Topics in Computer Science Spring 2025
COS 470 Lecture 4

Objectives

- K-Nearest Neighbors (KNN)
- Regression
- Scale Data
- "Homework" Problems
- Outline for KNN algorithm
- Classifier Function
- Class Code

1 K-Nearest Neighbors (KNN)

KNN is an algorithm that can be use for Regression and Classification tasks. Its function is to classify new data points by checking what is nearby them as its names indicate.

This is how the data looks:
$Data = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$

They look like a set of pairs where x represent the features, while y is the label that belongs to x.

2 Regression Case

2.1 Example of feature representation for a regression case

Predicting House Prices

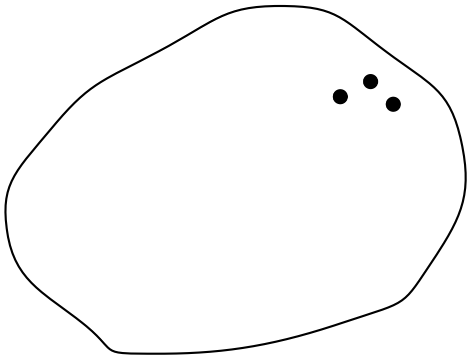
$$x = \begin{bmatrix} \text{Sq. ft} \\ \text{Dist to Ocean} \\ \text{Dist to School} \\ \vdots \end{bmatrix}$$

$$y = [325,000]$$

Here the input feature x consists of multiple attributes related to the house, like the square feet, the distance to the ocean, and the distance to school. The target variable y represents the price of the home.

2.2 Example of use case for regression case

For missing values on temperature for a particular city, instead of taking the average of every city in the country, we consider the average temperature of the 3 nearest cities. Let say we are trying to find the temperature of a city in New Hampshire.



Let's assume this dissorted circle is the map of The United States, and we are trying to find the temperature of a city in New Hampshire. The three dots inside the circle represent other cities near New Hampshire. We take the temperature of those 3 dots (cities), and then we average them to find the missing temperature from the city in New Hampshire. That's one way of doing it.

2.3 Dot Product and Cross-Validation in Regression

Understanding Dot Product and Cross-Validation for House Price Prediction

The dot product is a fundamental operation in linear regression, where the input feature vector x is multiplied by the learned weight vector w to predict the target y. Mathematically, this can be written as:

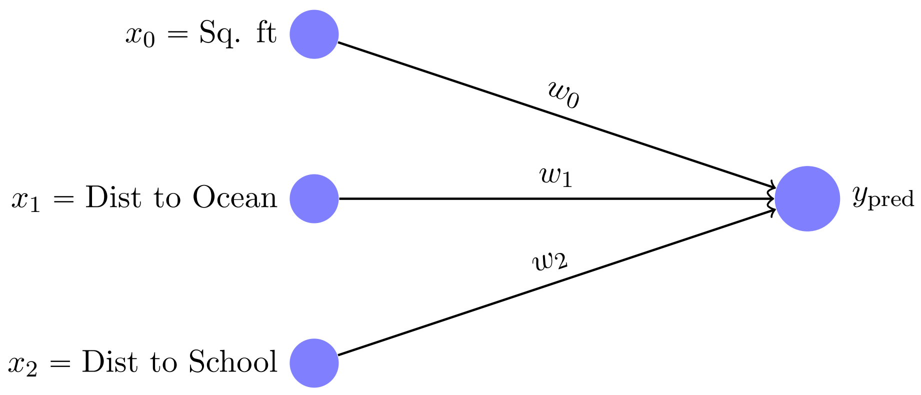
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \text{Sq. ft} \\ \text{Dist to Ocean} \\ \text{Dist to School} \end{bmatrix}, \quad w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$$

The predicted house price is computed using the dot product:

$$
\begin{aligned}
y_{\text{pred}} &= w \cdot x \\
&= \sum_{i=0}^{2} w_i x_i
\end{aligned}
$$

where: - w_0, w_1, w_2 are learned weights, - y_{pred} is the predicted house price.

This operation essentially performs a **weighted sum of the input features** to generate the prediction.



