k-Nearest Neighbour Classifiers - A Tutorial

PÁDRAIG CUNNINGHAM, School of Computer Science, University College Dublin
SARAH JANE DELANY, School of Computer Science, Technological University Dublin

Perhaps the most straightforward classifier in the arsenal or Machine Learning techniques is the Nearest Neighbour Classifier—classification is achieved by identifying the nearest neighbours to a query example and using those neighbours to determine the class of the query. This approach to classification is of particular importance, because issues of poor runtime performance is not such a problem these days with the computational power that is available. This article presents an overview of techniques for Nearest Neighbour classification focusing on: mechanisms for assessing similarity (distance), computational issues in identifying nearest neighbours, and mechanisms for reducing the dimension of the data.

This article is the second edition of a paper previously published as a technical report [16]. Sections on similarity measures for time-series, retrieval speedup, and intrinsic dimensionality have been added. An Appendix is included, providing access to Python code for the key methods.

CCS Concepts: • **Computing methodologies** → *Machine learning*

Additional Key Words and Phrases: *k*-Nearest neighbour classifiers

ACM Reference format:
Pádraig Cunningham and Sarah Jane Delany. 2021. k-Nearest Neighbour Classifiers - A Tutorial. *ACM Comput. Surv.* 54, 6, Article 128 (July 2021), 25 pages.
https://doi.org/10.1145/3459665

1 INTRODUCTION

The intuition underlying Nearest Neighbour Classification is quite straightforward: examples are classified based on the class of their nearest neighbours. It is often useful to take more than one neighbour into account, so the technique is more commonly referred to as ***k*-Nearest Neighbour (*k*-NN)** Classification, where k nearest neighbours are used in determining the class. Since the training examples are needed at runtime, i.e., they need to be in memory at runtime, it is sometimes also called Memory-based Classification. Because induction is delayed to runtime, it is considered a Lazy Learning technique. Because classification is based directly on the training examples, it is also called Example-based Classification, or Case-based Classification.

The basic idea is as shown in Figure 1, which depicts a 3-Nearest Neighbour Classifier on a two-class problem in a two-dimensional feature space. In this example the decision for q_1 is

This work was funded by Science Foundation Ireland through I-From: The SFI Centre for Advance Manufacturing Research (16/RC/3872) and the SFI Centre for Research Training in Machine Learning (Grant No. 18/CRT/6183).
Authors' addresses: P. Cunningham, School of Computer Science, University College Dublin; email: padraig.cunningham@ucd.ie; S. J. Delany, School of Computer Science, Technological University Dublin; email: sarahjane.delany@tudublin.ie.

0360-0300/2021/07-ART128 $15.00
https://doi.org/10.1145/3459665

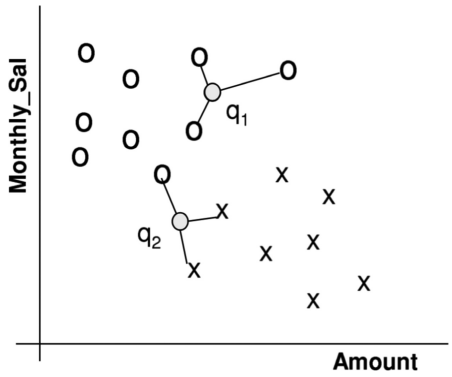

Fig. 1. 3-Nearest Neighbour Classification in a 2D feature space (Monthly_Sal and Amount).

straightforward—all three of its nearest neighbours are of class O, so it is classified as an O. The situation for q_2 is a bit more complicated, as it has two neighbours of class X and one of class O. This can be resolved by simple majority voting or by distance weighted voting (see below). So, k-NN classification has two stages: the first is the determination of the nearest neighbours, and the second is the determination of the class using those neighbours.

Let us assume that we have a training dataset D made up of $(\mathbf{x}_i)_{i\in[1,n]}$ training samples (where $n = |D|$). The examples are described by a set of features F and any numeric features have been normalised to the range [0,1]. Each training example is labelled with a class label $y_j \in Y$. Our objective is to classify an unknown example $\mathbf{q}$. For each $x_i \in D$, we can calculate the distance between $\mathbf{q}$ and $\mathbf{x}_i$ as follows:

$$d(\mathbf{q}, \mathbf{x}_i) = \sum_{f \in F} w_f \delta(\mathbf{q}_f, \mathbf{x}_{if}). \tag{1}$$

This is a summation over all the features in F with w_f the weight for each feature. There is a large range of possibilities for this distance metric; a basic version for continuous and discrete attributes would be:

$$\delta(\mathbf{q}_f, \mathbf{x}_{if}) = \begin{cases} 0 & f \text{ discrete and } \mathbf{q}_f = \mathbf{x}_{if}, \\ 1 & f \text{ discrete and } \mathbf{q}_f \neq \mathbf{x}_{if}, \\ |\mathbf{q}_f - \mathbf{x}_{if}| & f \text{ continuous.} \end{cases} \tag{2}$$

The k nearest neighbours are selected based on this distance metric. Then there is a variety of ways in which the k nearest neighbours can be used to determine the class of $\mathbf{q}$. The most straightforward approach is to assign the majority class among the nearest neighbours to the query.

It will often make sense to assign more weight to the nearer neighbours in deciding the class of the query. A fairly general technique to achieve this is distance-weighted voting where the neighbours get to vote on the class of the query case with votes weighted by the inverse of their distance to the query.

$$Vote(y_j) = \sum_{c=1}^{k} \frac{1}{d(\mathbf{q}, \mathbf{x}_c)^p} 1(y_j, y_c) \tag{3}$$

Thus, the vote assigned to class y_j by neighbour x_c is 1 divided by the distance to that neighbour, i.e., $1(y_j, y_c)$ returns 1 if the class labels match and 0 otherwise. In Equation (3) p would normally be 1 but values greater than 1 can be used to further reduce the influence of more distant neighbours.

Another approach to voting is based on Shepard's work [48] and uses an exponential function rather than inverse distance, i.e:

$$Vote(y_j) = \sum_{c=1}^{k} e^{d(\mathbf{q},\mathbf{x}_c)} 1(y_j, y_c). \tag{4}$$

It is worth mentioning that k-NN can also be effective for regression [2]. In regression, the dependant variable y is a real number ($y \in \mathbf{R}$), so the predicted value $\hat{y}$ can be the mean or weighted mean of the y value for the neighbours. The weighted mean would be defined as follows:

$$\hat{y} = \frac{1}{k} \sum_{c=1}^{k} \frac{1}{d(\mathbf{q}, \mathbf{x}_c)^p} y_c. \tag{5}$$

The objective for this article is to present a comprehensive tutorial resource on the use of k-NN. While other k-NN resources exist, most notably the 1991 book by Dasarathy [17] and the more recent survey by Bhatia [11], our focus is on the implementation of a wide variety of k-NN methods and we provide supporting code in Python. To this end, we consider three important factors that must be considered with the use of k-NN. In the next section, we look at the core issue of similarity and distance measures and explore some *exotic* (dis)similarity measures to illustrate the generality of the k-NN idea. In Section 3, we look at computational complexity issues and review some speedup techniques for k-NN. In Section 4, we look at dimension reduction—both feature selection and sample selection. Dimension reduction is of particular importance with k-NN, as it has a big impact on computational performance and accuracy. The article concludes with a summary of the advantages and disadvantages of k-NN.

2 SIMILARITY AND DISTANCE METRICS

While the terms *similarity metric* and *distance metric* are often used colloquially to refer to any measure of affinity between two objects, the term metric has a formal meaning in mathematics. A metric must conform to the following four criteria (where $d(x, y)$ refers to the distance between two objects x and y):

(1) $d(x, y) \geq 0$; non-negativity
(2) $d(x, y) = 0$ only if $x = y$; identity
(3) $d(x, y) = d(y, x)$; symmetry
(4) $d(x, z) \geq d(x, y) + d(y, z)$; triangle inequality.

It is possible to build a k-NN classifier that incorporates an affinity measure that is not a proper metric, however, there are some performance optimisations to the basic k-NN algorithm that require the use of a proper metric [10, 46, 59, 59]. In brief, these techniques can identify the nearest neighbour of an object without comparing that object to every other object but the affinity measure must be a metric; in particular, it must satisfy the triangle inequality.

The basic distance metric described in Equations (1) and (2) is a special case of the Minkowski Distance metric—in fact, it is the 1-norm (L_1) Minkowski distance. The general formula for the Minkowski distance is

$$MD_p(\mathbf{q}, \mathbf{x}_i) = \left(\sum_{f \in F} |\mathbf{q}_f - \mathbf{x}_{if}|^p \right)^{\frac{1}{p}}. \tag{6}$$

The L_1 Minkowski distance is the Manhattan distance and the L_2 distance is the Euclidean distance. It is unusual but not unheard of to use p values greater than 2. Larger values of p have the effect of giving greater weight to the attributes on which the objects differ most. To illustrate this,

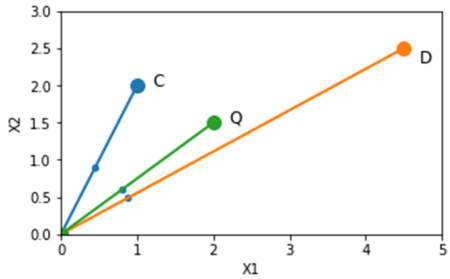

Fig. 2. Cosine Similarity: Q, C, and D are three examples in a 2D feature space, using Cosine Similarity the nearest neighbour to Q is D.

we can consider three points in 2D space: $A = (1, 1), B = (5, 1)$, and $C = (4, 4)$. Since A and B differ on one attribute only, the $MD_p(A, B)$ is 4 for all p, whereas $MD_p(A, C)$ is 6, 4.24, and 3.78 for p values of 1, 2, and 3, respectively. So, C becomes the nearer neighbour to A for p values of 3 and greater.

The other important Minkowski distance is the L^∞ or Chebyshev distance:

$$MD_\infty(\mathbf{q}, \mathbf{x}_i) = \max_{f \in F} |\mathbf{q}_f - \mathbf{x}_{if}|.$$

This is simply the distance in the dimension in which the two examples are most different; it is sometimes referred to as the chessboard distance, as it is the number of moves it takes a chess king to reach any square on the board.

