ECE521H1 - Assignment 1

Rahul Chandan (999781801) & Christine Jieun Lee (999836646)

February 7, 2017

1 k-Nearest Neighbour

1.1 Geometry of k-NN

Question 1: Describe a 1-D dataset of two classes such that when applying k-NN, the classification accuracy of the k-NN classifier on the training set is a periodic function of the hyper-parameter k.

Solution:

For the classification accuracy of the k-NN classifier on the training set to be a periodic function of the hyper-parameter k, when applying k-NN on a 1-D dataset of two classes, that dataset should contain points with alternating classes. This can be seen below...

Figure 1: 1-D dataset of two classes with alternating points.

Question 2: Owing to something called the curse of dimensionality, the k-NN model can encounter severe difficulty with high-dimensional data using the sum-of-squares distance function. In this question, we will investigate the curse of dimensionality in its worst-case scenario. Consider a dataset consisting of a set of N-dimensional input data $\mathbf{x}$, $\mathbf{x} \in R^N$.The dataset is i.i.d. and is sampled from an N-dimensional Gaussian distribution such that each dimension is independent with mean zero and variance σ^2, i.e. P(x) = $\prod_{n=1}^{N} \mathcal{N}(x_n; 0; \sigma^2)$. Show that for two random samples $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ from this dataset,

$$var\left(\frac{\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|_2^2}{E[\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|_2^2]}\right) = \frac{N+2}{N} - 1$$

Solution:

Using the variance identity, $Var(X) = E[X^2] - (E[X])^2$, we get Eq. 1:

$$var\left(\frac{\|x^{(i)} - x^{(j)}\|_2^2}{E\left[\|x^{(i)} - x^{(j)}\|_2^2\right]}\right) = E\left[\left(\frac{\|x^{(i)} - x^{(j)}\|_2^2}{E\left[\|x^{(i)} - x^{(j)}\|_2^2\right]}\right)^2\right] - \left(E\left[\frac{\|x^{(i)} - x^{(j)}\|_2^2}{E\left[\|x^{(i)} - x^{(j)}\|_2^2\right]}\right]\right)^2 \tag{1}$$

By evaluating each of the terms on the right-hand side of Eq. 1, the equality can be satisfied. Let's start with the subtracted term...

$$\left(E\left[\frac{\|x^{(i)} - x^{(j)}\|_2^2}{E\left[\|x^{(i)} - x^{(j)}\|_2^2\right]}\right]\right)^2 = \left(\frac{1}{E\left[\|x^{(i)} - x^{(j)}\|_2^2\right]} E\left[\|x^{(i)} - x^{(j)}\|_2^2\right]\right)^2$$
$$= (1)^2 = 1$$

The evaluation of the remaining term is slightly more involved. Start by listing some identities that will be used in the derivation...

- The difference between two independent Gaussian random variables is an independent Gaussian, i.e. $d_n = x_n^{(i)} - x_n^{(j)}$ is a Gaussian;
- $E[d_n^2 d_m^2] = E[d_n^2]E[d_m^2]$ if d_n and d_m are i.i.d;
- The second moment is $E[d_n^2] = (\sqrt{2}\sigma)^2$ and the fourth moment is $E[d_n^4] = 3(\sqrt{2}\sigma)^4$.

Now we are ready to begin our derivation...

$$\begin{aligned}
E\left[\left(\frac{\|x^{(i)} - x^{(j)}\|_2^2}{E\left[\|x^{(i)} - x^{(j)}\|_2^2\right]}\right)^2\right] &= \frac{E\left[\left(\|x^{(i)} - x^{(j)}\|_2^2\right)^2\right]}{\left(E\left[\|x^{(i)} - x^{(j)}\|_2^2\right]\right)^2} \\
&= \frac{E\left[\left(\sum_{k=1}^N d_k^2\right)^2\right]}{\left(E\left[\sum_{k=1}^N d_k^2\right]\right)^2} \\
&= \frac{E\left[\sum_{k=1}^N d_k^4 + \sum_{k=1}^N \sum_{l=1, l\neq k}^N d_k^2 d_l^2\right]}{\left(\sum_{k=1}^N E\left[d_k^2\right]\right)^2} \\
&= \frac{\sum_{k=1}^N E\left[d_k^4\right] + \sum_{k=1}^N \sum_{l=1, l\neq k}^N E\left[d_k^2\right] E\left[d_l^2\right]}{\sum_{k=1}^N \sum_{l=1}^N E\left[d_k^2\right] E\left[d_l^2\right]} \\
&= \frac{3N(\sqrt{2}\sigma)^4 + N(N-1)(\sqrt{2}\sigma)^2(\sqrt{2}\sigma)^2}{N^2(\sqrt{2}\sigma)^2(\sqrt{2}\sigma)^2} \\
&= \frac{3N + N^2 - N}{N^2} \\
&= \frac{2+N}{N}
\end{aligned}$$

