Question 2: Feature Maps

(a) This data is not seperable by a linear classifier. We know that a linear classifier separating this data must partition $\mathbb{R}^2$ into half-spaces H_1 (positive) and H_0 (negative) based on a threshold t.

In this dataset, $p_1 = (-2, -1)$ and $p_2 = (2, 3)$ have label 1, and so are in H_1. We know that half-spaces are convex, and so any convex combination of p_1 and p_2 must lie in H_1. If we take $\lambda = 3/4$, then

$$p_2 = p_1 + \lambda(p_2 - p_1) = (-2, -1) + \frac{3}{4}(2 - (-2), 3 - (-1)) = (-2, -1) + \frac{3}{4}(4, 4) = (1, 2)$$

must lie in H_1 since it is convex combination of p_1 and p_2. However, we know that p_3 has label 0, and so must lie in H_0. This is a contradiction, and so no linear classifier can separate this data.

(b) We will assume that a threshold $t = 0$ is used. The constraints are as follows:

$$-2w_1 + w_2 \geq 0 \tag{1}$$
$$w_1 + 4w_2 < 0 \tag{2}$$
$$2w_1 + 9w_2 \geq 0 \tag{3}$$

(c) The feasible area is given by the intersection of the blue, red, and green half-spaces below, which is the shaded region to the left of the red line, above the green line, and below the blue line. The feasible area is shown in Figure 1.

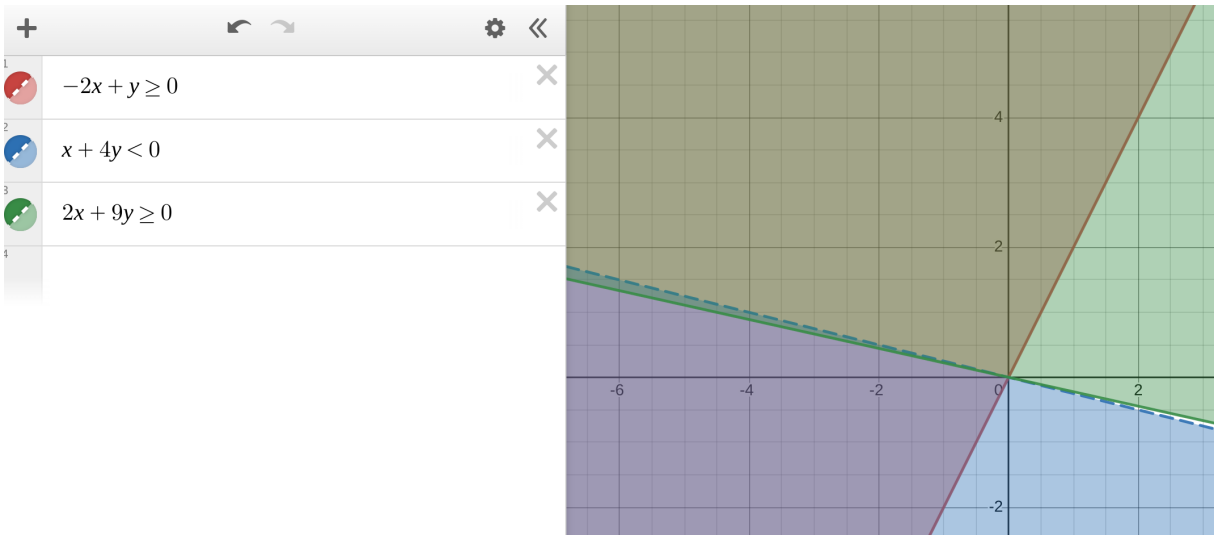

Figure 1: Feasible area for w_1, w_2

Question 3: kNN vs Logistic Regression

3.1 (a) The plot of classification vs validation accuracy is given below in Figure 2.