While the Euclidean and Manhattan distances are probably the most popular k-NN distance measures, much of the usefulness of k-NN derives from the potential to work with metrics that are specific to the data under analysis. In the next subsections, we will look at the merits of Cosine Similarity and (Pearson) Correlation. Then, we will look at some more complex distance measures that are specialised for particular data types, i.e., Earth Mover's Distance for image data and Dynamic Time Warping for time-series data. These were chosen because they enable the application of machine learning on data that is not in a feature vector format. This section concludes with a brief introduction to Metric Learning whereby a metric can be *induced* from the data.

2.1 Cosine Similarity

Like Minkowski distance, Cosine Similarity works with feature vector data. However, similarity is based on the angles between the feature vectors—see Figure 2. While C would be the closer example to Q based on Euclidean distance, D is closer to Q when the angles between the features vectors is considered. The Cosine similarity between a query $\mathbf{q}$ and $\mathbf{x}_i$ is as follows:

$$\text{Cos}(\mathbf{q}, \mathbf{x}_i) = \frac{\sum_{f \in F} \mathbf{q}_f \cdot \mathbf{x}_{if}}{\sqrt{\sum_{f \in F} \mathbf{q}_f^2} \sqrt{\sum_{f \in F} \mathbf{x}_{if}^2}}. \tag{7}$$

This is the dot product of the feature values normalised by the lengths of the feature vectors. Cosine similarity is a popular metric in text analytics. When text is processed as a bag of words the features are word counts and Cosine similarity has the advantage that it is independent of the magnitude of the feature vectors. Thus, it is insensitive to document size. Cosine similarity requires that all feature values are positive real numbers.

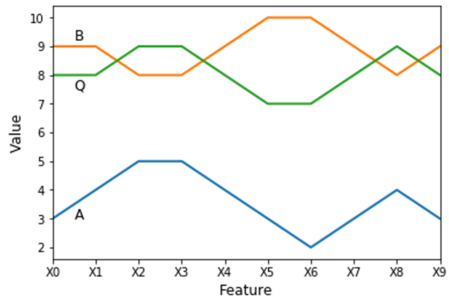

Fig. 3. Correlation: Q, A, and B are three examples described by 10 features (X0–X9), correlation recognises that Q is more similar to A than to B.

If feature values are positive, then the Cosine similarity will be in the interval $[0, 1]$. So, we can define a Cosine distance measure:

$$\text{CosD}(\mathbf{q}, \mathbf{x}_i) = 1 - \text{Cos}(\mathbf{q}, \mathbf{x}_i). \tag{8}$$

2.2 Correlation

Figure 3 shows a scenario where correlation would be the appropriate similarity measure. While B is the more similar example to the query Q in terms of feature values, the *pattern* in A better correlates with Q. Sometimes this correlation is the key to the underlying similarity. This would be appropriate where the feature values reflect resource allocation (for example, household expenditure on 10 categories (X0–X9)). The magnitudes might be quite different, but the allocation pattern could be the same.

The two most popular correlation coefficients are the Pearson and Spearman measures [20]. The Pearson is applicable for features that are normally distributed. When reference is made to a correlation coefficient without specifying which one, it is probably the Pearson. The Pearson correlation between a query $\mathbf{q}$ and a sample $\mathbf{x}_i$ is defined as follows:

$$r(\mathbf{q}, \mathbf{x}_i) = \frac{\sum_{f \in F}(\mathbf{q}_f - \bar{\mathbf{q}})(\mathbf{x}_{if} - \bar{\mathbf{x}}_i)}{(n-1)s_q s_{x_i}} = \frac{\sum_{f \in F}(\mathbf{q}_f - \bar{\mathbf{q}})(\mathbf{x}_{if} - \bar{\mathbf{x}}_i)}{\sqrt{\sum_{f \in F}(\mathbf{q}_f - \bar{\mathbf{q}})^2 \sum_{f \in F}(\mathbf{x}_{if} - \bar{\mathbf{x}}_i)^2}}, \tag{9}$$

where $\bar{\mathbf{x}}_i$ and s_x are the mean and standard deviation of $\mathbf{x}_i$. This is the dot product of the mean-adjusted $\mathbf{q}$ and $\mathbf{x}_i$ vectors divided by their standard deviations. This mean adjustment makes the measure insensitive to variations in scale.

In circumstances where the features are not normally distributed the Spearman (rank) correlation can be used. The feature values are ranked and the statistic is calculated using ranks rather than the original values.

Correlation scores range from $[-1, 1]$. A score of 1 represents a perfect correlation, 0 is no correlation and -1 means the samples are anti-correlated. A correlation is a similarity score and so can be converted to a distance in the same manner as for Cosine—see Equation (8).

2.3 Other Distances Metrics for Multimedia Data

The Minkowski distance defined in Equation (6) is a very general metric that can be used in a k-NN classifier for any data that is represented as a feature vector. When working with image data a convenient representation for the purpose of calculating distances is a colour histogram. An image can be considered as a grey-scale histogram H of N levels or bins where h_i is the number of pixels that fall into the interval represented by bin i (this vector h is the feature vector). The

Minkowski distance formula (6) can be used to compare two images described as histograms. L_1, L_2, and less often L_∞ norms are used. Other popular measures for comparing histograms are the Kullback-Leibler divergence (10) [32] and the χ^2 statistic (11) [45].

$$d_{KL}(H,K) = \sum_{i=1}^{N} h_i \log\left(\frac{h_i}{k_i}\right), \tag{10}$$

$$d_{\chi^2}(H,K) = \sum_{i=1}^{N} \frac{h_i - m_i}{h_i}, \tag{11}$$

where H and K are two histograms, h and k are the corresponding vectors of bin values, and $m_i = \frac{h_i+k_i}{2}$.

While these measures have sound theoretical support, in information theory and in statistics they have some significant drawbacks. The first drawback is that they are not metrics in that they do not satisfy the symmetry requirement. However, this problem can easily be overcome by defining a modified distance between x and y that is in some way an average of $d(x,y)$ and $d(y,x)$—see Reference [45] for the Jeffrey divergence, which is a symmetric version of the Kullback-Leibler divergence.

A more significant drawback is that these measures are prone to errors due to bin boundaries. The distance between an image and a slightly darker version of itself can be great if pixels fall into an adjacent bin, as there is no consideration of adjacency of bins in these measures.

Earth Mover's Distance. The E**arth Mover's Distance (EMD)** is a distance measure that overcomes many of these problems that arise from the arbitrariness of binning. As the name implies, the distance is based on the notion of the amount of effort required to convert one image to another based on the analogy of transporting *mass* from one distribution to another. If we think of two images as distributions and view one distribution as a mass of earth in space and the other distribution as a hole (or set of holes) in the same space, then the EMD is the minimum amount of work involved in filling the holes with the earth.

In their analysis of the EMD Rubner et al. argue that a measure based on the notion of a *signature* is better than one based on a histogram. A signature $\{\mathbf{s}_j = \mathbf{m}_j, w_{\mathbf{m}_j}\}$ is a set of j clusters where $\mathbf{m}_j$ is a vector describing the mode of cluster j and $w_{\mathbf{m}_j}$ is the fraction of pixels falling into that cluster. Thus, a signature is a generalisation of the notion of a histogram where boundaries and the number of partitions are not set in advance; instead j should be "appropriate" to the complexity of the image [45].

The example in Figure 4 illustrates this idea. We can think of the clustering as a quantisation of the image in some colour space so the image is represented by a set of cluster modes and their weights. In the figure the source image is represented in a 2D space as two points of weights 0.6 and 0.4; the target image is represented by three points with weights 0.5, 0.3, and 0.2. In this example, the EMD is calculated to be the sum of the amounts moved (0.2, 0.2, 0.1, and 0.5) multiplied by the distances they are moved. Calculating the EMD involves discovering an assignment that minimises this amount.

For two images described by signatures $S = \{\mathbf{m}_j, w_{\mathbf{m}_j}\}_{j=1}^{n}$ and $Q = \{\mathbf{p}_k, w_{\mathbf{p}_k}\}_{k=1}^{r}$, we are interested in the work required to transfer from one to the other for a given flow pattern $\mathbf{F}$:

$$WORK(S,Q,\mathbf{F}) = \sum_{j=1}^{n}\sum_{k=1}^{r} d_{jk} f_{jk}, \tag{12}$$

where d_{jk} is the distance between clusters $\mathbf{m}_j$ and $\mathbf{p}_k$ and f_{jk} is the flow between $\mathbf{m}_j$ and $\mathbf{p}_k$ that minimises overall cost. An example of this in a 2D colour space is shown in Figure 4. Once

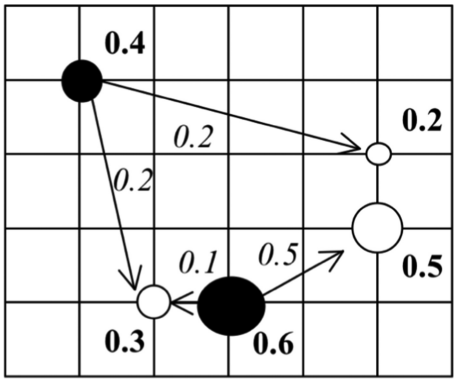

Fig. 4. An example of the EMD between two 2D signatures with two points (clusters) in one signature and three in the other (based on example in Reference [44]).

the transportation problem of identifying the flow that minimises effort is solved (using dynamic programming), the EMD is defined to be:

$$EMD(S,Q) = \frac{\sum_{j=1}^{n}\sum_{k=1}^{r} d_{jk} f_{jk}}{\sum_{j=1}^{n}\sum_{k=1}^{r} f_{jk}}. \tag{13}$$

Efficient algorithms for the EMD are described in Reference [45], however, this measure is expensive to compute with cost increasing more than linearly with the number of clusters. Nevertheless, it is an effective measure for capturing similarity between images.

