The kNN, or k-nearest neighbors algorithm, is a simple supervised learning algorithm, which works under the assumption that samples with similar features are more likely to share the same label. The algorithm is very straight-forward, although differing between trying to predict categoric and numeric labels: using a given distance metric (such as Euclidean, Manhattan, etc.), the algorithm finds the k closest samples to the sample we want to classify, and assigns the label that:

1. is most common (i.e. the mode) among the k neighbors, in the case of categoric labels;
2. is the average of the labels of the k neighbors, in the case of numeric labels.

1. Considering the following data set:

	y_1	y_2	z_1	z_2
x_1	1	1	A	1.4
x_2	2	1	B	0.5
x_3	2	3	B	2
x_4	3	3	B	2.2
x_5	2	2	A	0.7
x_6	1	2	A	1.2

Assuming kNN, with $k = 3$ applied within a leave-one-out schema:

(a) Considering an z_1 categoric output variable and the Euclidean distance, provide the prediction for x_1.

(b) Considering an z_2 numeric output variable and the cosine similarities, provide the mean regression estimate for x_1.

(c) Considering a weighted-distance kNN, with Manhattan distance, identify both the **weighted-mode** estimate of x_1 for a z_1 outcome and the **weighted-mean** estimate of x_1 for a z_2 outcome.

Here, working with a leave-one-out schema means that, for each sample, we can pick its k neighbors from the pool of all the other samples, excluding itself (which, in theory, is always its closest neighbor, with null distance).

(a) The Euclidean distance is defined as the square root of the sum of the squared differences between the features of the two samples:

$$d(x_i, x_j) = \sqrt{\sum_{l=1}^{p} (x_i^{(l)} - x_j^{(l)})^2}$$

Since we're working with a leave-one-out schema, and trying to estimate the z_1 label for x_1, we can pick the k neighbors from all samples except x_1. Below are illustrated the Euclidean distances between those samples and x_1, with the k closest neighbors highlighted in teal:

$$d(x_1, x_2) = \sqrt{(1-2)^2 + (1-1)^2} = 1$$

$$d(x_1, x_3) = \sqrt{5}, \quad d(x_1, x_4) = 2\sqrt{2}, \quad d(x_1, x_5) = \sqrt{2}, \quad d(x_1, x_6) = 1$$

Knowing the k closest neighbors, we can now estimate the z_1 label for x_1, by picking the most common label among them. In this case, the estimated label will be $\textbf{mode}(B, A, A) = A$.

(b) Here, instead of the usual Euclidean/Manhattan distance, we're using the cosine similarities, which are defined as the cosine of the angle between the two vectors of features. As we know since high school, the cosine of the angle between two vectors is equal to the dot product of the two vectors divided by the product of their norms:

$$\cos(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\|\|\vec{b}\|}$$

Here, each sample is essentially a vector of features, hence the usage of this distance metric. The higher the cosine similarity, the closer the two vectors are, and vice versa: the cosine is equal to 1 when the two vectors are identical, and 0 when they are orthogonal.

Let's now compute the cosine similarities between x_1 and the other samples, picking the k closest neighbors (once again, in teal):

$$\cos(x_1, x_2) = \frac{1 \cdot 2 + 1 \cdot 1}{\sqrt{2}\sqrt{5}} = \frac{3}{\sqrt{10}} = 0.94868, \quad \cos(x_1, x_3) = \frac{1 \cdot 2 + 1 \cdot 3}{\sqrt{2}\sqrt{13}} = \frac{5}{\sqrt{26}} = 0.98058$$

$$\cos(x_1, x_4) = \frac{1 \cdot 3 + 1 \cdot 3}{\sqrt{2}\sqrt{18}} = \frac{6}{\sqrt{36}} = 1, \quad \cos(x_1, x_5) = \frac{1 \cdot 2 + 1 \cdot 2}{\sqrt{2}\sqrt{8}} = \frac{4}{\sqrt{16}} = 1$$

$$\cos(x_1, x_6) = \frac{1 \cdot 1 + 1 \cdot 2}{\sqrt{2}\sqrt{5}} = \frac{3}{\sqrt{10}} = 0.94868$$

Note, as mentioned above, that the closest neighbors here are the ones with a higher cosine similarity: x_3, x_4, and x_5. Since we're working with numeric labels for z_2, we can now estimate the mean regression estimate for x_1 by averaging the values of z_2 for those samples. In this case, the estimated value will be $\textbf{mean}(2, 2.2, 0.7) = 1.6(3)$.

