# University of Prishtina “Hasan Prishtina”

## Faculty of Electrical and Computer Engineering


# Technical documentation of the project

**Subject: Artificial Intelligence**

**Project Title: k-Nearest-Neighbors (k-NN) algorithm implementation from scratch for regression**

**Professor**

Prof. Dr. Nysret Musliu

**Student / email addresse**

Uran Lajçi / **uran.lajci@student.uni-pr.edu**

Prishtinë, Kosovë, 2023

**K-Nearest Neighbors Algorithm**

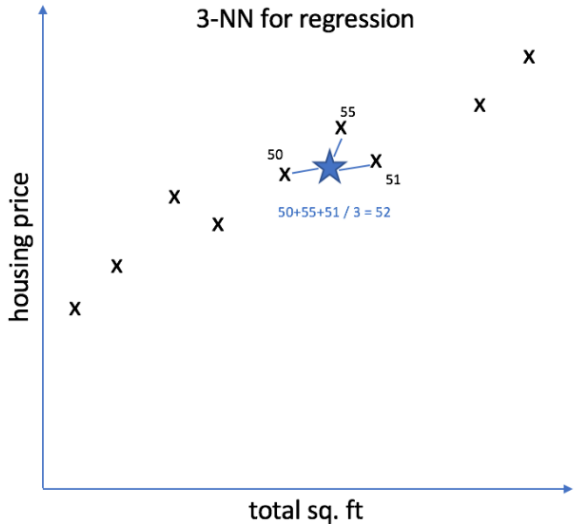
The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. While it can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another.

We are going to implement the KNN for regression problems. In regression problems KNN uses the average of the k nearest neighbors to make a prediction.

There are several distance metrics that can be used in KNN depending on the nature of our data. Some common distance metrics include:

1. Euclidean Distance: The most commonly used distance metric, it calculates the straight-line distance between two points in Euclidean space.
   Formula: **'sqrt(sum(x1 – x2)^2)'**
2. Manhattan Distance: Also known as City Block Distance, it calculates the sum of the absolute differences between coordinates of two points.
   Formula: **'sum(|x1 – x2|)'**
3. Chebyshev Distance: It is the maximum absolute difference between the coordinates of two points, also known as the chessbord distance since it represents the minimum number of moves a king needs to move between two points on a chessbord.
   Formula: **'max(|x1 – x2|)'**
4. Minkowski Distance: A generalization of Euclidean and Manhattan distances, it depends on a parameter 'p'. When p = 2, it becomes Euclidean distance, and when p = 1, it becomes Manhattan distance.
   Formula: **'(sum(|x1 – x2|^p))^(1/p)'**

There are other distance metrics like the Hamming and Jaccard distances but we are going to use the 4 that are mentiont above.




For example the KNN algorithem would be used to predict the value of the housing price based on the 3 neighbors like in the example above.

**Detailed Process of building the KNN algorithm**

1. Calculate the distance between data points:

The first step is to choose a distance metric that measures the similarity between data points. Some commonly used distance metrics include Euclidean, Manhattan, Minkowski, and Cosine distance. The choice of distance metric depends on the nature of the data and the problem you are trying to solve.

2. Find the k-nearest neighbors for a given data point:

For each data point you want to make a prediction for, you need to identify its k-nearest neighbors in the training set. To do this, you'll calculate the distance between the data point and every other point in the training set, using the chosen distance metric. Then, sort these distances and select the k points with the smallest distances. These k points are considered the k-nearest neighbors of the data point in question.

3. Make predictions based on the k-nearest neighbors for regression problems:

The process is similar to classification, but instead of selecting the majority class label, you'll take the average of the target values of the k-nearest neighbors. This average value will be the predicted target value for the data point.

4. Evaluate the algorithm's performance:

To assess the performance of the k-NN algorithm, you can use metrics such as accuracy for classification problems, and mean squared error or mean absolute error for regression problems. Additionally, you can perform cross-validation to get a more accurate estimate of the algorithm's performance.

5. Fine-tune the algorithm:

Experiment with different values of k, distance metrics, and feature scaling methods to find the optimal combination for your problem. You can use techniques like grid search or random search to systematically explore the hyperparameter space and find the best model.

In summary, building the k-NN algorithm from scratch involves calculating distances between data points, finding the k-nearest neighbors, making predictions based on these neighbors, and evaluating the model's performance. By following these steps and fine-tuning the algorithm's hyperparameters, you can create a custom k-NN model for your specific problem.