Compression-based Dissimilarity. In recent years the idea of basing a similarity metric on compression has received a lot of attention [28, 34]. Indeed, Li et al. [34], refer to this as *The* similarity metric. The basic idea is quite straightforward; if two documents are very similar, then the compressed size of the two documents concatenated together will not be much greater than the compressed size of a single document. This will not be true for two documents that are very different. Slightly more formally, the difference between two documents A and B is related to the compressed size of document B when compressed using the codebook produced when compressing document A.

The theoretical basis of this metric is in the field of Kolmogorov complexity, specifically, in conditional Kolmogorov complexity:

$$d_{Kv}(x,y) = \frac{Kv(x|y) + Kv(y|x)}{Kv(xy)}, \tag{14}$$

where $Kv(x|y)$ is the length of the shortest program that computes x when y is given as an auxiliary input to the program, and $Kv(xy)$ is the length of the shortest program that outputs y concatenated to x. While this is an abstract idea, it can be approximated using compression:

$$d_C(x,y) = \frac{C(x|y) + C(y|x)}{C(xy)}. \tag{15}$$

$C(x)$ is the size of data x after compression, and $C(x|y)$ is the size of x after compressing it with the compression model built for y. If we assume that $Kv(x|y) = Kv(xy) - Kv(y)$, then we can define a normalised compression distance:

$$d_{NC}(x,y) = \frac{C(xy) - \min(C(x), C(y))}{\min(C(x), C(y))}. \tag{16}$$

It is important that $C(.)$ should be an appropriate compression metric for the data. Delany and Bridge [21] show that compression using Lempel-Ziv (GZip) is effective for text. They show that this compression-based metric is more accurate in k-NN classification than distance-based metrics on a bag-of-words representation of the text.

2.4 Similarity Metrics for Time Series

Time-series data can be as diverse as human activity measured by wearable sensors [37] or measurements coming from a manufacturing process. There is a long history of machine learning research on time-series analysis and 1-NN is the baseline metric for time-series classification [5]. However, the special characteristics of time series do present challenges for k-NN. Consider a query time-series $\mathbf{q}$ and a target $\mathbf{x}$:

$$\mathbf{q} = q_1, q_2, \ldots, q_j, \ldots, q_m, \tag{17}$$

$$\mathbf{x} = x_1, x_2, \ldots, x_j, \ldots, x_n. \tag{18}$$

While both time series are vectors, the Euclidean distance between these two vectors may be quite large even if they have the same general shape (see Figure 5). Furthermore, the two time-series might be of different lengths. To complicate things further, similarity might depend on specific features (motifs) in the time series rather than similarity across the time series as a whole.

A number of methods for scoring similarity between time series have been developed that allow k-NN to work with time-series data. Three popular methods are:

- **Dynamic Time Warping (DTW):** Because two time series may be fundamentally similar but offset or slightly distorted, DTW allows the time axis to be *warped* to identify underlying similarities [29].
- **Symbolic Aggregate Approximation (SAX):** The idea with SAX is to discretise the time series so it can be represented as a sequence of symbols [35]. Then methods for scoring sequence similarity can be applied.
- **Symbolic Fourier Approximation (SFA):** SFA is like SAX except the sequence representation is produced from a discrete Fourier transform representation of the signal rather than a discretisation of the signal itself. So, SFA is a frequency domain rather than a time domain representation of the signal [47].

By far, the most popular of these is DTW, so we will provide some detail here on how DTW works. As the name suggests, the idea is to allow the time series to be stretched (warped) to find the best mapping. The DTW distance is defined as follows:

$$DTW(\mathbf{q}, \mathbf{x}) = \min_{\pi} \sqrt{\sum_{(i,j)\in\pi} d(q_i, x_j)^2}, \tag{19}$$

where $\pi = [\pi_1, \ldots, \pi_l, \ldots, \pi_L]$ is the optimum path (mapping) having the following properties:

- $m = |\mathbf{q}|, n = |\mathbf{x}|$,
- $\pi_1 = (1, 1), \pi_L = (m, n)$,
- $\pi_{l+1} - \pi_l \in \{(1, 0), (0, 1), (1, 1)\}$.

The DTW path for the two time series in Figure 5 is shown on the right. It starts at the top left (1,1) and finishes at the bottom right (m, n). Each point (i, j) on the path indicates the mapping between q_i and x_j. The extent of the deviation from the main diagonal reflects the *warping*. In practice, the path may be restricted to a band around the main diagonal to restrict warping.

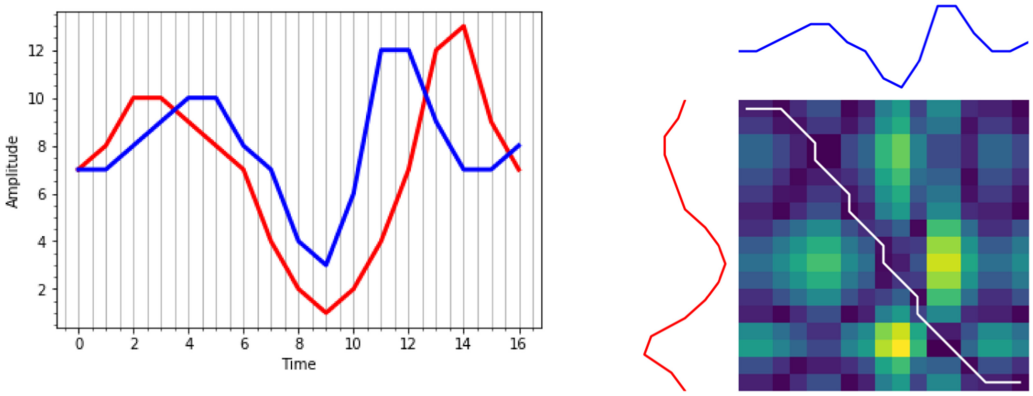

Fig. 5. The image on the left illustrates the main challenge in quantifying similarity between time series. The two series are similar but the Euclidean distance between them is large. The image on the right shows the DTW mapping between the two time series (produced using tslearn [51]).

The computational complexity of DTW is $O(n, m)$, because it entails a search through the matrix shown on the right is Figure 5. This is effectively $O(n^2)$ in the length of the time-series—so DTW is computationally expensive.

Finally, DTW is not a proper metric, because it fails two of the criteria laid out at the beginning of this section. $DTW(\mathbf{q}, \mathbf{x}) = 0 \nRightarrow \mathbf{x} = \mathbf{x}$ and the triangle inequality may not hold. This means that speedup mechanisms such as Ball Trees (Section 3.2) that work for proper similarity metrics cannot be applied. Neither can mechanisms that work for vector space representations, i.e., Kd-Trees (Section 3.1) and Random Projection Trees (Section 3.3.2).

2.5 Metric Learning

So far, we have considered scenarios where the system designer selects a metric that is considered appropriate based on their insight into the data. It is also possible to *induce* a metric from the data. This Metric Learning has been the subject of extensive research [53, 54]. In this section, we briefly outline two general strategies:

- **Linear Discriminant Analysis (LDA)** is a linear projection method similar to PCA (see Section 4.1). However, LDA is *supervised* in the sense that the objective is to discover a projection that does a good job of separating the classes. Thus, LDA can be used as the basis of a learned metric for k-NN.
- **The Mahalanobis Distance** is defined to be:

$$d_M(x, y) = \sqrt{(\mathbf{x} - \mathbf{y})^\top \mathbf{S}(\mathbf{x} - \mathbf{y})}, \tag{20}$$

  where $\mathbf{S}$ is the covariance matrix of the data. If we replace $\mathbf{S}$ with any Positive Semi-Definite Matrix $\mathbf{M}$, then we have a very general metric that can be learned from the data—this is the Generalized Mahalanobis distance [53]:

$$d_{GM}(x, y) = \sqrt{(\mathbf{x} - \mathbf{y})^\top \mathbf{M}(\mathbf{x} - \mathbf{y})}. \tag{21}$$

Of these two approaches, strategies based on Generalized Mahalanobis distance have received the most attention [53]. If the data is described by m features, then $\mathbf{M}$ is an $m \times m$ matrix so the learned metric has considerable expressive power. Appendix I provides a link to Python code presenting an example of metric learning based on the large margin nearest neighbor method [54].

Code for metric learning is not included in scikit-learn, but a dedicated metric learning extension for scikit-learn is available [19].

3 COMPUTATIONAL COMPLEXITY

Computationally expensive metrics such as the Earth-Mover's Distance and compression-based (dis)similarity metrics focus attention on the computational issues associated with k-NN classifiers. Basic k-NN classifiers that use a simple Minkowski distance will have a time behaviour that is $O(dn)$ where n is the number of data samples and d is the number of features that describe the data, i.e., the distance metric is linear in the number of features and the comparison process increases linearly with the amount of data. The computational complexity of the EMD and compression metrics is more difficult to characterise, but a k-NN classifier that incorporates an EMD metric is likely to be $O(nc^3 \log c)$ where c is the number of clusters [45].

For these reasons there has been considerable research on editing down the training data and on reducing the number of features used to describe the data (see Section 4). There has also been considerable research on alternatives to the exhaustive search strategy (brute force) that is used in the standard k-NN algorithm. In the remainder of this section, we review Kd-Trees and Cover Trees, the two speedup strategies included in Scikit-learn. We also review some approximate k-NN algorithms that do not guarantee to retrieve nearest neighbours but offer dramatic speedup with little loss of accuracy. In the final sub-section, a simple comparison of Kd-Trees and Cover Trees against brute force search is presented.

3.1 Kd Trees

Kd-Trees represent the longest established strategy for speedup in k-NN [6]. It is best to think of Kd-Trees as a general strategy rather than a single algorithm. The general idea is that a binary tree is used to successively partition the dataset with training samples sorted to the leaves of the tree. This offers the potential for retrieval time that is $O(d \log(n))$ rather than $O(dn)$.

A sample Kd-Tree is shown in Figure 6. The data is described by two features so it can be represented as a 2D plot. The plot on the left corresponds to the binary tree on the right. The Kd-Tree always partitions the data along hyperplanes (lines in the 2D case) that are perpendicular to the axes.

