CSC411 Introduction to Machine Learning
Assignment 1
Xi Chen (Student ID: 1002266013)

1 **Logistic Regression**

1.1

$$posterior \propto p(\boldsymbol{w})p(\boldsymbol{t}|\boldsymbol{x};\boldsymbol{w},w_0)$$

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|0,\alpha^{-1}\boldsymbol{I}) = \frac{1}{(2\pi)^{\frac{D}{2}}}\frac{1}{|\alpha^{-1}\boldsymbol{I}|^{\frac{1}{2}}}\exp\left\{-\frac{1}{2}\boldsymbol{w}^T(\alpha^{-1}\boldsymbol{I})^{-1}\boldsymbol{w}\right\}$$

$$= \frac{1}{(2\pi\alpha^{-1})^{\frac{D}{2}}}\exp\{-\frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w}\}$$

So, the loss function is:

$$Loss = -\log\left[\frac{1}{(2\pi\alpha^{-1})^{\frac{D}{2}}}\exp\{-\frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w}\}\prod_{i=1}^{N}(P(t^{(i)} = 1|x^{(i)})^{t^{(i)}}(1 - P(t^{(i)} = 1|x^{(i)})^{1-t^{(i)}})\right]$$

$$= -\left[-\frac{D}{2}\log(2\pi\alpha^{-1}) - \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w} + \sum_{\boldsymbol{i=1}}^{\boldsymbol{N}}\boldsymbol{t}^{(i)}logP(\boldsymbol{t}^{(i)} = 1|\boldsymbol{x}^{(i)}) + \sum_{i=1}^{N}(1-\boldsymbol{t}^{(i)})\log(1 - P(\boldsymbol{t}^{(i)} = 1|\boldsymbol{x}^{(i)})\right]$$

$$= \frac{D}{2}\log(2\pi\alpha^{-1}) + \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w} + \sum_{\boldsymbol{i=1}}^{\boldsymbol{N}}\boldsymbol{t}^{(i)}\log(1 + exp(-\boldsymbol{z}^{(i)})) + \sum_{i=1}^{N}(1-\boldsymbol{t}^{(i)})\boldsymbol{z}^{(i)}$$
$$+ \sum_{i=1}^{N}(1-\boldsymbol{t}^{(i)})\log(1 + \exp(-(\boldsymbol{z}^{(i)}))$$

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

$$= \frac{D}{2}\log(2\pi\alpha^{-1}) + \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w} + \sum_{\boldsymbol{i=1}}^{\boldsymbol{N}}\log\left(1 + exp\left(-w_0 - \sum_{d=1}^{D}\boldsymbol{w}_d\boldsymbol{x}_d^{(i)}\right)\right)$$
$$+ \sum_{i=1}^{N}(1-\boldsymbol{t}^{(i)})(w_0 + \sum_{d=1}^{D}\boldsymbol{w}_d\boldsymbol{x}_d^{(i)})$$

1.2

$$\frac{\partial Loss}{\partial \boldsymbol{w}_i} = \alpha\boldsymbol{w}_i - \sum_{j=1}^{N}\frac{\exp(-\boldsymbol{z}^{(j)})}{1 + \exp(-\boldsymbol{z}^{(j)})}\boldsymbol{x}_i^{(j)} + \sum_{j=1}^{N}(1-\boldsymbol{t}^{(j)})\boldsymbol{x}_i^{(j)}$$

$$= \alpha\boldsymbol{w}_i - \sum_{j=1}^{N}\boldsymbol{x}_i^{(j)}\left[\boldsymbol{t}^{(j)} - \frac{1}{1 + \exp(-\boldsymbol{z}^{(j)})}\right]$$

$$= \alpha\boldsymbol{w}_i - \sum_{j=1}^{N}\boldsymbol{x}_i^{(j)}[\boldsymbol{t}^{(j)} - p(\boldsymbol{t}^{(j)} = 1|\boldsymbol{x}^{(j)};\boldsymbol{w},w_0)]$$

