ECE521 - Assignment 1 - k-NN and Linear Regression

K. L. Barry Fung - 999871125
Wen Tao Zhao - 999721127

Due February 7th, 2017

1 k-Nearest Neighbour

1.1 Geometry of k-NN

Problem 1.1.1. *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. We can have a 1D dataset that is exponentially distributed, with alternating labels of Class 1 and Class 2, beginning at 0, and ending at 2N-1, where N is the number of data points.Thus, the classification accuracy is periodic on average, with the expected value of the classification accuracy given as:

$$Accuracy\% = 50 * cos(0.5 * k) + 50$$

■

Problem 1.1.2. *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 x, $x \in \mathbb{R}^N$. 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 $x^{(i)}$, $x^{(j)}$ from this dataset,*

$$\mathrm{var}\Big(\frac{||\boldsymbol{x}^{(\mathrm{i})} - \boldsymbol{x}^{(\mathrm{j})}||_2^2}{\mathbb{E}[||\boldsymbol{x}^{(\mathrm{i})} - \boldsymbol{x}^{(\mathrm{j})}||_2^2]}\Big) = \frac{\mathrm{N}+2}{\mathrm{N}} - 1$$

(This is known as the variance of the squared l_2 (squared Euclidean) distance ratio.) Next, show that it vanishes as $N \to \infty$, i.e. a test data point drawn from this dataset is equally close to all the training examples in high dimensional space. k.

Solution. Simply expanding the definition of variance $var(x) = \mathbb{E}(x^2) - \mathbb{E}(x)^2$ we obtain the following:

$$
\begin{aligned}
& var\Big(\frac{||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2}{\mathbb{E}[||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2]}\Big) \\
=& \mathbb{E}\Big[\Big(\frac{||\mathbf{d}||_2^2}{\mathbb{E}[||\mathbf{d}||_2^2]}\Big)^2\Big] - \mathbb{E}\Big[\Big(\frac{||\mathbf{d}||_2^2}{\mathbb{E}[||\mathbf{d}||_2^2]}\Big)\Big]^2 \\
=& \mathbb{E}\Big[\Big(\frac{\sum_{n=1}^N d_n^2}{\mathbb{E}[\sum_{n=1}^N d_n^2]}\Big)^2\Big] - \Big(\frac{\mathbb{E}[||\mathbf{d}||_2^2]}{\mathbb{E}[||\mathbf{d}||_2^2]}\Big)^2 \\
=& \mathbb{E}\Big[\frac{(\sum_{n=1}^N d_n^2) * (\sum_{m=1}^N d_m^2)}{(\sum_{n=1}^N \mathbb{E}[d_n^2])^2}\Big] - 1 \\
=& \mathbb{E}\Big[\frac{(\sum_{n=1}^N d_n^4 + 2\sum_{n=1}^N \sum_{m=n+1}^N d_n^2 d_m^2)}{(\sum_n^N \text{var}(d_n) - \mathbb{E}[d_n]^2)^2}\Big] - 1 \\
=& \frac{\sum_{n=1}^N \mathbb{E}[d_n^4] + 2\sum_{n=1}^N \sum_{m=n+1}^N \mathbb{E}[d_n^2 d_m^2]}{\mathbb{E}[(\sum_n^N \text{var}(d_n)^2]} - 1 \\
=& \frac{12N\sigma^4 + 2\sum_{n=1}^N \sum_{m=n+1}^N \mathbb{E}[d_n^2]\mathbb{E}[d_m^2]}{\mathbb{E}[(\sum_n^N 2\sigma^2)^2]} - 1 \\
=& \frac{12N\sigma^4 + 2\sum_{n=1}^N \sum_{m=n+1}^N \text{var}(d_n)\text{var}(d_m)}{4N^2\sigma^4} - 1 \\
=& \frac{12N\sigma^4 + 2 * \frac{(N)(N-1)}{2} * ((2\sigma)^2)^2}{4N^2\sigma^4} - 1 \\
=& \frac{12N\sigma^4 + 8N^2\sigma 4 - 4N\sigma^4}{4N^2\sigma^4} - 1 \\
=& \frac{12N\sigma^4 + 4N^2\sigma 4 - 4N\sigma^4}{4N^2\sigma^4} - 1 \\
=& \frac{N+2}{N} - 1
\end{aligned}
$$

as required. Note that as $N \to \infty$, the sum $\frac{N+2}{N} = \frac{1+\frac{2}{N}}{1} \to 1$. As such, the variance of the squared Euclidean distance approaches 0 as the dimensionality of the problem goes to infinity. ■

1.2 Euclidean distance function

Problem 1.2.1. ***Inner Product:*** *Consider the special case in which all the input vectors have the same magnitude in a dataset, $||\boldsymbol{x}^{(1)}||_2^2 = ... = ||\boldsymbol{x}^{(M)}||_2^2$. Show that in order to find the nearest neighbour of a test point **x*** among the training set, it is just sufficient to compare and rank the negative inner product between the training set and test data: $-x^{(m)^T} x^*$.*

Solution. To find the distance between the i^{th} point in the training set x^{i^*} and the jth

point in the training set x^j:

$$\begin{aligned}||x^{i^*} - x^j||_2^2 &= \sum_{k=1}^{n} \left(x_k^{i^*} - x_k^j\right)^2 \\ &= \sum_{k=1}^{n} \left(x_k^{i^*} - x_k^j\right)^2 \\ &= ||x^{i^*}|| + ||x^j|| - 2x^{i^*} \cdot x^j\end{aligned}$$

Given that the magnitude of all points in the training set are equal, and the magnitude of the point in the testing set is unchanged when doing the comparison, it is sufficient to just compare and rank the third term of this expression, i.e. the negative inner product between the points in the training and data set. For all datasets, the exact expression is as suggested: $x^{(m)^T} x^*$ ■

Problem 1.2.2. ***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 relevant portions of the TensorFlow code are shown in Figure 1 below.