Cross-Validation for Model Generalization

To ensure that our model generalizes well to unseen data, we use **k-fold cross-validation**, which consists of the following steps:

1. Split the dataset into k subsets (folds).
2. Train the model on $k-1$ folds while leaving one fold for validation.
3. Repeat this process k times, using a different fold for validation each time.
4. Average the performance across all k runs to obtain a reliable estimate of model accuracy.

3 Preprocessing

3.1 Scale Data

Scaling data is important in machine learning algorithm to maintain consistency. Some features might have large values while other have small, meaning that models will be more biased to the larger values. So, when they are in the same range, it is easier for models to compare the data impact on the result. For example, when dealing with distances metrics, 1000 miles compared to 50 fts, the 1000 miles will completely dominated the distance calculation, so, it is always necessary to scale this type of data.

3.2 Scaling Techniques

1. **Standarize**

$$x_i' = \frac{x_i - \bar{x}}{\sigma} = Z$$

- x_i': the new data point
- x_i: the old data point
- σ: the standard deviation
- $\bar{x}$: mean of entire column of values
- Z: Z-score

Where:

$$x_i = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_p \end{bmatrix}$$

For this standarize formula, we take the original values (x_i), subtract it with the mean of the entire column ($\bar{x}$), to later divide it with the standard deviation (σ), we would get the Z-score. This is one way of scaling data.

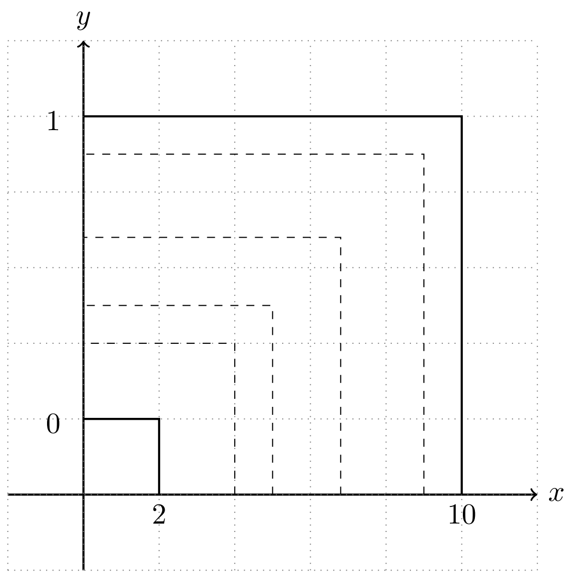
2. **Normalize (min max scaler)**

For this scaling technique, we take the values of x, and then map them to the interval from 0 to 1:

$$x \rightarrow [0, 1]$$

For example, we have values that ranges from 2 to 10, then we would want to map 2 to 0, and 10 to 1.

$$[2, 10] \rightarrow [0, 1]$$



As we can see in the graph, we mapped 2 to 0, and 10 to 1, and they can be denoted by the solid lines, while the dashed lines represent a number in between 2 and 10 that connect to a value in between 0 and 1.

4 Exploration (XPL) Problems

1. Code the Standarize Scaler formula.

2. Write down the formula for linear transformation, and then code it. e.g.$[2, 10] \rightarrow [0, 1]$ and anything in between.

3. Write a linear transformation that maps $[a, b]$ to $[c, d]$ (draw a line for them).

5 Outline for the KNN algorithm

- Given a dataset: $D = \{x_i, y_i\}$
- Compute the distance from new point x_t that is not in the set D, to every x_i
- Sort distances
- Select first K of the sorted distances
- If **Classification** problem, take the mode of labels $y_1, y_2, y_3 ... y_k$, else, for **Regression** problem, take the mean of labels $y \in \mathbb{R} \rightarrow y_1, y_2, y_3 ... y_k$ (as an alternative, using median could be ideal in some cases since it ignore outliers)

