Ten algorithms

Daniel Walther Berns

April 5, 2023

1 KNN

1.1 Introduction

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.

1. Non-parametric models do not make assumptions about the underlying distribution of the data, which can make them more flexible but also potentially more computationally expensive.

2. Supervised learning means that the algorithm is trained on labeled data, where the correct classifications are already known, and uses this training to make predictions on new, unlabeled data.

3. In KNN, proximity refers to the distance between data points in a feature space, where features are the measurable characteristics of the data points. The k in KNN refers to the number of nearest neighbors that are considered when making a prediction. For example, if k=3, the algorithm would consider the three closest data points to the one being classified, and assign the class label that is most frequent among those three.

4. While it is true that KNN can be used for both classification and regression problems, it is worth mentioning that the algorithm is more commonly used for former working off the assumption that similar points can be found near one another. In regression problems, KNN would predict a numerical value for the target variable, rather than a class label.

5. For classification problems, a class label is assigned on the basis of a majority vote, i.e. the label that is most frequently represented around a given data point is used. While this is technically considered "plurality voting", the term, "majority vote" is more commonly used in literature. The distinction between these terminologies is that "majority voting" technically requires a majority of greater than 50%, which primarily works when there are only two categories. When you have multiple classes, four categories, you don't necessarily need 50% of the vote to make a conclusion about a class; you could assign a class label with the greatest vote.

6. Regression problems use a similar concept as classification problem, but in this case, the average of the k nearest neighbors is taken to make a prediction about a classification. The main distinction here is that classification is used for discrete values, whereas regression is used with continuous ones.

7. It's also worth noting that the KNN algorithm is also part of a family of "lazy learning" models, meaning that it only stores a training dataset versus undergoing a training stage. This also means that all the computation occurs when a prediction is being made. Since it heavily relies on memory to store all its training data, it is also referred to as an instance-based or memory-based learning method.

8. Evelyn Fix and Joseph Hodges are credited with the initial ideas around the KNN model in this 1951 paper while Thomas Cover expands on their concept in his article "Nearest Neighbor Pattern Classification." While it's not as popular as it once was, it is still one of the first algorithms one learns in data science due to its simplicity and accuracy. However, as a dataset grows, KNN becomes increasingly inefficient, compromising overall model performance. It is commonly used for simple recommendation systems, pattern recognition, data mining, financial market predictions, intrusion detection, and more.

1.2 KNN Code

```
import numpy as np

class KNN:
    """
    This implementation defines a KNN class
    that takes a value k as a parameter
    during initialization.
    The fit method is used to train the model
    on a training set X and corresponding labels y.

    The predict method takes a set of
    data points X and returns predicted labels
    for each data point.

    In this implementation, we take advantage
    of numpy's ability to perform operations
    on arrays element-wise to vectorize
    the calculation of distances between X
    and self.X_train.

    First, we reshape X and self.X_train using
    np.newaxis to create a new axis so that we
    can perform broadcasting, which allows
    numpy to perform operations on arrays
    with different shapes.
    Then, we calculate the squared differences
    between each data point in X and each
    training example in self.X_train
    using the ** operator, sum over the
    feature dimension using np.sum, and take
    the square root using np.sqrt to obtain
    the distances.

    Next, we use np.argsort to obtain the
    indices of the k smallest distances
    along the second axis (axis 1), corresponding
    to the k nearest neighbors for each
    data point in X.

    We then extract the labels of the
    k nearest neighbors from self.y_train
    using numpy's array indexing syntax,
    and calculate the mode of the labels
    for each data point using np.apply_along_axis
    with a lambda function that applies np.bincount
    and argmax along the first axis (axis 1).

    Finally, we return the predicted labels y_pred,
    which is a numpy array with shape (n_samples,)
    containing the predicted label for each
    data point in X.
    """
    def __init__(self, k):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def predict(self, X):
        distances =
        np.sqrt(
            np.sum((X[:, np.newaxis, :] - self.X_train) ** 2,
                   axis=2))
        neighbors = np.argsort(distances, axis=1)[:, :self.k]
        labels = self.y_train[neighbors]
        y_pred = np.apply_along_axis(
            lambda x: np.bincount(x).argmax(),
            axis=1,
            arr=labels)
        return y_pred
```

2 Linear regression

Linear regression

3 Logistic regression

Logistic regression

4 Decision trees

Decision trees

5 Random forest

Random Forest

6 Naive Bayes

Naive Bayes

7 PCA

PCA

8 Perceptron

Perceptron

9 SVM

10 K means

Kmeans

11 Bibliography

Write