P04 - KNN Linear and Regression

Luis Sa-Couto[1] and Andreas Wichert[2]

INESC-ID, Instituto Superior Tecnico, Universidade de Lisboa
{luis.sa.couto,andreas.wichert}@tecnico.ulisboa.pt

1 Nearest Neighbour

The k Nearest Neighbors (kNN) classifier works under the assumption that examples that share similar features will likely have the same class. With that, if we represent every example as a feature vector, we can, for instance, compute the euclidean distance between a new example and some known training data to find out what previously seen examples are similar to the new one. We can then use that information to perform classification.

To illustrate this idea let us describe a simple example task. Assuming that 1 means $True$ and 0 means $False$, consider the following features and class:

- X_1: "Weight (Kg)"
- X_2: "Height (Cm)"
- $Class$: "NBA (National Basketball Association) Player"

Consider as well that you are given the following training set:

	$\mathbf{x}^{(1)}$	$\mathbf{x}^{(2)}$	$\mathbf{x}^{(3)}$	$\mathbf{x}^{(4)}$	$\mathbf{x}^{(5)}$	$\mathbf{x}^{(6)}$	$\mathbf{x}^{(7)}$	$\mathbf{x}^{(8)}$	$\mathbf{x}^{(9)}$	$\mathbf{x}^{(10)}$
X_1	170	80	90	60	50	70	90	100	110	80
X_2	160	220	200	160	150	190	170	180	178	210
$Class$	0	1	1	0	0	1	0	0	0	0

Because these data are two-dimensional, we can plot them in figure 1 to get a general sense.

If we are given a new test example, represented in figure 1 by a dark cross, a nearest neighbor classifier starts by finding the k training examples that are most similar to this new point. In figure 2 we are assuming that $k = 1$, so figure 1 highlights the closest training example in euclidean distance. Having the set of closest neighbors, the classifier outputs the label of the majority within that set. In this case, the only element in the set is an NBA player, so the classifier outputs 1 as we see in figure 1.

The idea of choosing the majoritary label implies that we typically set k to be an odd number. Otherwise there would be ties to deal with.

To consolidate our undestanding of the process, let us focus on the example in figure 3. This time we set $k = 3$, so figure 1 highlights the three closest training examples. This three point set has two NBA players and one regular person. For that reason, the classifier would, yet again, output label 1 as is shown in figure 1.

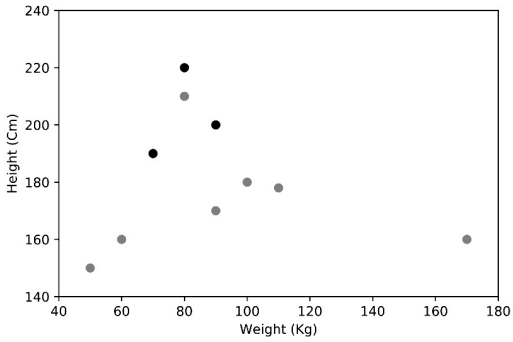


Fig. 1. The data collection of people represented in feature space where blue points are NBA players.

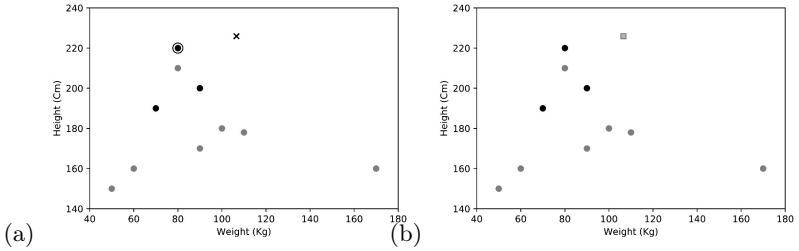


Fig. 2. An example of a one nearest neighbor classifier where in a) we find the closest training example and in b) we use the neighbor's label to classify the input.

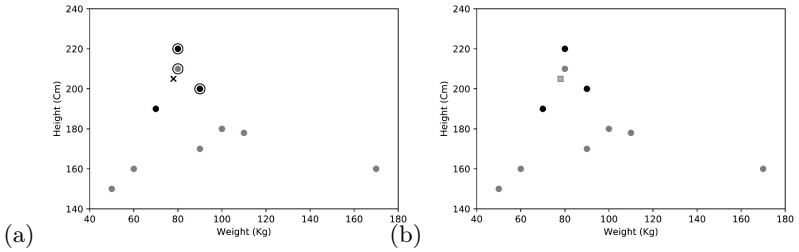


Fig. 3. An example of a three nearest neighbors classifier where in a) we find the set of closest training examples and in b) we use the set's majority label to classify the input.

In this case, increasing k did not produce an impact. However, this is not always the case. In fact, this is a key model parameter that needs to be chosen wisely. To see some of the key impacts let us focus on figure 4. In general, to build these plots, we have, for each scenario, classified every possible point in the feature space. This allows us to see which regions are labeled 1 and which are labeled 0 for four different scenarios for the parameter k. Let us analyze each one.

Figure 1 presents the regions for a one nearest neighbor. In this case, we see a manifestation of overfitting. An outlier example of a regular person in the middle of several NBA players leads the classifier to think that there is something particular about those heights and weights when there is not.

In figures 1 and 1, we increase k to three and five and we see that the overfitting problem seems to disappear. Both scenarios show a smoother decision boundary between classes and both look like reasonable classification rules. In fact, this pattern of reducing overfitting by increasing k usually appears. However, an increase of this parameter can also create problems.

The last scenario (figure 1), shows the classification regions for $k = 7$. In this case, we see that all points are classified as non-NBA players. This happens because there are not enough examples of this class.

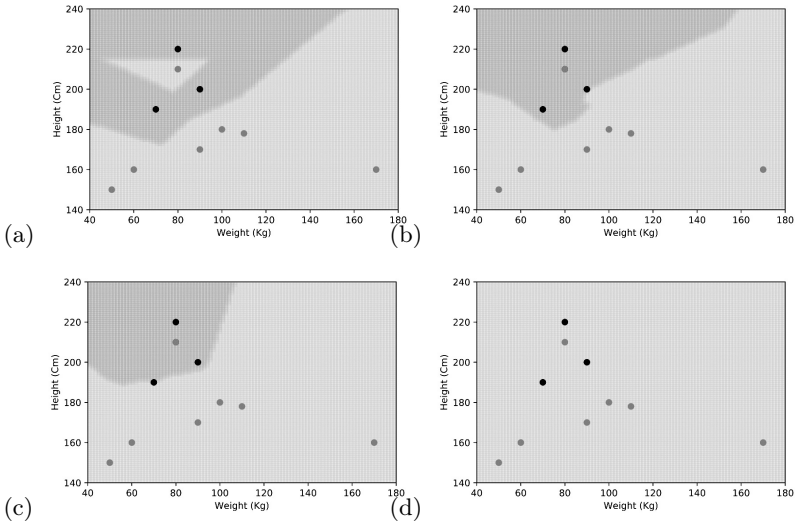


Fig. 4. In this figure we plot four scenarios with different values of k. In 1 we find a small classification region that appears due to overfitting. In 1 and 1 we see that increasing the number of neighbors yields a smoother, more general decision landscape. Finally, in 1 we see that a bigger increase of the number of neighbors creates problems and all new inputs will be classified as non-NBA players.

From the four described scenarios we can take away that the choice of k is bounded by trade-offs. As usual, to deal with this issue we need to conduct experiments with validation data to make an informed decision.

Unlike most classifiers we look at in this text, kNN does not use parameters and update rules to do learning. In fact one may wonder if there is any learning at all. This type of classifier is called non-parametric and because learning amounts to storing and using a collection of training instances, it is usually refered to as instance-based learning.

The fact that we need to store and use the whole collection of data can become a big drawback of this approach. However, for small data collections it can be very useful for three main reasons. First of all, it is extremely easy to implement and use. Second, learning is immediate. Finally, we can apply it to many kinds of data besides vectors. All we need is a similarity measure between examples to find out the most similar neighbors.

1) a) Using the same data from the example above, use a k nearest neighbors classifier to classify vector $\mathbf{x} = \begin{bmatrix} 100 & 210 \end{bmatrix}^T$ with $k = 1$, $k = 3$ and $k = 5$.

Solution:

The first step is to measure the $l1$ distance from the input to all labeled data vectors.

	$\mathbf{x}^{(1)}$	$\mathbf{x}^{(2)}$	$\mathbf{x}^{(3)}$	$\mathbf{x}^{(4)}$	$\mathbf{x}^{(5)}$	$\mathbf{x}^{(6)}$	$\mathbf{x}^{(7)}$	$\mathbf{x}^{(8)}$	$\mathbf{x}^{(9)}$	$\mathbf{x}^{(10)}$
X_1	170	80	90	60	50	70	90	100	110	80
X_2	160	220	200	160	150	190	170	180	178	210
$Class$	0	1	1	0	0	1	0	0	0	0
$\left\|\mathbf{x} - \mathbf{x}^{(i)}\right\|_2$	86.0	22.4	14.1	64.0	78.1	36.1	41.2	30.0	33.5	20.0

For $k = 1$, the classifier outputs the label of the nearest neighbor. In this case, the closest example is $\mathbf{x}^{(3)}$ at a distance of 14.14, so the output class will be 1.