$$\frac{\partial Loss}{\partial w_0} = -\sum_{j=1}^{N} \frac{\exp(-\boldsymbol{z}^{(j)})}{1+\exp(-\boldsymbol{z}^{(j)})} + \sum_{j=1}^{N}(1-\boldsymbol{t}^{(j)})$$
$$= -\sum_{j=1}^{N}\left[\boldsymbol{t}^{(j)} - \frac{1}{1+\exp(-\boldsymbol{z}^{(j)})}\right]$$
$$= -\sum_{j=1}^{N}\left[\boldsymbol{t}^{(j)} - p(\boldsymbol{t}^{(j)}=1|\boldsymbol{x}^{(j)};\boldsymbol{w},w_0)\right]$$

1.3

Pseudocode:

```
1. Setting learning rate λ, weight decay α
2. The $w_0, w_1, …, w_D$ are randomly generated value
3. Repeat until convergence or a large number of iterations {
        # calculate prediction
        for j = 1 to N {
            $p(\boldsymbol{t}^{(j)} = 1|\boldsymbol{x}^{(j)};\boldsymbol{w},w_0) = \frac{1}{1+\exp(-\boldsymbol{z}^{(j)})}$
        }

        #   calculate derivatives
        for i = 1 to D {
            $\frac{\partial Loss}{\partial \boldsymbol{w}_i} = \alpha\boldsymbol{w}_i - \sum_{j=1}^{N}\boldsymbol{x}_i^{(j)}[\boldsymbol{t}^{(j)} - p(\boldsymbol{t}^{(j)} = 1|\boldsymbol{x}^{(j)};\boldsymbol{w},w_0)]$
        }

        $\frac{\partial Loss}{\partial w_0} = -\sum_{j=1}^{N}[\boldsymbol{t}^{(j)} - p(\boldsymbol{t}^{(j)} = 1|\boldsymbol{x}^{(j)};\boldsymbol{w},w_0)]$

        # update the weights using gradient descent
        for i = 1 to D {
            $w_i \leftarrow w_i - \lambda\frac{\partial Loss}{\partial \boldsymbol{w}_i}$
                $= w_i + \lambda\sum_{j=1}^{N}\boldsymbol{x}_i^{(j)}[\boldsymbol{t}^{(j)} - p(\boldsymbol{t}^{(j)} = 1|\boldsymbol{x}^{(j)};\boldsymbol{w},w_0)] - \lambda\alpha\boldsymbol{w}_i$
        }
        $w_0 \leftarrow w_0 - \lambda\frac{\partial Loss}{\partial \boldsymbol{w}_0}$
            $= w_0 + \lambda\sum_{j=1}^{N}[\boldsymbol{t}^{(j)} - p(\boldsymbol{t}^{(j)} = 1|\boldsymbol{x}^{(j)};\boldsymbol{w},w_0)]$
   }
```

2. **Decision Trees**

2.1

For the first attribute:

$IG(Satisfied|Overcooked\ pasta)$
$= H(Satisfied) - \ H(Satisfied|Overcooked\ pasta)$
$= \left(-\frac{3}{5}log_2\frac{3}{5}-\frac{2}{5}log_2\frac{2}{5}\right) - \ H(Satisfied|Overcooked\ pasta)$
$= 0.971 - \left[\frac{3}{5}\times\left(-\frac{1}{3}log_2\frac{1}{3}-\frac{2}{3}log_2\frac{2}{3}\right)+\frac{2}{5}\times(-log_2 1)\right]$
$= 0.42$

$IG(Satisfied|Waiting\ time)$
$= H(Satisfied) - \ H(Satisfied|Waiting\ time)$
$= \ 0.971 - H(Satisfied|Waiting\ time)$
$= 0.971 - \left[\frac{3}{5}\times\left(-\frac{2}{3}log_2\frac{2}{3}-\frac{1}{3}log_2\frac{1}{3}\right)+\frac{2}{5}\times\left(-\frac{1}{2}log_2\frac{1}{2}-\frac{1}{2}log_2\frac{1}{2}\right)\right]$
$= 0.02$

