets' classification rate converged at the 800^{th}, 205^{th}, and the 130^{th} iteration when using a learning rate of 0.07, but when the learning rate of 0.1 was used they converged at the 560^{th}, 140^{th}, and 90^{th}. Inspecting the validation set with the chosen learning rate, it is found that the validation cross entropy is at its minimum after roughly 200 iterations. There number of iteration is set to 200.

With the chosen hyperparameter setting, the final cross entropy and classification error on the three data sets are as follows:

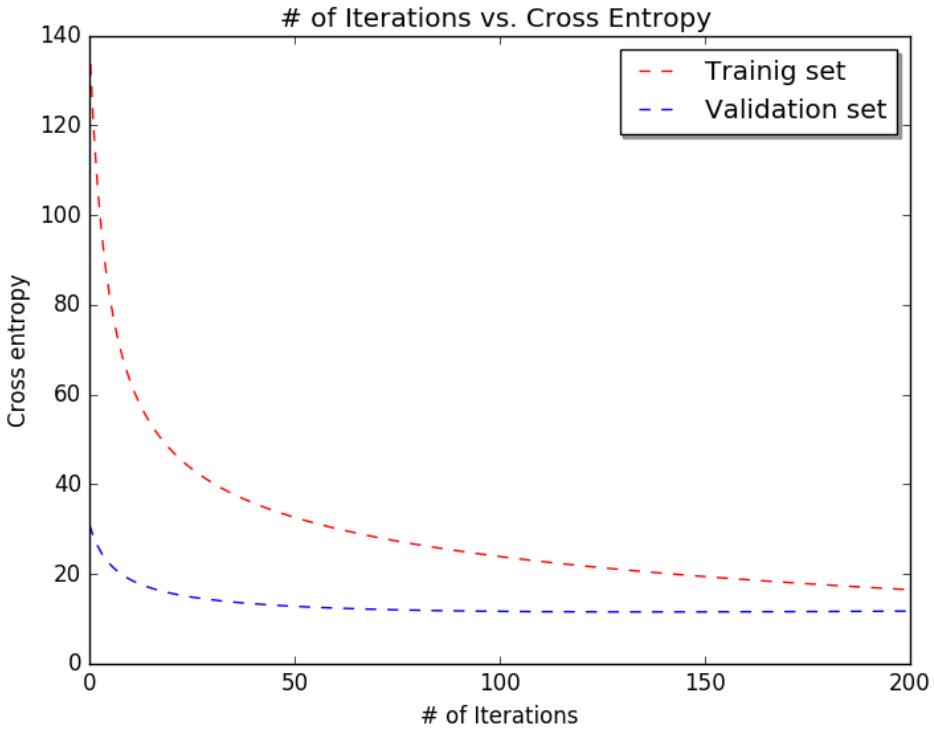
Learning rate = 0.1, number of iterations = 200			
	Training	**Validation**	**Test**
Final Cross Entropy	16.504934	11.689231	6.563695
Classification Error	0.025	0.08	0.04

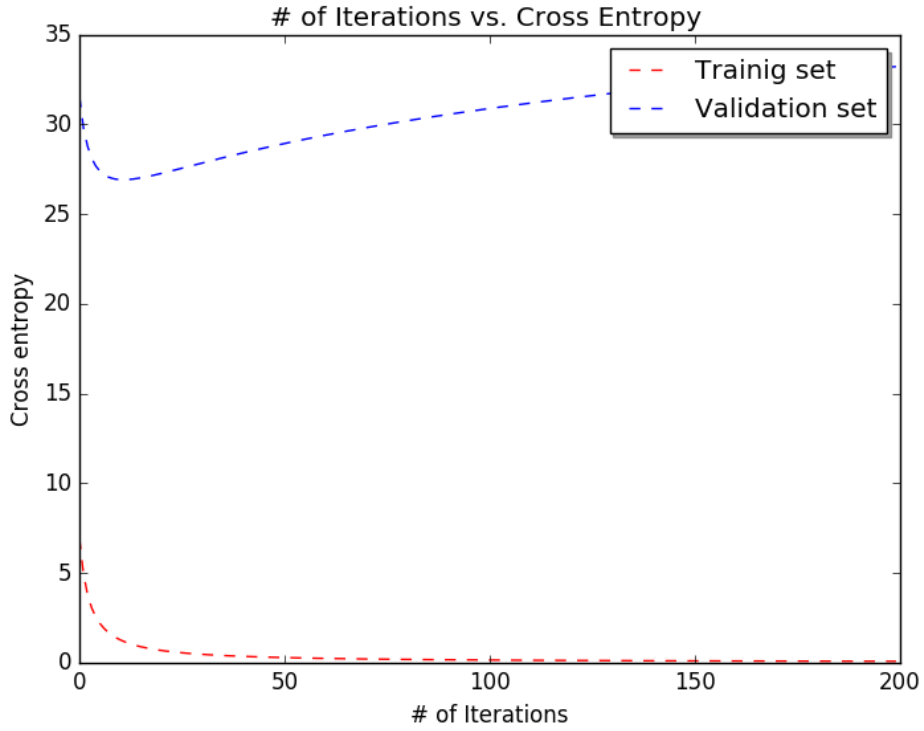
2.