Figure 1: Code for calculating pairwise distances

DistanceMatrix.py

```python
import tensorflow as tf

def get_distance_matrix (train_dataset,test_dataset):

  x_expanded = tf.expand_dims(train_dataset,1)
  y_expanded = tf.expand_dims(test_dataset,0)

  z = tf.squared_difference(x_expanded,y_expanded)
  return tf.reduce_sum(z,2)

def main():

  train = tf.placeholder(tf.int64,[None,None])
  test = tf.placeholder(tf.int64,[None,None])

  dmatrix = get_distance_matrix(train, test)
  train_in =  [[9,1],[1,2],[1,3]]
  test_in = [[1,2],[4,12]]

  sess = tf.Session()
  print(sess.run(dmatrix,feed_dict={train:train_in, test:test_in}))

  sess.close()



if __name__ == "__main__":
    main()
```

■

1.3 Making predictions

Problem 1.3.1. ***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 portions of the TensorFlow code are shown in Figure 2 below.

Figure 2: Code for calculating the responsibility matrix for k-NN

```python
import numpy as np
import tensorflow as tf

sess = tf.Session()
def get_distance_matrix (train_dataset,test_dataset):

  x_expanded = tf.expand_dims(train_dataset,1)
  y_expanded = tf.expand_dims(test_dataset,0)

  z = tf.squared_difference(x_expanded,y_expanded)

  dist_matrix = tf.reduce_sum(z,2)
  return dist_matrix

def hard_resp(pw_matrix, k):
    '''
    hard_resp
        Calculates the standard KNN responsibility vector
    '''
   #We need to index the closest values
    ref_matrix = tf.neg(tf.transpose(pw_matrix))
    values, indices = tf.nn.top_k(ref_matrix, k, sorted=False)
    #Generate the indices from top_k (adapted liberally from Stack Overflow)
    range_repeated = tf.tile(tf.expand_dims(tf.range(0, tf.shape(indices)[0]), 1), [1, k])
    # Time to update
    full_indices = tf.reshape(tf.concat(2, [tf.expand_dims(range_repeated, 2),\
                            tf.expand_dims(indices, 2)]), [-1, 2])
    update = tf.mul(tf.truediv(tf.constant(1.0,dtype=tf.float64), \
                    tf.cast(k,tf.float64)),tf.ones(tf.shape(values), dtype=tf.float64))
    #update, and desparsify
    return tf.sparse_to_dense(full_indices, tf.shape(ref_matrix), tf.reshape(update, [-1]), \
                              default_value=0., validate_indices=False)

d_matrix = get_distance_matrix([[1.0],[4.4],[9.0]],[[9.2],[1.1],[2.0]])
print(sess.run(d_matrix))
resp = hard_resp(d_matrix,1)
print(sess.run(resp))

sess.close()
```

■

Problem 1.3.2. ***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 $x \in [0, 11]$ similar to the figure shown above. (You may create an input matrix $X = np.linspace(0.0, 11.0, num = 1000)[:, np.newaxis]$ and use Matplotlib to plot the predictions.) Comment on this plot and how you would pick the best k from it.*

Solution. The output of the training MSE is shown in Figure 3.

Figure 3: The recorded MSE for the training and validation sets for various k

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
Running KNN for K = 1

The MSE from the test data was 0.13943234651568065, while the MSE from the validation data was 0.2715496697215269

Running KNN for K = 3

The MSE from the test data was 0.158796884506008, while the MSE from the validation data was 0.3243762960195306

Running KNN for K = 5

The MSE from the test data was 0.1850959723619398, while the MSE from the validation data was 0.3166989901357597