**The code of KNN implementation in Python**

```
import math
import operator
import warnings

import numpy as np
import pandas as pd
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score

warnings.filterwarnings("ignore")


def euclidean_distance(instance1, instance2, length):
    distance = 0
    for x in range(length):
        distance += pow((instance1[x] - instance2[x]), 2)
    return math.sqrt(distance)


def manhattan_distance(instance1, instance2, length):
    distance = 0
    for x in range(length):
        distance += abs(instance1[x] - instance2[x])
    return distance


def chebyshev_distance(instance1, instance2, length):
    distance = 0
    for x in range(length):
        distance = max(distance, abs(instance1[x] - instance2[x]))
    return distance


def get_neighbors(training_set, test_instance, k, distance_function):
    distances = []
    length = len(test_instance) - 1
    for x in range(len(training_set)):
        dist = distance_function(test_instance, training_set[x], length)
        distances.append((training_set[x], dist))
    distances.sort(key=operator.itemgetter(1))
    neighbors = []
    for x in range(k):
        neighbors.append(distances[x][0])
    return neighbors
```

```
def predict_regression(neighbors):
    prediction = sum(neighbor[-1] for neighbor in neighbors) /
len(neighbors)
    return prediction


# Load the dataset
boston = load_boston()
data = pd.DataFrame(boston.data, columns=boston.feature_names)
data['MEDV'] = boston.target

# Split the dataset into training and testing sets
train_data, test_data, train_target, test_target =
train_test_split(data[boston.feature_names], data['MEDV'], test_size=0.2,
random_state=0)
train_data['MEDV'] = train_target
test_data['MEDV'] = test_target

# Test with different k values and distance functions
distance_functions = [euclidean_distance, manhattan_distance,
chebyshev_distance]
k_values = [3, 5, 7]

for distance_function in distance_functions:
    for k in k_values:
        predictions = []
        for i in range(len(test_data)):
            neighbors = get_neighbors(train_data.values,
test_data.values[i], k, distance_function)
            prediction = predict_regression(neighbors)
            predictions.append(prediction)

        # Compute the performance measures
        rmse = math.sqrt(mean_squared_error(test_target, predictions))
        mae = mean_absolute_error(test_target, predictions)
        r2 = r2_score(test_target, predictions)

        print(f"Distance: {distance_function.__name__}, k: {k}")
        print(f"RMSE: {rmse}")
        print(f"MAE: {mae}")
        print(f"R2 Score: {r2}\n")
```

**Explenation of the code**

1. Import libraries:

Import necessary libraries, including math, operator, and warnings. The warnings library is used to suppress any warnings that may arise during execution.

2. Define Euclidean or some other distance function:

The euclidean_distance() function calculates the Euclidean distance between two instances, given the length of the feature vectors.

3. Define get_neighbors function:

The get_neighbors() function finds the k-nearest neighbors of a test_instance from the training_set using the Euclidean distance. It returns a list of k-nearest neighbors.

4. Define predict_regression function:

The predict_regression() function computes the average of the target values (the last element) of the k-nearest neighbors. This average is used as the predicted target value for the test_instance.

5. Load the Boston Housing dataset:

Load the dataset and create a DataFrame with the feature names as columns. Add the target variable (MEDV) to the DataFrame.

6. Split the dataset into training and testing sets:

Use the train_test_split() function from scikit-learn to split the dataset into training and testing sets, with a test size of 20% and a fixed random state for reproducibility.

7. Make predictions on the test data:

For each instance in the test_data, find the k-nearest neighbors, compute the prediction using the predict_regression() function, and append the prediction to the predictions list.

8. Compute the root-mean-square-error (RMSE):

Calculate the RMSE by taking the square root of the mean squared difference between the test target values and the predictions.

9. Print the RMSE:

Print the calculated RMSE for the k-NN regression model.

**Tests**

```
Distance: euclidean_distance, k: 3
RMSE: 6.971114538472075
MAE: 4.723856209150326
R2 Score: 0.40320066958970824

Distance: euclidean_distance, k: 5
RMSE: 7.193321521369938
MAE: 4.756078431372549
R2 Score: 0.36454787656595133

Distance: euclidean_distance, k: 7
RMSE: 7.361064876119771
MAE: 4.702100840336135
R2 Score: 0.3345657029081168

Distance: manhattan_distance, k: 3
RMSE: 6.4791142580315295
MAE: 4.266339869281046
R2 Score: 0.48446854621987245

Distance: manhattan_distance, k: 5
RMSE: 6.505071023409955
MAE: 4.219411764705882
R2 Score: 0.4803296058832637

Distance: manhattan_distance, k: 7
RMSE: 6.85588150747209
MAE: 4.390056022408963
R2 Score: 0.4227678709551642

Distance: chebyshev_distance, k: 3
RMSE: 7.028514131254641
MAE: 4.876797385620915
R2 Score: 0.39333221338439395

Distance: chebyshev_distance, k: 5
RMSE: 7.557650843649089
MAE: 4.981960784313725
R2 Score: 0.29854867743331703

Distance: chebyshev_distance, k: 7
RMSE: 7.434677650894428
MAE: 4.8203081232493
R2 Score: 0.32119008531293036
```

**References**

[1] KNN algorithm explenation
https://www.ibm.com/topics/knn#:~:text=The%20k%2Dnearest%20neighbors%20algorithm%2C%20also%20known%20as%20KNN%20or,of%20an%20individual%20data%20point.

[2] KNN example
https://www.jeremyjordan.me/k-nearest-neighbors/

[3] KNN algorithm implementation source
https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/

[4] Boston Housing dataset source
https://scikit-learn.org/stable/datasets/toy_dataset.html#boston-dataset

[5] Pandas library for data preprocessing
https://pandas.pydata.org/

[6] Scikit-learn library for train-test split
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

[7] Math library for computing RMSE
https://docs.python.org/3/library/math.html

[8] Brownlee, J. (2016). Master Machine Learning Algorithms - Discover how they work and Implement Them From Scratch. Machine Learning Mastery.

[9] ChatGPT (Versions 3.5 and 4)

[10] https://github.com/uran-lajci/KNN-Implementation-From-Scratch