REGULARIZED K-NEAREST NEIGHBORS ALGORITHM

Arthur Mello
Independent author
arthur.bmello@gmail.com

ABSTRACT

The K-Nearest Neighbors algorithm (KNN) is a simple yet powerful machine learning algorithm, used in both classification and regressions tasks. For classification, for instance, it makes estimations for new observations by identifying the majority class within the k training data points that are the closest to the new data. Despite its effectiveness and interpretability, KNN has a few weaknesses. One weakness in particular comes from the fact that the distances between points usually do not take into account the importance of the features for predicting the target variable. This paper aims to present a method to deal with that weakness, by weighting the features according to their correlation with the target variable.

***K*eywords** Nearest neighbor · Classification · Regression · Weight adjustment · Regularization · Machine Learning

1 Introduction

Machine learning consists of making computers able to learn and deal with new information without giving it specific instructions. Instead, computers are given algorithms and data, from which they can infer existing patterns, that can then be applied to new data. It is a fast-growing field, with many different applications, from healthcare to finance.
One type of learning is called "supervised learning", when there is one specific variable for which we need estimations, so we use algorithms in which we specify our target variable. One of the most basic, yet efficient, algorithms used for this kind of task is called K-Nearest Neighbors (KNN).
In KNN, the values for new observations are calculated based on the most similar observations for which the target variable is known. For example, when calculating the probability of default of a bank's new client, that would mean looking at the probability of default of the K clients that are the most similar to that new one.
By using the terms "similar" and "nearest", we imply there is some sort of measure distance used to determine which clients will be used. Indeed, some common distance measures used in KNN are Euclidean, Manhattan, Mahalanobis, Minkowski and Chebyshev, in addition to some weighted variations of those.
One limitation of using the non-weighted distance metrics is that every dimension is given the same importance when defining the nearest neighbors, even though variables will have different degrees of importance depending on the task. Even after filtering out the least important features, the remaining ones will be considered equally important when calculating the distances. This limitation can be addressed by using a weighted variation of the distance metric, in a way that gives more weight to more important features (features that are better predictors of the target variable).
In this paper we propose a regularized version of the KNN algorithm that takes feature importance into account and works for both classification and regression tasks. This approach will change the way the distance measure is calculated, by adding to it a feature importance component.

2 Related work

The KNN algorithm was introduced in 1951 by Fix and Hodges, in an unpublished US Air Force School of Aviation Medicine report, as a "non-parametric method for pattern classification". It was then further developed by Cover in 1967 [1], who showed the admissibility of using k = 1.

The idea of calculating weighted distances is not novel, and most implementations of the KNN algorithm have some way of including weight parameters for that, at least for the most commonly used distance metrics. What to use as weights for those distances, however, remains under-unexplored.

One example of a weighted distance metric is the Mahalanobis distance [2], which weights features using the covariance matrix, in order to take into account the covariance between variables. This can be quite useful when calculating the distance between observations and centroids, in order to detect outliers.

In 2001, Han et al [3]. proposed a weight-adjusted version of KNN for text categorization, called WAKNN, which showed promising results. It works by inserting a weight vector W in the cosine similarity measures between words. That vector has to be initialized and then updated iteratively, in an attempt to improve the objective function. Those updates are done through random, small perturbations on the weights. That weight adjustment step is of complexity $O(cn^2)$, where c is the number of iterations in the weight adjustment step and n is the number of data points. Therefore, it can be time-consuming and make implementation more difficult.

3 KNN algorithm

Let x be a new observation for which we want to estimate the value of the target variable y. The KNN algorithm works as follows:

1. Calculate the distance between x and all the other data points for which we know the the value of y.
2. Arrange the distances in increasing order
3. Given a positive integer k, select the k-first distances from the arranged list
4. Select the k points corresponding to those distances
5. If it's a classification task, label x with the majority class amongst the k observations. If it's a regression task, estimate y using the average value of y for the k observations.

The distance measure used on step 1 can vary according to the task in hand and the type of data, but most common ones include Euclidean, Manhattan, Minkowski and Mahalanobis distances. The Euclidean and Manhattan distances being special cases of the Minkowsky distance, we will only look at Minkowski and Mahalanobis distances. The Minkowski distance between two points X and Y is calculated as:

$$(\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}} \tag{1}$$

Where n is the number of dimensions, and p is the order of the distance, being p≥1. When p=1, we have the Manhattan distance; when p=2, we have the Euclidean distance. The Mahalanobis distance between two points X and Y is calculated as:

$$\sqrt{(x-y)'V^{-1}(x-y)} \tag{2}$$

Where V is is the covariance matrix between all the data features.

4 Regularized version

On the KNN algorithm, regularization can be done during step 1, when calculating the distance measure between x and the other points. In order to take into account the relationship between each variable and the target, we have a few alternatives. In this paper, we'll use the absolute value of the Pearson correlation coefficient as a weight for each feature, adjusted by a regularization parameter:

$$w_j = |\rho_{X_j Y}|^{\lambda} \tag{3}$$

where j represents the rank of the feature in question, and λ represents the regularization parameter. The greater the regularization parameter is, the more it penalizes features that are not correlated to the target variable, and the less they will weight in the distance calculation.

That has the advantage of being non-negative and easily interpretable. On the downside, it will not properly take into account non-linear relationships.

A weight vector w can then be formed from the weights of each feature, and then be applied to the more general Minkowski distance:

$$(\sum_{i=1}^{n} w|x_i - y_i|^p)^{\frac{1}{p}} \tag{4}$$

Or to the Mahalanobis distance:

$$\sqrt{w(x-y)'V^{-1}(x-y)} \tag{5}$$

5 Experimental results

In order to test its predictive power, the regularized version has been tested on the diabetes dataset [4], included on sklearn [5].