Running KNN for K = 50

The MSE from the test data was 0.702631655073554, while the MSE from the validation data was 1.228701629604592
```

The resultant prediction graphs are also shown below in Figure 4 - 7, where the dots represents the training data, and the lines represent the prediction for each of the k-NN graphs.

Figure 4: Prediction graph, k = 1

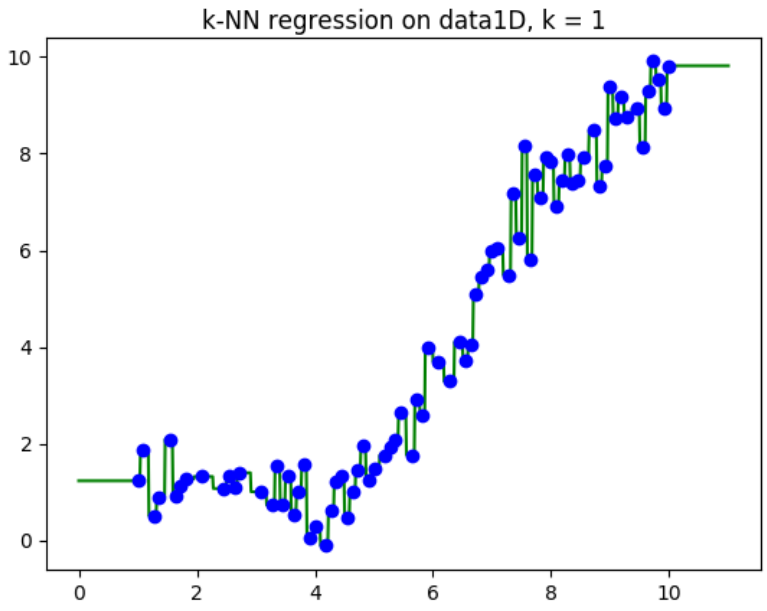


Figure 5: Prediction graph, k = 3

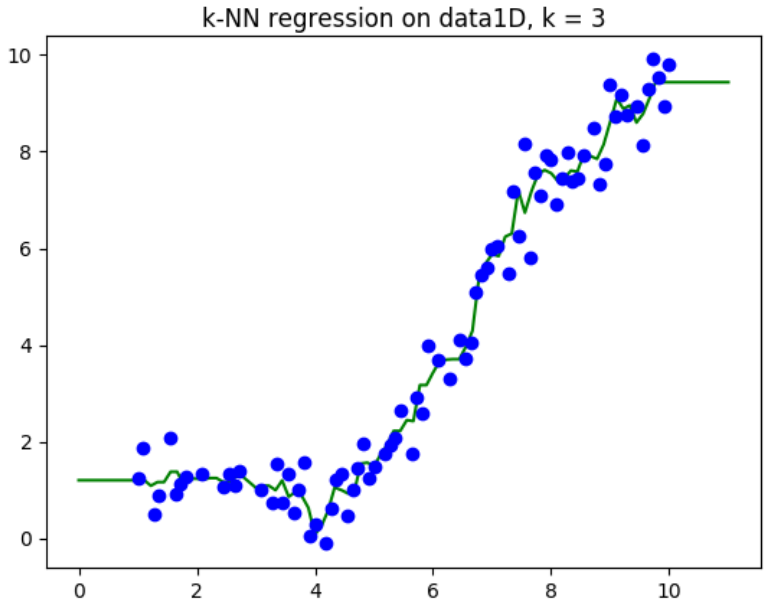


Figure 6: Prediction graph, k = 5

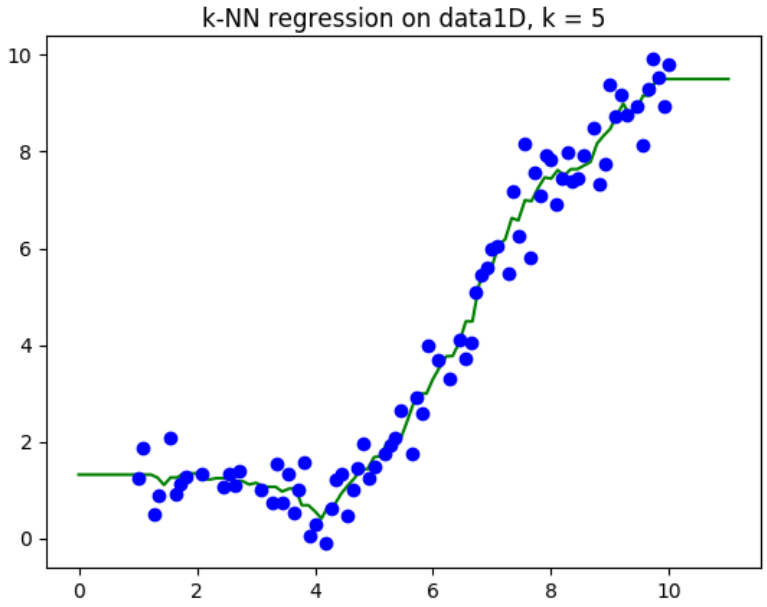
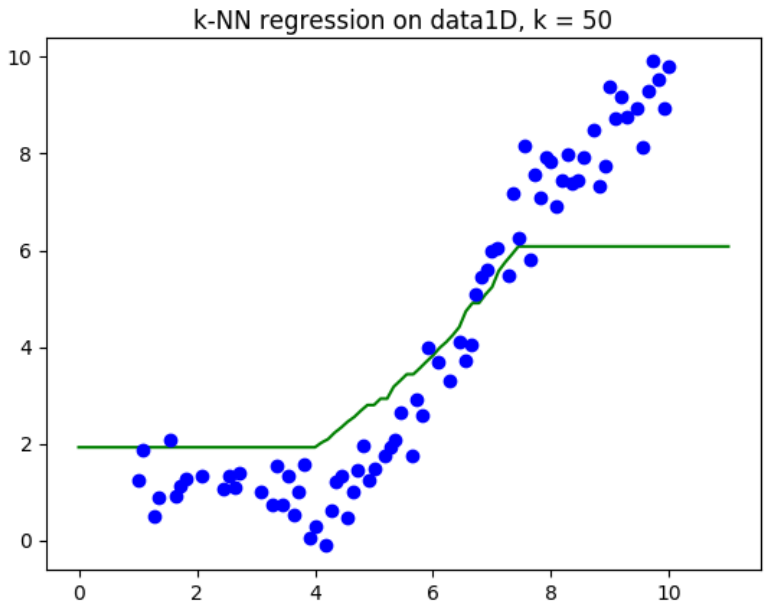


Figure 7: Prediction graph, k = 50



Depending on the metric of selection, the optical k value differs. When examining the MSE, k = 1 is shown to be the best choice, while examining the prediction graphs over the range $x \in [0, 11]$ would indicate that k = 5 has the most generalizability, given its lack of sudden changes. ■

1.4 Soft k-NN and Gaussian process

Problem 1.4.1. ***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 key functions of the TensorFlow code implementing soft-k-NN and Gaussian process regression are shown in Figure 8 below. These functions take in a training set, a target for the training set, an input to predict, and a hyper-parameter λ (l in the code) to calculate their weighting functions.

Figure 8: Essential functions for calculating soft k-NN and Gaussian process regression

question1-4-1.py

```python
def get_distance_matrix (train_dataset,test_dataset):
    x_expanded = tf.expand_dims(train_dataset,1)
    y_expanded = tf.expand_dims(test_dataset,0)
    z = tf.squared_difference(x_expanded,y_expanded)
    dist_matrix = tf.reduce_sum(z,2)
    return dist_matrix

def soft_resp_knn(pw_matrix, l):
    '''
    soft_resp
        Calculates the soft KNN responsibility vector
    '''
    #We need to index the closest values
    ref_matrix = tf.exp(l*tf.neg(tf.transpose(pw_matrix)))
    ref_sums = tf.reduce_sum(ref_matrix,axis=1)
    ref_sums_exp = tf.expand_dims(ref_sums,1)
    ref_sums = tf.tile( ref_sums_exp, (1,tf.shape(ref_matrix)[1]))
    #Generate the indices
    return tf.truediv(ref_matrix,ref_sums)

def sqk_gpr(input1, input2, l):
    '''
    sqk_gpr
        Calculates the squared exponential kernel
    '''
    #We need to index the closest values
    return tf.exp(l*tf.neg(get_distance_matrix(input1,input2)))

def knn(train, target, input, l):
    '''
    knn
        runs the KNN for a set of input values
    '''
    resp = soft_resp_knn(get_distance_matrix(train, input), l)
    return tf.matmul(resp,target)