For $k = 3$, we need the three nearest neighbors. For this input the neighbors will be $\mathbf{x}^{(3)}$, $\mathbf{x}^{(10)}$ and $\mathbf{x}^{(2)}$ from nearest to farthest. Out of this group, a majority of two examples have class 1, so this would be the output.

Finally, for $k = 5$, the set of neighbors is $\mathbf{x}^{(3)}$, $\mathbf{x}^{(10)}$, $\mathbf{x}^{(2)}$, $\mathbf{x}^{(8)}$ and $\mathbf{x}^{(9)}$. In this case, we have a majority of three examples with label 0.

---

b) Redo the exercise computing the distance with the $l1$ norm $\|\mathbf{x}\|_1 = \sum_i |x_i|$.

Solution:

Since we are working in two dimensions, it would be easy to plot the point and check the region where it falls in the previously shown region plots. However, we want to use a more general approach. The first step is to measure the euclidean distance from the query vector to all labeled data vectors.

	$\mathbf{x}^{(1)}$	$\mathbf{x}^{(2)}$	$\mathbf{x}^{(3)}$	$\mathbf{x}^{(4)}$	$\mathbf{x}^{(5)}$	$\mathbf{x}^{(6)}$	$\mathbf{x}^{(7)}$	$\mathbf{x}^{(8)}$	$\mathbf{x}^{(9)}$	$\mathbf{x}^{(10)}$
X_1	170	80	90	60	50	70	90	100	110	80
X_2	160	220	200	160	150	190	170	180	178	210
$Class$	0	1	1	0	0	1	0	0	0	0
$\left\|\mathbf{x}-\mathbf{x}^{(i)}\right\|_1$	120	30	20	90	110	50	50	30	42	20

For $k = 1$, the classifier outputs the label of the nearest neighbor. In this case, we have a tie for the closest example between $\mathbf{x}^{(3)}$ and $\mathbf{x}^{(10)}$ at a distance of 20. In practice one can choose randomly and take the class from that example.

For $k = 3$, we need the three nearest neighbors. For this input the neighbors will be $\mathbf{x}^{(3)}$, $\mathbf{x}^{(10)}$ and $\mathbf{x}^{(2)}$ from nearest to farthest. Out of this group, a majority of two examples have class 1, so this would be the output. It was also possible to choose $\mathbf{x}^{(8)}$ instead of $\mathbf{x}^{(2)}$ and in that case the label would be 0.

Finally, for $k = 5$, the set of neighbors is $\mathbf{x}^{(3)}$, $\mathbf{x}^{(10)}$, $\mathbf{x}^{(2)}$, $\mathbf{x}^{(8)}$ and $\mathbf{x}^{(9)}$. In this case, we have a majority of three examples with label 0.

---

c) Again, repeate the exercise computing the distance with the infinity norm $\|\mathbf{x}\|_\infty = max_i\,|x_i|$.

Solution:

Since we are working in two dimensions, it would be easy to plot the point and check the region where it falls in the previously shown region plots. However, we want to use a more general approach. The first step is to measure the euclidean distance from the query vector to all labeled data vectors.

	$\mathbf{x}^{(1)}$	$\mathbf{x}^{(2)}$	$\mathbf{x}^{(3)}$	$\mathbf{x}^{(4)}$	$\mathbf{x}^{(5)}$	$\mathbf{x}^{(6)}$	$\mathbf{x}^{(7)}$	$\mathbf{x}^{(8)}$	$\mathbf{x}^{(9)}$	$\mathbf{x}^{(10)}$
X_1	170	80	90	60	50	70	90	100	110	80
X_2	160	220	200	160	150	190	170	180	178	210
$Class$	0	1	1	0	0	1	0	0	0	0
$\left\|\mathbf{x}-\mathbf{x}^{(i)}\right\|_\infty$	70	20	10	50	60	30	40	30	32	20

For $k = 1$, the classifier outputs the label of the nearest neighbor. In this case, the closest example is $\mathbf{x}^{(3)}$ at a distance of 10, so the output class will be 1.

For $k = 3$, we need the three nearest neighbors. For this input the neighbors will be $\mathbf{x}^{(3)}$, $\mathbf{x}^{(10)}$ and $\mathbf{x}^{(2)}$ from nearest to farthest. Out of this group, a majority of two examples have class 1, so this would be the output.

Finally, for $k = 5$, the set of neighbors is $\mathbf{x}^{(3)}$, $\mathbf{x}^{(10)}$, $\mathbf{x}^{(2)}$, $\mathbf{x}^{(6)}$ and $\mathbf{x}^{(8)}$. In this case, we have a majority of three examples with label 1.

---

2) Assuming that 1 means $True$ and 0 means $False$, consider the following features and class:

- X_1: "Fast processing"
- X_2: "Decent Battery"
- X_3: "Good Camera"
- X_4: "Good Look and Feel"
- X_5: "Easiness of Use"
- $Class$: "iPhone"

You are given the following training set:

	$\mathbf{x}^{(1)}$	$\mathbf{x}^{(2)}$	$\mathbf{x}^{(3)}$	$\mathbf{x}^{(4)}$	$\mathbf{x}^{(5)}$	$\mathbf{x}^{(6)}$	$\mathbf{x}^{(7)}$
X_1	1	1	0	0	1	0	0
X_2	1	1	1	0	0	0	0
X_3	0	1	1	0	1	1	0
X_4	1	0	1	1	1	0	0
X_5	0	0	0	1	1	0	1
$Class$	1	0	0	0	1	1	1

Use a k nearest neighbors classifier based on the Hamming distance to classify vector $\begin{bmatrix}1 & 1 & 1 & 1 & 1\end{bmatrix}^T$ with $k = 1$ and $k = 3$.

Solution:

When working with binary data it is common to use different, binary specific distance measures. In this example we will use the Hamming distance which counts the number of different bits between the two vectors. This is a good example to remind us that the nearest neighbor classifier is not tied to the euclidean distance measure.

Having chosen the Hamming distance, we need to compute it for all training examples.

	$\mathbf{x}^{(1)}$	$\mathbf{x}^{(2)}$	$\mathbf{x}^{(3)}$	$\mathbf{x}^{(4)}$	$\mathbf{x}^{(5)}$	$\mathbf{x}^{(6)}$	$\mathbf{x}^{(7)}$
X_1	1	1	0	0	1	0	0
X_2	1	1	1	0	0	0	0
X_3	0	1	1	0	1	1	0
X_4	1	0	1	1	1	0	0
X_5	0	0	0	1	1	0	1
$Class$	1	0	0	0	1	1	1
$Hamming\left(\mathbf{x}, \mathbf{x}^{(i)}\right)$	2	3	3	4	1	4	4

For $k = 1$, the classifier outputs the label of the nearest neighbor. In this case, the closest example is $\mathbf{x}^{(5)}$ at a distance of 1, so the output class will be 1.

For $k = 3$, we need the three nearest neighbors. For this input these neighbors will be $\mathbf{x}^{(5)}$, $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ (it could also be $\mathbf{x}^{(3)}$) from nearest to farthest. Out of this group, a majority of two examples have class 1, so this would be the output.

---

3) Consider the following data where a few preprocessed restaurant reviews (without stopwords) are classified as positive (1) or negative (0).

Sentence	*Class*
{"*Great*","*place*","*go*","*with*","*friends*"}	1
{"*Food*","*amazing*"}	1
{"*What*","*terrible*","*experience*","*no*","*words*"}	0
{"*Waiting*","*time*","*too*","*long*"}	0
{"*Terrible*","*place*","*for*","*family*","*dinner*"}	0

Consider as well the Jaccard similarity between two sets:

$$Jaccard_{sim}(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Using the similarity measure, compute the k nearest neighbor output for input {"*Terrible*","*food*","*overall*","*lousy*","*dinner*"} using $k = 1$ and $k = 3$.

Solution:

Much like before we start by computing the similarity between the input and the training data.

	Sentence	*Class*	*Jaccard*
$x^{(1)}$	{"*Great*","*place*","*go*","*with*","*friends*"}	1	$\frac{0}{10} = 0$
$x^{(2)}$	{"*Food*","*amazing*"}	1	$\frac{1}{6}$
$x^{(3)}$	{"*What*","*terrible*","*experience*","*no*","*words*"}	0	$\frac{1}{9}$
$x^{(4)}$	{"*Waiting*","*time*","*too*","*long*"}	0	$\frac{0}{9} = 0$
$x^{(5)}$	{"*Terrible*","*place*","*for*","*family*","*dinner*"}	0	$\frac{2}{8} = \frac{1}{4}$

For $k = 1$, we need to find the closest neighbor. Working with a similarity measure instead of a distance measure, the closest example will be the one with highest similarity. In this case $x^{(5)}$ is most similar, so the classifier outputs its label of 0.