Combining both the now simplified terms from the right-hand side of Eq. 1, we get the desired identity, namely...

$$Var\left(\frac{\|x^{(i)} - x^{(j)}\|_2^2}{E\left[\|x^{(i)} - x^{(j)}\|_2^2\right]}\right) = \frac{N+2}{N} - 1 \tag{2}$$

We can also see that this expression vanishes when N $\to \infty$ as shown below:

$$\frac{N+2}{2} - 1 = \frac{N+2-N}{N} = \frac{2}{N} \lim_{N\to\infty} \frac{2}{N} = \infty$$

1.2 Euclidean distance function

Question 1: Consider the special case in which all the input vectors have the same magnitude in a training dataset, $\|x^{(1)}\|_2^2 \ldots \|x^{(M)}\|_2^2$. Show that in order to find the nearest neighbour of a test point $\mathbf{x}^*$ among the training set, it is sufficient to just compare and rank the negative inner product between the training and test data: -$\mathbf{x}^{(m)T}\mathbf{x}^*$.

Solution:
To find the nearest neighbour of a test point x^* among the training set, we want to find an $x^{(m)}$ such that we have a minimum $\|x^{(m)} - x^*\|_2^2$ for $m = 1, \ldots, M$.

$$\|x^{(m)} - x^*\|_2^2 = \left(x^{(m)} - x^*\right)^T \left(x^{(m)} - x^*\right) = \sum_{n=1}^N \left(x_n^{(m)} - x_n^*\right)^2$$

Since $\|x^{(1)}\|_2^2 = \cdots = \|x^{(m)}\|_2^2$, the first term in the above equation can be treated as a constant. Then we have,

$$\|x^{(m)} - x^*\|_2^2 = const - 2\sum_{n=1}^{N} x_N^{(m)} x_n^* = const - 2x^{(m)T}x^*$$

Therefore, it is sufficient to compare and rank the negative inner product between the training and test data, i.e. $-x^{(m)T}x^*$, to find the nearest neighbour.

Question 2 - Pairwise distances: Write a vectorized Tensorflow Python function that implements the pairwise squared Euclidean distance function for two input matrices. It should not contain loops and you should make use of Tensorflow broadcasting. Include the snippets of the Python code.

Solution:

The vectorized Tensorflow Python function to implement the pairwise squared Euclidan distance function for two input matrices can be found below.

```
def D(X, Z):
    sub = X[:,:,tf.newaxis] - tf.transpose(Z)
    matD = tf.matmul(tf.transpose(sub, perm=[0,2,1]), sub)
    return tf.matrix_diag_part(matD)
```

This function takes in a B x N matrix **X** and a C x N matrix **Z** and returns a B x C matrix containing the pairwise Euclidean distances.

1.3 Making predictions

Question 1 - Choosing the nearest neighbours: Write a vectorized Tensorflow Python function that takes a pairwise distance matrix and returns the responsibilities of the training examples to a new test data point. It should not contain loops. You may find the tf.nn.top k function useful. Include the relevant snippets of your Python code.

Solution:

The relevant snippets of code can be found below.

```
def find_knn(diff):
        resp = np.zeros(80)
        for idx_i in range(80):
                resp[idx_i] = diff[idx_i] /sum(diff)
        return resp
```

Question 2 - Prediction: For the dataset data1D, compute the above k-NN prediction function with k = 1, 3, 5, 50. For each of these values of k, compute and report the training MSE loss, validation MSE loss and test MSE loss. Choose the best k using the validation error. For the different k value, plot the prediction function for x $\in$ [0; 11].