$IG(Satisfied|Rude\ waiter)$
$= H(Satisfied) - \ H(Satisfied|Rude\ waiter)$
$= \ 0.971 - H(Satisfied|Rude\ waiter)$
$= 0.971 - \left[\frac{4}{5}\times\left(-\frac{1}{2}log_2\frac{1}{2}-\frac{1}{2}log_2\frac{1}{2}\right)+\frac{1}{5}\times(-log_2 1)\right]$
$= 0.171$

So we **choose the "Overcooked pasta" as the first attribute** to be split on since it has the largest information gain.

For the second attribute:

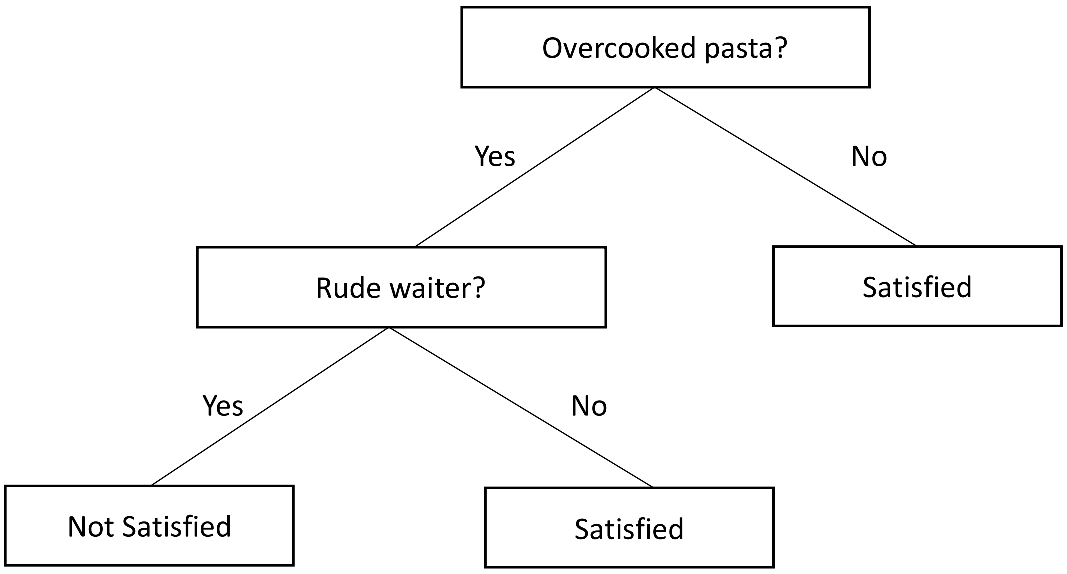
$IG(Satisfied|Waiting\ time)$
$= H(Satisfied) - \ H(Satisfied|Waiting\ time)$
$= \left(-\frac{1}{3}log_2\frac{1}{3}-\frac{2}{3}log_2\frac{2}{3}\right) - \ H(Satisfied|Waiting\ time)$

$= \ 0.918 - H(Satisfied|Waiting\ time)$
$= 0.918 - \left[\frac{2}{3}\times\left(-\frac{1}{2}log_2\frac{1}{2}-\frac{1}{2}log_2\frac{1}{2}\right)+\frac{1}{3}\times(-log_2 1)\right]$
$= 0.251$

$IG(Satisfied|Rude\ waiter)$
$= H(Satisfied) - \ H(Satisfied|Rude\ waiter)$
$= \ 0.918 - H(Satisfied|Rude\ waiter)$
$= 0.918 - \left[\frac{2}{3}\times(-log_2 1)+\frac{1}{3}\times(-log_2 1)\right]$
$= 0.918$

So we **choose the “Rude Waiter” as the second attribute** to be split on since it has larger IG.