The figure shows a query point $Q(2, 5)$. The search for nearest neighbours for Q will locate it to the appropriate node in the tree $G(2, 6)$. It can be seen in the plot that the nearest neighbour for Q is not guaranteed to be located in the hypercube represented by G. However, the distance between G and Q gives us an upper bound on the distance to the nearest neighbour. It is clear that the grey box (hypercube) needs also to be considered; the rest of the tree can be bounded out from consideration. It is this potential to bound out large parts of the data that yields the $O(d \log(n))$ performance.

Some other aspects of Kd-Trees that need to be considered are as follows:

- **Constructing Kd-Trees** entails a straightforward binary partitioning of the data and sorting the data to leaf nodes so the construction process is comparatively quick. The partition is typically at the median value for the selected feature.
- At each step in the building of the tree, a decision has to be made on **feature selection**. The policy could be to cycle through the features in order or to select the feature in which the variance (spread) in the data is highest.
- The query time will increase with the number of neighbours required (k). For very large k, query time will exceed that for brute force search.

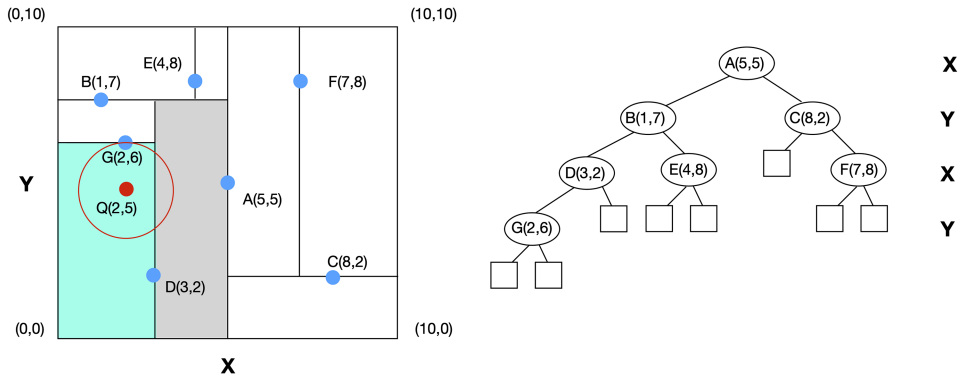

Fig. 6. A Kd-Tree based on the example in the original paper by Bentley [7]). The partitioning of the 2D feature space shown on the left corresponds to the tree on the right.

- The $O(d\log(n))$ retrieval time depends on a **balanced tree**. If the tree is not well balanced, then some retrieval times will be poor [6].
- The main drawback with Kd-Trees is that the **curse of dimensionality** still applies. The benefits of using a binary tree as an indexing structure cease to apply when d is large (say, > 20). Kleinberg [30] points out that when $d > log(n)$, $O(d\log(n))$ is no better than $O(dn)$.

3.2 Ball Trees

Figure 6 shows that a Kd-Tree indexes the data by partitioning the feature space. By contrast, a Ball Tree is a "metric tree" in the sense that it is based on a metric defined on pairs of samples [23, 59]. The construction of the ball tree is akin to a hierarchical clustering problem that can be tackled top-down or bottom-up:

- **Bottom-up:** Initially, each data point is a point sized bounding ball. At each step, select the closest pair of balls, the pair that have the smallest bounding ball that covers them. Join these balls. Continue until the top-level bounding ball is reached.
- **Top-down:** At each step, two data points are chosen that have the maximum distance between them. The remaining points are partitioned by allocating to the closer or these. This process is repeated recursively until a stopping criterion is met, e.g., number of samples at a leaf node.

In contrast to Kd-Trees, the construction of a Ball tree depends on a metric defined on the data rather than a feature space representation. However, it should be noted that the distance measure must be a metric so a Ball Tree cannot be applied for measures such as Earth Mover's Distance or Dynamic Time Warping. Compared with Kd-Trees, Ball Trees have the potential to perform better for high-dimension data, for example, in image analysis [33].

3.3 Approximate k-NN

Brute force search for k-NN is $O(dn)$. As we have seen in the preceding sections, we can get over the linear dependence on n but high dimension data is still a problem. Fortunately, for many applications, it is not essential to retrieve the absolute nearest neighbours. For instance, in recommender systems, the most similar item is not necessarily required—indeed, items that are reasonably close may offer some serendipitous discovery. In k-NN classification, neighbours that are close (but not necessarily closest) are probably of the correct class. So, in this section, we review the two most

Table 1. Overview of the Datasets Used to Test the Speedup Algorithms

	HTRU	Shuttle	Letter	Credit
Samples	17,898	43,494	20,000	30,000
Features	8	9	16	23
Winning Alg.	kd-tree	kd-tree	kd-tree	brute-force

popular strategies for Approximate k-NN; these are Locality Sensitive Hashing and Random Projection Trees [36].

3.3.1 Locality Sensitive Hashing. With **Locality Sensitive Hashing (LSH)** the objective is to map similar items into the same "buckets" with high probability. This contrasts with conventional hashing where the objective of minimising hashing collisions means that similar items will have very different hashes. Given that LSH maps similar items to the same buckets, it can be used to implement approximate nearest neighbour search. The strategy is to use a number of variants of LSH algorithms to retrieve a candidate set of nearest neighbours. This candidate set is the union of the items in the buckets returned by the LSH algorithms. Then the similarity metric can be applied to these candidates to find nearest neighbours that will be near-optimal [27, 36].

3.3.2 Random Projection Trees. In Section 3.1, we saw that exact nearest neighbour search is a two-stage process. First, the query is located to the correct leaf node in the tree and candidate nearest neighbours are identified. Then there is a backtracking process that finds better candidates or bounds out sections of the tree from consideration. The retrieval of nearest neighbours is guaranteed without explicitly measuring against all data points. Random Projection Trees depends on two extensions to this basic Kd-tree idea:

- **Defeatist Search:** The query item is located to the correct leaf node but the backtracking process to ensure optimality is dropped or at least greatly curtailed [36]. The search gives up early, which might be considered a bit *defeatist*.
- **Multiple Trees:** If the search returns without backtracking, then the prospect of finding good neighbours can be improved by repeating with multiple trees. Different variants of the tree can be produced by including a random element in the Kd-Tree generation process [49]. Since there is a risk that there will not be great variety in the Kd-Tree variants, it is common to produce different trees by randomly projecting the data into a different space (i.e., perform a simple linear transformation on the data) [30].

In the next section, we provide a demonstration of the effectiveness of Approximate k-NN on a number of datasets. The results show significant speedup with little or no loss of accuracy. The caveat is that it only works for feature vector data.

3.4 Speedup Evaluation

The objective in this section is to show the potential speedup that is possible with these methods—it is not meant as a comprehensive evaluation. In our first evaluation, we assess the three options available in the k-NN implementation in scikit-learn (scikit-learn.org). These are brute-force search, Kd-Trees, and Ball Trees. The evaluation covers the four datasets summarised in Table 1. Two of these datasets are low dimension (< 10). The Credit dataset would be considered high-dimension with 23 features.

We present two sets of results (see Figure 7), one using 2-fold cross-validation and one using 10-fold. The objective is to show the impact of the tree building phase; while both cross-validations

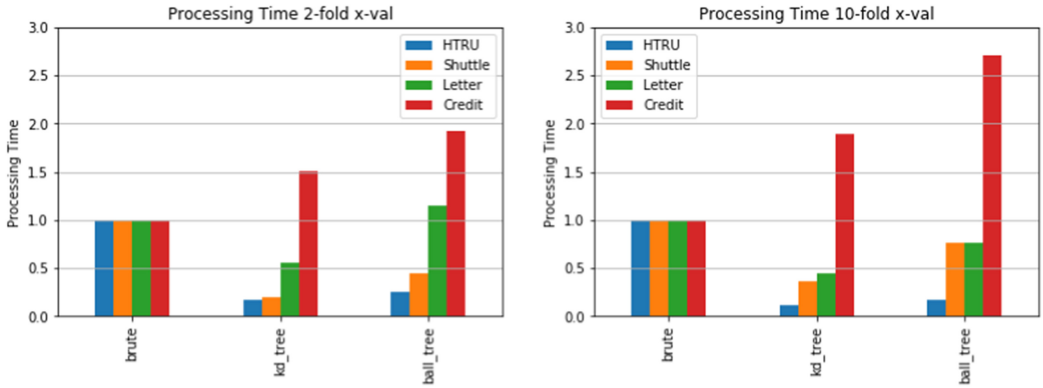

Fig. 7. Processing time for Kd-Tree and Ball Tree compared with brute force search. Times are normalised to the brute force time. (<1 is an improvement). 2-fold cross-validation on the left and 10-fold on the right.

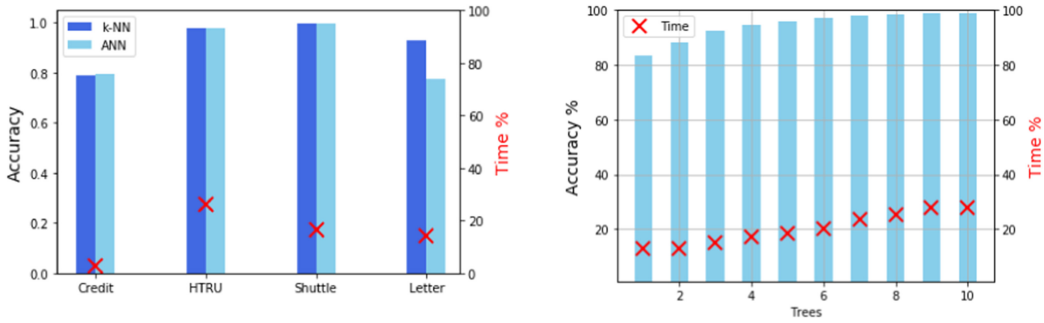

Fig. 8. The plot on the left shows the accuracy of ANN using a single tree compared with "full search" k-NN. The time saving is significant. The plot on the right shows that accuracy on the Letter dataset can be improved with the addition of more trees.

use all the data for testing, the 10-fold cross-validation incurs the tree building overhead 10 times instead of twice.