For $k = 3$, we need the three nearest neighbors. For this query set these neighbors will be $x^{(5)}$, $x^{(2)}$ and $x^{(3)}$. Out of this group, a majority of two examples have class 0, so this would be the output.

Alternatively, we could have followed the same reasoning as in previous exercises by converting the similarity measure to a distance through:

$$Jaccard_{dist}(A,B) = 1 - Jaccard_{sim}(A,B)$$

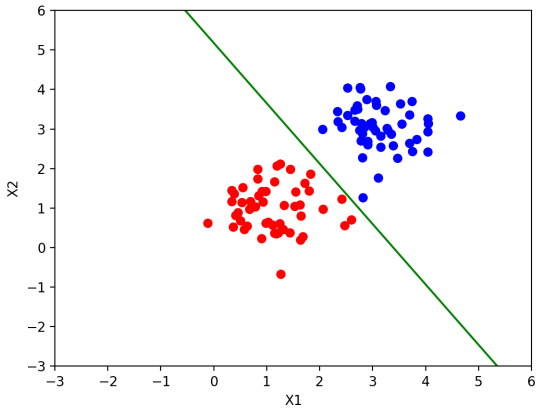
2 Closed form learning

Linear Regression

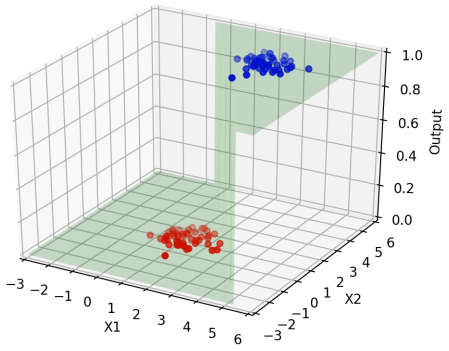
In this section we will be focusing on the model of linear regression. Much like perceptron, this model works by learning a weight vector $\mathbf{w} = \begin{pmatrix} w_0 & w_1 & \cdots & w_d \end{pmatrix}^T$ that is used to compute a linear combination of the input features $\mathbf{x} = \begin{pmatrix} x_1 & \cdots & x_d \end{pmatrix}^T$. However, this model focuses on regression instead of classification. So, for that reason, its output is not binary. It is, in fact, a real number $\mathit{output}(\mathbf{x}; \mathbf{w}) = o \in \mathbb{R}$. So achieve that, we get rid of the sign function and work directly with the linear combination (dot product).

We can think about the linear regression as providing a score for each point. Instead of making a binary decision for each point (i.e. belongs to the class or not) we give a score to each point where higher scores are more likely to belong to the class.

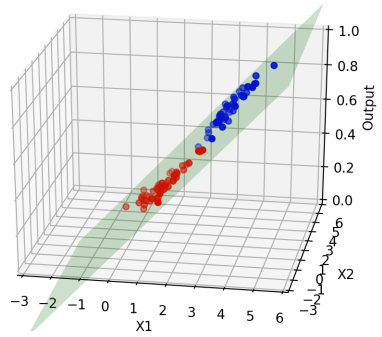
The following image shows a linearly separable problem with two input features:



If we add a third dimension for the output, we can think of the perceptron as putting all points that don't belong to the class on the $\mathit{Output} = 0$ plane and all points that belong to the class on the $\mathit{Output} = 1$ plane:



In linear regression, we have a similar picture but we don't make binary decisions, we provide an output score:



Looking at both plots we see that as the formulas suggest, the perceptron basically squashes the linear regression scores into a binary decision $perceptron\left(\mathbf{x};\mathbf{w}\right) = \Theta\left(linear\_regression\left(\mathbf{x};\mathbf{w}\right)\right)$.

Closed form solution of squared error

So, let us assume we have training data $\left\{\left(\mathbf{x}^{(1)},t^{(1)}\right),\left(\mathbf{x}^{(2)},t^{(2)}\right),\ldots,\left(\mathbf{x}^{(n)},t^{(n)}\right)\right\}$. For a given weight vector $\mathbf{w}$, the linear regression outputs are:

$$
\begin{aligned}
output^{(1)}\left(\mathbf{x}^{(1)};\mathbf{w}\right) &= o^{(1)} = \mathbf{w}^T\mathbf{x}^{(1)} \\
output^{(2)}\left(\mathbf{x}^{(2)};\mathbf{w}\right) &= o^{(2)} = \mathbf{w}^T\mathbf{x}^{(2)} \\
&\vdots \\
output^{(n)}\left(\mathbf{x}^{(n)};\mathbf{w}\right) &= o^{(n)} = \mathbf{w}^{\mathbf{T}}\mathbf{x}^{(n)}
\end{aligned}
$$

All computations we will need to do will be made easier by working in vector notation. So, we can write the whole system of equations as follows:

$$
\begin{aligned}
\begin{pmatrix} o^{(1)} \\ o^{(2)} \\ \vdots \\ o^{(n)} \end{pmatrix} &= \begin{pmatrix} 1 & x_1^{(1)} & \cdots & x_d^{(1)} \\ 1 & x_1^{(2)} & \cdots & x_d^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(n)} & \cdots & x_d^{(n)} \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{pmatrix} \\
\mathbf{O} &= \mathbf{X}\mathbf{w}
\end{aligned}
$$

Having stated how linear regression maps inputs to outputs, we can focus on the central problem of learning. We will look into the most common form of learning. That is, learning by optimization. More specifically, we want to find a weight vector such that our outputs closely approximate the targets $o^{(i)} \sim t^{(i)}$. To find that vector, it is common to define an error function that measures how wrong the outputs are. A well known error function is the sum of squared errors:

$$
E\left(\mathbf{w}\right) = \sum_{i=1}^{n} \left(t^{(i)} - o^{(i)}\right)^2
$$

Again, we can switch to vector notation as follows:

$$
\begin{aligned}
E\left(\mathbf{w}\right) &= \sum_{i=1}^{n} \left(t^{(i)} - o^{(i)}\right)^2 \\
&= \sum_{i=1}^{n} \left(t^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)}\right)^2 \\
&= \left(\mathbf{T} - \mathbf{X}\mathbf{w}\right)^T \left(\mathbf{T} - \mathbf{X}\mathbf{w}\right)
\end{aligned}
$$

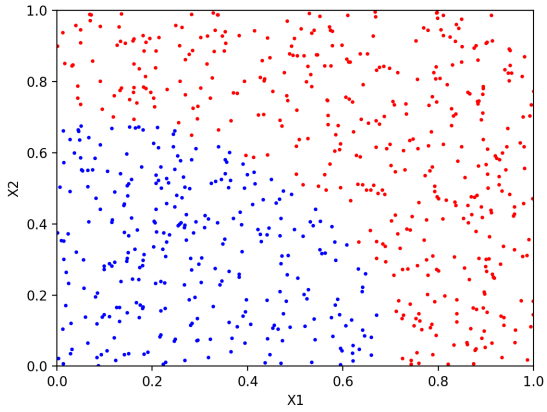
Having defined the error function, the learning problem can be solved by finding its minimum. For linear regression, we can compute a closed form solution for this problem. We do it by computing the derivative and finding its zero:

$$
\begin{aligned}
\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} &= 0 \\
\frac{\partial (\mathbf{T}-\mathbf{X}\mathbf{w})^T (\mathbf{T}-\mathbf{X}\mathbf{w})}{\partial \mathbf{w}} &= 0 \\
\left(\frac{\partial}{\partial \mathbf{w}} (\mathbf{T}-\mathbf{X}\mathbf{w})^T\right)(\mathbf{T}-\mathbf{X}\mathbf{w}) + (\mathbf{T}-\mathbf{X}\mathbf{w})^T \left(\frac{\partial}{\partial \mathbf{w}} (\mathbf{T}-\mathbf{X}\mathbf{w})\right) &= 0 \\
\left(-\mathbf{X}^T\right)(\mathbf{T}-\mathbf{X}\mathbf{w}) + (\mathbf{T}-\mathbf{X}\mathbf{w})^T (-\mathbf{X}) &= 0 \\
\left(-\mathbf{X}^T\right)(\mathbf{T}-\mathbf{X}\mathbf{w}) + (-\mathbf{X})^T (\mathbf{T}-\mathbf{X}\mathbf{w}) &= 0 \\
-2\mathbf{X}^T (\mathbf{T}-\mathbf{X}\mathbf{w}) &= 0 \\
\mathbf{X}^T (\mathbf{T}-\mathbf{X}\mathbf{w}) &= 0 \\
\mathbf{X}^T\mathbf{T} - \mathbf{X}^T\mathbf{X}\mathbf{w} &= 0 \\
\mathbf{X}^T\mathbf{X}\mathbf{w} &= \mathbf{X}^T\mathbf{T} \\
\mathbf{w} &= \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{T}
\end{aligned}
$$

So, unlike perceptron where we had to take multiple iterative steps to learn the correct weights, in linear regression we can learn them directly.