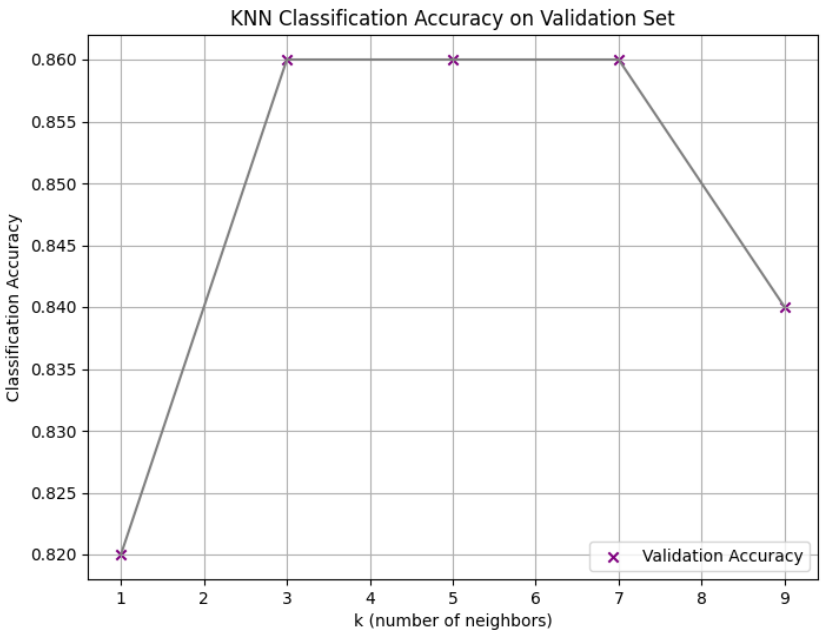

Figure 2: Classification vs Validation Accuracy

(b) The value of k^* chosen is $k^* = 3$. This is because 3 is the smallest value of the hyperparameter k which achieves the maximum accuracy seen across all values of k (0.860). I have chosen this value of k in adherence with the principle of Occam's Razor, which states that the simplest solution is often the best one. In this case, the simplest solution is the least complex model (lowest k), as the model complexity increases with k. Here is a report of the validation and test accuracies for $k^*, k^* + 2, k^* - 2$:

```
Classification accuracies for k_star=3
    Validation accuracy: 0.86
    Test accuracy: 0.92
Classification accuracies for (k    _star + 2)=5
    Validation accuracy: 0.86
    Test accuracy: 0.94
Classification accuracies for (k_star - 2)=1
    Validation accuracy: 0.82
    Test accuracy: 0.88
```

We see that the maximum validation accuracy is reached when $k^* = 3$, increasing from 0.82 when $k = 1$ and staying at the same value of 0.86 when $k = 5$.

However, the test accuracy increases from 0.88 to 0.92 when going from $k^* - 2$ to k^*, and continues to increase with $k = k^* + 2 = 5$. This illustrates that the simplest model that performs best on the validation set may not always be the one that performs best on the test set.

3.2 (a) Done!

(b) I've completed the missing parts in `run_logistic_regression`, and run the code on both `mnist_train` and `mnist_train_small`. The value returned by `run_check_grad` is small, and is given below:

```
diff = 2.246931370781212e-08
```

I used an initial value of $w = \vec{0}$, and a learning rate of $\eta = 0.2$. I tried values in the range $\{0.1, \ldots, 1.0\}$, and I chose this value as it was the largest value of η that did not cause the loss to fluctuate in the early training iterations on both the training and validation sets.

I used 500 iterations, as it appears that both the cross entropy and classification accuracies converge on the training and validation sets after 500 iterations.

The cross entropy and classification accuracies for the training and validation sets for $\eta = 0.2$ and 500 iterations are given below:

```
Classification accuracy on training set: 1.0
Classification accuracy on validation set: 0.88
Classification accuracy on test set: 0.92
Cross entropy on training set: 0.02149375612870086
Cross entropy on validation set: 0.19186085743772488
Cross entropy on test set: 0.2142534589131518
```

(c) Here are figures showing how cross-entropy changes as training progresses, one for the `mnist_train` and one for the `mnist_train_small` datasets.