Solution:

For each value of k, the training MSE loss, validation MSE loss, and test MSE loss can be found below:

k = 1:

Training MSE Loss = 0
Validation MSE Loss = 1.08619868
Test MSE Loss = 0.55772939

k = 3:

Training MSE Loss = 0.42585956
Validation MSE Loss = 1.29750518
Test MSE Loss = 0.63518754

k = 5:
Training MSE Loss = 0.48427401
Validation MSE Loss = 1.26679596
Test MSE Loss = 0.74038389

k = 50:
Training MSE Loss = 4.98394131
Validation MSE Loss = 4.91480652
Test MSE Loss = 2.81052662

Looking at the validation MSE loss for the different k values, the best value of k is 1 as it has the lowest MSE loss out of the 4 values. In practice, a more robust choice would be 5, which performs well in the presence of less regular data, and with more outliers.

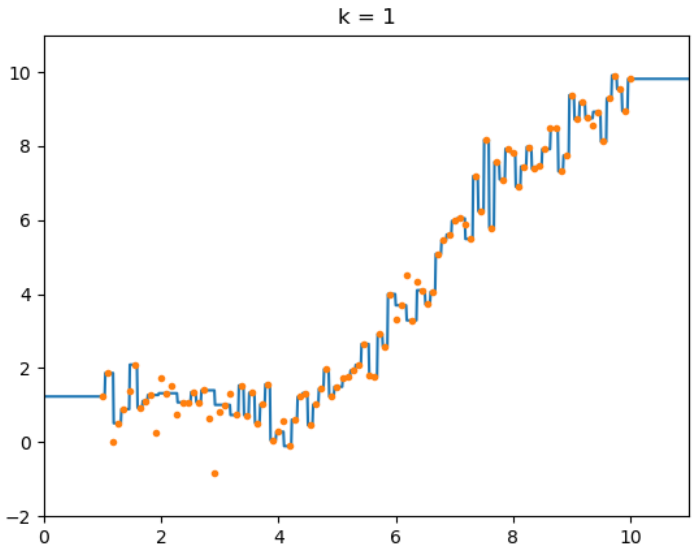

Figure 2: k-Nearest-Neighbours with k = 1.

The rest of the plots can be found at the end of this document.

1.4 Soft k-NN and Gaussian process

Question 1 - Soft decisions: Write a Tensorflow Python program based on the soft k-NN model to compute predictions on the data1D.npy dataset. Set $\lambda = 10$ and plot the test-set prediction of the model. Repeat all for the Gaussian process regression model. Comment on the difference you observe between your two programs. Include the relevant snippets of Python code.

Solution:

The soft decision reduces the jaggedness of the regression. This is because the decision on each point is performed by calculating over every point in a continuous spread, which contributes to a higher dimensional fit. (See Fig. 6 at the end of this document).

The snippets of code relevant can be found below:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
```

```
init_op = tf.initialize_all_variables()

def D(X, Z):
    sub = X[:,:,tf.newaxis] - tf.transpose(Z)
    matD = tf.matmul(tf.transpose(sub, perm=[0,2,1]), sub)
    return tf.matrix_diag_part(matD)

def find_knn(diff):
    resp = np.zeros(80)
    for idx_i in range(80):
            resp[idx_i] = diff[idx_i] /sum(diff)
    return resp

np.random.seed(521)

with tf.Session() as sess:
    Data = np.linspace(1.0 , 10.0 , num =100) [:, np. newaxis]
    Target = np.sin( Data ) + 0.1 * np.power( Data , 2) \
    + 0.5 * np.random.randn(100 , 1)

    randIdx = np.arange(100)
    np.random.shuffle(randIdx)

    hype_param = 100

    trainData, trainTarget = Data[randIdx[:80]], Target[randIdx[:80]]
    validData, validTarget = Data[randIdx[80:90]], Target[randIdx[80:90]]
    testData, testTarget = Data[randIdx[90:100]], Target[randIdx[90:100]]