def gpr(train, target, input, l, noise):
    '''
    gpr
        runs Gaussian process regression using a set of input values
    '''
    noise.set_shape([])
    left_m= tf.matrix_inverse(tf.add(sqk_gpr(train, train, l),tf.scalar_mul(noise,tf.eye(tf.shape(train)[0], dtype=tf.float64))))
    resp = tf.matmul(left_m, sqk_gpr(train, input, l))
    return tf.matmul(tf.transpose(target), resp)
```

The fit graph generated by soft k-NN follows in Figure 9, while the fit graph generated by Gaussian process regression follows in Figure 10 for $\lambda = 100$

Figure 9: Prediction graph, soft k-NN, $\lambda = 100$

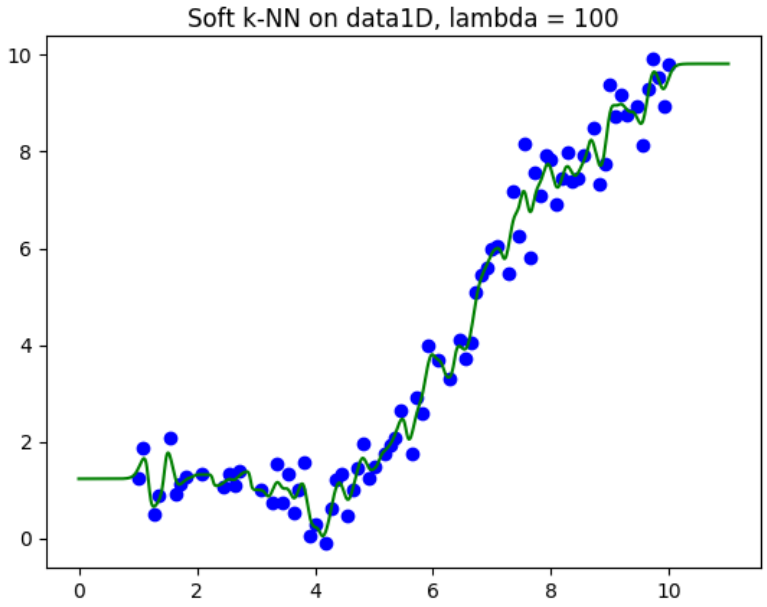
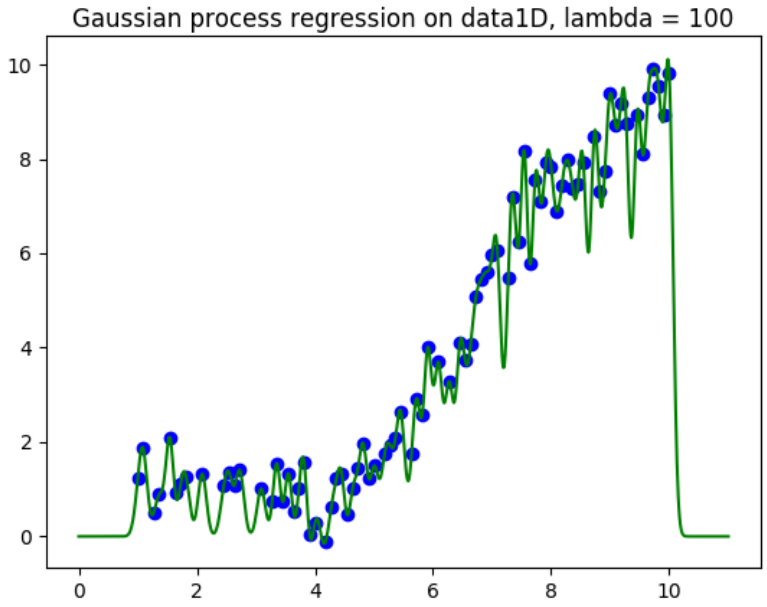


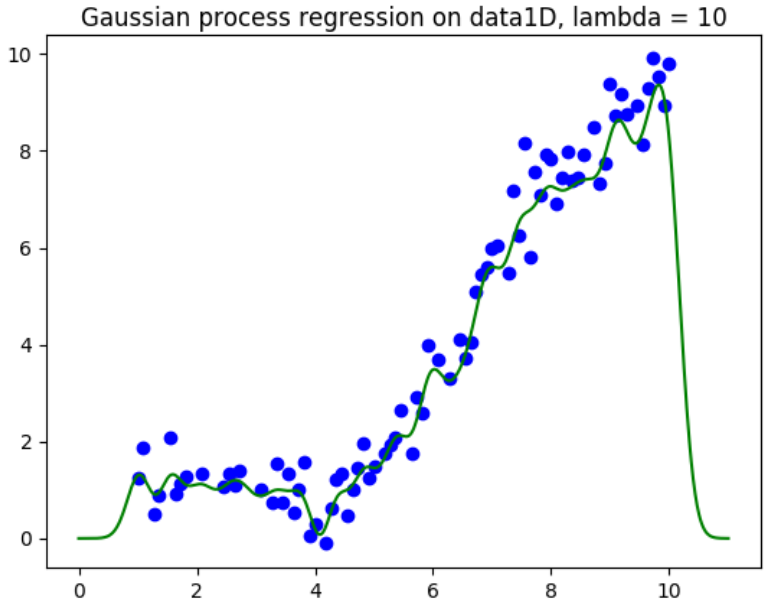
Figure 10: Prediction graph, GPR, $\lambda = 100$



It is obvious from the graphs that the better fit is achieved by the soft k-NN fit. The Gaussian process regression yields wild changes from point to point, and clearly is not generalizable as a whole. The reason for this wild variation is due to the amount of noise in the data. The given formula for the Gaussian process regression assumes that the training

data is noise free, and thus forces the predicted graph to cross each one of the test points (see Figure 10), and as such, due to the noise in the graph, results in wild variation in the prediction. By adding a constant value to each diagonal of the matrix $\mathcal{K}_{XX}^{-1}$ in the formula, this constraint is relaxed, allowing for much better fits. Figure 11 demonstrates a fit with a constant factor of 0.25.

Figure 11: Prediction graph, GPR with increased tolerances, $\lambda = 10$



■

Problem 1.4.2. ***Conditional distribution of a Gaussian:*** *Abridged - Derive the expression for the conditional mean* μ_{cond} *and variance* Σ_{cond} *in terms of* $y_{train}, \Sigma_{y^*y^*}, \Sigma_{y_{train}y_{train}}$, *and* $\Sigma_{y_{train}y^*}$.

Solution. Let y^*, μ^*, y_t, and μ_t denote the prediction input, prediction mean ,training input, and training mean respectively

$$P(\mathbf{y}) = \frac{1}{2\pi^{-\frac{M+1}{2}}|\Sigma|^{0.5}} exp\{-\frac{1}{2}(\mathbf{y}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{y}-\mu)\}$$

$$\begin{aligned}(\mathbf{y}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{y}-\mu) &= (y_t-\mu_t, y^*-\mu^*)^T \begin{bmatrix}\Sigma_{y_ty_t} & \Sigma_{y_ty^*} \\ \Sigma_{y^*y_t} & \Sigma_{y^*y^*}\end{bmatrix} (y_t-\mu_t, y^*-\mu^*) \\ &= (y_t-\mu_t)^T\Lambda_{y_ty_t}(y_t-\mu_t) + (y_t-\mu_t)^T\Lambda_{y_ty^*}(y^*-\mu^*) \\ &+ (y^*-\mu^*)^T\Lambda_{y^*y_t}(y_t-\mu_t) + (y^*-\mu^*)^T\Lambda_{y^*y^*}(y^*-\mu^*)\end{aligned}$$

From the identity given in the assignment and the identity:

$$(A+CBD)^{-1} = A^{-1} - A^{-1}C(B^{-1}+DA^{-1}C)^{-1}DA^{-1}$$

We have:

$$\begin{aligned}
\Lambda_{y_t y_t} &= \Sigma_{y_t y_t}^{-1} + \Sigma_{y_t y_t}^{-1}\Sigma_{y_t y^*}(\Sigma_{y^* y^*} - \Sigma_{y_t y^*}^T\Sigma_{y_t y_t}^{-1}\Sigma_{y_t y^*})^{-1}\Sigma_{y_t y^*}^T\Sigma_{y_t y_t}^{-1} \\
\Lambda_{y^* y^*} &= (\Sigma_{y^* y^*} - \Sigma_{y_t y^*}^T\Sigma_{y_t y_t}^{-1}\Sigma_{y_t y^*})^{-1} \\