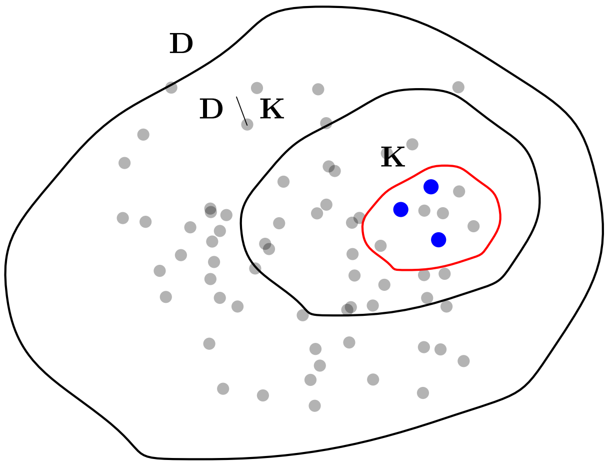
x_t is a test point

$K \subseteq D$

$|K| = k$, where $|K|$ represent the number of elements in that subset

$\forall (x', y') \in D \setminus K$, for every point that is outside of K, the distance from the test point to all the points outside of K have to be bigger than the maximum distance inside of K.

$$d(x', y') \geqslant max\ d\ (x_t, x'')$$



In the graph shown above, D represents the whole data points, $D \setminus K$ represent the points outside of K, and K contains its own points that is also part of D. There are three blue dots (K=3) inside of K surrounded by a red circle. This three dots are the 3 nearest neighbors used for classification or regression task in KNN. So, given a new test point, KNN calculates the distance between this new test point, and every point in the dataset. It then selects the K closest point as its neighbors (K=3 in this case), and then, it would either assign a class or predict a value depending of the task to that new data point by comparing the proximity of the data point to the nearest neighbor.

6 Classifier Function

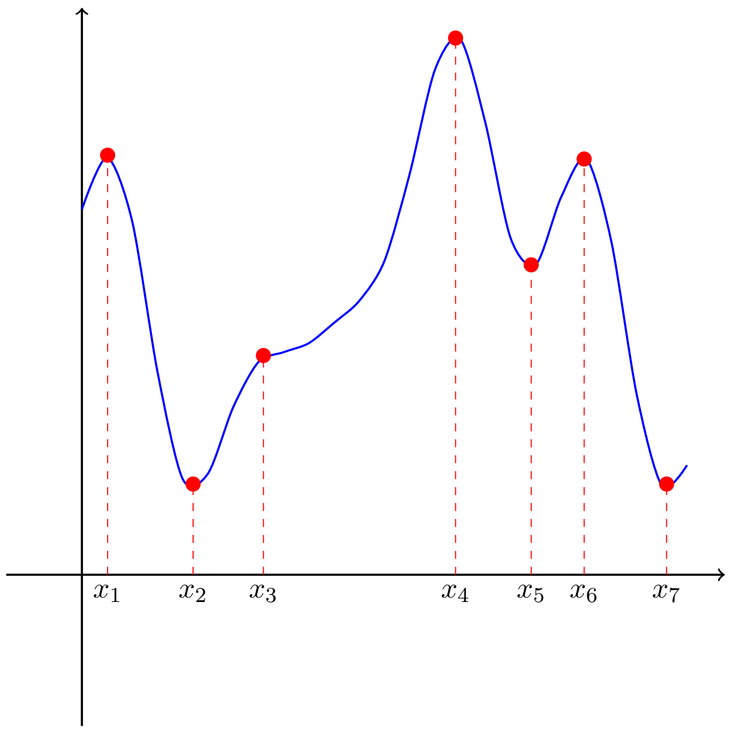
$$h(x) = \underset{j=\{1,2,\ldots,m\}}{\operatorname{argmax}} \sum_{i=N_k} I_{cj}(y_i)$$

This can also be called **mode function**, which is used to find the value that appears most frequently in the dataset.