Generally, with each iteration the cross entropy for both the training set and the validation set decreases. It is noticeable that after the classification rate of the validation set converges, the training set still improves. This shows a bias towards the training set, hence the cross entropy of the validation set will start increasing as shown in the two plots on the next page.

Since the hyperparameter settings were chosen based on mnist_train set and the validation set, the plot with mnist_train shows both sets' cross entropy decreasing for the most part. On the other hand, the plot with mnist_train_small does not look like the first plot at all. This is due to the fact that there are less training data, hence the small training set converges faster and starts over-fitting which causes the cross entropy of the validation set to increase.

Running the code several times resulted in the same exact result as the data is not changing, therefore the same calculations are done in each run.





3.3 Logistic regression

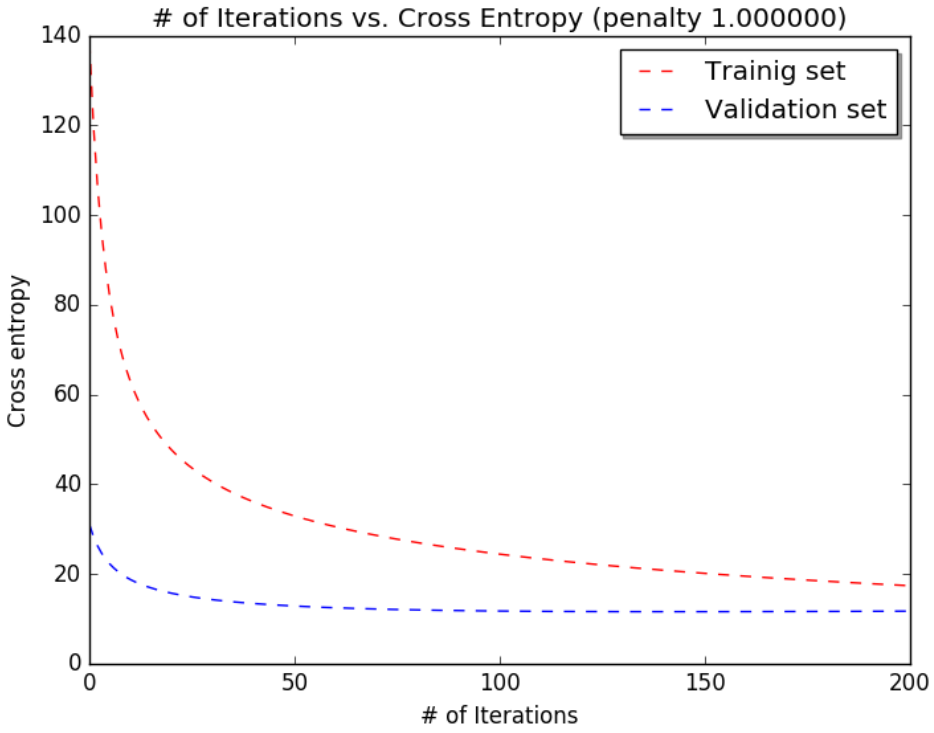
1.