Thus, we get the following decision tree:



2.2

According to the decision, we get the following result:

Person ID	Overcooked pasta?	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

3.1.1

For k = 1, 3, 5, 7 and 9, I get the following results:

K	Classification Rate
1	0.94
3	0.98
5	0.98
7	0.98
9	0.96

Table 1. Classification rate for different value of k on validation data

According the results listed above, I get the following graph:

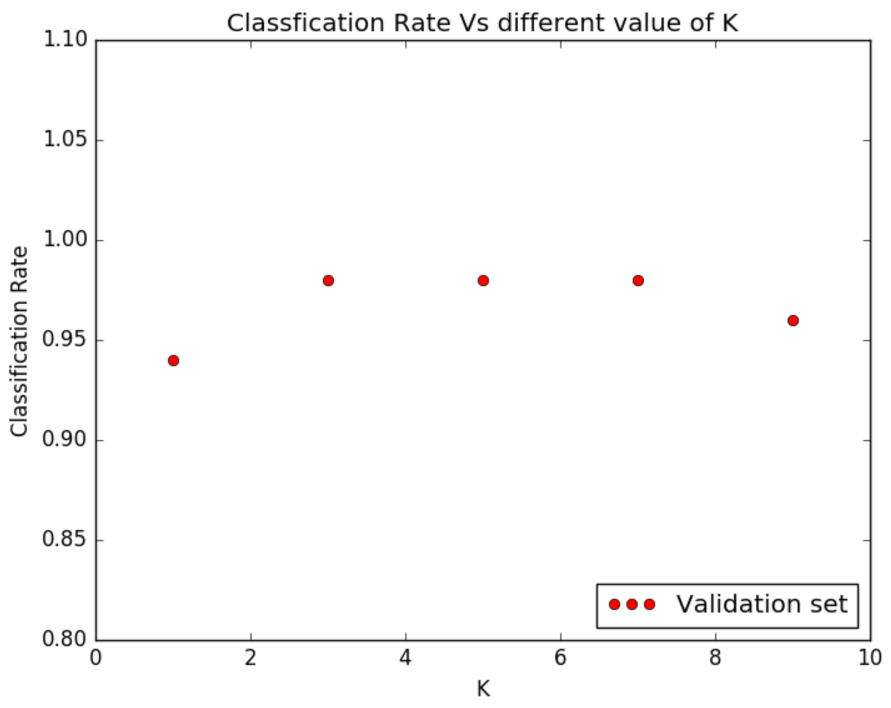


Fig 1. Classification rate vs. k values in validation set

As we can see, for the validation data, we get a high classification rate using KNN method. The classification rate increases when the value of k increases and reaches the peak (0.98) when k = 3, 5 and 7. If the value of k continues to increase to 9, the classification rate begins to drop.

In the given data set, k = 3, 5 and 7 give us same results. We know that when k is small, the result will be affected by noise (mislabeled) data points; when k is too large, some data points which are far away will be considered as the neighbor point, which may also make the result worse. So, I would **choose k = 5 to be the best value of k.**

For the test data set, we can get the classification rate using the chosen k*, k* -2 and k* + 2. The results are listed as below:

K	K* - 2	K*	K* + 2
Classification rate	0.98	0.98	0.92

Table 2. Classification rate for different value of k on test data

3.1.2

In order to analyze the test performance, we can also draw the results for the test data set.