The bar charts show the processing time divided by the time for brute force search. It is clear that significant speedup is possible for the low-dimension datasets. The Kd-Tree results are slightly better than Ball Tree in all cases. However, the performance for the Credit dataset is worse than brute-force search. This is to be expected, given that it is high-dimension.

This poor performance on high-dimension data shows that the curse of dimensionality cannot be avoided in exact nearest neighbour search. Figure 8 shows the speedup that can be achieved with Approximate Nearest Neighbour. The method we evaluate is called Annoy (github.com/spotify/annoy) and uses Random Projection Trees [9]. Annoy stands for "Approximate Nearest Neighbor Oh Yeah" but the name probably stems from the fact that the method is "annoyingly" effective. The results in Figure 8 show dramatic speedup with almost no loss of accuracy except for the Letter dataset. When more trees are added, the accuracy reaches that achievable with exact k-NN with a four-fold improvement in processing time.

4 DIMENSION REDUCTION

Given the high dimension nature of the data, Dimension Reduction is a core research topic in Machine Learning. Research on Dimension Reduction has itself two dimensions; the dimensions

of a dataset of $|D|$ examples described by $|F|$ features can be reduced by selecting a subset of the examples or by selecting a subset of the features (an alternative to this is to transform the data into a representation with less features). It is important to emphasise that dimension reduction through feature selection can be achieved without loss of information, because the *intrinsic* dimension of the data may be considerably less than the number of features. This notion of intrinsic dimension is discussed in Section 4.1.

Dimension reduction as achieved by supervised feature selection is described in Section 4.2. Unsupervised feature transformation using **Principal Component Analysis (PCA)** [15] can be used as a preprocessing step for k-NN [4]. PCA is discussed in the next section in the context of intrinsic dimension. However, there is no evidence that PCA can be combined with k-NN without sacrificing accuracy, so PCA will not be covered in this article. The other aspect of dimension reduction is the deletion of redundant or noisy instances in the training data—this is reviewed in Section 4.3.

4.1 Intrinsic Dimension

Colloquially, we can think of the intrinsic dimension as the minimum number of features required to provide a "good" representation of the data. This notion of a "good" representation can be considered in terms of PCA. We can represent a dataset D as a rectangular matrix $\mathbf{D}$ of dimension $n \times p$, that is, n samples described by p features. If we perform PCA on $\mathbf{D}$, then we get:

$$\mathbf{T}_{n\times r} = \mathbf{D}_{n\times p}\mathbf{W}_{p\times r}. \tag{22}$$

The PCA provides a linear mapping of the data into a lower dimension representation $\mathbf{T}$. The PCA also provides a ranking of the **principal components (PCs)** in terms of the variance in the data that they capture. We can select the top s PCs that together capture $(1-\epsilon)$ fraction of the variance in the data. This s is an approximation of the intrinsic dimension of the data.

$$\mathbf{T}_{n\times s} = \mathbf{D}_{n\times p}\mathbf{W}_{p\times s} \tag{23}$$

The variance captured by the first four PCs for the HTRU and Shuttle datasets is shown in Figure 9. The four PCs capture almost all of the variance for the HTRU data but less than 80% for Shuttle. We can think of four as a reasonable assessment of the intrinsic dimension of the HTRU data, whereas the intrinsic dimension for Shuttle is greater than four.

This PCA-inspired notion of intrinsic dimension is a global approximation and there may be parts of the space where the intrinsic dimension is locally less than s. Imagine a neighbourhood of radius r around a point $\mathbf{q}$ (e.g., among the k nearest neighbours), $(1-\epsilon)$ fraction of the variance will be covered by s' features, where $s' < p$.

Dasgupta & Freund [18] provide an insightful example to explain intrinsic dimension. Imagine a motion capture system with 13 markers attached to a person to facilitate processing (see Figure 10). In a 2D image, these markers can be represented by 26 x, y coordinates. So, the dimension of the motion capture data will be 26. However, it is clear from Figure 10(c) that very many points in the 26D space are not reachable. This system does not really have 26 degrees of freedom. Instead, the person could be represented by the joint angles in the body, a number much smaller than 26.

4.2 Feature Selection

When the objective is to reduce the number of features used to describe data there are two strategies that can be employed. Techniques such as PCA may be employed to *transform* the data into a lower dimension representation. Alternatively, feature selection may be employed to *discard* some

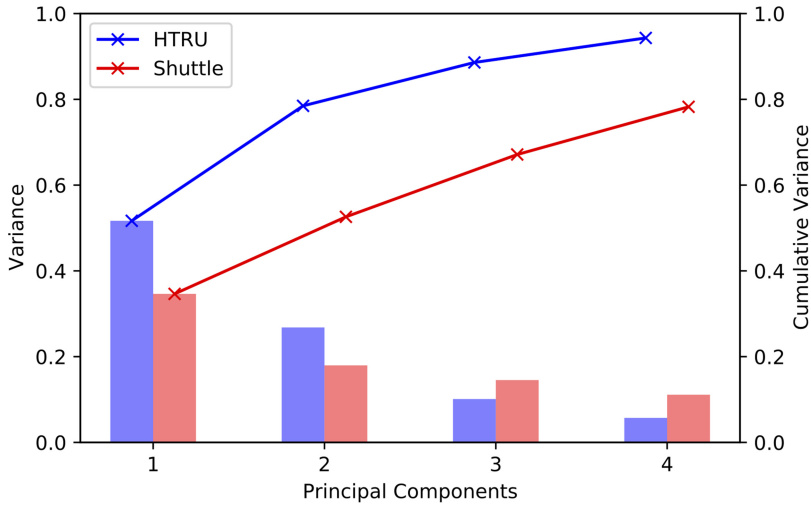

Fig. 9. The first four principal components of the HTRU and Shuttle datasets.

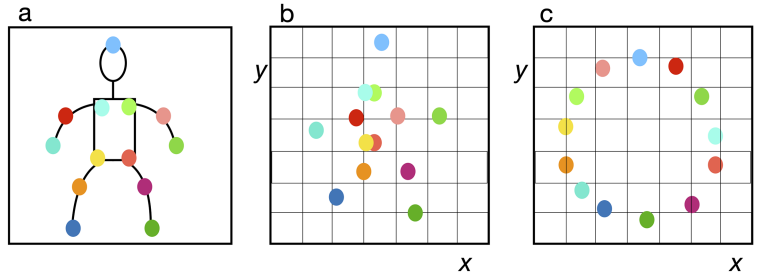

Fig. 10. Consider a simple motion capture system where 13 coloured balls capture the motion of the stick figure (a). (b) shows a valid configuration of these balls. (c) shows a configuration in this space that is not reachable. (Motivated by example in Reference [18].)

of the features. In using k-NN with high dimension data, there are several reasons why it is useful to perform feature selection:

— For many distance measures, the retrieval time increases directly with the number of features (see Section 3).
— Noisy or irrelevant features can have the same influence on retrieval as predictive features, so they will impact negatively on accuracy.
— Things look more similar on average the more features used to describe them (see Figure 11).

Feature Selection techniques typically incorporate a search strategy for exploring the space of feature subsets, including methods for determining a suitable starting point and generating successive candidate subsets, and an evaluation criterion to rate and compare the candidates, which serves to guide the search process. The evaluation schemes can be divided into two broad categories:

— **Filter** approaches attempt to remove irrelevant features from the feature set prior to the application of the learning algorithm. Initially, the data is analysed to identify those dimensions

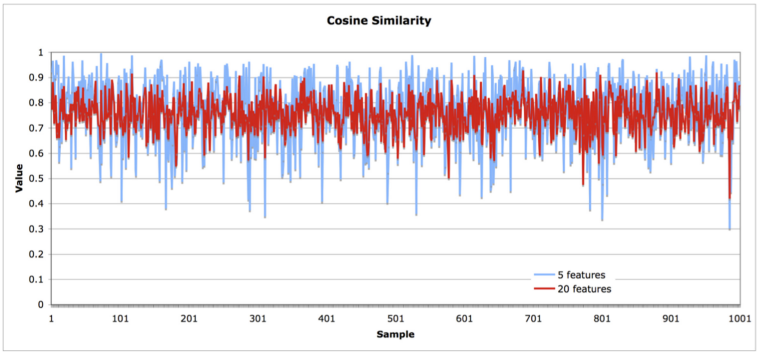

Fig. 11. The more dimensions used to describe objects, the more similar on average things appear. This figure shows the cosine similarity between objects described by 5 and by 20 features. It is clear that in 20 dimensions similarity has a lower variance than in 5.

that are most relevant for describing its structure. The chosen feature subset is subsequently used to train the learning algorithm. Feedback regarding an algorithm's performance is not required during the selection process, though it may be useful when attempting to gauge the effectiveness of the filter.

— **Wrapper** methods for feature selection make use of the learning algorithm itself to choose a set of relevant features. The wrapper conducts a search through the feature space, evaluating candidate feature subsets by estimating the predictive accuracy of the classifier built on that subset. The goal of the search is to find the subset that maximises this criterion.

It is worth mentioning at this point that some other classification techniques perform implicit feature selection. For instance, the process of building a decision tree will very often not select all the features for use in the tree. Features not used in the tree have no role then in classification.