(c) Note that, here, we're working with a weighted-distance kNN, which means that we're assigning different weights to each sample based on its distance from the sample we're trying to estimate - a closer neighbor will have a bigger impact on the final estimate than a further neighbor. Note, also, that we're now working with the Manhattan distance, which is defined as the sum of the absolute differences between the features of the two samples:

$$d(x_i, x_j) = \sum_{l=1}^{p} |x_i^{(l)} - x_j^{(l)}|$$

Let's now compute the Manhattan distances between x_1 and the other samples, picking the k closest neighbors (once again, in teal):

$$d(x_1, x_2) = |1 - 2| + |1 - 1| = 1$$

$$d(x_1, x_3) = 3, \quad d(x_1, x_4) = 4, \quad d(x_1, x_5) = 2, \quad d(x_1, x_6) = 1$$

Now, the fun part begins: we have to correctly weigh each neighbor's label to estimate the label for x_1. We can do this by assigning a weight to each neighbor, based on its distance from x_1. The most common way to do this is by using the inverse of the distance. Regarding the weighted mode estimate of x_1 for z_1:

$$\hat{z}_1 = \textbf{weighted\_mode}({}^{1}/_{1}B, ({}^{1}/_{1} + {}^{1}/_{2})A) = A$$

The mean, of course, will take into account for the denominator the weights of each neighbor, instead of the amount of neighbors:

$$\hat{z}_2 = \textbf{weighted\_mean} = \frac{{}^{1}/_{1} \cdot 0.5 + {}^{1}/_{2} \cdot 0.7 + {}^{1}/_{1} \cdot 1.2}{{}^{1}/_{1} + {}^{1}/_{2} + {}^{1}/_{1}} = 0.82$$

2. Consider the following training data set:

	y_1	y_2	z
x_1	1	1	1.4
x_2	2	1	0.5
x_3	1	3	2
x_4	3	3	2.5

(a) Find the closed form solution for a linear regression, minimizing the sum of squared errors.

(b) Predict the target value for the query vector $x_{new} = \begin{bmatrix} 2 & 3 \end{bmatrix}^T$.

(c) Sketch the predicted three-dimensional hyperplane.

(d) Compute both the MSE and MAE produced by the linear regression.

(e) Are there biases on the residuals against any of the input variables?

(f) Compute the closed form solution, considering Ridge regularization term with $\lambda = 0.2$.

(g) Compare the hyperplanes obtained utilizing ordinary least squares and ridge regression.

(h) Why is the Lasso regression usually preferred over Ridge regression for data spaces with a larger number of features?

In linear regression, we're trying to learn a linear function that maps the input features to the target variable. In other words, we're trying to find a function f such that:

$$f(x) = \sum_{i=1}^{p} w_i x_i + b$$

Here, the **bias**, b, is commonly written as w_0, and the **weights**, w_i, are the coefficients of the linear function - the bias is the intercept of the function, allowing us to effectively shift the function up or down.

Note that we always want to make the best predictions possible, of course: for that purpose, we'll want our function f to be as close as possible to the actual target values. In other words, we want to minimize the error between the predicted values and the actual values. The most common way to do this is by minimizing the sum of squared errors (SSE), which is defined as:

$$\text{SSE} = (XW - Z)^T(XW - Z) = (\hat{Z} - Z)^T(\hat{Z} - Z)$$

Considering that we're trying to minimize the error, we can take the derivative of the SSE with respect to the weights and set it to zero:

$$\frac{\partial \text{SSE}}{\partial w_i} = 0 \leftrightarrow \frac{\partial}{\partial w_i}(\hat{Z} - Z)^T(\hat{Z} - Z) = 0$$

After performing some maniacal algebra, we can find the closed form solution for the weights:

$$W = (X^T X)^{-1} X^T Z$$

(a) We've derived the closed form solution for the weights, above, so most of our work is done here. Considering the training data set, we can say that X and Z are the following matrices (note that X's first column is all ones, letting the neutral element for the bias, w_0, shift the function up or down):

$$X = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 3 \\ 1 & 3 & 3 \end{bmatrix}, \quad Z = \begin{bmatrix} 1.4 \\ 0.5 \\ 2 \\ 2.5 \end{bmatrix}$$

