Weighted Symbols-Based Edit Distance for String-Structured Image Classification*

Cécile Barat, Christophe Ducottet, Elisa Fromont,
Anne-Claire Legrand, and Marc Sebban

Université de Lyon, F-42023, Saint-Etienne, France
CNRS, UMR 5516, Laboratoire Hubert Curien, 42023, Saint-Etienne, France
Université de Saint-Etienne, Jean Monnet, F-42023, Saint-Etienne, France

Abstract. As an alternative to vector representations, a recent trend in image classification suggests to integrate additional structural information in the description of images in order to enhance classification accuracy. Rather than being represented in a p-dimensional space, images can typically be encoded in the form of strings, trees or graphs and are usually compared either by computing suited metrics such as the (string or tree)-edit distance, or by testing subgraph isomorphism. In this paper, we propose a new way for representing images in the form of strings whose symbols are weighted according to a TF-IDF-based weighting scheme, inspired from information retrieval. To be able to handle such real-valued weights, we first introduce a new weighted string edit distance that keeps the properties of a distance. In particular, we prove that the triangle inequality is preserved which allows the computation of the edit distance in quadratic time by dynamic programming. We show on an image classification task that our new weighted edit distance not only significantly outperforms the standard edit distance but also seems very competitive in comparison with standard histogram distances-based approaches.

1 Introduction

Classification of images is of considerable interest in many image processing and computer vision applications. A common approach to represent the image content is to use histograms of color, texture and edge direction features (1; 2). Although they are computationally efficient, such histograms only use global information and so provide a crude representation of the image content. The current trend in image classification is towards the use of the *bag-of-visual-words* model that comes from the *bag-of-words* representation of text documents (3). This model requires four basic stages: (i) keypoints detection (ii) description, (iii) codebook creation and (iv) image representation. Keypoints refer to small regions of interest in the image. They can be sampled densely (4), randomly (5) or extracted with various detectors (6). Once extracted, keypoints are characterized

* This work is part of the ongoing ANR SATTIC 07-1_184534 research project and the Pascal2 Network of Excellence.

J.L. Balcázar et al. (Eds.): ECML PKDD 2010, Part I, LNAI 6321, pp. 72–86, 2010.

using a local descriptor, the most widely used being SIFT (7). A visual codebook is then learned over the collection of descriptors of a training set typically by using the k-means algorithm (8). Each cluster gives a visual word and each image can then be mapped into this new space of visual words leading to a *bag-of-visual-words*. Each word can be weighted either according to its frequency in the image, or using more sophisticated techniques such as mutual-information-based binarization (9) or TF-IDF-based weighting (Term Frequency-Inverse Document Frequency) (10; 11). Whatever the weighting scheme, two images are usually compared in a classification task by computing either a dot product or a given distance (*e.g.* L_1, L_2, L_∞) between their corresponding weighted feature vectors in the considered vector space.

Although working in a vector space brings many advantages, it does not allow the integration of additional structural information or topological relationships between the objects of the images. To overcome this drawback, an alternative to the vector representation-based approaches consists in representing images in the form of structured data such as strings, trees, or graphs. For instance, in order to code the topological relationship of so-called iconic objects in an image, the 2D string representation (12; 13; 14) uses the relative location of the visual words in the original 2D-space to build a graph of similarities. The comparison between two images coded in the form of strings is then achieved by searching for the largest common subsequence that satisfies a clique in that graph. In order to represent binary objects, Freeman (15) codes the boundaries of such objects in the form of sequences of symbols. Extending this principle in (16), Daliri proposes to map each contour of the objects into a string whose components are pairs of symbols, the first one representing the angle computed between the contour point and its neighbors and the other describing the normalized distance from the center of mass.

In the previous two approaches, the comparison between two images is achieved by computing the edit distance (17) between the corresponding string representations. Let us recall that the standard edit distance between two structured data $\mathbf{x}$ and $\mathbf{y}$ is defined as the less costly set of edits needed to transform $\mathbf{x}$ into $\mathbf{y}$, with the allowable edit operations being insertion, deletion, or substitution of symbols. If a usual way to use the edit distance is to assign a unit cost to each of the edit operations, many efforts have been made during the past few years to automatically learn more performing edit costs. In this context, a recent line of research has investigated the ways to model those costs in the form of the parameters of probabilistic state machines, such as pair-Hidden Markov models, stochastic transducers, or probabilistic automata (18; 19; 20; 21). The resulting *stochastic* edit distance (usually learned using an EM-based approach) can either be used in more performing neighborhood-based classifiers, or wrapped into *edit kernels* for improving Support Vector Machines (22).