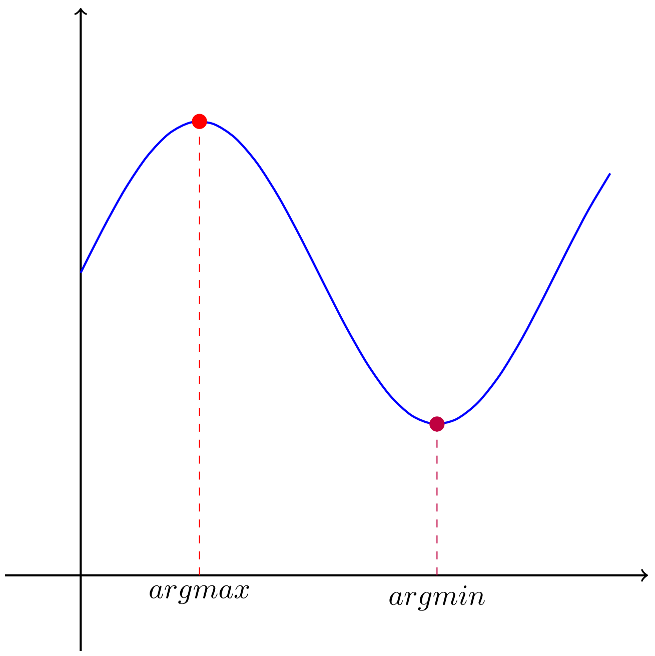
$$I_{cj}(y_i) = \begin{cases} 1 & y_i = c_j \\ 0 & y_i \neq c_j \end{cases}$$

This is a flag function, that will determine if a value is true or false, and the classifier function will accumulate all of those, and output the mode.

6.1 argmax and argmin



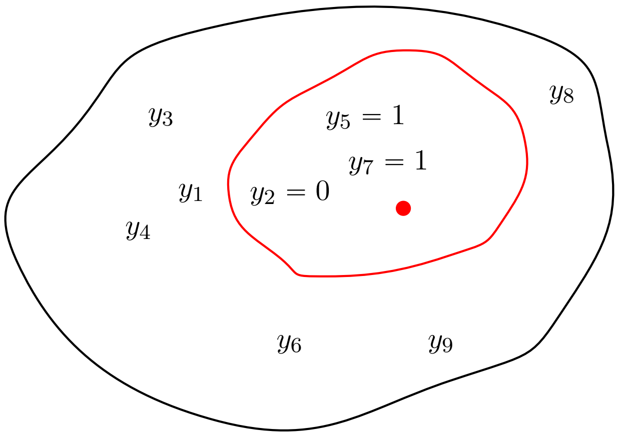
Given a function $f(x)$, try to find the min and max for the function. We take the derivative of the function and set it to zero ($f'(x) = 0$). A reason for that is because max and min can only occur when the derivate is set to zero, and then we solve for x. In the graph above, we can see different sets of x that could be used to find the min and max of the function. If we plug x_4 to the function, $f(x_4)$, we will find the **argmax**, which is the x value that makes the function maximum. This concept also applies to find the **argmin**, where we look for the x value (in this case x_2), which will make the function minimum.



As shown in the graph above, the argmax is the input value (x) to produce the maximum output value for the function. We must remember that $f(x) = y$, so, the **argmax** will produce the highest y value.

6.2 Example of mode function

Let say we have a new data point as shown in the graph below represented by a red dot, and there are already a set of 9 y's values:



Let say we assign $K = 3$, then it will select the nearest neighbors to that new data point. As seen in the graph, y_2 has a value of 0, y_5 is 1, and y_7 is 1 (0 and 1 are classes in this case), and they are the closest to the new data point. Later, we find the mode, and since the class 1 was the predominant one, the new data point would be assigned to be class 1.
One important thing to do, if there are let say 2 classes, K should not be a multiple of 2 to break ties, so K could be 3 or 5 or 7. This concept also applies for cases where there are 3 classes, we avoid choosing a K that is a multiple of 3.

7 Class Code

Language: Julia

```julia
# Installing the necessary packages
using Pkg
Pkg.add("RDatasets")
Pkg.add("MLJBase")
Pkg.add("Distances")

# Import packages for data handling
using Distances
using RDatasets

# Load the Iris dataset from the RDatasets package
iris = dataset("datasets", "iris")

# Extract the first four columns of the Iris dataset as a matrix (feature variables)
X = Matrix(iris[:,1:4])

# Extract the target variable from the Iris dataset and store it in y
y = iris.Species

# Convert the target variable into binary labels:
# Assign 1 to "setosa" and -1 to all other species
y = @. ifelse(iris.Species == "setosa", 1, -1)

# Get the unique class labels in the target variable
c = unique(y)

# Find the most frequent class label in y (this is mode technically)
c[argmax(map(i -> sum(y .== c[i]), 1:lastindex(c)))]
```