    X = np.linspace(0.0, 11.0, num = 1000)[:, np.newaxis]

    y = np.zeros(1000)
    j = 0

    diff = D(X, trainData).eval()
    K = np.exp(-hype_param * diff)
    for row in K:
            resp = find_knn(row)#, k)
            y[j] = np.dot(trainTarget.transpose(), resp)
            j += 1
    plt.plot(X, y, Data, Target, '.')
    plt.axis([0,11,-2,11])
    plt.show()
```

Question 2 - Conditional distribution of a Gaussian: Derive the expression for the conditional mean μ^* and variance Σ^* in terms of $\mathbf{y}_{train}$, $\Sigma_{y^*y^*}$, $\Sigma_{\mathbf{y}_{train}\mathbf{y}_{train}}$, and $\Sigma_{\mathbf{y}_{train}y^*}$.

Solution:

Let y_{train}, μ_{train}, y^*, μ^*, and $\mathbf{\Sigma}$ represent the training target, the training mean, the prediction, the prediction mean, and the join covariance matrix, respectively.

From the question, we are given the following joint probability:

$$P(\mathbf{y}) = \frac{1}{(\sqrt{2\pi})^{M+1}\sqrt{|\mathbf{\Sigma}|}} exp\{-\frac{1}{2}\mathbf{y}^T\mathbf{\Sigma}^{-1}\mathbf{y}\}$$

We can rewrite this as follows:

$$P(\mathbf{y}) = \frac{1}{(\sqrt{2\pi})^{M+1}\sqrt{|\mathbf{\Sigma}|}} exp\{-\frac{1}{2}\big(\mathbf{y}-\mu\big)^T\mathbf{\Sigma}^{-1}\big(\mathbf{y}-\mu\big)\}$$

The inside of the exponential term can be simplified as follows:

$$\big(\mathbf{y}-\mu\big)^T\mathbf{\Sigma}^{-1}\big(\mathbf{y}-\mu\big) = \big[\big(\mathbf{y}_{train}-\mu_{train}\big),\big(y^*-\mu^*\big)\big]^T\begin{bmatrix}\Sigma_{11} & \Sigma_{12}\\ \Sigma_{21} & \Sigma_{22}\end{bmatrix}\big[\big(\mathbf{y}_{train}-\mu_{train}\big),\big(y^*-\mu^*\big)\big]$$

$$where\mathbf{\Sigma}^{-1} = \begin{bmatrix}\Sigma_{11} & \Sigma_{12}\\ \Sigma_{21} & \Sigma_{22}\end{bmatrix} = \begin{bmatrix}\Sigma_{y^*y^*} & \Sigma_{y^*\mathbf{y}_{train}}\\ \Sigma_{\mathbf{y}_{train}y^*} & \Sigma_{\mathbf{y}_{train}\mathbf{y}_{train}}\end{bmatrix}^{-1}$$

$$\big(\mathbf{y}-\mu\big)^T\mathbf{\Sigma}^{-1}\big(\mathbf{y}-\mu\big) = (y_{train}-\mu_{train})^T\Sigma_{11}(y_{train}-\mu_{train}) + 2(y_{train}-\mu_{train})^T\Sigma_{12}(y^*-\mu^*) + (y_{train}-\mu_{train})^T\Sigma_{22}(y^*-\mu^*))$$

$$\Sigma_{11} = (\Sigma_{y^*y^*} - \Sigma_{y^*y} - \Sigma_{y^*\mathbf{y}_{train}}(\Sigma_{\mathbf{y}_{train}\mathbf{y}_{train}}-))$$

2 Linear and logistic regression

2.1 Geometry of linear regression

Question 1: Is the l_2 penalized mean squared error loss L a convex function of W? Show your results using the Jensen inequality (see the convexity inequality from the lecture of 16 January). Is the error a convex function of the bias, b?

Solution:

To determine whether or not the *l2* penalized mean squared error loss L is a convex function of a particular variable, x, we can use the Jensen inequality (convexivity inequality).

L is a convex function of W if and only if

$$\alpha L(W_1) + (1-\alpha)L(W_2) \geq L(\alpha W_1 + (1-\alpha)W_2).$$

We'll start by listing the expressions required for the left side:

$$\alpha L(W_1) = \sum_{m=1}^{M}\frac{1}{2M}\|\alpha W_1^T\mathbf{x}^{(m)} + \alpha b - \alpha y^{(m)}\|_2^2 + \frac{\alpha\lambda}{2}\|W_1\|_2^2$$

$$(1-\alpha)L(W_2) = \sum_{m=1}^{M}\frac{1}{2M}\|(1-\alpha)W_2^T\mathbf{x}^{(m)} + (1-\alpha)b - (1-\alpha)y^{(m)}\|_2^2 + \frac{(1-\alpha)\lambda}{2}\|W_2\|_2^2$$

Adding the two, we use the Triangle Inequality to get the following:

$$\begin{aligned}\alpha L(W_1) + (1-\alpha)L(W_2) &= \sum_{m=1}^{M}\frac{1}{2M}\Big(\|\alpha W_1^T\mathbf{x}^{(m)} + \alpha b - \alpha y^{(m)}\|_2^2 \\ &+ \|(1-\alpha)W_2^T\mathbf{x}^{(m)} + (1-\alpha)b - (1-\alpha)y^{(m)}\|_2^2\Big) + \frac{\alpha\lambda}{2}\|W_1\|_2^2 + \frac{(1-\alpha)\lambda}{2}\|W_2\|_2^2 \\ &\geq \sum_{m=1}^{M}\frac{1}{2M}\Big(\|\alpha W_1^T\mathbf{x}^{(m)} + (1-\alpha)W_2^T\mathbf{x}^{(m)} + b - y^{(m)}\|_2^2\Big) + \frac{\lambda}{2}\|\alpha W_1 + (1-\alpha)W_2\|_2^2\end{aligned}$$

The right side can be expanded as follows:

$$L(\alpha W_1 + (1-\alpha)W_2) = \sum_{m=1}^{M} \frac{1}{2M}\Big(\|(\alpha W_1^T + (1-\alpha)W_2^T)\mathbf{x}^{(m)} + b + y^{(m)}\|_2^2\Big) + \frac{\lambda}{2}\|(\alpha W_1 + (1-\alpha)W_2)\|_2^2$$

From this, we can conclude that $\alpha L(W_1) + (1-\alpha)L(W_2) \geq L(\alpha W_1 + (1-\alpha)W_2)$; therefore, the MSE loss is a convex function of W.

We can carry out a similar method to determine if the error is also a convex function of the bias, b.
L is a convex function of b if and only if

$$\alpha L(b_1) + (1-\alpha)L(b_2) \geq L(\alpha b_1 + (1-\alpha)b_2).$$

We'll start by listing the expressions required for the left side:

$$\alpha L(b_1) = \sum_{m=1}^{M} \frac{1}{2M}\|\alpha W^T\mathbf{x}^{(m)} + \alpha b_1 - \alpha y^{(m)}\|_2^2 + \frac{\alpha\lambda}{2}\|W\|_2^2$$

$$(1-\alpha)L(b_2) = \sum_{m=1}^{M} \frac{1}{2M}\|(1-\alpha)W^T\mathbf{x}^{(m)} + (1-\alpha)b_2 - (1-\alpha)y^{(m)}\|_2^2 + \frac{(1-\alpha)\lambda}{2}\|W\|_2^2$$

Adding the two and using the Triangle Inequality, we get the following:

$$\begin{aligned}\alpha L(b_1) + (1-\alpha)L(b_2) &= \sum_{m=1}^{M} \frac{1}{2M}\Big(\|\alpha W^T\mathbf{x}^{(m)} + \alpha b_1 - \alpha y^{(m)}\|_2^2 \\ &\quad + \|(1-\alpha)W^T\mathbf{x}^{(m)} + (1-\alpha)b_2 - (1-\alpha)y^{(m)}\|_2^2\Big) + \frac{\alpha\lambda}{2}\|W\|_2^2 + \frac{(1-\alpha)\lambda}{2}\|W\|_2^2 \\ &\geq \sum_{m=1}^{M} \frac{1}{2M}\Big(\|W^T\mathbf{x}^{(m)} + \alpha b_1 + (1-\alpha)b_2 - y^{(m)}\|_2^2\Big) + \frac{\lambda}{2}\|W\|_2^2\end{aligned}$$