While all the previously cited works mainly dealt with the improvement of weighted edit distances where the weights concern the *edit operations* between two symbols, we can note that no effort has been really made to develop new weighted edit distances where the weights are assigned to the *symbols* themselves.

And yet, such distances would allow us to take advantage of both vector and structured approaches. The objective of this paper is to fill this gap by presenting a new weighted symbols-based edit distance (Section 2). By allowing each symbol to be real-valued, this opens the door to new image representations. In this context, we propose a new way to encode images in the form of strings whose symbols are real-valued according to a TF-IDF weighting scheme (Section 3). A series of experiments is carried out in Section 4 that shows that our new weighted edit distance not only significantly outperforms the standard edit distance but also seems very competitive in comparison with standard histogram distances-based approaches. Before concluding, we propose an original way to automatically learn the costs of the edit operations, by taking into account the number of edges separating the visual words in a minimum spanning tree built on the visual codebook. Plugged in our weighted edit distance, we show that these edit costs allow us to improve the classification accuracy.

2 Weighted Edit Distance

In its basic form, the Levenshtein distance (or edit distance (17)) between two strings $\mathbf{x}(T)$ and $\mathbf{y}(V)$ of length T and V is defined as the minimum number of edits needed to transform $\mathbf{x}(T)$ into $\mathbf{y}(V)$, with the allowable edit operations being insertion, deletion, or substitution of a single character. Using the dynamic programming Algorithm 1, the edit distance $D(T, V)$ is computable in $\mathcal{O}(T \times V)$ and boils down to filling in a $(T + 1) \times (V + 1)$ matrix.

Rather than simply counting the number of required edit operations to change $\mathbf{x}(T)$ into $\mathbf{y}(V)$, the additive term 1 in the expressions d_1, d_2 and d_3 in Algorithm 1 can be replaced by the value of an edit cost function $c(x_r, y_k)$ that takes into account the nature of the symbols $x_r, y_k \in \Sigma \cup \{\lambda\}$ involved in the edit operation, where Σ is the alphabet and λ the empty symbol. In this case, the edit distance between $\mathbf{x}(T)$ and $\mathbf{y}(V)$ becomes the minimum cost of all sequences of edit operations which transform $\mathbf{x}(T)$ into $\mathbf{y}(V)$. As mentioned in (17), $D(T, V)$ remains a metric if the edit cost function $c(x_r, y_k)$ satisfies the two properties of *positive definiteness* and *symmetry*. It is computable in $\mathcal{O}(T \times V)$ if the *triangle inequality* is also fulfilled.

Rather than allowing the use of weights on each edit operation (as usually done in the literature), we propose in this section to authorize the management of weighted symbols during the calculation of the edit distance. By this way, an edit operation becomes a transformation of a weighed symbol into another one. This enables us to take into account the TF-IDF of each symbol of a string-structured image as a weight, and to compute the edit distance between two images represented with such strings of weighted symbols. We propose in the following an edit cost function that is able to manage at once two symbols and their corresponding weights. Henceforth, a string $\mathbf{x}(T)$ will be composed of T weighted symbols $\mathbf{x_1} \ldots \mathbf{x_T}$ where $\forall i = 1..T, \mathbf{x_i} = (x_i, w_{x_i})$ is made of a symbol $x_i \in \Sigma$ and a weight $w_{x_i} \in \mathbb{R}_+^*$.