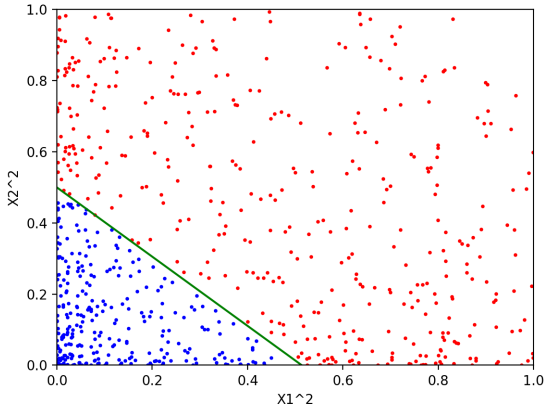
Feature Transformations

Much like perceptron, the typical linear regression can only work well with linear problems. However, in many problems we can fix this by changing the feature space we are working in. For example, the following image shows points of two different classes that cannot be separated by a line:



However, if we notice that the difference between classes is related to their distance from the origin (point $(0, 0)$) we can work with the squares of the

features x_1^2, x_2^2 instead of the features themselves. The following plot shows how this transformation turns the problem into a linear one. In this space, we could use a perceptron to solve the classification issue perfectly.



This idea of feature transformation can be used in general (both in classification and regression) to transform non-linear problems into linear ones. In practice, we take the input matrix:

$$\mathbf{X} = \begin{pmatrix} 1 & x_1^{(1)} & \cdots & x_d^{(1)} \\ 1 & x_1^{(2)} & \cdots & x_d^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(n)} & \cdots & x_d^{(n)} \end{pmatrix}$$

and we choose feature transforming functions (which in the previous example correspond to taking the squares of the features $\phi_k^{(i)}\left(x_1^{(i)}, \ldots, x_d^{(i)}\right) = x_k^{(i)2}$) to build the new input matrix:

$$\mathbf{\Phi} = \begin{pmatrix} 1 & \phi_1^{(1)}\left(x_1^{(1)}, \ldots, x_d^{(1)}\right) & \cdots & \phi_l^{(1)}\left(x_1^{(1)}, \ldots, x_d^{(1)}\right) \\ 1 & \phi_1^{(2)}\left(x_1^{(2)}, \ldots, x_d^{(2)}\right) & \cdots & \phi_l^{(2)}\left(x_1^{(2)}, \ldots, x_d^{(2)}\right) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \phi_1^{(n)}\left(x_1^{(n)}, \ldots, x_d^{(n)}\right) & \cdots & \phi_l^{(n)}\left(x_1^{(n)}, \ldots, x_d^{(n)}\right) \end{pmatrix}$$

the rest of the process stays exactly the same. We just have a different input!

1) Consider the following training data:

$$\left\{\mathbf{x}^{(1)} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \mathbf{x}^{(2)} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \mathbf{x}^{(3)} = \begin{pmatrix} 1 \\ 3 \end{pmatrix}, \mathbf{x}^{(4)} = \begin{pmatrix} 3 \\ 3 \end{pmatrix}\right\}$$

$$\left\{t^{(1)} = 1.4, t^{(2)} = 0.5, t^{(3)} = 2, t^{(4)} = 2.5\right\}$$

a) Find the closed form solution for a linear regression that minimizes the sum of squared errors on the training data..

Solution:

First, we need to build the $n \times (d+1)$ design matrix to account for the bias parameter, where n is the number of examples and d is the original number of input features.

$$X = \begin{pmatrix} 1\ 1\ 1 \\ 1\ 2\ 1 \\ 1\ 1\ 3 \\ 1\ 3\ 3 \end{pmatrix}$$

Second, we construct a target vector:

$$Y = \begin{pmatrix} 1.4 \\ 0.5 \\ 2.0 \\ 2.5 \end{pmatrix}$$

Now, the goal is to find the weight vector $\mathbf{w} = \begin{pmatrix} w_0\ w_1\ w_2 \end{pmatrix}^T$ that minimizes the sum of squared errors. We can do it using the pseudo-inverse:

$$\mathbf{w} = \left(X^T X\right)^{-1} X^T Y$$

$$
\begin{aligned}
\mathbf{w} &= \left(X^T X\right)^{-1} X^T Y \\
&= \left[\begin{pmatrix} 1\,1\,1 \\ 1\,2\,1 \\ 1\,1\,3 \\ 1\,3\,3 \end{pmatrix}^T \begin{pmatrix} 1\,1\,1 \\ 1\,2\,1 \\ 1\,1\,3 \\ 1\,3\,3 \end{pmatrix}\right]^{-1} \begin{pmatrix} 1\,1\,1 \\ 1\,2\,1 \\ 1\,1\,3 \\ 1\,3\,3 \end{pmatrix}^T \begin{pmatrix} 1.4 \\ 0.5 \\ 2.0 \\ 2.5 \end{pmatrix} \\
&= \left[\begin{pmatrix} 1\,1\,1\,1 \\ 1\,2\,1\,3 \\ 1\,1\,3\,3 \end{pmatrix} \begin{pmatrix} 1\,1\,1 \\ 1\,2\,1 \\ 1\,1\,3 \\ 1\,3\,3 \end{pmatrix}\right]^{-1} \begin{pmatrix} 1\,1\,1\,1 \\ 1\,2\,1\,3 \\ 1\,1\,3\,3 \end{pmatrix} \begin{pmatrix} 1.4 \\ 0.5 \\ 2.0 \\ 2.5 \end{pmatrix} \\
&= \begin{pmatrix} 4 & 7 & 8 \\ 7 & 15 & 15 \\ 8 & 15 & 20 \end{pmatrix}^{-1} \begin{pmatrix} 1\,1\,1\,1 \\ 1\,2\,1\,3 \\ 1\,1\,3\,3 \end{pmatrix} \begin{pmatrix} 1.4 \\ 0.5 \\ 2.0 \\ 2.5 \end{pmatrix} \\
&= \begin{pmatrix} 1.875 & -0.5 & -0.375 \\ -0.5 & 0.4 & -0.1 \\ -0.375 & -0.1 & 0.275 \end{pmatrix} \begin{pmatrix} 1\,1\,1\,1 \\ 1\,2\,1\,3 \\ 1\,1\,3\,3 \end{pmatrix} \begin{pmatrix} 1.4 \\ 0.5 \\ 2.0 \\ 2.5 \end{pmatrix} \\
&= \begin{pmatrix} 1.0 & 0.5 & 0.25 & -0.75 \\ -0.2 & 0.2 & -0.4 & 0.4 \\ -0.2 & -0.3 & 0.35 & 0.15 \end{pmatrix} \begin{pmatrix} 1.4 \\ 0.5 \\ 2.0 \\ 2.5 \end{pmatrix} \\
&= \begin{pmatrix} 0.275 \\ 0.02 \\ 0.645 \end{pmatrix}
\end{aligned}
$$

---

b) Predict the target value for $x_{query} = \begin{pmatrix} 2\,3 \end{pmatrix}^T$.

Solution:
From the previous question, we have our weights:

$$\mathbf{w} = \begin{pmatrix} 0.275 \\ 0.02 \\ 0.645 \end{pmatrix}$$

So, to compute the predicted value, we just need to augment the query vector with a bias dimension and apply the linear regression:

$$output\,(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} = \begin{pmatrix} 0.275 \\ 0.02 \\ 0.645 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = 2.25$$

---

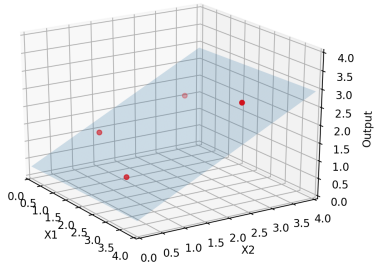
c) Sketch the predicted hyperplane along which the linear regression predicts points will fall.

Solution:

We can get the hyperplane's equation by taking the linear regression output for a general input $\begin{pmatrix} 1 & x_1 & x_2 \end{pmatrix}^T$ and equating it to zero:

$$\begin{aligned} \mathit{output}\,(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} &= 0 \\ = \begin{pmatrix} 0.275 \\ 0.02 \\ 0.645 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix} &= 0 \\ = 0.02x_1 + 0.645x_1 + 0.275 &= 0 \end{aligned}$$

From the equation, we get the following plot:



---

d) Compute the mean squared error produced by the the linear regression.