\Lambda_{y_t y^*} &= -\Sigma_{y_t y_t}^{-1}\Sigma_{y_t y^*}(\Sigma_{y^* y^*} - \Sigma_{y_t y^*}^T\Sigma_{y_t y_t}^{-1}\Sigma_{y_t y^*})^{-1} = \Lambda_{y^* y_t}^T
\end{aligned}$$

Substituting back, we get:

$$\begin{aligned}
(\mathbf{y}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{y}-\mu) &= (y_t-\mu_t)^T(\Sigma_{y_t y_t}^{-1} + \Sigma_{y_t y_t}^{-1}\Sigma_{y_t y^*}(\Sigma_{y^* y^*} - \Sigma_{y_t y^*}^T\Sigma_{y_t y_t}^{-1}\Sigma_{y_t y^*})^{-1}\Sigma_{y_t y^*}^T\Sigma_{y_t y_t}^{-1})(y_t-\mu_t) \\
&\quad + 2(y_t-\mu_t)^T(-\Sigma_{y_t y_t}^{-1}\Sigma_{y_t y^*}(\Sigma_{y^* y^*} - \Sigma_{y_t y^*}^T\Sigma_{y_t y_t}^{-1}\Sigma_{y_t y^*})^{-1})(y^*-\mu^*) \\
&\quad + (y^*-\mu^*)^T(\Sigma_{y^* y^*} - \Sigma_{y_t y^*}^T\Sigma_{y_t y_t}^{-1}\Sigma_{y_t y^*})^{-1}(y^*-\mu^*) \\
&= (y_t-\mu_t)^T(\Sigma_{y_t y_t}^{-1})(y_t-\mu_t) \\
&\quad + (y_t-\mu_t)^T(\Sigma_{y_t y_t}^{-1}\Sigma_{y_t y^*}(\Sigma_{y^* y^*} - \Sigma_{y_t y^*}^T\Sigma_{y_t y_t}^{-1}\Sigma_{y_t y^*})^{-1}\Sigma_{y_t y^*}^T\Sigma_{y_t y_t}^{-1})(y_t-\mu_t) \\
&\quad - 2(y_t-\mu_t)^T(\Sigma_{y_t y_t}^{-1}\Sigma_{y_t y^*}(\Sigma_{y^* y^*} - \Sigma_{y_t y^*}^T\Sigma_{y_t y_t}^{-1}\Sigma_{y_t y^*})^{-1})(y^*-\mu^*) \\
&\quad + (y^*-\mu^*)^T(\Sigma_{y^* y^*} - \Sigma_{y_t y^*}^T\Sigma_{y_t y_t}^{-1}\Sigma_{y_t y^*})^{-1}(y^*-\mu^*) \\
&= (y_t-\mu_t)^T(\Sigma_{y_t y_t}^{-1})(y_t-\mu_t) \\
&\quad + [(y^*-\mu^*) - \Sigma_{y_t y^*}^T\Sigma_{y_t y_t}^{-1}(y_t-\mu_t)]^T(\Sigma_{y^* y^*} - \Sigma_{y_t y^*}^T\Sigma_{y_t y_t}^{-1}\Sigma_{y_t y^*})^{-1}[(y^*-\mu^*) \\
&\quad - \Sigma_{y_t y^*}^T\Sigma_{y_t y_t}^{-1}(y_t-\mu_t)]
\end{aligned}$$

We define the conditional mean and covariance, μ_{cond}, Σ_{cond} as follows:

$$\begin{aligned}
\mu_{cond} &= \mu^* + \Sigma_{y_t y^*}^T\Sigma_{y_t y_t}^{-1}(y_t-\mu_t) \\
\Sigma_{cond} &= \Sigma_{y^* y^*} - \Sigma_{y_t y^*}^T\Sigma_{y_t y_t}^{-1}\Sigma_{y_t y^*}
\end{aligned}$$

Thus we have:

$$(\mathbf{y}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{y}-\mu) = (y_t-\mu_t)^T(\Sigma_{y_t y_t}^{-1})(y_t-\mu_t) + (y^*-\mu_{cond})^T\Sigma_{cond}^{-1}(y^*-\mu_{cond})$$

Using these results, we can decompose the initial probability function $P(\mathbf{y})$ into a joint probability:

$$\begin{aligned}
P(\mathbf{y}) &= P(y_t, y^*) \\
&= \frac{1}{2\pi^{-\frac{M+1}{2}}|\Sigma|^{0.5}} exp\{-\frac{1}{2}((y_t-\mu_t)^T(\Sigma_{y_t y_t}^{-1})(y_t-\mu_t) + (y^*-\mu_{cond})^T\Sigma_{cond}^{-1}(y^*-\mu_{cond}))\} \\
&= \mathcal{N}(x;\mu_t,\Sigma_{y_t y_t})\mathcal{N}(x;\mu_{cond},\Sigma_{cond})
\end{aligned}$$

We can thus express the conditional probability $P(y_t|y^*)$ as:

$$\begin{aligned}
P(y_t|y^*) &= \frac{P(y_t,y^*)}{P(y^*)} = \frac{P(y_t,y^*)}{\mathcal{N}(x;\mu_t,\Sigma_{y_t y_t})} \\
&= \frac{1}{2\pi^{-\frac{M+1}{2}}|\Sigma|^{0.5}} exp\{-\frac{1}{2}(y^*-\mu_{cond})^T\Sigma_{cond}^{-1}(y^*-\mu_{cond})\} \\
&= \mathcal{N}(x;\mu_{cond},\Sigma_{cond}).
\end{aligned}$$

Thus the expressions for the conditional mean and covariance are as defined. ■

2 Linear and logistic regression

2.1 Geometry of logistic regression

Problem 2.1.1. *Is the l_2 penalized mean squared error loss $\mathcal{L}$ 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. It can be shown that the sum of two convex functions is also convex. Let $g(x)$ and $f(x)$ be convex functions, and let $h(x) = g(x) + f(x)$. Thus, both f(x) and g(x) satisfy Jensen's inequality:

$$g(\alpha x_1 + (1-\alpha)x_2) \leq \alpha g(x_1) + (1-\alpha)g(x_2)$$
$$f(\alpha x_1 + (1-\alpha)x_2) \leq \alpha f(x_1) + (1-\alpha)f(x_2)$$

for $\alpha \in [0, 1]$. Adding these two inequalities together yields a third inequality

$$\begin{aligned} g(\alpha x_1 + (1-\alpha)x_2) + f(\alpha x_1 + (1-\alpha)x_2) \leq & \alpha g(x_1) + (1-\alpha)g(x_2) + \alpha * f(x_1) + (1-\alpha)f(x_2) \\ h(\alpha x_1 + (1-\alpha)x_2) \leq & \alpha(g(x_1) + f(x_1)) + (1-\alpha)(g(x_2) + f(x_2)) \\ h(\alpha x_1 + (1-\alpha)x_2) \leq & \alpha h(x_1) + (1-\alpha)h(x_2) \end{aligned}$$

for $\alpha \in [0, 1]$, which is Jensen's inequality.

Thus, we examine the two loss parameters separately as a function of W. The l_2 Euclidean distance function can be shown to obey Jensen's inequality. This is done by applying the definition of Jensen's inequality, and showing that it holds for all $\alpha \in$ [0,1], as shown below:

$$\mathcal{L}_D(\alpha W_1 + (1-\alpha)W_2) \leq \alpha \mathcal{L}_D(W_1) + (1-\alpha)\mathcal{L}_D(W_2)$$

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

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

Since b and $y^{(m)}$ are constants with respect to the weights, treat them as a constant, i.e. $b - y^{(m)} = C$. In addition, we consider the general N-D training example $x^{(m)}$, which is generalizable for any set of training examples (sum of convex functions are convex) .

$$||(\alpha W_1^T + (1-\alpha)W_2^T)x^{(m)} + C||_2^2 \leq \alpha ||W_1^T x^{(m)} + C||_2^2 + (1-\alpha)||W_2^T x^{(m)} + C||_2^2$$
$$((\alpha W_1^T + (1-\alpha)W_2^T)x_n^{(m)} + C)^2 \leq (\alpha W_1^T x_n^{(m)} + C)^2 + (1-\alpha)(W_2^T x_n^{(m)} + C)^2$$

$$\left(\alpha^2 W_1 W_1^T + \alpha(1-\alpha)(W_2 W_1^T + W_1 W_2^T) + (1-\alpha)^2 W_2 W_2^T\right)(x^{(m)})^2 (\alpha W_1^T + (1-\alpha)W_2^T) C x^{(m)} + (C)^2 \leq$$
$$(\alpha W_1 W_1^T + (1-\alpha)W_2 W_2^T)(x^{(m)})^2 + (\alpha W_1^T + (1-\alpha)W_2^T) C x^{(m)} + (\alpha + 1 - \alpha)(C)^2$$

Since the only difference between the two sides of the inequality are the quadratic terms (i.e. the linear and constant terms wrt $x_n^{(m)}$ are identical on either side), it is sufficient to compare only the quadratic coefficients.

$$\alpha^2 W_1 W_1^T + \alpha(1-\alpha)(W_2 W_1^T + W_1 W_2^T) + (1-\alpha)^2 W_2 W_2^T \leq \alpha W_1 W_1^T + (1-\alpha) W_2 W_2^T$$
$$0 \leq (\alpha - \alpha^2) W_1 W_1^T + (1 - \alpha - 1 + 2\alpha - \alpha^2) W_2 W_2^T + \alpha(1-\alpha)(W_2 W_1^T + W_1 W_2^T)$$
$$0 \leq (\alpha - \alpha^2)(W_1 W_1^T + W_2 W_2^T + W_2 W_1^T + W_1 W_2^T)$$
$$0 \leq (W_1^T + W_2^T)^T (W_1^T + W_2^T)$$
$$0 \leq ||W_1^T + W_2^T||_2^2$$

which is true for all α, and is thus true for $\alpha \in [0, 1]$. Thus Jensen's inequality is shown to be true for $\mathcal{L}_D(W)$

The proof for $\mathcal{L}_W(W)$ follows in a similar manner.

$$\mathcal{L}_W(\alpha W_1 + (1-\alpha) W_2) \leq \alpha \mathcal{L}_W(W_1) + (1-\alpha)\mathcal{L}_W(W_2)$$