The right side can be expanded as follows:

$$L(\alpha b_1 + (1-\alpha)b_2) = \sum_{m=1}^{M} \frac{1}{2M}\Big(\|(W^T\mathbf{x}^{(m)} + \alpha b_1 + (1-\alpha)b_2 + y^{(m)}\|_2^2\Big) + \frac{\lambda}{2}\|W\|_2^2$$

From this, we can conclude that $\alpha L(b_1) + (1-\alpha)L(b_2) \geq L(\alpha b_1 + (1-\alpha)b_2)$; therefore, the MSE loss is a convex function of b.

Question 2: Consider learning a linear regression model of a single scalar output with a weight vector W $\in R^N$, bias $\in R$ and no regularizer, on a dataset D $= \{(x^{(1)}, y^{(1)}), \ldots, (x^{(M)}, y^{(M)})\}$. If the n^{th} input dimensions, x_n, is scaled by a constant factor of $\alpha > 1$ and shifted by a constant $\beta > 1$ for every datum in the training set, what will happen to the optimal weights and bias after learning as compared to the optimal weights and biases learned from the non-transformed (original) data? What will happen to the new minimum value of the loss function comparing to the original minimum loss? Explain your answer clearly.

Solution:

If the n^{th} input dimensions, x_n, is scaled by some constant factor $\alpha > 1$ and shifted by a constant $\beta > 1$ for every datum in the training set, the optimal weights as well as the bias after learning will differ from the optimal weights and biases learned from the original data.

Specifically, the n^{th} dimension of the new optimal weight vector will be scaled by a factor of $\frac{1}{\alpha}$ and the bias will be offset by a factor of β.

Since scaling the input dimension by a constant and shifting it by an offset are both linear transformations, the resulting transformation of the loss function is also liner; therefore, the minimum value of the loss function compared to the original minimum loss is equal.

Question 3: On the same linearly transformed dataset from the last question, consider learning an l_2-regularized linear regression model in which the weight-decay coefficient is set to a constant λ. Describe what will happen to the weights and the bias learned on this transformed dataset as compared to those learned from non-transformed (original) data. What will happen to the new minimum value of the loss function, again comparing to the minimum loss before the transformation? Explain your answer clearly.

Solution:

Taking the same linearly transformed dataset from the previous question, we now consider learning a l_2-regularized linear regression model with a constant weight-decay coefficient, λ.

Since the penalization term added is a constant, the difference in the new and original biases will still be the same as described in the previous section. In other words, the new bias will be offset by a factor of β.

Similarly, the n^{th} dimension of the new weight vector will be the same as the previous section: scaled by a factor of $\frac{1}{\alpha}$.

Unlike the previous case, since this is a l_2-regularized loss function, there exists a regularization term. Since the n^{th} dimension of the new weight vector is scaled by a factor of $\frac{1}{\alpha}$, i.e. decreases, the regularization term also decreases. Therefore, the changes in the weights and the bias are the same as the previous section; however, the new minimum value of the loss function is less than the minimum loss before the transformation.

Question 4: Consider a multi-class classification task with D $>$ 2 classes. Suppose that somehow you were constrained by the software packages such that you were only able to train binary classifiers. Design a procedure/scheme to solve a multi-class classification task using a number of binary classifiers.

Solution:

To solve a multi-class classification task using a number of just binary classifiers, there are two main approaches: one-vs-rest and one-vs-one.

For a case of a D class classification task where D $>$ 2, the one-vs-rest approach trains one classifier per each class in a multi-class classification task. For a certain class i, it will assume all samples labelled as class i as positive and the reset as negative. One of the main problems for this approach is that the binary classification learner will typically encounter a much larger set of negatives than positives.

In the one-vs-one approach, a separate classifier is trained for each different pair of labels. Therefore, this leads to $\frac{D(D-1)}{2}$ classifiers. Although this method is less sensitive to the problem of imbalanced datasets than the one-vs-rest approach, it is much more computationally expensive.

2.2 Stochastic gradient descent

Question 1 - Tuning the learning rate: Write a Tensorflow script that implements linear regression and the stochastic gradient descent algorithm with mini-batch size B = 50. Train the linear regression model on the tiny MNIST dataset by using SGD to optimize the total loss L. Set the weight decay coefficient λ = 1. Tune the learning rate η to obtain the best overall convergence speed of the algorithm. Plot the total loss function L vs. the number of updates for the best learning rate found and discuss the effect of the learning-rate value on training convergence