Solution:

For each point in the training data, we must compute the linear regression prediction and then compute its squared error:

$$\left(t^{(1)} - \mathit{output}\left(\mathbf{x}^{(1)}\right)\right)^2 = \left(t^{(1)} - \mathbf{w} \cdot \mathbf{x}^{(1)}\right)^2 = \left(1.4 - \begin{pmatrix} 0.275 \\ 0.02 \\ 0.645 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}\right)^2 = (1.4 - 0.94)^2 = 0.2116$$

$$\left(t^{(2)} - \mathit{output}\left(\mathbf{x}^{(2)}\right)\right)^2 = \left(t^{(2)} - \mathbf{w} \cdot \mathbf{x}^{(2)}\right)^2 = \left(0.5 - \begin{pmatrix} 0.275 \\ 0.02 \\ 0.645 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}\right)^2 = (0.5 - 0.96)^2 = 0.2116$$

$$\left(t^{(3)} - \mathit{output}\left(\mathbf{x}^{(3)}\right)\right)^2 = \left(t^{(3)} - \mathbf{w} \cdot \mathbf{x}^{(3)}\right)^2 = \left(2.0 - \begin{pmatrix} 0.275 \\ 0.02 \\ 0.645 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix}\right)^2 = (2.0 - 2.23)^2 = 0.0529$$

$$\left(t^{(4)} - output\left(\mathbf{x}^{(4)}\right)\right)^2 = \left(t^{(4)} - \mathbf{w}\cdot\mathbf{x}^{(4)}\right)^2 = \left(2.5 - \begin{pmatrix}0.275\\0.02\\0.645\end{pmatrix}\cdot\begin{pmatrix}1\\3\\3\end{pmatrix}\right)^2 = (2.5-2.27)^2 = 0.0529$$

So, the mean error is:

$$\frac{0.2116 + 0.2116 + 0.0529 + 0.0529}{4} = 0.13225$$

---

2) Consider the following training data:

$$\left\{\mathbf{x}^{(1)} = (-2.0), \mathbf{x}^{(2)} = (-1.0), \mathbf{x}^{(3)} = (0.0), \mathbf{x}^{(4)} = (2.0)\right\}$$

$$\left\{t^{(1)} = 2.0, t^{(2)} = 3.0, t^{(3)} = 1.0, t^{(4)} = -1.0\right\}$$

a) Find the closed form solution for a linear regression that minimizes the sum of squared errors on the training data..

Solution:

First, we need to build the $n \times (d+1)$ design matrix to account for the bias parameter, where n is the number of examples and d is the original number of input features.

$$X = \begin{pmatrix}1 & -2.0\\1 & -1.0\\1 & 0.0\\1 & 2.0\end{pmatrix}$$

Second, we construct a target vector:

$$Y = \begin{pmatrix}2.0\\3.0\\1.0\\-1.0\end{pmatrix}$$

Now, the goal is to find the weight vector $\mathbf{w} = \begin{pmatrix}w_0 & w_1\end{pmatrix}^T$ that minimizes the sum of squared errors. We can do it using the pseudo-inverse:

$$\mathbf{w} = \left(X^T X\right)^{-1} X^T Y$$

$$
\begin{aligned}
\mathbf{w} &= \left(X^T X\right)^{-1} X^T Y \\
&= \left[\begin{pmatrix} 1 & -2.0 \\ 1 & -1.0 \\ 1 & 0.0 \\ 1 & 2.0 \end{pmatrix}^T \begin{pmatrix} 1 & -2.0 \\ 1 & -1.0 \\ 1 & 0.0 \\ 1 & 2.0 \end{pmatrix}\right]^{-1} \begin{pmatrix} 1 & -2.0 \\ 1 & -1.0 \\ 1 & 0.0 \\ 1 & 2.0 \end{pmatrix}^T \begin{pmatrix} 2.0 \\ 3.0 \\ 1.0 \\ -1.0 \end{pmatrix} \\
&= \left[\begin{pmatrix} 1 & 1 & 1 & 1 \\ -2.0 & -1.0 & 0.0 & 2.0 \end{pmatrix} \begin{pmatrix} 1 & -2.0 \\ 1 & -1.0 \\ 1 & 0.0 \\ 1 & 2.0 \end{pmatrix}\right]^{-1} \begin{pmatrix} 1 & 1 & 1 & 1 \\ -2.0 & -1.0 & 0.0 & 2.0 \end{pmatrix} \begin{pmatrix} 2.0 \\ 3.0 \\ 1.0 \\ -1.0 \end{pmatrix} \\
&= \begin{pmatrix} 4.0 & -1.0 \\ -1.0 & 9.0 \end{pmatrix}^{-1} \begin{pmatrix} 1 & 1 & 1 & 1 \\ -2.0 & -1.0 & 0.0 & 2.0 \end{pmatrix} \begin{pmatrix} 2.0 \\ 3.0 \\ 1.0 \\ -1.0 \end{pmatrix} \\
&= \begin{pmatrix} 0.2571 & 0.0286 \\ 0.0286 & 0.1143 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ -2.0 & -1.0 & 0.0 & 2.0 \end{pmatrix} \begin{pmatrix} 2.0 \\ 3.0 \\ 1.0 \\ -1.0 \end{pmatrix} \\
&= \begin{pmatrix} 0.2 & 0.2286 & 0.2571 & 0.3143 \\ -0.2 & -0.0857 & 0.0286 & 0.2571 \end{pmatrix} \begin{pmatrix} 2.0 \\ 3.0 \\ 1.0 \\ -1.0 \end{pmatrix} \\
&= \begin{pmatrix} 1.0286 \\ -0.8857 \end{pmatrix}
\end{aligned}
$$

---

b) Predict the target value for $x_{query} = (1)^T$.

Solution:
From the previous question, we have our weights:

$$\mathbf{w} = \begin{pmatrix} 1.0286 \\ -0.8857 \end{pmatrix}$$

So, to compute the predicted value, we just need to augment the query vector with a bias dimension and apply the linear regression:

$$output\left(\mathbf{x}\right) = \mathbf{w} \cdot \mathbf{x} = \begin{pmatrix} 1.0286 \\ -0.8857 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = 0.1429$$

---

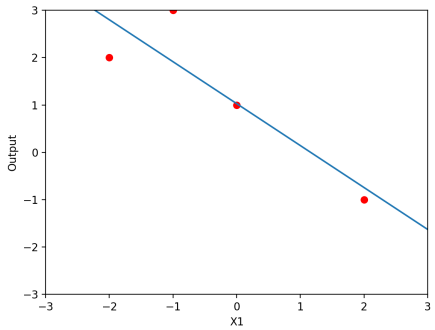
c) Sketch the predicted hyperplane along which the linear regression predicts points will fall.

Solution:

We can get the hyperplane's equation by taking the linear regression output for a general input $\begin{pmatrix} 1 & x_1 \end{pmatrix}^T$ and equating it to zero:

$$\begin{aligned} \textit{output}\,(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} &= 0 \\ = \begin{pmatrix} 1.0286 \\ -0.8857 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ x_1 \end{pmatrix} &= 0 \\ = -0.8857 x_1 + 1.0286 &= 0 \end{aligned}$$

From the equation, we get the following plot:



---

d) Compute the mean squared error produced by the the linear regression.

Solution:

For each point in the training data, we must compute the linear regression prediction and then compute its squared error:

$$\left(t^{(1)} - \mathbf{w} \cdot \mathbf{x}^{(1)}\right)^2 = \left(2.0 - \begin{pmatrix} 1.0286 \\ -0.8857 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ -2.0 \end{pmatrix}\right)^2 = (2.0 - 2.800)^2 = 0.64$$

$$\left(t^{(2)} - \mathbf{w} \cdot \mathbf{x}^{(2)}\right)^2 = \left(3.0 - \begin{pmatrix} 1.0286 \\ -0.8857 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ -1.0 \end{pmatrix}\right)^2 = (3.0 - 1.9143)^2 = 1.1788$$

$$\left(t^{(3)} - \mathbf{w} \cdot \mathbf{x}^{(3)}\right)^2 = \left(1.0 - \begin{pmatrix} 1.0286 \\ -0.8857 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0.0 \end{pmatrix}\right)^2 = (1.0 - 1.0286)^2 = 0.0008$$

$$\left(t^{(4)} - \mathbf{w} \cdot \mathbf{x}^{(4)}\right)^2 = \left(-1.0 - \begin{pmatrix} 1.0286 \\ -0.8857 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2.0 \end{pmatrix}\right)^2 = (-1.0 - (-0.7429))^2 = 0.0661$$

So, the mean error is:

$$\frac{0.64 + 1.1788 + 0.0008 + 0.0661}{4} = 0.4714$$

---

3) Consider the following training data:

$$\left\{\mathbf{x}^{(1)} = \begin{pmatrix}1\\1\end{pmatrix}, \mathbf{x}^{(2)} = \begin{pmatrix}2\\1\end{pmatrix}, \mathbf{x}^{(3)} = \begin{pmatrix}1\\3\end{pmatrix}, \mathbf{x}^{(4)} = \begin{pmatrix}3\\3\end{pmatrix}\right\}$$

$$\left\{t^{(1)} = 1, t^{(2)} = 1, t^{(3)} = 0, t^{(4)} = 0\right\}$$

a) Find the closed form solution for a linear regression that minimizes the sum of squared errors on the training data..

