Optimisation of Object Classification using KNN Classifiers

Ben Stockil

June 2021

Abstract

Algorithms which are able to classify arbitrary objects into groups based on their attributes are fundamentally important to many artificial intelligence applications, such as image recognition or personalised recommendation algorithms. A KNN classifier is an algorithm which is able to do this by comparing the attributes of an unknown object with attributes of objects with known classes, and deciding which class they belong to accordingly. In order to maximise the accuracy of the KNN classifer, the grid search technique is employed to find the best possible configuration for the model, for any given dataset. Preprocessing techniques such as imputation, oversampling and normalisation are also employed to "clean" the data before it is fed into the classifier.

1 The KNN Classifier

The KNN (k-Nearest-Neighbours) classifier is initialised by placing all objects from the dataset with a **known class** into an n-dimensional space, where the number of dimensions is equal to the number of features per object. These points are assigned labels based on their class. [1]

For example, take a dataset containing the **heights** and **weights** of a group of rugby and basketball players. Since there are only two features, the data can be plotted on a 2-dimensional graph. In the following graphs, basketball players are displayed in **red**, while rugby players are displayed in **green**.

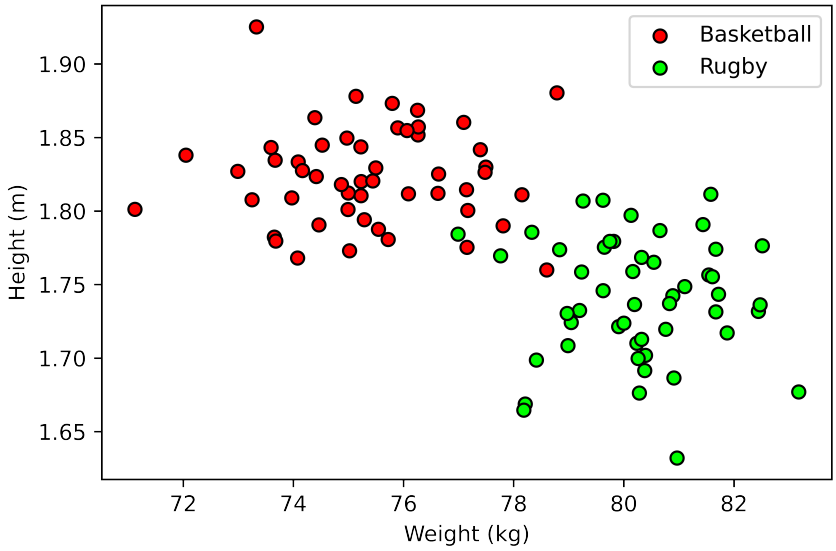

Figure 1: Graph representing heights and weights of basketball and rugby players.

As expected, we can see two distinct groups. The basketball players are typically taller, while the rugby players are typically heavier, but shorter. There are a few outliers in the data, but those will generally not affect the classification in a well-tuned model.

In order to classify an unknown object as a rugby or basketball player, we first place it into this two-dimensional space, as we did with the other objects.

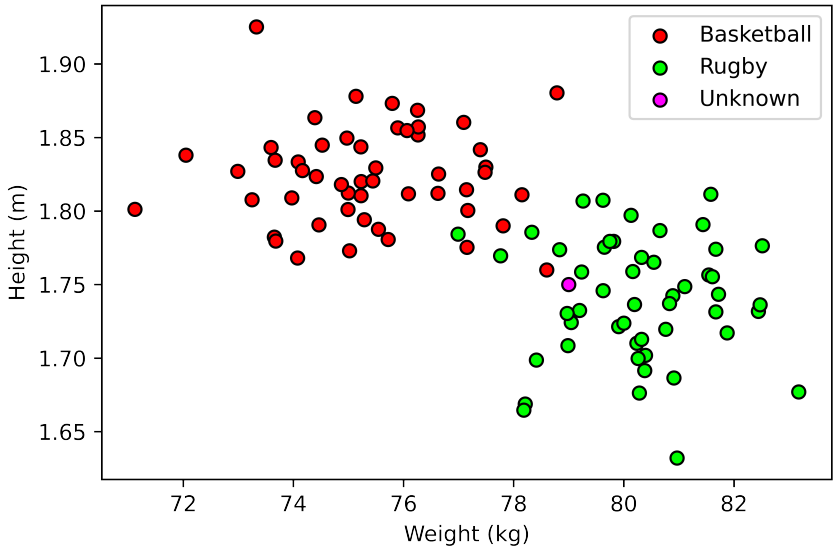

Figure 2: A new point has been added, representing an unknown player that the KNN model must classify.

Classification is then performed by calculating the distance from each known object to the new unclassified point, and then selecting the k nearest objects (k was defined when the model was initialised; for this example, k=3). Once the 3 nearest objects have been found, a majority vote is used to determine the class of the new point. By vote of 2–1, the new object is determined to be a rugby player.