After plugging these matrices into the closed form solution, we gather the following weight vector (`numpy` calculations in the notebook):

$$W = \begin{bmatrix} 0.275 \\ 0.02 \\ 0.645 \end{bmatrix}$$

(b) As we have mentioned in the motivation for this exercise, the predictions, $f(x) = \hat{z}$, are given by the following equation:

$$\hat{z} = \sum_{i=1}^{p} w_i x_i + w_0$$

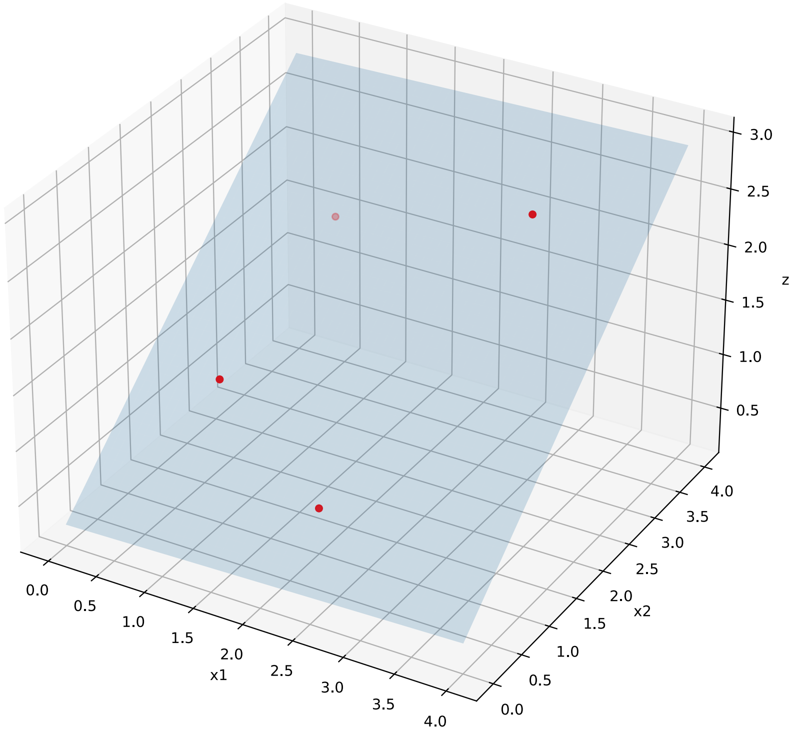
Considering an initial bias of zero (since no bias is given in the question's statement), we can say that:

$$\hat{z} = W^T \cdot x_{new} = \begin{bmatrix} 0.275 \\ 0.02 \\ 0.645 \end{bmatrix}^T \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = 2.25$$

(c) The three-dimensional hyperplane for the linear regression in question is the one defined by the following equation:

$$\hat{z} = \begin{bmatrix} 0.275 \\ 0.02 \\ 0.645 \end{bmatrix}^T \cdot x = 0.275 + 0.02x_1 + 0.645x_2$$

With the aid of Python's `matplotlib` library, we can plot the hyperplane in three dimensions, as shown in the figure below:



(d) We're asked to compute both the MSE (Mean Squared Error) and the MAE (Mean Absolute Error) produced by the linear regression in hands. They're defined as follows:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{z}_i - z_i)^2$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\hat{z}_i - z_i|$$

We'll need, of course, to compute the estimates, $\hat{z}$, for each of the training set's original samples. We can do this by plugging the training set's input vectors into the linear regression's equation:

$$\hat{z} = \begin{bmatrix} 0.275 \\ 0.02 \\ 0.645 \end{bmatrix}^T \cdot x$$

The obtained estimates are the following:

$$\hat{z} = \begin{bmatrix} 0.94 \\ 0.96 \\ 2.23 \\ 2.27 \end{bmatrix}$$

With these estimates, we can compute the MSE and MAE:

$$\mathbf{MSE} = \frac{1}{4}\sum_{i=1}^{4}(\hat{z}_i - z_i)^2 = \frac{1}{4}\left((0.94 - 1.4)^2 + \cdots + (2.27 - 2.5)^2\right) = 0.0.13225$$

$$\mathbf{MAE} = \frac{1}{4}\sum_{i=1}^{4}|\hat{z}_i - z_i| = \frac{1}{4}\left(|0.94 - 1.4| + \cdots + |2.27 - 2.5|\right) = 0.345$$

(e) TODO

(f) The Ridge regularization technique aims to **tune** a linear regression model, by adding a penalty term to the SSE. This penalty, usually denoted by λ, changes the closed form solution for the weights, W, to the following:

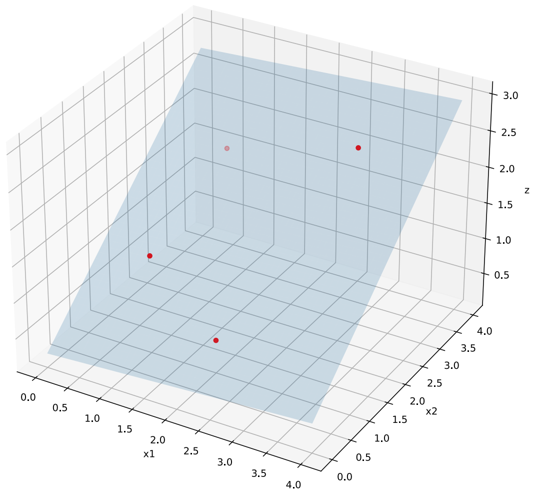
$$W = (X^T X + \lambda I)^{-1} X^T Z$$

where I is the identity matrix. The regularization term, λ, is a hyperparameter that controls the amount of regularization applied to the model, in order to avoid overfitting. The larger the value of λ, the more the regression will shift from its original solution.