Solution:

First, we need to build the $n \times (d+1)$ design matrix to account for the bias parameter, where n is the number of examples and d is the original number of input features.

$$X = \begin{pmatrix}1\ 1\ 1\\1\ 2\ 1\\1\ 1\ 3\\1\ 3\ 3\end{pmatrix}$$

Second, we construct a target vector:

$$Y = \begin{pmatrix}1\\1\\0\\0\end{pmatrix}$$

Now, the goal is to find the weight vector $\mathbf{w} = \begin{pmatrix}w_0\ w_1\ w_2\end{pmatrix}^T$ that minimizes the sum of squared errors. We can do it using the pseudo-inverse:

$$\mathbf{w} = \left(X^T X\right)^{-1} X^T Y$$

$$
\begin{aligned}
\mathbf{w} &= \left(X^T X\right)^{-1} X^T Y \\
&= \left[\begin{pmatrix}1&1&1\\1&2&1\\1&1&3\\1&3&3\end{pmatrix}^T \begin{pmatrix}1&1&1\\1&2&1\\1&1&3\\1&3&3\end{pmatrix}\right]^{-1} \begin{pmatrix}1&1&1\\1&2&1\\1&1&3\\1&3&3\end{pmatrix}^T \begin{pmatrix}1\\1\\0\\0\end{pmatrix} \\
&= \left[\begin{pmatrix}1&1&1&1\\1&2&1&3\\1&1&3&3\end{pmatrix} \begin{pmatrix}1&1&1\\1&2&1\\1&1&3\\1&3&3\end{pmatrix}\right]^{-1} \begin{pmatrix}1&1&1&1\\1&2&1&3\\1&1&3&3\end{pmatrix} \begin{pmatrix}1\\1\\0\\0\end{pmatrix} \\
&= \begin{pmatrix}4&7&8\\7&15&15\\8&15&20\end{pmatrix}^{-1} \begin{pmatrix}1&1&1&1\\1&2&1&3\\1&1&3&3\end{pmatrix} \begin{pmatrix}1\\1\\0\\0\end{pmatrix} \\
&= \begin{pmatrix}1.875&-0.5&-0.375\\-0.5&0.4&-0.1\\-0.375&-0.1&0.275\end{pmatrix} \begin{pmatrix}1&1&1&1\\1&2&1&3\\1&1&3&3\end{pmatrix} \begin{pmatrix}1\\1\\0\\0\end{pmatrix} \\
&= \begin{pmatrix}1.0&0.5&0.25&-0.75\\-0.2&0.2&-0.4&0.4\\-0.2&-0.3&0.35&0.15\end{pmatrix} \begin{pmatrix}1\\1\\0\\0\end{pmatrix} \\
&= \begin{pmatrix}1.5\\0.0\\-0.5\end{pmatrix}
\end{aligned}
$$

---

b) Use your linear regression to classify $x_{query} = \begin{pmatrix}2&2.5\end{pmatrix}^T$, assuming a threshold similarity of 0.5.

Solution:
From the previous question, we have our weights:

$$\mathbf{w} = \begin{pmatrix}1.5\\0.0\\-0.5\end{pmatrix}$$

So, to compute the predicted value, we just need to augment the query vector with a bias dimension and apply the linear regression:

$$output\left(\mathbf{x}_{\text{query}}\right) = \mathbf{w} \cdot x_{\text{query}} = \begin{pmatrix}1.5\\0.0\\-0.5\end{pmatrix} \cdot \begin{pmatrix}1\\2\\2.5\end{pmatrix} = 0.25$$

The input is classified as not belonging to the class.

---

4) Consider the following training data:

$$\left\{\mathbf{x}^{(1)} = (-2.0)\,, \mathbf{x}^{(2)} = (-1.0)\,, \mathbf{x}^{(3)} = (0.0)\,, \mathbf{x}^{(4)} = (2.0)\right\}$$

$$\left\{t^{(1)} = 1, t^{(2)} = 0, t^{(3)} = 0, t^{(4)} = 0\right\}$$

a) Find the closed form solution for a linear regression that minimizes the sum of squared errors on the training data..

Solution:

First, we need to build the $n \times (d+1)$ design matrix to account for the bias parameter, where n is the number of examples and d is the original number of input features.

$$X = \begin{pmatrix} 1 & -2.0 \\ 1 & -1.0 \\ 1 & 0.0 \\ 1 & 2.0 \end{pmatrix}$$

Second, we construct a target vector:

$$Y = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Now, the goal is to find the weight vector $\mathbf{w} = \begin{pmatrix} w_0 & w_1 \end{pmatrix}^T$ that minimizes the sum of squared errors. We can do it using the pseudo-inverse:

$$\mathbf{w} = \left(X^T X\right)^{-1} X^T Y$$

$$
\begin{aligned}
\mathbf{w} &= \left(X^T X\right)^{-1} X^T Y \\
&= \left[\begin{pmatrix} 1 & -2.0 \\ 1 & -1.0 \\ 1 & 0.0 \\ 1 & 2.0 \end{pmatrix}^T \begin{pmatrix} 1 & -2.0 \\ 1 & -1.0 \\ 1 & 0.0 \\ 1 & 2.0 \end{pmatrix} \right]^{-1} \begin{pmatrix} 1 & -2.0 \\ 1 & -1.0 \\ 1 & 0.0 \\ 1 & 2.0 \end{pmatrix}^T \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \\
&= \left[\begin{pmatrix} 1 & 1 & 1 & 1 \\ -2.0 & -1.0 & 0.0 & 2.0 \end{pmatrix} \begin{pmatrix} 1 & -2.0 \\ 1 & -1.0 \\ 1 & 0.0 \\ 1 & 2.0 \end{pmatrix} \right]^{-1} \begin{pmatrix} 1 & 1 & 1 & 1 \\ -2.0 & -1.0 & 0.0 & 2.0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \\
&= \begin{pmatrix} 4.0 & -1.0 \\ -1.0 & 9.0 \end{pmatrix}^{-1} \begin{pmatrix} 1 & 1 & 1 & 1 \\ -2.0 & -1.0 & 0.0 & 2.0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \\
&= \begin{pmatrix} 0.2571 & 0.0286 \\ 0.0286 & 0.1143 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ -2.0 & -1.0 & 0.0 & 2.0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \\
&= \begin{pmatrix} 0.2 & 0.2286 & 0.2571 & 0.3143 \\ -0.2 & -0.0857 & 0.0286 & 0.2571 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \\
&= \begin{pmatrix} 0.2 \\ -0.2 \end{pmatrix}
\end{aligned}
$$

---

b) Use your linear regression to classify $x_{query} = (-0.3)^T$, assuming a threshold similarity of 0.15.

Solution:
From the previous question, we have our weights:

$$\mathbf{w} = \begin{pmatrix} 0.2 \\ -0.2 \end{pmatrix}$$

So, to compute the predicted value, we just need to augment the query vector with a bias dimension and apply the linear regression:

$$output\left(\mathbf{x}_{\text{query}}\right) = \mathbf{w} \cdot \mathbf{x}_{\text{query}} = \begin{pmatrix} 0.2 \\ -0.2 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ -0.3 \end{pmatrix} = 0.26$$

The input is classified as belonging to the class.

---

5) Consider the following training data:

$$\mathbf{x}^{(1)} = \begin{pmatrix} -0.95 \\ 0.62 \end{pmatrix}, \mathbf{x}^{(2)} = \begin{pmatrix} 0.63 \\ 0.31 \end{pmatrix}, \mathbf{x}^{(3)} = \begin{pmatrix} -0.12 \\ -0.21 \end{pmatrix}, \mathbf{x}^{(4)} = \begin{pmatrix} -0.24 \\ -0.5 \end{pmatrix},$$
$$\mathbf{x}^{(5)} = \begin{pmatrix} 0.07 \\ -0.42 \end{pmatrix}, \mathbf{x}^{(6)} = \begin{pmatrix} 0.03 \\ 0.91 \end{pmatrix}, \mathbf{x}^{(7)} = \begin{pmatrix} 0.05 \\ 0.09 \end{pmatrix}, \mathbf{x}^{(8)} = \begin{pmatrix} -0.83 \\ 0.22 \end{pmatrix}$$

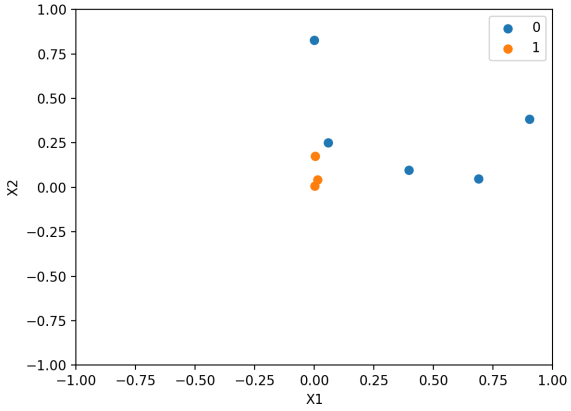
$$\left\{ t^{(1)} = 0, t^{(2)} = 0, t^{(3)} = 1, t^{(4)} = 0, t^{(5)} = 1, t^{(6)} = 0, t^{(7)} = 1, t^{(8)} = 0 \right\}$$

a) Plot the data points and try to choose a non-linear transformation to apply.