Filter Techniques. Central to the Filter strategy for feature selection is the criterion used to score the predictiveness of the features. In recent years **Information Gain (IG)** has become perhaps the most popular criterion for feature selection. The Information Gain of a feature is a measure of the amount of information that a feature brings to the training set [41]. It is defined as the expected reduction in entropy caused by partitioning the training set D using the feature f as shown in Equation (24), where D_v is that subset of the training set D where feature f has value v.

$$IG(D, f) = Entropy(D) - \sum_{v \in values(f)} \frac{|D_v|}{|D|} Entropy(D_v). \tag{24}$$

Entropy is a measure of how much randomness or impurity there is in the dataset. It is defined in Equation (14), where c equals the number of classes in the training set and pi is the proportion of class i in the data—entropy is highest when the proportions are equal.

$$Entropy(D) = \sum_{i=1}^{c} -p_i \log_2 p_i \tag{25}$$

In binary classification, this can be simplified to $Entropy(D) = -p_+ \log_2 p_+ - p_- \log_2 p_-$ where p_+ represents the class and p_- the non-class. For comparison purposes, we will also consider **Odds Ratio (OR)** [39], which is an alternative filtering criterion. For binary classification, OR calculates

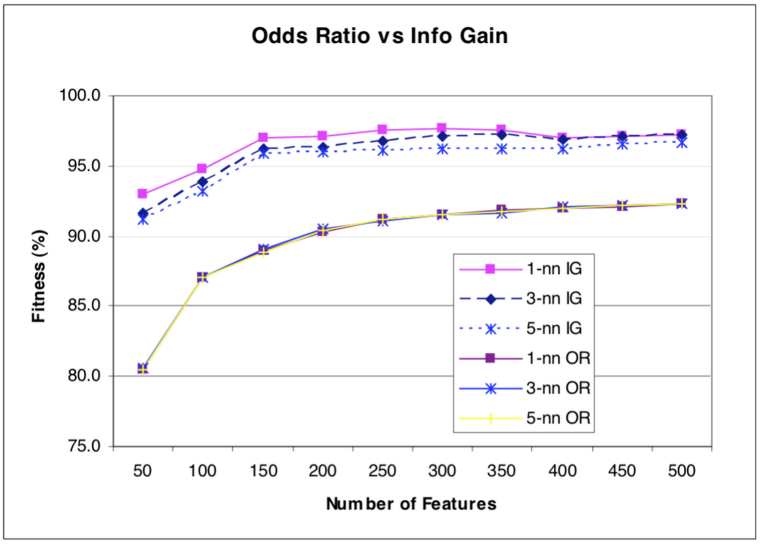

Fig. 12. Comparing Information Gain with Odds Ratio. Results of the average of three 10-fold cross-validation experiments on a dataset of 1,000 emails—500 spam and 500 legitimate—where word features only were used.

the ratio of the odds of a feature occurring in the class to the odds of the feature occurring in the non-class:

$$OR(f, c) = \frac{Odds(f|c)}{Odds(f|\bar{c})}. \tag{26}$$

Where a specific feature does not occur in a class, it can be assigned a small fixed value so the OR can still be calculated. For feature selection, the features can be ranked according to their OR with high values, indicating features that are very predictive of the class. The same can be done for the non-class to highlight features that are predictive of the non-class.

We can look at the impact of these feature selection criteria in an email spam classification task. In this experiment, we selected the $\frac{n}{2}n$ features with the highest IG value and n features each from $OR(f, spam)$ and $OR(f, nonspam)$ sets. The results, displayed in Figure 12, show that IG performed significantly better than OR. The reason for this is that OR is inclined to select features that occur rarely but are very strong indicators of the class. This means that some objects (emails) are described by no features and thus have no similarity to any cases in the case base. In this experiment, this occurs in 8.8% of cases with OR compared with 0.2% for the IG technique. This shows a simple but effective strategy for feature selection in very high dimension data. IG can be used to rank features, then a cross-validation process can be employed to identify the number of features above which classification accuracy is not improved. This evaluation suggests that the top 350 features as ranked by IG are adequate.

While this is an effective strategy for feature selection, it has the drawback that features are considered in isolation, so redundancies or dependancies are ignored. Two strongly correlated features may both have high IG scores but one may be redundant once the other is selected. More sophisticated filter techniques that address these issues using Mutual Information to score *groups* of features have been researched by Novovičová et al. [40] and have been shown to be more effective than these simple Filter techniques.

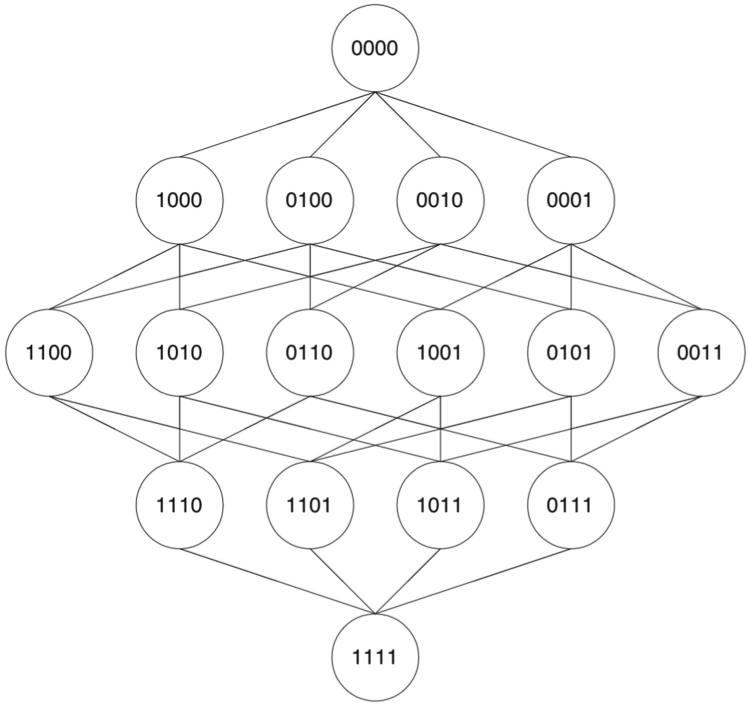

Fig. 13. The feature subspace.

Wrapper Techniques. The obvious criticism of the Filter approach to feature selection is that the filter criterion is separate from the induction algorithm used in the classifier. This is overcome in the Wrapper approach by using the performance of the classifier to guide search in feature selection—the classifier is wrapped in the feature selection process [31]. In this way, the merit of a feature subset is the generalisation accuracy it offers as estimated using cross-validation on the training data. If 10-fold cross-validation is used, then 10 classifiers will be built and tested for each feature subset evaluated—so the wrapper strategy is very computationally expensive. If there are p features under consideration, then the search space is of size 2^p, so it is an exponential search problem.

A simple example of the search space for feature selection where $p = 4$ is shown in Figure 13. Each node is defined by a feature mask; the node at the top of the figure has no features selected, while the node at the bottom has all features selected. For large values of p, an exhaustive search is not practical because of the exponential nature of the search.

The two most popular strategies are:

— Forward Selection, which starts with no features selected, evaluates all the options with just one feature, selects the best of these, and considers the options with that feature plus one other, and so on.
— Backward Elimination starts with all features selected, considers the options with one feature deleted, selects the best of these, and continues to eliminate features.

These strategies will terminate when adding (or deleting) a feature will not produce an improvement in classification accuracy as assessed by cross-validation. Both of these are greedy search strategies and so are not guaranteed to discover the best feature subset. More sophisticated search

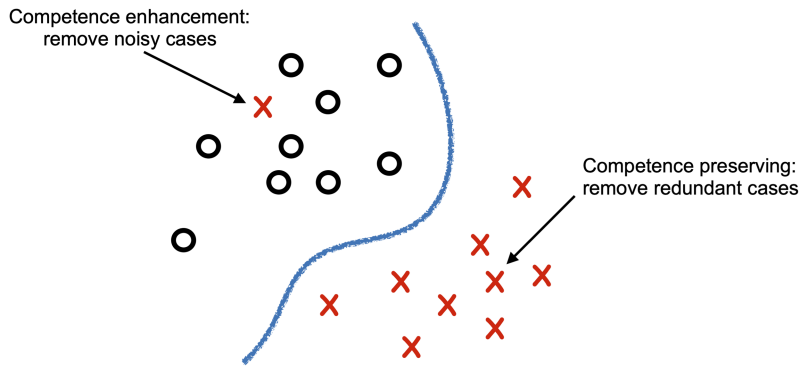

Fig. 14. Instance selection techniques demonstrating competence preservation (redundancy reduction) and competence enhancement (noise reduction).

strategies can be employed to better explore the search space; however, Reunanen [42] cautions that more intensive search strategies are more likely to overfit the training data.

4.3 Instance Selection and Noise Reduction

The second aspect of dimension reduction is instance selection, reducing the size of the training set by removing redundant or noisy instances while maintaining or even improving performance. This aspect of dimension reduction is explored and researched in two different areas, nearest-neighbour classification, and **case-based reasoning (CBR)**. It is known as Instance Selection or Prototype Selection by those who focus on nearest neighbour classification [24] and Case-base Editing or Case-base Maintenance by the CBR community [38].

Instance selection techniques can be categorised as competence preservation or competence enhancement techniques [12]. Competence preservation corresponds to redundancy reduction, removing superfluous instances that do not contribute to classification competence. Competence enhancement is effectively noise reduction, removing noisy or corrupt instances from the training set. Figure 14 illustrates both of these with a classification example, where instances of one class are represented by stars, and instances of the other class are represented by circles. Competence preservation techniques aim to remove internal instances in a cluster of instances of the same class and can predispose towards preserving noisy instances as exceptions or border cases.

Noise reduction, however, aims to remove noisy or corrupt instances but can remove exceptional or border instances that may not be distinguishable from true noise, so a balance of both can be useful. Techniques that combine a balance of both redundancy reduction and noise removal are known as hybrid approaches.

Editing strategies normally operate in one of two ways: *incremental*, which involves adding selected instances from the training set to an initially empty edited set, and *decremental*, which involves contracting the training set by removing selected instances.

Early Techniques. An early competence preservation technique is Hart's **Condensed Nearest Neighbour (CNN)** [26]. CNN is an incremental technique that adds to an initially empty edited set any instance from the training set that cannot be classified correctly by the edited set. This technique is very sensitive to noise and to the order of presentation of the training set instances—in fact, CNN by definition will tend to preserve noisy instances. Improvements on CNN included the **Selective NN (SNN)** [43], which imposes the rule that every instance in the training set must be closer to an instance of the same class in the edited set than to any other training instance of a different class. The Reduced NN Rule [25] took the opposite, decremental approach, removing

an instance from the training set where its removal does not cause any other instance to be misclassified. This technique will allow for the removal of noisy cases but is sensitive to the order of presentation of cases.