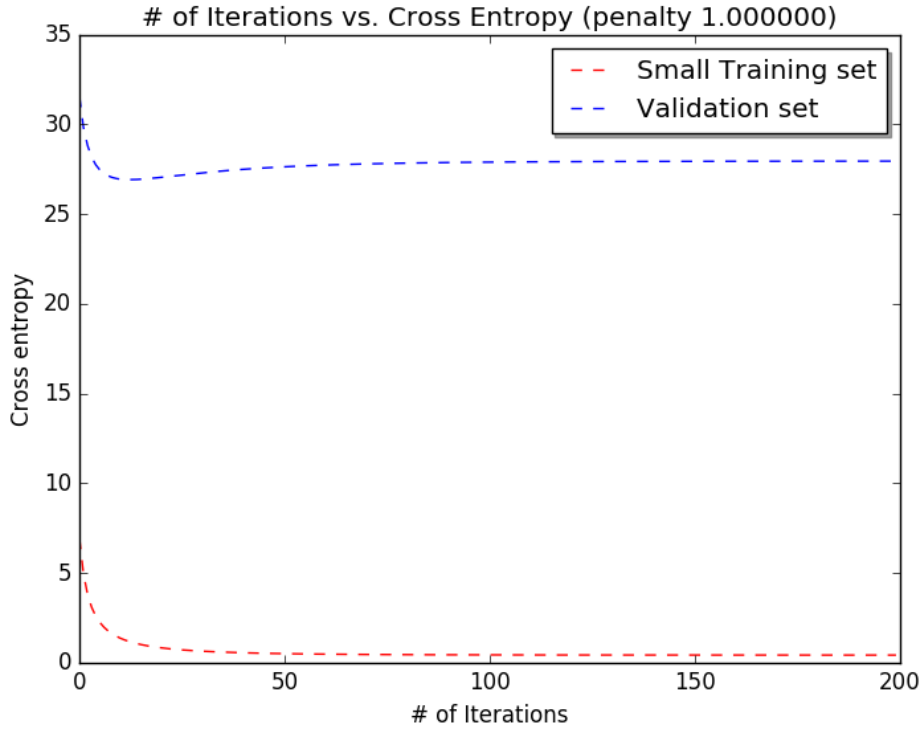
Similar to 3.2, at first the number of iteration was set to large constant. In this case it was set to 500. An algorithm was written which tested the penalty parameters of 0.001, 0.01, 0.1, 1. The result of the test are as follows:

		0.001	**0.01**	**0.1**	**1**
Training	**CE**	8.770545	8.785841	8.939039	10.484660
	FRAC	99.5	99.5	99.5	99.5
Validation	**CE**	13.465966	13.459892	13.400350	12.911905
	FRAC	92	92	92	92

As evident, the cross entropy for the training set increases with larger penalty parameters and for the validation set it decreases. When the penalty parameter is 1, the cross entropy for validation set is the smallest and closest to the training set cross entropy, therefore choose penalty parameter value of 1. Inspecting the cross entropy at each iterations when the penalty is set, it can be found that 200 iterations are sufficient.

In the two plots below, the cross entropy of the training set, small training set, and validation set is shown. Alike, question 3.2 there are the same tendencies in these plots. In the plot of the training set and the validation set there is no noticeable difference, but in the plot of the small training set there is. In this plot the cross entropy of the small training set is not noticeably different, but for the validation data the cross entropy start increasing after reaching its minimum but then quickly settles and does not change significantly.


# of Iterations vs. Cross Entropy (penalty 1.000000)
Trainig set
Validation set
Cross entropy
# of Iterations
0
20
40
60
80
100
120
140
0
50
100
150
200


# of Iterations vs. Cross Entropy (penalty 1.000000)
Small Training set
Validation set
Cross entropy
# of Iterations
0
5
10
15
20
25
30
35
0
50
100
150
200

2.

For this section the penalty parameters of 0.001, 0.01, 0.1, and 1 are used. In the table below the trend of CE and FRAC can be seen.

		0.001	**0.01**	**0.1**	**1**
Training	**CE**	16.505823	16.513824	16.593875	17.397693
	FRAC	97.5	97.5	97.5	97.5
Validation	**CE**	11.689213	11.689053	11.687563	11.682892
	FRAC	92	92	92	92
Test	**CE**	6.563943	6.566184	6.588594	6.812937
	FRAC	96	96	96	96

With increasing penalty parameter, the cross entropy for the training and test set increases and for the validation set decrease slightly. The classification change error stays the same. Testing penalty parameter of 10, it was found that the CE for the validation set will eventually start increasing. The increasing cross entropy can be attributed to the fact that with larger penalty parameter the weights are regulated more. Assuming the training data set is large enough to avoid overfitting, the optimal penalty parameter here is 0.001.

3.

In the case of using the larger training set (mnist_train), regularizing did not really have an effect on the final result. This could be due to the fact that there was enough training data points to prevent over-fitting. In the case of using the small training set (mnist_train_small), regularizing helped significantly. The small training set had fewer data points, hence it was more prone to over-fitting, and regularizing reduced the risk of over-fitting.

4.

In terms of classification rate, kNN gave the best results for the validation and test set of 98%. Both logistic regression and regularized logistic regression gave a rate of 92% for the validation set and 96% for the test set. Here kNN outperformed the other two algorithms. In general, kNN would be implemented in problems with non-linear boundaries. The advantage of kNN is that it is very easy to implements as no learning is needed, and the disadvantage is that it can be computationally heavy. The advantage of logistic regression is that it is not effected by mislabeled data and the disadvantages are that it has to be heavily tuned to get optimal results and it can over-fit the data.