Solution:

Plotting the data points we see that the labels seem to change with the distance from the origin. A way to capture is this is to perform a quadratic feature transform:

$$\phi\left(x_1, x_2\right) = \left(x_1^2, x_2^2\right)$$



---

b) Adopt the non-linear transform you chose in a) and find the closed form solution.

Solution:

First, we need to build the $n \times (d+1)$ design matrix to account for the bias parameter, where n is the number of examples and d is the original number of input features. However, unlike previous exercises where we used the features themselves directly ($\phi\left(x\right) = x$), in this case, we have a non-linear transformation. So, we apply it:

$$\Phi = \begin{pmatrix} 1 & (-0.95)^2 & (0.62)^2 \\ 1 & (0.63)^2 & (0.31)^2 \\ 1 & (-0.12)^2 & (-0.21)^2 \\ 1 & (-0.24)^2 & (-0.5)^2 \\ 1 & (0.07)^2 & (-0.42)^2 \\ 1 & (0.03)^2 & (0.91)^2 \\ 1 & (0.05)^2 & (0.09)^2 \\ 1 & (-0.83)^2 & (0.22)^2 \end{pmatrix} = \begin{pmatrix} 1 & 0.9025 & 0.3844 \\ 1 & 0.3969 & 0.0961 \\ 1 & 0.0144 & 0.0441 \\ 1 & 0.0576 & 0.2500 \\ 1 & 0.0049 & 0.1764 \\ 1 & 0.0009 & 0.8281 \\ 1 & 0.0025 & 0.0081 \\ 1 & 0.6889 & 0.0484 \end{pmatrix}$$

Second, we construct a target vector:

$$Y = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

Now, the goal is to find the weight vector $\mathbf{w} = \begin{pmatrix} w_0 & w_1 & w_2 \end{pmatrix}^T$ that minimizes the sum of squared errors. We can do it using the pseudo-inverse:

$$\mathbf{w} = \left(X^T X\right)^{-1} X^T Y$$

$$\mathbf{w} = \left(X^T X\right)^{-1} X^T Y$$

$$= \left[\begin{pmatrix} 1 & 0.9025 & 0.3844 \\ 1 & 0.3969 & 0.0961 \\ 1 & 0.0144 & 0.0441 \\ 1 & 0.0576 & 0.2500 \\ 1 & 0.0049 & 0.1764 \\ 1 & 0.0009 & 0.8281 \\ 1 & 0.0025 & 0.0081 \\ 1 & 0.6889 & 0.0484 \end{pmatrix}^T \begin{pmatrix} 1 & 0.9025 & 0.3844 \\ 1 & 0.3969 & 0.0961 \\ 1 & 0.0144 & 0.0441 \\ 1 & 0.0576 & 0.2500 \\ 1 & 0.0049 & 0.1764 \\ 1 & 0.0009 & 0.8281 \\ 1 & 0.0025 & 0.0081 \\ 1 & 0.6889 & 0.0484 \end{pmatrix}\right]^{-1} \begin{pmatrix} 1 & 0.9025 & 0.3844 \\ 1 & 0.3969 & 0.0961 \\ 1 & 0.0144 & 0.0441 \\ 1 & 0.0576 & 0.2500 \\ 1 & 0.0049 & 0.1764 \\ 1 & 0.0009 & 0.8281 \\ 1 & 0.0025 & 0.0081 \\ 1 & 0.6889 & 0.0484 \end{pmatrix}^T \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$= \begin{pmatrix} 8.0000 & 2.0686 & 1.8356 \\ 2.0686 & 1.4502 & 0.4351 \\ 1.8356 & 0.4351 & 0.9407 \end{pmatrix}^{-1} \begin{pmatrix} 1 & 0.9025 & 0.3844 \\ 1 & 0.3969 & 0.0961 \\ 1 & 0.0144 & 0.0441 \\ 1 & 0.0576 & 0.2500 \\ 1 & 0.0049 & 0.1764 \\ 1 & 0.0009 & 0.8281 \\ 1 & 0.0025 & 0.0081 \\ 1 & 0.6889 & 0.0484 \end{pmatrix}^T \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$= \begin{pmatrix} 0.3099 & -0.3026 & -0.4647 \\ -0.3026 & 1.0962 & 0.0835 \\ -0.4647 & 0.0835 & 1.9311 \end{pmatrix} \begin{pmatrix} 1 & 0.9025 & 0.3844 \\ 1 & 0.3969 & 0.0961 \\ 1 & 0.0144 & 0.0441 \\ 1 & 0.0576 & 0.2500 \\ 1 & 0.0049 & 0.1764 \\ 1 & 0.0009 & 0.8281 \\ 1 & 0.0025 & 0.0081 \\ 1 & 0.6889 & 0.0484 \end{pmatrix}^T \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$= \begin{pmatrix} -0.1419 & 0.1451 & 0.2850 & 0.1763 & 0.2264 & -0.0752 & 0.3053 & 0.0789 \\ 0.7188 & 0.1405 & -0.2831 & -0.2186 & -0.2825 & -0.2325 & -0.2992 & 0.4566 \\ 0.3529 & -0.2459 & -0.3783 & 0.0229 & -0.1236 & 1.1346 & -0.4488 & -0.3137 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$= \begin{pmatrix} 0.8168 \\ -0.8648 \\ -0.9508 \end{pmatrix}$$

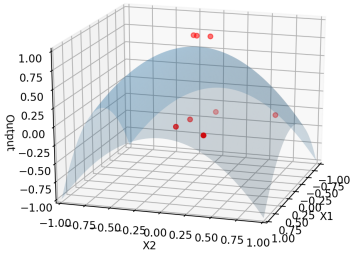
---

c) Sketch the predicted surface along which the predictions will fall.

Solution:

$$
\begin{aligned}
output\left(\mathbf{x}\right) &= \mathbf{w}\cdot\phi\left(\mathbf{x}\right) && = 0\\
&= \begin{pmatrix} 0.8168 \\ -0.8648 \\ -0.9508 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ x_1^2 \\ x_2^2 \end{pmatrix} && = 0\\
&= -0.8648x_1^2 - 0.9508x_2^2 + 0.8168 && = 0
\end{aligned}
$$

From the equation, we get the following plot:



---

6) Consider the following training data:

$$\left\{\mathbf{x}^{(1)} = (3), \mathbf{x}^{(2)} = (4), \mathbf{x}^{(3)} = (6), \mathbf{x}^{(4)} = (10), \mathbf{x}^{(5)} = (12)\right\}$$

$$\left\{t^{(1)} = 1.5, t^{(2)} = 9.3, t^{(3)} = 23.4, t^{(4)} = 45.8, t^{(5)} = 60.1\right\}$$

a) Adopt a logarithmic feature transformation $\phi\left(x_1\right) = \log\left(x_1\right)$ and find the closed form solution for this non-linear regression that minimizes the sum of squared errors on the training data.