With $\lambda = 0.2$, as given by the question's statement, we can compute the new weights, W, as follows:

$$W = (X^T X + 0.2I)^{-1} X^T Z = \begin{bmatrix} 0.238123 \\ 0.0522205 \\ 0.629293 \end{bmatrix}$$

(g) The Ridge regression's hyperplane is the one shown in the figure below:



The norm of the vector describing the model's hyperplane is, as expected, smaller than the one obtained in the previous question. This is due to the regularization term, which aims precisely to reduce the model's weights' magnitude.

(h) LASSO, or Least Absolute Shrinkage and Selection Operator, is another regularization technique that aims to tune a linear regression model. It differs from Ridge regression in that it adds a penalty term to the SSE, but this term is the sum of the absolute values of the weights, instead of the sum of their squares. With LASSO, the coefficients are shrunk towards zero, which means that some of them will be reduced to zero, effectively removing them from the model - this is a so-called **feature selection**, since it'll happen to the least relevant features of the model. Ridge regression, on the other hand, will only reduce the weights' magnitude, but will never set them to zero. This way, LASSO is more likely to produce a sparse (and better) model, with fewer features, than Ridge regression, regarding data sets with a large number of features.

3. Considering the following training data, with z as an ordinal variable:

	y_1	y_2	z
x_1	1	1	1
x_2	2	1	1
x_3	1	3	0
x_4	3	3	0

(a) Find a linear regression using the closed form solution.

(b) Assuming an output threshold $\theta = 0.5$, provide the predicted class for $x_{new} = \begin{bmatrix} 2 & 2.5 \end{bmatrix}^T$.

(a) We already know that the closed form solution of a linear regression is given by:

$$W = (X^T X)^{-1} X^T Z$$

Looking at the data set, we can already write both X and Z:

$$X = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 3 \\ 1 & 3 & 3 \end{bmatrix}, \quad Z = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Plugging them into the closed form solution, we obtain:

$$W = (X^T X)^{-1} X^T Z = \begin{bmatrix} 1.5 \\ -2.22045e - 16 \\ -0.5 \end{bmatrix}$$

(b) Output thresholds are used to convert the output of a linear regression model into a class label - considering binary labels, the threshold θ is used to determine whether the output is 0 or 1 (considering whether or not $\hat{z}$ is greater than θ). In this case, we can compute the output of the model for the new sample, x_{new}, as follows:

$$\hat{z} = W^T \cdot x_{new} = \begin{bmatrix} 1.5 \\ -2.22045e - 16 \\ -0.5 \end{bmatrix}^T \cdot \begin{bmatrix} 1 \\ 2 \\ 2.5 \end{bmatrix} = 0.25$$

Since this estimation does not surpass the defined threshold, $\theta = 0.5$, the predicted class for x_{new} is 0.

4. Considering the data below to learn the following model, compare:

$$z = w_1 y_1 + w_2 y_2 + \epsilon, \epsilon \sim \mathcal{N}(0, 0.1)$$

	y_1	y_2	z
x_1	3	-1	2
x_2	4	2	1
x_3	2	2	1

(a) $w = \begin{bmatrix} w_1 & w_2 \end{bmatrix}^T$, using an MLE approach.

(b) w using the Bayesian approach, assuming $p(w) = N(w \mid \mu = [0,0], \Sigma = \begin{bmatrix} 0.2 & 0 \\ 0 & 0.2 \end{bmatrix})$.

(a)

(b)

5. Identify a transformation to aid linearly modelling the data set above. Sketch the predicted surface.

6. Consider both logarithmic and quadratic transformations:

$$\phi_1(x_1) = \log(x_1), \quad \phi_2(x_2) = x_2^2$$

(a) Plot both of the closed form regressions.

(b) Which transformation minimizes the sum of squared errors on the original data?

(a)

(b)

7. Select the criteria promoting a smoother regression model:

(a) Applying Ridge and Lasso regularizations to linear regression models.

(b) Increasing the depth of a decision tree regressor.

(c) Increasing the k parameter of a kNN regressor.

(d) Parametrizing a kNN regressor with uniform weights, instead of the default distance-based weights.