$$\frac{\lambda}{2}||\alpha W_1 + (1-\alpha) W_2||_2^2 \leq \alpha \frac{\lambda}{2}||W_1||_2^2 + (1-\alpha)\frac{\lambda}{2}||W_2||_2^2$$
$$\alpha^2 W_1^T W_1 + \alpha(1-\alpha)(W_2^T W_1 + W_1^T W_2) + (1-\alpha)^2 W_2^T W_2 \leq \alpha W_1^T W_1 + (1-\alpha) W_2^T W_2$$
$$0 \leq (\alpha - \alpha^2) W_1^T W_1 + (1 - \alpha - 1 + 2\alpha - \alpha^2) W_2^T W_2 + \alpha(1-\alpha)(W_2^T W_1 + W_1^T W_2)$$
$$0 \leq (W_1 + W_2)^T (W_1 + W_2)$$
$$0 \leq ||W_1 + W_2||_2^2$$

which is true for all α, and thus holds for $\alpha \in [0, 1]$. Thus $\mathcal{L}_W(W)$ is a convex function.

Since $\mathcal{L}_D(W)$ and $\mathcal{L}_W(W)$ are convex functions, the total penalized l_2 loss function$\mathcal{L}(W) = \mathcal{L}_D(W) + \mathcal{L}_W(W)$ is a convex function of W.

The proof is similar for b. Since $W^T x^{(m)}$ is constant for the generalized training set as examined in the proof for the convexity with respect to W, one can perform the following simplifications:

- Set $K = W^T x^{(m)} - y^{(m)}$ as constant
- Examine only one training example - summation preserves convexity
- Ignore the regularization term, since it isn't a function of b - that is, we only need to examine the convexity of $\mathcal{L}_D$ to prove convexity of the entire term.

The proof of the convexity of $\mathcal{L}_D$ is shown below. We apply Jensen's inequality with respect

to b, yielding:

$$\mathcal{L}_D(\alpha b_1 + (1-\alpha)b_2) \leq \alpha\mathcal{L}_D(b_1) + (1-\alpha)\mathcal{L}_D(b_2)$$
$$((\alpha b_1 + (1-\alpha)b_2) + K)^2 \leq (\alpha b_1 + K)^2 + (1-\alpha)(b+K)^2$$

$$\left(\alpha^2 b_1^2 + 2\alpha(1-\alpha)(b_1 b_2) + (1-\alpha)^2 b_2^2\right) + (\alpha b_1 + (1-\alpha)b_2)K + (D)^2 \leq$$
$$(\alpha b_1^2 + (1-\alpha)b_2^2) + (\alpha b_1 + (1-\alpha)b_2)K + (\alpha + 1 - \alpha)(K)^2$$

$$0 \leq \alpha(1-\alpha)(b_1^2 + 2(b_1 b_2) + b_2^2$$
$$0 \leq (b_1 + b_2)^2$$

which is true for all α, and thus holds for $\alpha \in [0, 1]$. Thus the loss function is convex in b as well. ■

Problem 2.1.2. *Consider learning a linear regression model of a single scalar output with a weight vector $W \in \mathbb{R}^N$, bias $b \in \mathbb{R}$ and no regularizer, on a dataset $\mathcal{D} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(M)}, y^{(M)})\}$. If the nth 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 one scales the nth dimension of the training data by a factor of α, the corresponding weight in the nth dimension will be scaled by a factor of $\frac{1}{\alpha}$. This will produce the same l_2 loss. Similarly, when offsetting the nth dimension of the training data by a constant β, the bias will be offset by -β. This will also produce the same l_2 loss. As scaling the nth dimension by a constant and introducing an offset are both linear transformations, the loss function will undergo the same transformations, thus remaining the same. ■

Problem 2.1.3. *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. Given the similarity of this question with the previous question, this answer will enumerate the differences between the two cases. The addition of a penalization term does not impact the effect of the biases. Thus, the bias will still be offset by $-\beta$. As for the weight vector, the nth term is still scaled by a factor of $\frac{1}{\alpha}$, producing the same l_2 loss. The regularization term will be decreased as the nth term in the weight vector will decrease. Thus the minimum value of the loss function is less than the minimum loss before the transformation, and is greater than the loss from problem 2.1.2. ■