Competence enhancement or noise reduction techniques start with Wilson's **Edited Nearest Neighbour (ENN)** algorithm [55], a decremental strategy, which removes instances from the training set that do not agree with their k nearest neighbours. These instances are considered to be noise and appear as exceptional examples in a group of instances of the same class. Extensions to ENN include the **repeated ENN (RENN)** and the *all k-NN* algorithms [52]. Both make multiple passes over the training set, the former repeating the ENN algorithm until no further eliminations can be made from the training set and the latter using incrementing values of k. These techniques focus on noisy or exceptional instances and do not result in the same storage reduction gains as the competence preservation approaches.

Hybrid techniques were introduced with a series of instance based learning IBn algorithms[1]. IB2 is similar to CNN, adding only instances that cannot be classified correctly by the reduced training set. IB2's susceptibility to noise is handled by IB3, which records how well instances are classifying and only keeps those that classify correctly to a statistically significant degree. Other researchers have provided variations on the IBn algorithms [13, 14, 58].

Competence-based Case-base Editing. Approaches to case-base editing build a competence model of the training data and use the competence properties of the cases to determine which cases to include in the edited set. Measuring and using case competence to guide case-base maintenance was first introduced by Smyth and Keane [50] and developed by Zhu and Yang [59]. Smyth and Keane [50] introduce two important competence properties, the *reachability* and *coverage* sets for a case in a case-base. The *reachability set* of a case c is the set of all cases that can successfully classify c, and the coverage set of a case c is the set of all cases that c can successfully classify. The coverage and reachability sets represent the local competence characteristics of a case and are used as the basis of a number of editing techniques.

A family of competence-guided editing methods for case-bases combine both incremental and decremental strategies using a combination of rules [38]:

(1) an *ordering policy* for the presentation of the cases that is based on the competence characteristics of the cases;
(2) an *addition rule* to determine the cases to be added to the edited set;
(3) a *deletion rule* to determine the cases to be removed from the training set; and
(4) an *update policy*, which indicates whether the competence model is updated after each editing step.

One of these algorithms, **Conservative Redundancy Removal (CRR)** [22], is included in the assessment in Section 4.3.1. This algorithm is similar in concept to the FCNN rule [3], which can be applied to huge collections of data.

Other approaches also use the coverage and reachability properties of cases. **Iterative Case Filtering (ICF)** [12] is a decremental strategy contracting the training set by removing those cases c, where the number of other cases that can correctly classify c is higher that the number of cases that c can correctly classify. This strategy focuses on removing cases far from class borders. After each pass over the training set, the competence model is updated and the process repeated until no more cases can be removed. ICF includes a pre-processing noise reduction stage, effectively RENN, to remove noisy cases. McKenna and Smyth compared their family of algorithms to ICF and concluded that the overall best algorithm of the family delivered improved accuracy (albeit marginal, 0.22%) with less than 50% of the cases needed by the ICF edited set [38].

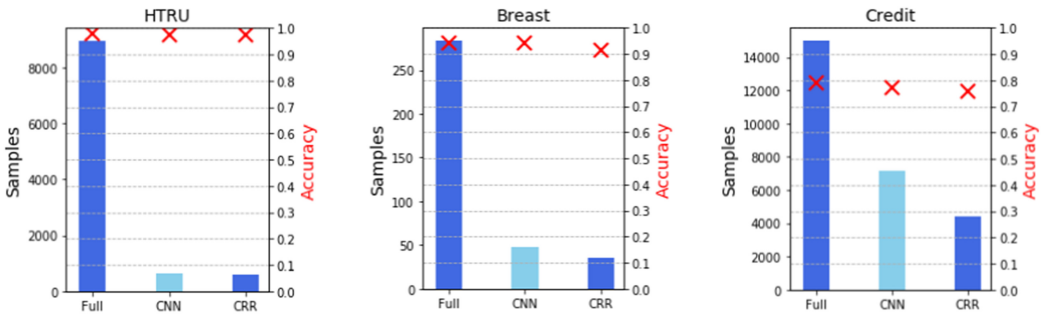

Fig. 15. The impact of CNN and CRR on training set size and accuracy.

Wilson and Martinez [57] present a series of reduction techniques called DROP1 to DROP5,[1] which, although published before the definitions of coverage and reachability, could also be considered to use a competence model. They define the set of associates of a case c, which is comparable to the coverage set of McKenna and Smyth except that the associates set will include cases of a different class from case c, whereas the coverage set will only include cases of the same class as c.

The DROPn algorithms use a decremental strategy. Their comprehensive evaluation found DROP3 to be the best mix of generalisation accuracy and storage requirements, performing consistently well in comparison with other reduction techniques. A comparison of ICF against DROP3 found that neither algorithm consistently out-performed the other and both represented the "cutting edge in instance set reduction techniques" [12].

4.3.1 Instance Selection Performance. Figure 15 shows the impact of two instance selection techniques (CNN & CRR) on training set size and generalisation accuracy. The evaluation shows that, at least for some datasets, the training set size can be dramatically reduced with almost no impact on generalisation accuracy. If there is a lot of redundancy in the training data, then dramatic speedup can be achieved through instance selection without any significant impact on accuracy.

5 MODEL SELECTION

With the large number of decisions to be made in setting up a k-NN classifier, model selection and hyper-parameter tuning is a key part of the process. Model selection is recognised as fundamental to Machine Learning, and toolkits such as scikit-learn include extensive support for model selection and parameter tuning. The fundamental strategy is to use cross-validation to evaluate the model/hyper-parameter combinations [8].

Even a simple k-NN deployment will require some model choices, for example:

- What is the best value for k?
- What distance measure is best for the data, e.g., Cosine, Euclidean, Correlation?
- Does it help to include distance weighting?

The normal practice is to use grid-search to explore this hyper-parameter space, as shown in Figure 16. If five alternatives for k are considered and three alternative distance measures, then a grid of 3×5 needs to be explored. When two instance weighting alternatives are added, the grid expands to $3 \times 5 \times 2$. These alternatives can be tested using cross-validation on the training data and the generalisation accuracy of the best combination can be assessed using a hold-out set. A link to Python code for grid search is provided in the Appendix.

[1]Three of these algorithms were originally published in [56] as Reduction Techniques (RT1 to RT3).

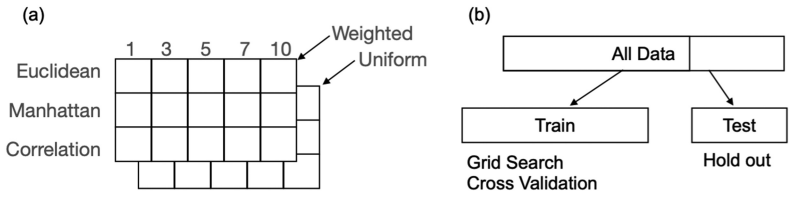

Fig. 16. (a) Grid Search for k-NN considering k, the distance metric and instance weighting policy (Weighted or Uniform).
(b) Typically, Grid Search will be performed on the training data with the impact assessed on a hold-out set.

6 CONCLUSION: ADVANTAGES AND DISADVANTAGES

k-NN is very simple to understand and easy to implement. So, it should be considered in seeking a solution to any classification problem. Some advantages of k-NN are as follows (many of these derive from its simplicity and interpretability):

— Because the process is transparent, it is easy to implement and debug.
— k-NN can be applied to data that cannot be described as a feature vector provided a similarity measure is available. Thus, k-NN can be used in situations where other ML mechanisms will not be applicable.
— In situations where an explanation of the output of the classifier is useful, k-NN can be very effective if an analysis of the neighbours is useful as explanation.
— There are some noise reduction techniques that work only for k-NN that can be effective in improving the accuracy of the classifier [22].
— In some circumstances, speedup mechanisms such as Kd-Trees or Ball Trees can improve retrieval times without any loss of accuracy.
— Approximate Nearest Neighbour techniques can greatly improve retrieval times, sometimes with minimal impact on accuracy [30].

These advantages of k-NN, particularly those that derive from its interpretability, should not be underestimated. However, some significant disadvantages are as follows:

— Because all the work is done at runtime, k-NN can have poor runtime performance if the training set is large.
— k-NN is very sensitive to irrelevant or redundant features, because all features contribute to the similarity (see Equation (1)) and thus to the classification. This can be ameliorated by careful feature selection or feature weighting.
— On very difficult classification tasks, k-NN may be outperformed by more *exotic* techniques such as Support Vector Machines or Neural Networks.

A APPENDIX I: PYTHON CODE

The GitHub repository[2] associated with this article contains the following Python Notebooks:

- kNN-Basic: Code for a basic k-NN classifier in scikit-learn.
- kNN-Correlation: How to use correlation as the k-NN metric in scikit-learn (see Section 2.2).
- kNN-Cosine: How to use Cosine as the k-NN metric in scikit-learn. Using Cosine similarity for text classification (Section 2.1).

[2] https://github.com/PadraigC/kNNTutorial.

- kNN-DTW: Using the tslearn library for time-series classification using DTW (Section 2.4).
- kNN-MetricLearn: Using the metric-learn library to learn a similarity metric.
- kNN-Speedup: scikit-learn provides some options for speeding up nearest neighbour retrieval. This notebook tests the speedup on four datasets (Section 3.4).
- kNN-Annoy: Testing the speedup offered by the Approximate Nearest Neighbour implementation in annoy (Section 3.4).
- kNN-PCA: Some code to use PCA to estimate the intrinsic dimension of the four datasets.
- kNN-InstSel: An evaluation of the impact of two Instance selection algorithms (CRR & CNN) on training set size and accuracy.
- kNN-Model-Selection: Using grid-search for model selection (hyper-parameter tuning).

REFERENCES

[1] David W. Aha, Dennis Kibler, and Marc K. Albert. 1991. Instance-based learning algorithms. *Mach. Learn.* 6, 1 (1991), 37–66.

[2] Naomi S. Altman. 1992. An introduction to kernel and nearest-neighbor nonparametric regression. *Amer. Statist.* 46, 3 (1992), 175–185.

[3] Fabrizio Angiulli. 2007. Fast nearest neighbor condensation for large data sets classification. *IEEE Trans. Knowl. Data Eng.* 19, 11 (2007), 1450–1464. DOI: https://doi.org/10.1109/TKDE.2007.190645

[4] Kyungim Baek, Bruce A. Draper, J. Ross Beveridge, and Kai She. 2002. PCA vs. ICA: A comparison on the FERET data set. In *Joint Conference on Information Sciences*. 824–827.