Solution:

The validation loss for each learning rate was evaluated after 40 epochs. As can be seen below, $\eta = 0.003$ was found to be the best learning rate.

$\eta = 0.001$**:**
Validation Loss = 6.58327428

$\eta = 0.002$**:**
Validation Loss = 6.35352567

$\eta = 0.003$**:**
Validation Loss = 6.33040334

$\eta = 0.004$**:**

Validation Loss = 7.0507901

$\eta = 0.008$**:**
Validation Loss = 7.23372033e+204

Question 2 - Effect of the mini-batch size: Run the SGD algorithm with $B = \{10; 50; 100; 700\}$ and tune the learning rate separately for each mini-batch size. Plot the total loss function L vs. the number of updates for each mini-batch size. What is the overall best mini-batch size in terms of training time? Comment on your observation.

Solution:

For batch size of 50, the best learning rate was around 0.003.
For batch size of 100, the best learning rate was around 0.002.
For batch size of 700, the best learning rate was around 0.0002.
The best mini-batch size in terms of training time is TODO. This is probably because TODO.

Question 3 - Generalization: Run SGD with mini-batch size $B = 50$ and use the validation set performance to choose the best weight decay coefficient that gives the best classification accuracy on the test set from $\lambda = \{0, 0 : 0001, 0 : 001, 0 : 01, 0 : 1, 1\}$. Plot λ vs the test set accuracy. Comment on your plot and the effect of weight-decay regularization on the test performance. Also comment on why we need to tune the hyper-parameter λ using a validation set instead of the training set.

Solution:

The best weight decay coefficient value, as can be seen below, was found to be $\lambda = 0$. This corresponds to a validation loss of 5.66903812

$\lambda = 0.$**:**
Validation Loss = 5.66903812

$\lambda = 0.0001$**:**
Validation Loss = 5.8586712

$\lambda = 0.001$**:**
Validation Loss = 5.69632471

$\lambda = 0.01$**:**
Validation Loss = 5.69510954

$\lambda = 0.1$**:**
Validation Loss = 6.00347569

$\lambda = 1$**:**
Validation Loss = 6.29424693

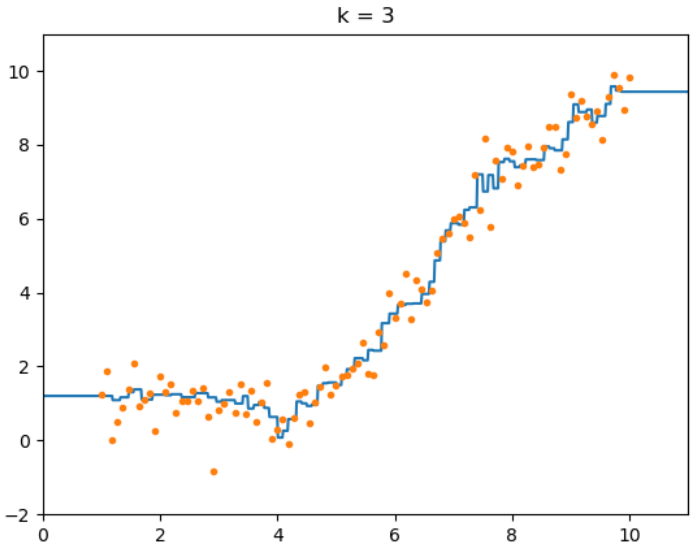

Figure 3: k-Nearest-Neighbours with k = 3.

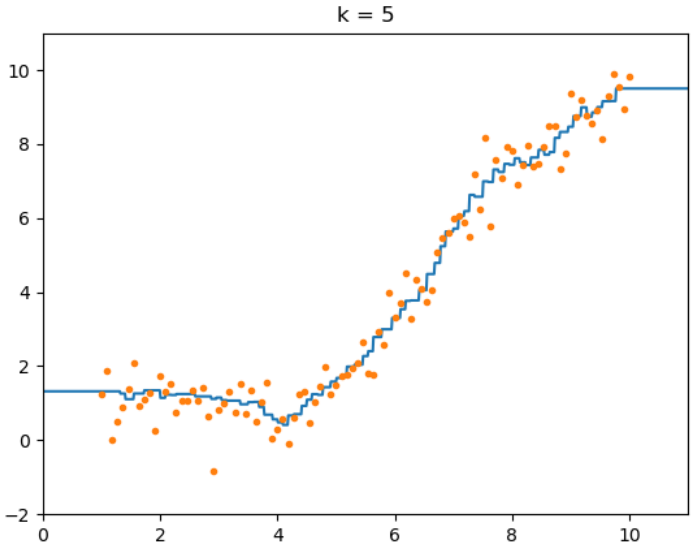

Figure 4: k-Nearest-Neighbours with k = 5.

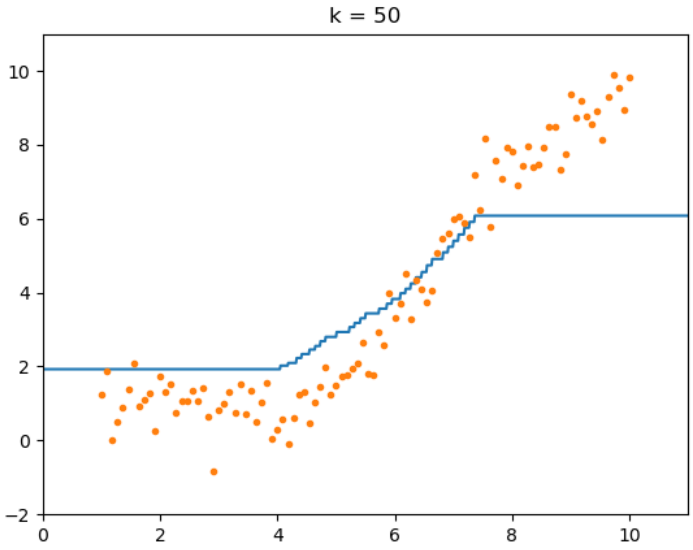

Figure 5: k-Nearest-Neighbours with k = 50.

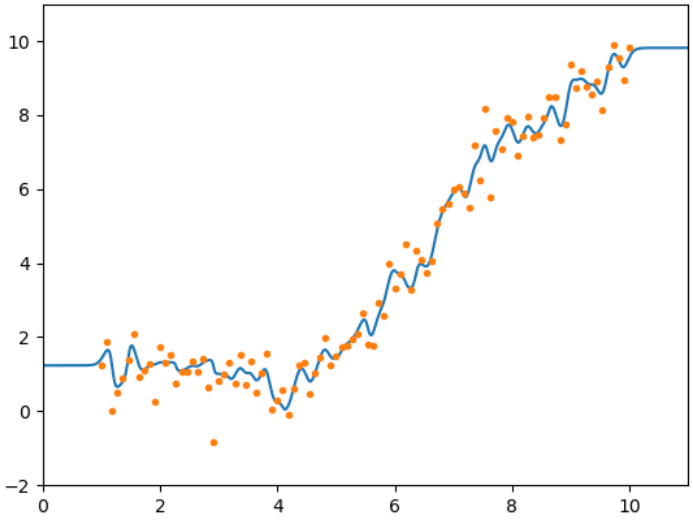

Figure 6: Soft k-Nearest-Neighbours with $\lambda = 100$.

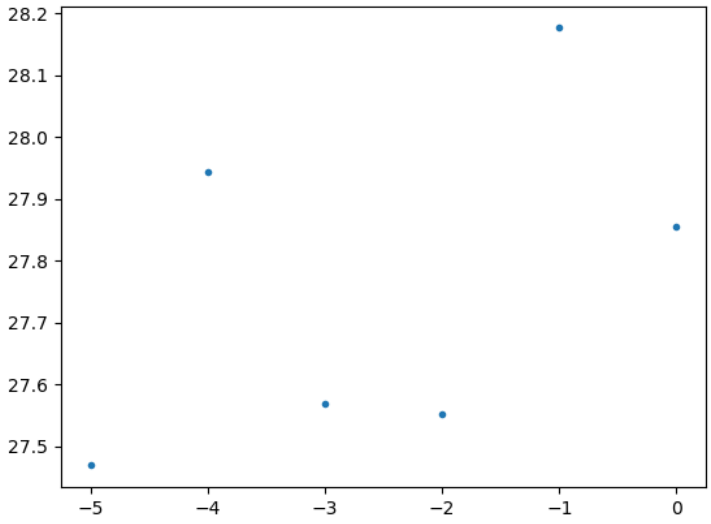

Figure 7: Plot of λ vs the test set accuracy.