Problem 2.1.4. *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*

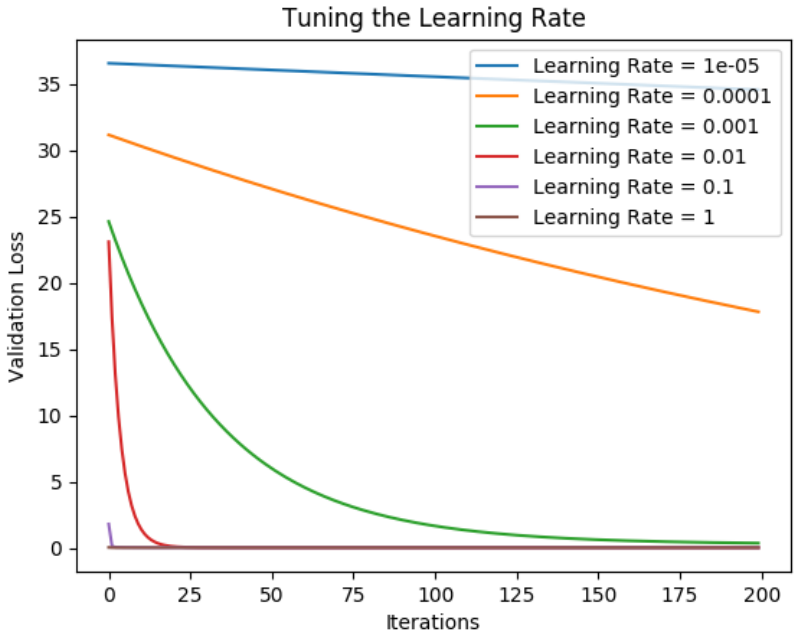
Solution. One could take a one vs all approach and use N classifiers to determine whether or not each point belongs to each individual class. A possible issue with this is when points are labeled as either more than one class, or no class at all. An alternative approach is to use many one vs one classifiers, one for each possible pair of classes. However, a similar problem occurs due to the non-transitive nature of two-way preferences. A point can be more class 2 than class 1, more class 3 than class 2, and yet more class 1 than class 3. ■

2.2 Stochastic gradient descent

Problem 2.2.1. ***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 $\mathcal{L}$. Set the weight decay coefficient $\lambda = 1$. Tune the learning rate η to obtain the best overall convergence speed of the algorithm. Plot the total loss function $\mathcal{L}$ vs. the number of updates for the best learning rate found and discuss the effect of the learning-rate value on training convergence*

Solution. As the learning rate is increased, the speed of convergence increases as well. However, if the learning rate increases past a certain threshold (approximately 1.2 in our case), the model does not converge. The rates are shown in Figure 12 .

Figure 12: Tuning the learning rate

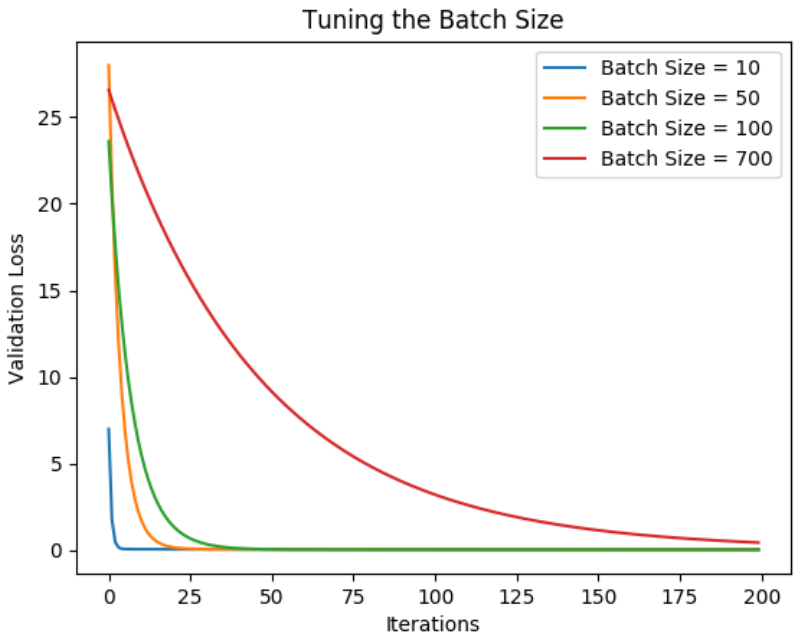


■

Problem 2.2.2. *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 $\mathcal{L}$ 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. The best mini-batch size in terms of training time was the largest value, 700. This is because the calculations are done in parallel, rather than sequentially, thus drastically speeding up the process. However, larger batch sizes take more iterations to converge. The results are summarized in Figure 13 below.

Figure 13: Tuning the batch size

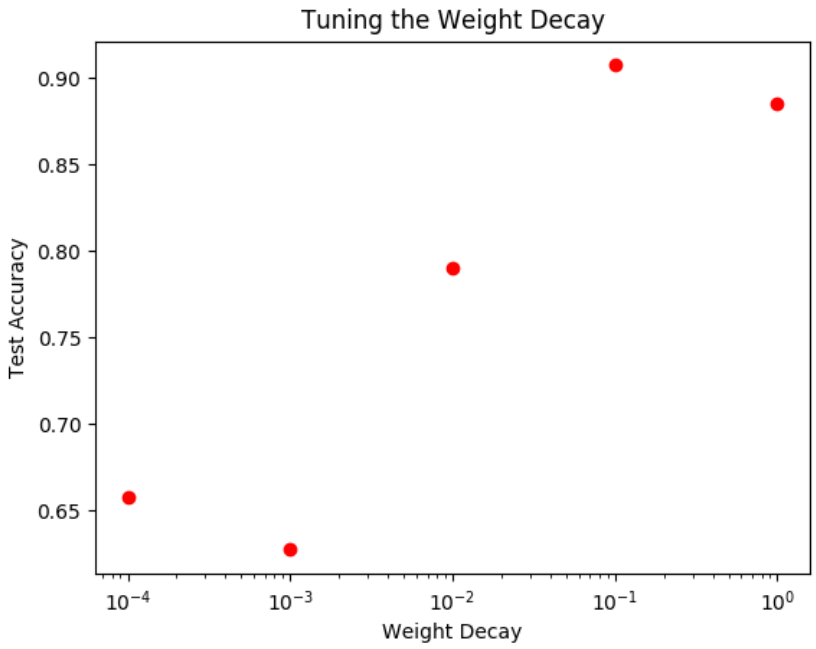


■

Problem 2.2.3. *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. From our investigation, as shown in Figure 14, 0.1 was the best value for the weight decay coefficient λ. As the weight decay coefficient increases, the weights are more incentivized to take on lower values and generalization of the model increases, shown by the increasing test accuracy. However, after a certain point, the weight decay penalizes the weights too much and force the weights to take values that result in lower accuracies. The plot illustrates this through an increase and subsequent decrease of test accuracy with increasing weight decay. It is also important to tune the hyper-parameters using the validation set instead of the training set. This is to avoid overtraining on the training set and improve generalization of the model.

Figure 14: Tuning the weight decay



■