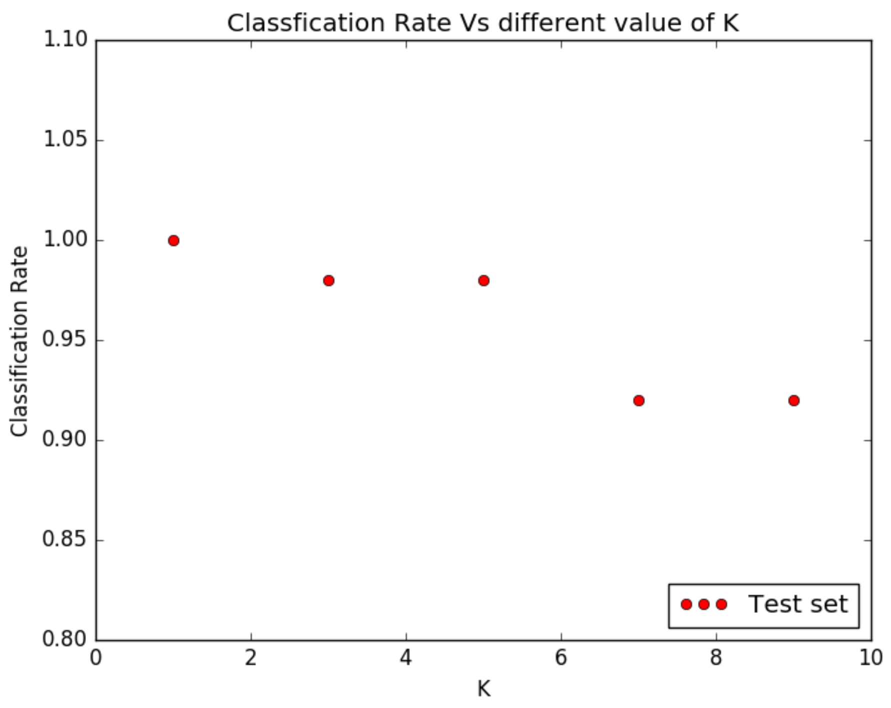


Fig 2. Classification rate vs. k values in test set

As the result above shows, it seems that the test performance does not correspond to the validation performance. For the test data, the classification rate begins to drop when k = 7, while in the validation data, the corresponding classification rate is much higher. Another difference is that when k = 1, the test data set shows 100% correction rate and it continues to drop when k increases.

The reasons should be:

The best value of k depends on the data distribution. For the given dataset, it is possible that the test data has high similarity with the training data, which means that each test data has similar distribution with the training data in all dimensions. So when the value of k is 1, each test data is labeled as the target of the nearest training data with high probability to be the right answer.

When the value of k increases, some training data which is far away from the test data is treated as the neighbors. Thus, the increasing k will make the noise to be the majority of the k nearest neighbors, which results in the mislabel and decreasing classification rate. As k becomes larger, more uncorrelated data is treated as the k nearest neighbors and the classification may be heavily affected by these data such that the result gets worse.

Therefore, the test performance does not correspond to the validation performance.

3.2 Logistic Regression

3.2.1

Since the number of iterations may be affected by the initialization of weights, I initialize the weights by a **normal distribution with mean 0 and variance 0.05** such that the ideal number of iterations is a stable number.

The number of iterations has impact on whether the model will be overfitting or underfitting. When we train the training set in large number of iterations, the model may get overfitting and the cross entropy of validation set may increase; when the number of iterations is too small, the model will be underfitting such that we cannot get the best performance. Thus, the **early stopping** should be used in the training process to prevent overfitting.

Thus, the hyperparameter settings I used are:

weights	initialized by normal distribution with mean 0 and variance 0.05
learning_rate	0.05
weight_regularization	False
num_iterations	450