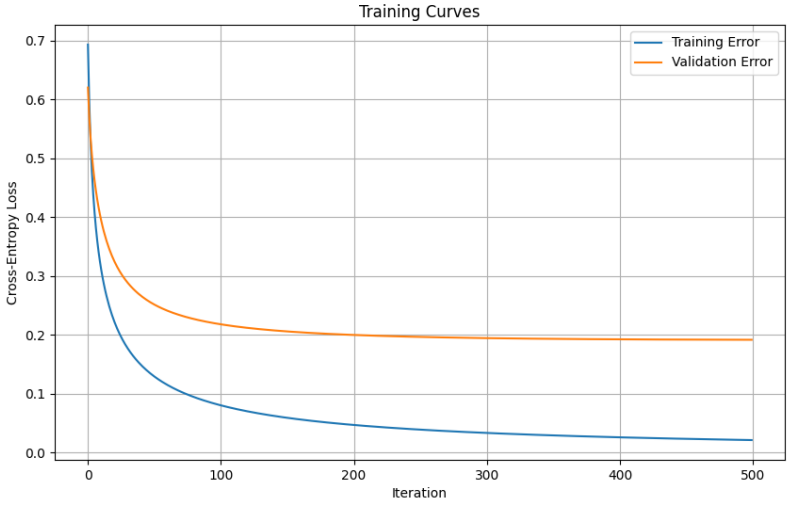

Figure 3: Cross entropy vs Training Iterations for `mnist_train`

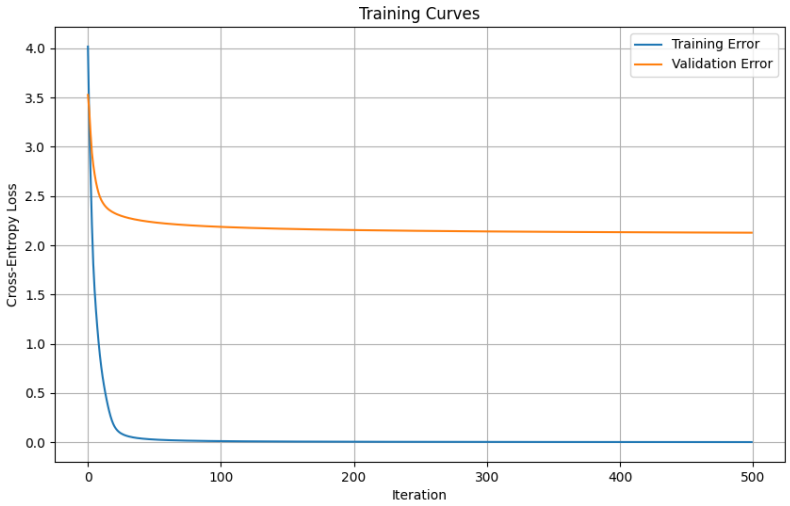

Figure 4: Cross entropy vs Training Iterations for `mnist_train_small`

We see that the validation error increases as the model trains on `mnist_train_small`, indicating that overfitting is taken place. No such phenomena is observed for `mnist_train`, indicating that the model is not overfitting on this dataset.

From running the code multiple times, we see that the results change slightly. This is due to the random initialization of weights (w) in the function `run_logistic_regression`. One way to choose the best hyperparameter that is the weight initialization is to run the code multiple times, and choose the weight initialization that gives the best validation accuracy.