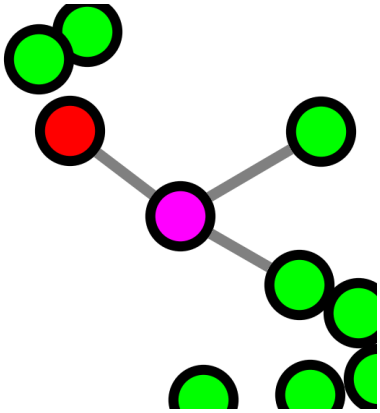

Figure 3: By vote of 2–1 from the nearest neighbours, the new object is assigned the class of basketball player.

1.1 Benefits and Drawbacks

One advantage of the KNN classifier is that it requires no training time, except for the time needed to populate the data space, which is often negligible. This allows it to be very quickly re-trained with new data or new parameters.

A limitation of the KNN model is that it can only work with numerical features, with **no values missing**. This is because all objects in the data must be representable as a single point in n-dimensional space, in order to calculate distances between them.

To solve the problem of missing values, rows with missing values can be removed from the data, or missing values can be estimated based on the features of other objects. This will be expanded on further.

To handle non-numerical/categorical features, a technique known as one-hot encoding can be employed. This will also be expanded on later.

Finally, due to the nature of the algorithm, the KNN model assumes that the distribution of classes within the data is equal. If, however, the data contains a dominant class, the model may be biased to classify unknown objects as this dominant class, affecting accuracy of results. In order to solve this, samples of minority classes can be synthetically generated to balance the distribution of classes.

2 Data Preprocessing

To combat the issues described above, the data is preprocessed through a series of steps, to ensure maximum accuracy for the given dataset.

2.1 Imputation of Missing Values

An imputer is an algorithm used to fill missing values within a dataset. The generated values for missing features can be determined by a number of factors, such as:

- the mean/median/mode of that feature among the entire dataset
- the mean/median/mode of the nearest neighbours

To increase the variety and prevent duplicates in values for missing features, noise can also be applied to the imputed values.

2.2 Oversampling (Synthetic Generation of Samples)

To equalise the distribution of classes, members of minority classes can be duplicated, adding small amounts of noise to increase variation. This prevents majority classes from dominating the data space, which preventing accurate classification.

2.3 One-Hot Encoding

One-hot encoding is where a categorical feature is "exploded" across multiple dummy features, so that it can be represented in vector space.

For example, a dataset containing the colours of 5 cupcakes can be converted using one-hot encoding. The left table can be rewritten as the right table, by expanding each category in the "colour" field to a separate dummy feature.

Cupcake	Colour
1	blue
2	red
3	blue
4	yellow

Cupcake	Red?	Blue?	Yellow?
1	0	1	0
2	1	0	0
3	0	1	0
4	0	0	1

The colour of the cupcake can now be represented in 3D space, with each colour expressed as one dimension.

2.4 Normalisation

The distance calculation used (usually euclidean distance) assumes that all features use the same scale with relatively similar ranges, so that no features outweigh each other in the calculation. However, in most data, this is not the

case. Consider a dataset containing people's ages and salaries; salary is recorded in dollars on a scale from \$15,000–\$250,000, while age is recorded in years from 18–95. In this case, salary would have much more of an effect on the distance calculation, due to the much larger numerical range.

To fix this, all features are simply normalised onto a scale of 0–1, where the minimum and maximum values of every feature are mapped to 0 and 1 respectively.

3 The Grid Search Algorithm

A KNN classifier has some parameters which are set before training begins (known as hyperparameters); the two most important of these are the value for k and the weighting strategy employed for the majority vote.

- The value for k, as described previously, dictates how many neighbours should be included in the classification vote for any prediction. A higher value for k may work better in higher noise datasets, while a lower value for k may work better in more heterogenous datasets. Usually, a value of 3,5 or 7 is optimal. Odd numbers are used to prevent the vote being a draw.
- The weighting strategy dictates how neighbours should be treated in the classification vote. The *uniform* strategy weighs all neighbours the same, while the *distance* strategy weights closer neighbours higher, meaning they have greater influence over the result of the vote.

While values for these parameters could be arbitrarily picked based on previous research or sensible estimates, there are optimisation techniques which enable the model to determine the best possible set of hyperparameters, in order to achieve the greatest classification accuracy.

The grid search algorithm is a hyperparameter optimisation algorithm which iterates over sets of possible hyperparameters, validating each combination and returning the best one. The speed of this process is determined by how many combinations it needs to check, as well as the speed of training and testing for the model (which, in the case of the KNN, is low). [2]

4 Practical Example

An example dataset can be used to evaluate the effectiveness of the KNN model in a simple classification task. The dataset chosen was the Adult Data Set from the UCI Machine Learning Repository [3]. It is a multivariate dataset containing data from the 1994 US Census database, and the proposed goal of the classifier is to determine whether someone makes less or more than $50,000 a year. It contains 32561 records, with 14 attributes each.