The dataset contains information on health indicators and diabetes, with 442 instances and 10 attributes, one of which being the target variable for our regression task: a quantitative measure of disease progression one year after baseline. All other variables are numeric or binary, and include information such as age, BMI and sugar blood levels.

All the variables are already centered and scaled.

5.1 Methodology

The following steps were followed in order to evaluate the regularized KNN predictive power:

1. For each distance metric (Manhattan, Euclidean and Minkowsky), the local optimal value for k was calculated within a certain arbitrary search space (in this case, all integers between 1 and 14), using the non-regularized version of KNN and optimizing for the MSE (Mean Squared Error)
2. That same k was applied to the regularized version, using different values of lambda
3. The MSE scores obtained by both algorithms were then compared

5.2 Results

For the Manhattan distance, using the non-regularized KNN yields us a locally-optimal value of 13 for k, and a MSE of 3278. For the same distance and k, the best the regularized version can do is 2827. Given the right value for lambda, regularization significantly improved the MSE (-14%).

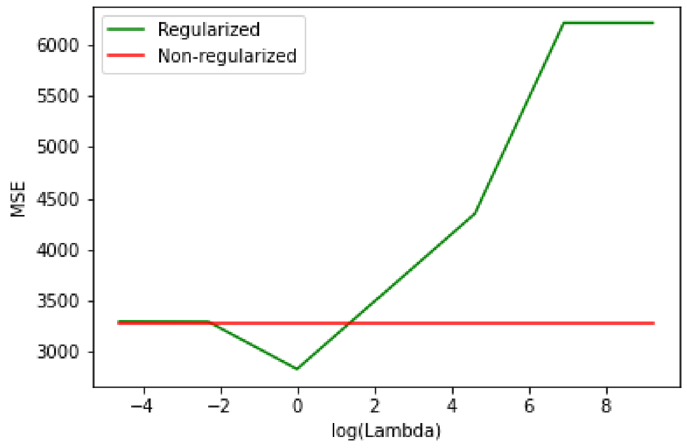

Figure 1: Scores for Manhattan distance

Looking at the Euclidean distance, the non-regularized KNN yields us a locally-optimal value of 13 for k, and a MSE score of 3278. For the same distance and k, the best the regularized version can do is 2827. The results were exactly the same as in the case of the Manhattan distance, so regularization has again significantly improved the MSE (-14%).

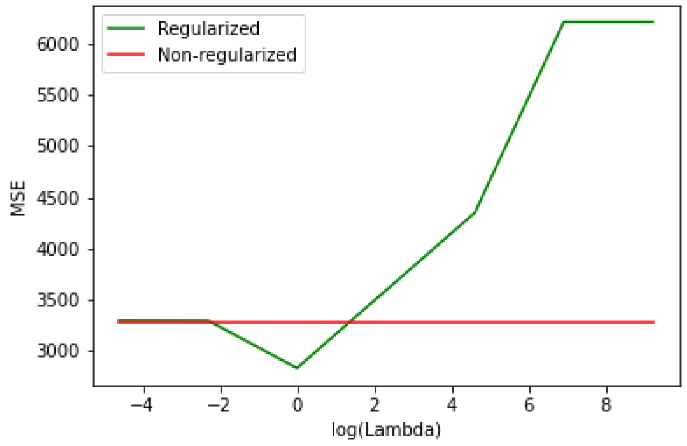

Figure 2: Scores for Euclidean distance

For the Mahalanobis distance, the non-regularized KNN yields us a locally-optimal value of 13 for k, and a MSE of 3482. For the same distance and k, the best the regularized version can do is 3314. Regularization has slightly improved the MSE (-5%).

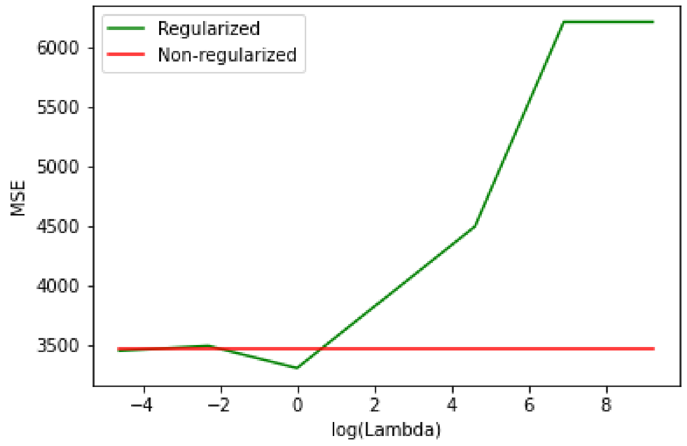

Figure 3: Scores for Mahalanobis distance

6 Conclusion and directions of future research

Overall, regularizing the KNN algorithm yielded better results, and the best overall MSE (3314) was obtained with the regularized version. This shows that regularizing the KNN algorithm can lead to improvements in regression tasks.

A potential area of further improvement would be to replace the Pearson's correlation coefficient by another metric that takes into account non-linear relationships.

References

[1] T.M. Cover. Nearest neighbor pattern classification. pages 21–27. IEEE, 1967.

[2] P.C. Mahalanobis. On tests and measures of group divergence. pages 541–588, 1930.

[3] Karypis G. Han, E. and V. Kumar. Text categorization using weight adjusted k-nearest neighbor classification. pages 53–65, 2001.

[4] Bradley E. et al. Diabetes data, 2004.

[5] F. et al. Pedregosa. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.