Question 4: Locally Weighted Regression

(a) First, define

$$J(w) = \frac{1}{2}\sum_{i=1}^{N} a^{(i)}(y^{(i)} - w^T x^{(i)})^2 + \frac{\lambda}{2}\|w\|^2 = \frac{1}{2}\langle y - Xw, A(y - Xw)\rangle + \frac{\lambda}{2}\langle w, w\rangle$$

We'll calculate the gradient of J by first finding its Jacobian, and then taking the transpose. We will use the following identities:

$$\begin{aligned}
\partial_x\langle x, A \cdot x\rangle &= x^T(A + A^T) = 2x^T A \qquad \text{if } A \text{ is symmetric} \\
\partial_x(A - Bx) &= -B
\end{aligned}$$

We compute the Jacobian as follows:

$$\begin{aligned}
\partial J(w) &= \frac{1}{2}\partial\langle y - Xw, A(y - Xw)\rangle + \frac{\lambda}{2}\partial\langle w, w\rangle \\
&= (y - Xw)^T \cdot A \cdot (-X) + \lambda w^T \\
&= (y - Xw)^T(-AX) + \lambda w^T
\end{aligned}$$

Taking the transpose, we get

$$\nabla_w J(w) = -X^T A^T(y - Xw) + \lambda w$$

Since J is convex, its minimum can be found by setting its gradient to 0 and solving for w. We do this as follows:

$$\begin{aligned}
\nabla_w J(w) &= 0 \\
-X^T A^T(y - Xw) + \lambda w &= 0 \\
-X^T A^T y + X^T A^T X w + \lambda w &= 0 \\
(X^T A^T X + \lambda I)w &= X^T A^T y \\
w &= (X^T A^T X + \lambda I)^{-1} X^T A^T y \\
&= (X^T A X + \lambda I)^{-1} X^T A y \qquad \text{since } A \text{ is symmetric}
\end{aligned}$$

which shows the desired closed-form solution.

(b) Done!

(c) Here are the training and validation losses as a function of τ:

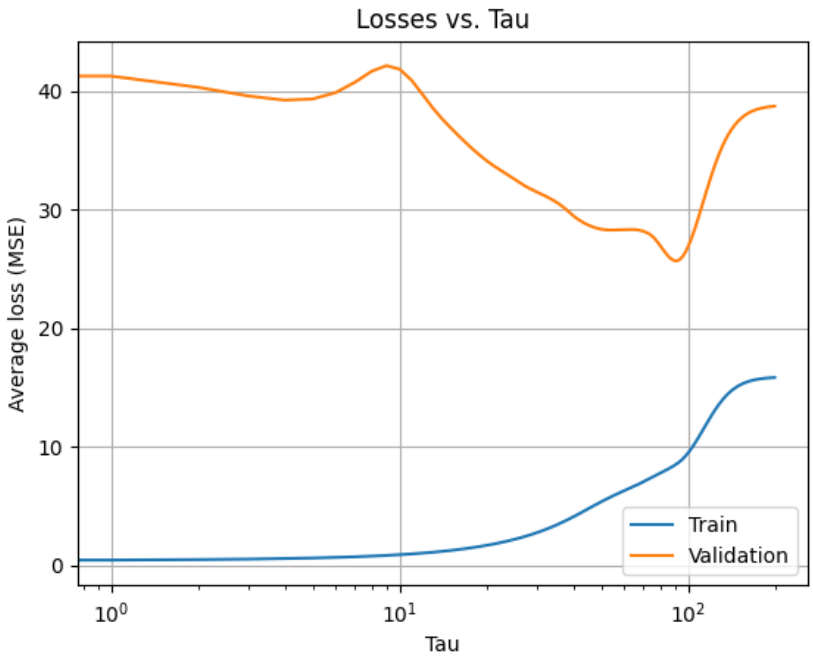

Figure 5: Training and Validation Losses vs τ

(d) By Lemma 1 below, we see that as $\tau \to 0$, the matrix A contains 1 in the ith diagonal entry if and only if $x^{(i)}$ is the closest point to x, and zeroes otherwise. Thus, in this case, our algorithm behaves like a k-nearest neighbors model with $k = 1$, since it only trains with respect to the closest point to x.