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
non-missing values	32561	32561	32561	32561	32561	32561
mean average	38.58	190,000	10.08	1077.65	87.30	40.44
standard deviation	13.64	106,000	2.57	7385.29	402.96	12.35
min value	17	12,300	1	0	0	1
25%	28	118,000	9	0	0	40
50%	37	178,000	10	0	0	40
75%	48	237,000	12	0	0	45
max value	90	1,480,000	16	99999	4356	99

	workclass	education	marital-status	occupation	relationship	race	sex	native-country	earns-over-50k
non-missing values	30725	32561	32561	30718	32561	32561	32561	31978	32561
unique categories	8	16	7	14	6	5	2	41	2
dominant category	Private	HS-grad	Married-civ-spouse	Prof-specialty	Husband	White	Male	United-States	<=50K
dominant category size	22696	10501	14976	4140	13193	27816	21790	29170	24720

The number of rows with missing values is 2399 out of 32561. This issue will be addressed in the preprocessing section of this paper.

Just looking at the data, it would be fair to predict that features such as education, work class and occupation would be the best proxies for income, while features such as marital status and sex should have less of an effect.

The classes of this dataset are unevenly distributed, with 24720 out of 32561 records (75.9%) earning <=50K a year.

4.1 Apply Preprocessing Steps

Firstly, missing values are filled using a simple imputer written using python's `pandas` library. Since only categorical features have missing values in this dataset, it is only necessary to write the imputer for categorical features. This imputer takes the most common (mode) category among all other objects and applies it to the missing features:

Oversampling is then used to balance the distribution of people who earn less and more than $50k a year. Shown by the pie charts below, this achieves a perfect 50–50 split among the classes, which is ideal for a KNN model.

Figure 4: After oversampling, the dataset is equally balanced, preventing bias in the KNN model.

Finally, one-hot encoding and normalisation are applied onto the categorical and numerical features. The preprocessed dataset is then ready for training.

4.2 Training

A parameter grid is established to indicate which parameters to include in the grid search, and which values to permute over. The parameter grid in this example represents the following:

- No. of neighbours / k-value: {3, 7, 15, 27, 43, 63, 87, 201, 501, 1001, 2001, 5001}
- Weighting strategy: {Uniform, Distance}

The grid search algorithm will run the model 24 times, each with a different set of parameters to use. Each model will be validated, and results are returned.

4.3 Results

The results from the grid search algorithm can be plotted on the following graph.

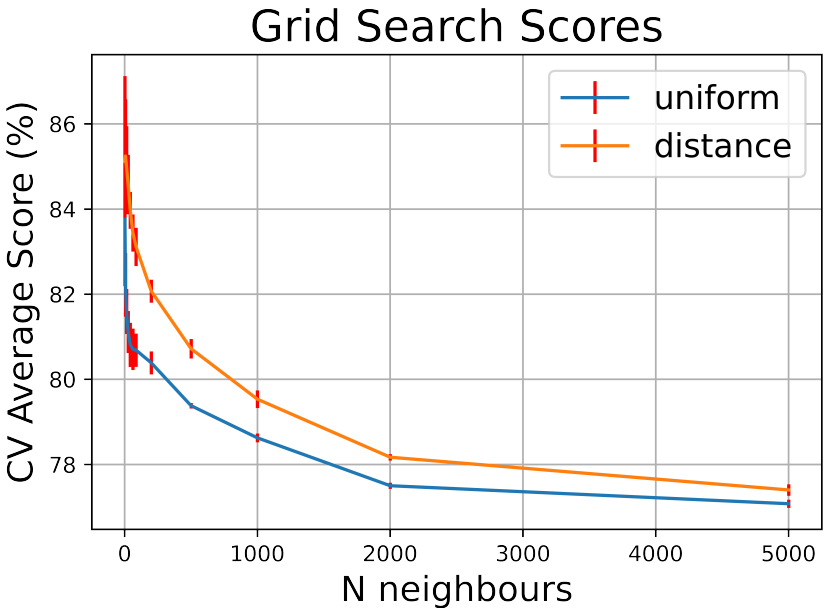

Figure 5: Results of training

As you can see, lower values of k performed much better, with accuracy continuously dropping as k increased. As k increases, the model is not only looking at immediate neighbours, but further away neighbours, which are more likely to be of the other class. However, the reason that a k-value of 7 performed better than a value of 3 is because more neighbours reduces noise, and it may therefore be beneficial to look at several surrounding neighbours rather than only the most immediate ones.

References

[1] B. W. Silverman and M. C. Jones, "E. fix and j.l. hodges (1951): An important contribution to nonparametric discriminant analysis and density estimation: Commentary on fix and hodges (1951)," *International Statistical Review / Revue Internationale de Statistique*, vol. 57, no. 3, pp. 233–238, 1989.

[2] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, vol. 415, pp. 295–316, 2020.

[3] D. Dua and C. Graff, "UCI machine learning repository," 2017.