[5] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. 2017. The great time series classification bake off: A review and experimental evaluation of recent algorithmic advances. *Data Mining Knowl. Discov.* 31, 3 (2017), 606–660.

[6] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (1975), 509–517.

[7] Jon Louis Bentley. 1979. Multidimensional binary search trees in database applications. *IEEE Trans. Softw. Eng.* SE-5, 4 (1979), 333–340.

[8] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* 13, 1 (2012), 281–305.

[9] Erik Bernhardsson. 2018. *Annoy: Approximate Nearest Neighbors in C++/Python*. Retrieved from https://pypi.org/project/annoy/.

[10] Alina Beygelzimer, Sham Kakade, and John Langford. 2006. Cover trees for nearest neighbor. In *23rd International Conference on Machine Learning*. ACM, 97–104.

[11] Nitin Bhatia et al. 2010. Survey of nearest neighbor techniques. *arXiv preprint arXiv:1007.0085* (2010).

[12] Henry Brighton and Chris Mellish. 2002. Advances in instance selection for instance-based learning algorithms. *Data Mining Knowl. Discov.* 6, 2 (2002), 153–172.

[13] Carla E Brodley. 1993. Addressing the selective superiority problem: Automatic algorithm/model class selection. In *Proceedings of the 10th International Conference on Machine Learning*. 17–24.

[14] R. Mike Cameron-Jones. 1992. Minimum description length instance-based learning. In *5th Australian Joint Conference on Artificial Intelligence*. World Scientific, 368–373.

[15] Pádraig Cunningham. 2008. Dimension reduction. In *Machine Learning Techniques for Multimedia*. Springer, 91–112.

[16] Padraig Cunningham and Sarah Jane Delany. 2007. *k-Nearest Neighbour Classifiers*. Technical Report UCD-CSI-2007-4. School of Computer Science & Informatics, University College Dublin.

[17] Belur V. Dasarathy. 1991. *Nearest Neighbor (NN) Norms: Nn Pattern Classification Techniques*. IEEE Computer Society Press.

[18] Sanjoy Dasgupta and Yoav Freund. 2008. Random projection trees and low dimensional manifolds. In *Proceedings of the 40th ACM Symposium on Theory of Computing*. 537–546.

[19] William de Vazelhes, C. J. Carey, Yuan Tang, Nathalie Vauquier, and Aurélien Bellet. 2020. metric-learn: Metric learning algorithms in Python. *J. Mach. Learn. Res.* 21, 138 (2020), 1–6.

[20] Joost C. F. de Winter, Samuel D. Gosling, and Jeff Potter. 2016. Comparing the Pearson and Spearman correlation coefficients across distributions and sample sizes: A tutorial using simulations and empirical data. *Psychol. Meth.* 21, 3 (2016), 273.

[21] Sarah Jane Delany and Derek Bridge. 2007. Catching the drift: Using feature-free case-based reasoning for spam filtering. In *Case-based Reasoning Research and Development*, Rosina O. Weber and Michael M. Richter (Eds.). Springer Berlin, 314–328.

[22] Sarah Jane Delany and Pádraig Cunningham. 2004. An analysis of case-base editing in a spam filtering system. In *European Conference on Case-based Reasoning*. Springer, 128–141.
[23] Keinosuke Fukunaga and Patrenahalli M. Narendra. 1975. A branch and bound algorithm for computing k-nearest neighbors. *IEEE Trans. Comput.* 100, 7 (1975), 750–753.
[24] Salvador García, Joaquín Derrac, José Ramón Cano, and Francisco Herrera. 2012. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE Trans. Pattern Anal. Mach. Intell.* 34, 3 (2012), 417–435. DOI: https://doi.org/10.1109/TPAMI.2011.142
[25] Geoffrey Gates. 1972. The reduced nearest neighbor rule (Corresp.). *IEEE Trans. Inf. Theor.* 18, 3 (1972), 431–433.
[26] Peter Hart. 1968. The condensed nearest neighbor rule (Corresp.). *IEEE Trans. Inf. Theor.* 14, 3 (1968), 515–516.
[27] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *30th ACM Symposium on Theory of Computing*. 604–613.
[28] Eamonn Keogh, Stefano Lonardi, and Chotirat Ann Ratanamahatana. 2004. Towards parameter-free data mining. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 206–215.
[29] Eamonn J. Keogh and Michael J. Pazzani. 2001. Derivative dynamic time warping. In *SIAM International Conference on Data Mining*. SIAM, 1–11.
[30] Jon M. Kleinberg. 1997. Two algorithms for nearest-neighbor search in high dimensions. In *29th ACM Symposium on Theory of Computing*. 599–608.
[31] Ron Kohavi and George H. John. 1997. Wrappers for feature subset selection. *Artif. Intell.* 97, 1–2 (1997), 273–324.
[32] Solomon Kullback and Richard A. Leibler. 1951. On information and sufficiency. *Ann. Math. Statist.* 22, 1 (1951), 79–86.
[33] Neeraj Kumar, Li Zhang, and Shree Nayar. 2008. What is a good nearest neighbors algorithm for finding similar patches in images? In *European Conference on Computer Vision*. Springer, 364–378.
[34] Ming Li, Xin Chen, Xin Li, Bin Ma, and Paul M. B. Vitányi. 2004. The similarity metric. *IEEE Trans. Inf. Theor.* 50, 12 (2004), 3250–3264.
[35] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. 2003. A symbolic representation of time series, with implications for streaming algorithms.In *8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*. DOI: https://doi.org/10.1145/882082.882086
[36] Ting Liu, Andrew W. Moore, Ke Yang, and Alexander G. Gray. 2005. An investigation of practical approximate nearest neighbor algorithms. In *International Conference on Advances in Neural Information Processing Systems*. 825–832.
[37] Vivek Mahato, William Johnston, and Pádraig Cunningham. 2019. Scoring performance on the Y-balance test. In *International Conference on Case-based Reasoning*. Springer, 281–296.
[38] Elizabeth McKenna and Barry Smyth. 2000. Competence-guided editing methods for lazy learning. In *14th European Conference on Artificial Intelligence*. IOS Press, 60–64.
[39] Dunja Mladenić. 1998. Feature subset selection in text-learning. In *European Conference on Machine Learning*. Springer, 95–100.
[40] Jana Novovičová, Antonín Malík, and Pavel Pudil. 2004. Feature selection using improved mutual information for text classification. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR'04) and Structural and Syntactic Pattern Recognition (SSPR'04)*. Springer, 1010–1017.
[41] J. Ross Quinlan. 2014. *C4. 5: Programs for Machine Learning*. Elsevier.
[42] Juha Reunanen. 2003. Overfitting in making comparisons between variable selection methods. *J. Mach. Learn. Res.* 3, Mar. (2003), 1371–1382.
[43] G. Ritter, H. Woodruff, S. Lowry, and T. Isenhour. 1975. An algorithm for a selective nearest neighbor decision rule (Corresp.). *IEEE Trans. Inf. Theor.* 21, 6 (1975), 665–669.
[44] Yossi Rubner, Leonidas J. Guibas, and Carlo Tomasi. 1997. The earth mover's distance, multi-dimensional scaling, and color-based image retrieval. In *ARPA Image Understanding Workshop*, Vol. 661. 668.
[45] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. 2000. The earth mover's distance as a metric for image retrieval. *Int. J. Comput. Vis.* 40, 2 (2000), 99–121.
[46] Jörg Walter Schaaf. 1996. Fish and Shrink. A next step towards efficient case retrieval in large scaled case bases. In *European Workshop on Advances in Case-based Reasoning*. Springer, 362–376.
[47] Patrick Schäfer and Mikael Högqvist. 2012. SFA: A symbolic fourier approximation and index for similarity search in high dimensional datasets. In *15th International Conference on Extending Database Technology*. ACM, 516–527.
[48] Roger N. Shepard. 1988. Toward a universal law of generalization. *Science* 242, 4880 (1988), 944–944.
[49] Chanop Silpa-Anan and Richard Hartley. 2008. Optimised KD-trees for fast image descriptor matching. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 1–8.
[50] Barry Smyth and Mark T. Keane. 1995. Remembering to forget. In *14th International Joint Conference on Artificial Intelligence*. 377–382.

[51] Romain Tavenard. 2017. Tslearn: A machine learning toolkit dedicated to time-series data. Retrieved from https://github.com/rtavenar/tslearn.
[52] Ivan Tomek. 1976. An experiment with the edited nearest-neighbor rule. *IEEE Trans. Syst., Man, Cyber.* 6, 6 (1976), 448–452.
[53] Fei Wang and Jimeng Sun. 2015. Survey on distance metric learning and dimensionality reduction in data mining. *Data Mining Knowl. Discov.* 29, 2 (2015), 534–564.
[54] Kilian Q. Weinberger and Lawrence K. Saul. 2009. Distance metric learning for large margin nearest neighbor classification. *J. Mach. Learn. Res.* 10, 2 (2009).
[55] Dennis L. Wilson. 1972. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Trans. Syst., Man,. Cyber.* SMC-2, 3 (1972), 408–421.
[56] D. Randall Wilson and Tony R. Martinez. 1997. Instance pruning techniques. In *International Conference on Machine Learning*, Vol. 97. 400–411.
[57] D. Randall Wilson and Tony R. Martinez. 2000. Reduction techniques for instance-based learning algorithms. *Mach. Learn.* 38, 3 (2000), 257–286. DOI: https://doi.org/10.1023/A:1007626913721
[58] Jianping Zhang. 1992. Selecting typical instances in instance-based learning. In *Machine Learning Proceedings 1992.* Elsevier, 470–479.
[59] Jun Zhu and Qiang Yang. 1999. Remembering to add: competence-preserving case-addition policies for case-base maintenance. In *International Joint Conference on Artificial Intelligence*, Vol. 99. 234–241.

Received April 2020; revised February 2020; accepted March 2021