By Lemma 2, we see that as $\tau \to \infty$, the matrix A is given by I/N, where I is the identity matrix of size $N \times N$. Thus, in this case, our algorithm behaves exactly like a linear regressor since it weights each point exactly the same and performs linear regression. Indeed, the closed-form solution in part (b) reduces to the closed-form solution for linear regression when $A = I/N$.

Lemma 1. *If the distances* $\left\|x - x^{(j)}\right\|$ *are distinct across all* j*, then*

$$\lim_{\tau \to 0} a^{(i)} = \lim_{\tau \to 0} \frac{\exp\left(-\left\|x - x^{(i)}\right\|^2 / 2\tau^2\right)}{\sum_j \exp\left(-\left\|x - x^{(j)}\right\|^2 / 2\tau^2\right)} = \begin{cases} 1 & \text{if } i = \operatorname{argmin}_j \left\|x - x^{(j)}\right\|^2 \\ 0 & \text{otherwise} \end{cases}$$

Proof. First, let's divide by $\exp\left(-\left\|x - x^{(i)}\right\|^2 / 2\tau^2\right)$ to get

$$\lim_{\tau \to 0} \frac{\exp\left(-\left\|x - x^{(i)}\right\|^2 / 2\tau^2\right)}{\sum_j \exp\left(-\left\|x - x^{(j)}\right\|^2 / 2\tau^2\right)} = \lim_{\tau \to 0} \frac{1}{1 + \sum_{j \neq i} \exp\left(\frac{\left\|x - x^{(i)}\right\| - \left\|x - x^{(j)}\right\|^2}{2\tau^2}\right)}$$

Let's assume that $i = \operatorname{argmin}_j \left\|x - x^{(j)}\right\|^2$. Then, we have that for $j \neq i$, the argument of the exponential is negative, and goes to $-\infty$ as $\tau \to 0$. Thus, we have that

$$\lim_{\tau \to 0} a^{(i)} = \frac{1}{1 + 0} = 1 \qquad \text{if } i = \operatorname{argmin}_j \left\|x - x^{(j)}\right\|^2$$

Conversely, if $i \neq \text{argmin}_j \left\|x - x^{(j)}\right\|^2$, then we have that for $j \neq i$, the argument of the exponential is positive, and goes to ∞ as $\tau \to 0$. Thus, we have that

$$\lim_{\tau \to 0} a^{(i)} = \frac{1}{1+\infty} = 0 \qquad \text{if } i \neq \text{argmin}_j \left\|x - x^{(j)}\right\|^2$$

which completes the proof.

□

Lemma 2. *We have that*

$$\lim_{\tau \to \infty} a^{(i)} = \lim_{\tau \to \infty} \frac{\exp\left(-\left\|x - x^{(i)}\right\|^2 / 2\tau^2\right)}{\sum_j \exp\left(-\left\|x - x^{(j)}\right\|^2 / 2\tau^2\right)} = \frac{1}{N}$$

where N is the number of training examples.

Proof. This is simple, and follows from

$$\lim_{\tau \to \infty} a^{(i)} = \frac{\exp(0)}{\sum_j \exp(0)} = \frac{1}{N}$$

□

(e) **Advantage**: The advantage of using locally weighted regression is that it allows one to train a model that is more expressive than a linear model (in the style of KNN) while still performing some non-trival training process. This training process learns from values of *important* training examples that have high values of $a^{(i)}$. This is in the spirit of linear regression, in which our model learns from training examples that are close to the point we are trying to predict.

Disadvantage: One disadvantage comes from the curse of dimensionality: if the distances of x to $x^{(j)}$, the training examples, are same due to being in a high-dimensional space, then $a^{(i)}$ will roughly be the same, causing the model to behave like a linear regressor. In this case, there is no benefit to the huge computational expense of *recomputing weights* everytime one wants to make a prediction.