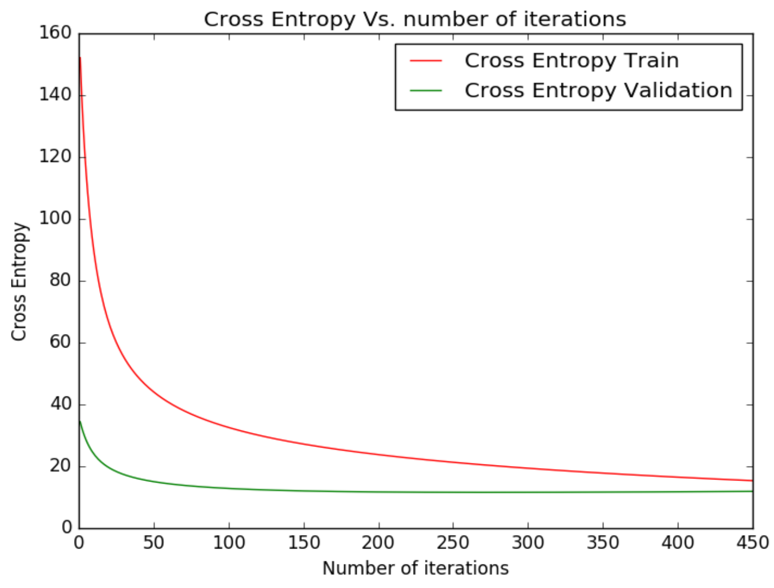
Under these settings, we get the following results:

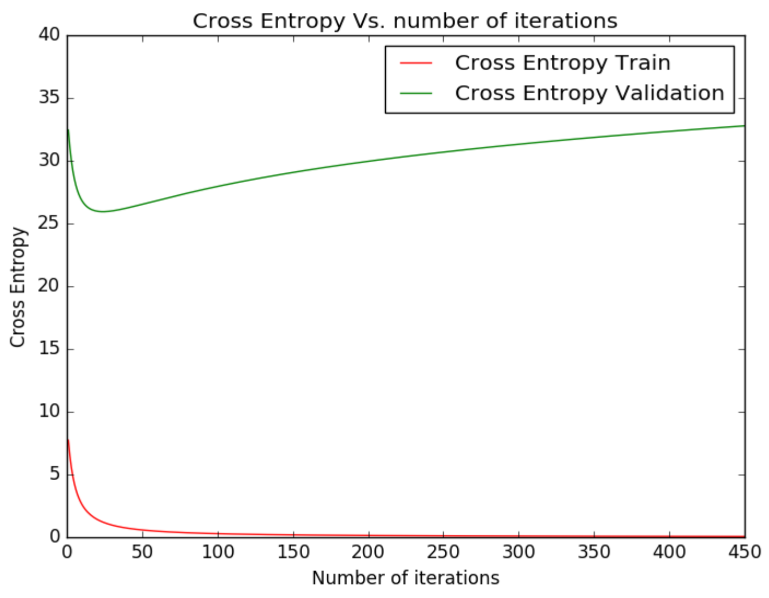
Data set	Cross Entropy	Classification Error
Training set	15.278520	0.02
Validation set	11.245727	0.08
Test set	6.560702	0.04

Table 3. Final cross entropy and classification error

3.2.2

Using the hyperparameter settings above, I get the following results for the change of cross entropy vs. number of iterations.

mnist_train	 Fig 3. Changes of Loss during training(mnist)

mnist_train_small	 Fig 4. Changes of Loss during training(mnist_small)

As the Fig 3 shows, when we use the mnist_train as training data, the cross entropy of training data and validation data both decrease as the number of iterations increases. If I continue to train the model, the weights will converge, which means that the training set may get 0 classification error and cross entropy should reach 0. However, the validation set may become overfitting and the cross entropy of validation data may go up again, so I just stop at this point (around 450 iterations) in order to have best performance for the unseen data.

However, when we use the mnist_train_small as the training data, the weights converge much faster than the the mnist_train data. The cross entropy of training set goes down quickly and reaches to a value near 0 after about 200 iterations. For the validation set, the cross entropy firstly decreases and begins to go up after 25 iterations, which is due to the overfitting.

I run my code for several time, it seems that **the tendency of how the cross entropy change is same** as the above results show, **but the final cross entropy for the two data sets change every time**, with different values in a small range.

For the mnist_train data set, the cross entropy of training data is about 15~16 and the value of validation data is about 11.3~12.5. For the mnist_train_small data set, the final cross entropy of training data is around 6.1e-2 with little difference, while for the validation data, the value falls in the area between 31 and 36.