Solution:

First, we need to build the $n \times (d+1)$ design matrix to account for the bias parameter, where n is the number of examples and d is the original number of input features. However, unlike previous exercises where we used the features themselves directly ($\phi\left(x\right) = x$), in this case, we have a non-linear transformation. So, we apply it:

$$\Phi = \begin{pmatrix} 1 & \log\left(3\right) \\ 1 & \log\left(4\right) \\ 1 & \log\left(6\right) \\ 1 & \log\left(10\right) \\ 1 & \log\left(12\right) \end{pmatrix} = \begin{pmatrix} 1 & 1.0986 \\ 1 & 1.3863 \\ 1 & 1.7918 \\ 1 & 2.3026 \\ 1 & 2.4849 \end{pmatrix}$$

Second, we construct a target vector:

$$Y = \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix}$$

Now, the goal is to find the weight vector $\mathbf{w} = \begin{pmatrix} w_0 & w_1 \end{pmatrix}^T$ that minimizes the sum of squared errors. We can do it using the pseudo-inverse:

$$\mathbf{w} = \left(X^T X\right)^{-1} X^T Y$$

$$\begin{aligned}
\mathbf{w} &= \left(X^T X\right)^{-1} X^T Y \\
&= \left[\begin{pmatrix} 1 & 1.0986 \\ 1 & 1.3863 \\ 1 & 1.7918 \\ 1 & 2.3026 \\ 1 & 2.4849 \end{pmatrix}^T \begin{pmatrix} 1 & 1.0986 \\ 1 & 1.3863 \\ 1 & 1.7918 \\ 1 & 2.3026 \\ 1 & 2.4849 \end{pmatrix} \right]^{-1} \begin{pmatrix} 1 & 1.0986 \\ 1 & 1.3863 \\ 1 & 1.7918 \\ 1 & 2.3026 \\ 1 & 2.4849 \end{pmatrix}^T \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix} \\
&= \left[\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1.0986 & 1.3863 & 1.7918 & 2.3026 & 2.4849 \end{pmatrix} \begin{pmatrix} 1 & 1.0986 \\ 1 & 1.3863 \\ 1 & 1.7918 \\ 1 & 2.3026 \\ 1 & 2.4849 \end{pmatrix} \right]^{-1} \begin{pmatrix} 1 & 1.0986 \\ 1 & 1.3863 \\ 1 & 1.7918 \\ 1 & 2.3026 \\ 1 & 2.4849 \end{pmatrix}^T \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix} \\
&= \begin{pmatrix} 5.0 & 9.0642 \\ 9.0642 & 17.8158 \end{pmatrix}^{-1} \begin{pmatrix} 1 & 1.0986 \\ 1 & 1.3863 \\ 1 & 1.7918 \\ 1 & 2.3026 \\ 1 & 2.4849 \end{pmatrix}^T \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix} \\
&= \begin{pmatrix} 2.5745 & -1.3098 \\ -1.3098 & 0.7225 \end{pmatrix} \begin{pmatrix} 1 & 1.0986 \\ 1 & 1.3863 \\ 1 & 1.7918 \\ 1 & 2.3026 \\ 1 & 2.4849 \end{pmatrix}^T \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix} \\
&= \begin{pmatrix} 1.1355 & 0.7587 & 0.2276 & -0.4415 & -0.6803 \\ -0.5160 & -0.3082 & -0.0152 & 0.3539 & 0.4856 \end{pmatrix} \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix} \\
&= \begin{pmatrix} -47.0212 \\ 41.3945 \end{pmatrix}
\end{aligned}$$

b) Repeat the exercise above for a quadratic feature transformation $\phi(x_1) = x_1^2$.

Solution:

First, we need to build the $n \times (d+1)$ design matrix to account for the bias parameter, where n is the number of examples and d is the original number of input features. However, unlike previous exercises where we used the features themselves directly ($\phi(x) = x$), in this case, we have a non-linear transformation. So, we apply it:

$$\Phi = \begin{pmatrix} 1 & 3^2 \\ 1 & 4^2 \\ 1 & 6^2 \\ 1 & 10^2 \\ 1 & 12^2 \end{pmatrix} = \begin{pmatrix} 1 & 9 \\ 1 & 16 \\ 1 & 36 \\ 1 & 100 \\ 1 & 144 \end{pmatrix}$$

Second, we construct a target vector:

$$Y = \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix}$$

Now, the goal is to find the weight vector $\mathbf{w} = \begin{pmatrix} w_0 & w_1 \end{pmatrix}^T$ that minimizes the sum of squared errors. We can do it using the pseudo-inverse:

$$\mathbf{w} = \left(X^T X\right)^{-1} X^T Y$$

$$
\begin{aligned}
\mathbf{w} &= \left(X^T X\right)^{-1} X^T Y \\
&= \left[\begin{pmatrix} 1 & 9 \\ 1 & 16 \\ 1 & 36 \\ 1 & 100 \\ 1 & 144 \end{pmatrix}^T \begin{pmatrix} 1 & 9 \\ 1 & 16 \\ 1 & 36 \\ 1 & 100 \\ 1 & 144 \end{pmatrix}\right]^{-1} \begin{pmatrix} 1 & 9 \\ 1 & 16 \\ 1 & 36 \\ 1 & 100 \\ 1 & 144 \end{pmatrix}^T \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix} \\
&= \left[\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 9 & 16 & 36 & 100 & 144 \end{pmatrix} \begin{pmatrix} 1 & 9 \\ 1 & 16 \\ 1 & 36 \\ 1 & 100 \\ 1 & 144 \end{pmatrix}\right]^{-1} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 9 & 16 & 36 & 100 & 144 \end{pmatrix} \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix} \\
&= \begin{pmatrix} 5.0 & 305.0 \\ 305.0 & 32369.0 \end{pmatrix}^{-1} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 9 & 16 & 36 & 100 & 144 \end{pmatrix} \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix} \\
&= \begin{pmatrix} 0.4703 & -0.0044 \\ -0.0044 & 0.00007 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 9 & 16 & 36 & 100 & 144 \end{pmatrix} \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix} \\
&= \begin{pmatrix} 0.4305 & 0.3994 & 0.3108 & 0.0272 & -0.1678 \\ -0.0038 & -0.0033 & -0.0018 & 0.0028 & 0.0060 \end{pmatrix} \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix} \\
&= \begin{pmatrix} 2.7895 \\ 0.4136 \end{pmatrix}
\end{aligned}
$$

---

c) Plot both regressions.

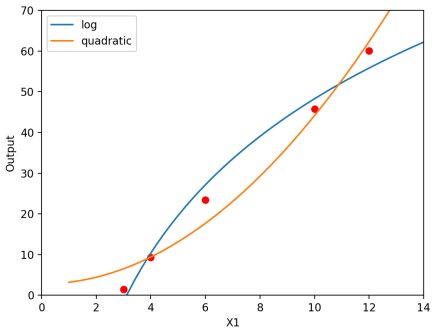
Solution:
For the logarithmic, we get:

$$output\left(\mathbf{x}\right) = w \cdot \phi\left(\mathbf{x}\right) = -47.0212 + 41.395 \log\left(x_1\right)$$

For the quadratic, we get:

$$output\left(\mathbf{x}\right) = w \cdot \phi\left(\mathbf{x}\right) = 2.7895 + 0.4136 x_1^2$$

With the equations, we can build the plots:



d) Which is a better fit a) or b)?

Solution:
Recall the equations for both regressions:

$$output_{log}\left(\mathbf{x}\right) = w \cdot \phi\left(\mathbf{x}\right) = -47.0212 + 41.395 \log\left(x_1\right)$$

$$output_{quadratic}\left(\mathbf{x}\right) = w \cdot \phi\left(\mathbf{x}\right) = 2.7895 + 0.4136 x_1^2$$

To measure which fit is better we will measure the mean squared errors. For the logarithmic transform, we have:

$$\left(\mathbf{\Phi w}_{\log} - \mathbf{Y}\right)^2 = \left(\begin{pmatrix} 1 & 1.0986 \\ 1 & 1.3863 \\ 1 & 1.7918 \\ 1 & 2.3026 \\ 1 & 2.4849 \end{pmatrix} \begin{pmatrix} -47.0212 \\ 41.3945 \end{pmatrix} - \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix}\right)^2 = \begin{pmatrix} 9.2704 \\ 1.1315 \\ 14.0455 \\ 6.2155 \\ 18.1460 \end{pmatrix}$$

Which yields a mean squared error of:

$$\frac{9.2704 + 1.1315 + 14.0455 + 6.2155 + 18.1460}{5} = 9.7618$$

For the quadratic transform, we have:

$$\left(\mathbf{\Phi w}_{\text{quadratic}} - \mathbf{Y}\right)^2 = \left(\begin{pmatrix} 1 & 9 \\ 1 & 16 \\ 1 & 36 \\ 1 & 100 \\ 1 & 144 \end{pmatrix} \begin{pmatrix} 2.7895 \\ 0.4136 \end{pmatrix} - \begin{pmatrix} 1.5 \\ 9.3 \\ 23.4 \\ 45.8 \\ 60.1 \end{pmatrix}\right)^2 = \begin{pmatrix} 25.1202 \\ 0.0152 \\ 32.7228 \\ 2.7192 \\ 5.0628 \end{pmatrix}$$

Which yields a mean squared error of:

$$\frac{25.1202 + 0.0152 + 32.7228 + 2.7192 + 5.0628}{5} = 13.1273$$

So, given the availale data, the logarithmic transform appears to fit the data better.

---

3 Thinking Questions

a) Until now we could only solve classification tasks where the two classes were separated by simple lines. Now we have seen that we can apply any feature transformations we want. Think about which kinds of problems we can solve now? Is it all a matter of finding the right transformation? Is it easy to choose the right transformation?

b) When using the linear regression for classification, think about how the threshold changes the sensitivity of the model. Is it more or less likely that the model will fail to recognize a class member as the threshold increases?