I think the reason should be the random weight initialization. Each time I run the code, the weights are initialized randomly by normal distribution. Since the cross entropy is determined by prediction and prediction is related to the weights in each iteration, the final value of cross entropy should be different with randomly generated weights.

The reason for the final value falls in small range is the small variance in normal distribution. After fixed number of iterations (450 in my model), the weights are updated to similar values for the same data set in each run. Thus, the final cross entropy for each run will fall in small range. I also try to initialize the weights with large variance, the final results differ a lot for in this case. Thus, the variance in the initialization will affect the final cross entropy given fixed number of iterations.

3.3 Regularized logistic regression

3.3.1

When I use regularization in logistic regression, I found that the results for mnist_train data set is similar to the one without regularization. But for the mnist_train_small data set, there were some differences between the two results.

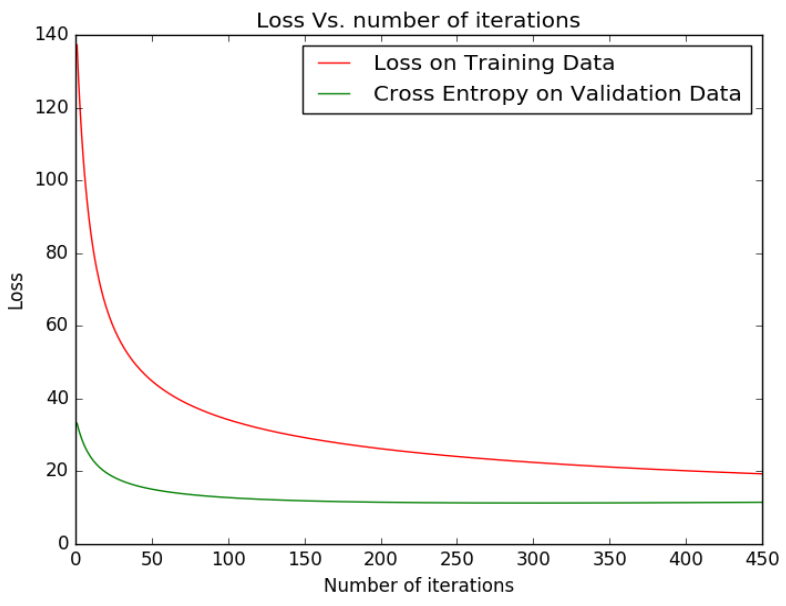
The hyperparameter settings are:

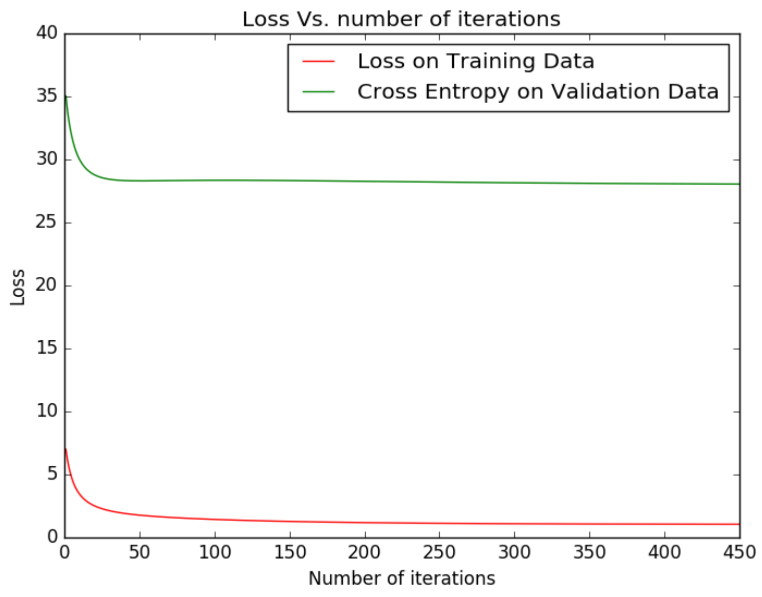
weights	initialized by normal distribution with mean 0 and variance 0.05
learning_rate	0.05
weight_regularization	True
num_iterations	450
weight_decay	1

The test error for the mnist_train training set is: 0.04

The test error for the mnist_train_small training set is: 0.26

The plots of the change of loss during training is shown below, I just ignore the constant term in the loss. The two curves are loss for training set and cross entropy for validation set.

mnist_train	 Fig 5. Changes of Loss during training(mnist)

mnist_train_small	 Fig 6. Changes of Loss during training(mnist_small)

3.3.2
When other hyperparameters are fixed (including the initial value of weights), I observed how the loss and classification rate changed during the training.

For the mnist_train training set, the loss of training data increases while the cross entropy of validation data decreases as the value of α increase, but the **change is not obvious**. For the classification error, the value for both training data and validation data does not change when α changes.

α	Training loss	Training error	Validation loss	Validation error
0.001	15.3569339	0.025	11.8228569	0.08
0.01	15.3657099	0.025	11.8223536	0.08
0.1	15.4535173	0.025	11.8174707	0.08
1	16.3351071e	0.025	11.7828119	0.08

For the mnist_train_small set, both the loss and classification error changes in the same way with the mnist_train set. However, **the change in loss is much more obvious than the first case.**

α	Training loss	Training error	Validation loss	Validation error
0.001	0.0614543277	0	33.6223712	0.20
0.01	0.0642032449	0	33.4554400	0.20
0.1	0.0941767083	0	32.0560030	0.20
1	0.41516998	0	27.96052996	0.20

The reason for this phenomenon is:

In the mnist_train dataset, I use early stopping to avoid overfitting, so the regularizer has little impact on the final performance. When we train the model using mnist_train_small dataset, the overfitting is severe. The weight penalty helps to push the weights towards 0, which slows down the weight update process and avoids overfitting, especially when α is large.

The unchanged classification rate is due to the fact that the weights change slowly after a number of iterations. So the predictions may keep nearly unchanged and are still at the same side of the decision boundary even when we increase the value of α.

From the above result, we can conclude that **α = 1 is the best value** in this case.

3.3.3
When I compare the results with and without regularizer, I found that for the mnist_train training set, the results are almost the same. I think the reason should be that when I train the model, I just stop when the model is becoming overfitting, so the regularizer has no effect on the final result.

But for the mnist_train_small data set, the model becomes overfitting quickly because of the small training data size. The mode fits the training data too well such that the noise is captured and has negative impact on unseen data. In this case, the regularizer begins to affect the result on the validation data. When the value of α is large, it helps to avoid the overfitting, which we can see from the decreasing cross entropy of validation data when α increases.

Thus, I think for the small training data, the regularized logistic regression model works better while in large data set, the regularizer has little impact on the result.

3.3.4
Comparing the results against that from KNN, I found that KNN has better performance in this case, especially when the value of k is small.

In general, KNN is used when there exist nonlinear boundaries in data, since logistic regression can only find a linear boundary in the classification problem. When the data has large dimensionalities, logistic regression has better performance than KNN, since KNN will be affected by curve of dimensionality.

The pros and cons for logistic regression and KNN are listed as below:

Logistic regression:
Pros:
1. Logistic regression is faster than KNN when the data size is large
2. It provides probability of the outcomes, which is convenient for observation

Cons:
1. When training size is too small, the result may be heavily affected by outliers
2. Hyperparameters need to be tuned carefully to build the model

KNN:

Pros:

1. KNN can handle nonlinear boundary problems
2. KNN can handle multi-class problems
3. KNN is nonparametric model, so it does not have training process and there is no need to tune the parameters to build the model

Cons:

1. KNN has high time complexity (O(ND)), the run time could be very long when training sample is large
2. KNN is sensitive to mislabeled data
3. Performance is poor when data has high dimensionality