Input: Two strings $\mathbf{x}(T)$ and $\mathbf{y}(V)$
Output: Edit Distance $D(T, V)$ between $\mathbf{x}(T)$ and $\mathbf{y}(V)$

```
1  D(0,0) ← 0;
2  for r=1 to T do
3      D(r,0) ← D(r−1,0) + 1;
4  end
5  for k=1 to V do
6      D(0,k) ← D(0,k−1) + 1;
7  end
8  for r=1 to T do
9      for k=1 to V do
10         if (x_r = y_k) then
11             D(r,k) = D(r−1,k−1);
12         end
13         else
14             d1 ← D(r−1,k−1) + 1;
15             d2 ← D(r−1,k) + 1;
16             d3 ← D(r,k−1) + 1;
17             D(r,k) ← min(d1,d2,d3);
18         end
19     end
20 end
21 Return D(T,V);
```

Algorithm 1. Edit distance algorithm that returns the number of edit operations required to change a string $\mathbf{x}(T)$ into another $\mathbf{y}(V)$.

Definition 1. *The edit cost function* $c : ((\Sigma \times \mathbb{R}_+^*) \cup (\{\lambda\} \times \{0\})) \times ((\Sigma \times \mathbb{R}_+^*) \cup (\{\lambda\} \times \{0\})) \rightarrow \mathbb{R}_+$ *is defined as follows*[1]*:*
Let the symbols a, b *and the positive reals* n, m *be the components of two weighted symbols* $\mathbf{a} = (a, n)$ *and* $\mathbf{b} = (b, m)$

$$c(\mathbf{a}, \mathbf{b}) = \begin{cases} \max(n, m) & \text{if } a \neq b \\ |n - m| & \text{otherwise} \end{cases}$$

Plugging this function in an edit distance algorithm, we obtain the Algorithm 2. The underlying idea of the function c is graphically described in Figure 1. The edit cost between two weighted symbols (a, n) and (b, m) is close to the one computed between two strings where a and b are "virtually" repeated n and m times respectively. An alternative to our cost function would have consisted in actually repeating each symbol according to its corresponding weight. Despite the fact that this would quadratically increase (in the average of the weights) the algorithmic complexity of the edit distance calculation, this would lead to a loss of information by discarding the decimal part of the weight. We will show this behavior in the experimental part of this paper.

[1] Note that the weight of the empty string λ is always equal to 0 such that $\boldsymbol{\lambda} = (\lambda, 0)$.

Input: Two strings $\mathbf{x}(T)$ and $\mathbf{y}(V)$
Output: Edit Distance $D(T, V)$ between $\mathbf{x}(T)$ and $\mathbf{y}(V)$
1 $D(0,0) \leftarrow 0$;
2 **for** *r=1 to T* **do**
3 $\quad D(r,0) \leftarrow D(r-1,0) + c(\mathbf{x_r}, \lambda)$;
4 **end**
5 **for** *k=1 to V* **do**
6 $\quad D(0,k) \leftarrow D(0,k-1) + c(\lambda, \mathbf{y_k})$;
7 **end**
8 **for** *r=1 to T* **do**
9 $\quad$ **for** *k=1 to V* **do**
10 $\quad\quad d_1 \leftarrow D(r-1,k-1) + c(\mathbf{x_r}, \mathbf{y_k})$;
11 $\quad\quad d_2 \leftarrow D(r-1,k) + c(\mathbf{x_r}, \lambda)$;
12 $\quad\quad d_3 \leftarrow D(r,k-1) + c(\lambda, \mathbf{y_k})$;
13 $\quad\quad D(r,k) \leftarrow min(d_1, d_2, d_3)$;
14 $\quad$ **end**
15 **end**
16 Return $D(T, V)$;

Algorithm 2. Weighted edit distance algorithm that returns the minimum cost required to change a weighted string $\mathbf{x}(T)$ into another $\mathbf{y}(V)$.

Proposition 1. *Given the edit cost function c of Definition 1, Algorithm 2 is a generalization of Algorithm 1.*

Proof. Algorithm 2 generalizes Algorithm 1 if, for two unweighted strings, they both return the same edit distance. Two symbols $\mathbf{a} = (a, n)$ and $\mathbf{b} = (b, m)$ are unweighted if $n = m = 1$. In this case, $c(\mathbf{a}, \mathbf{b})$ returns

- either $|n - m| = 0$ if $a = b$ (and so lines 10 and 11 of Algorithm 1 and line 10 of Algorithm 2 are the same),
- or $\max(n, m) = 1$ if $a \neq b$ (and so lines 14, 15 and 16 of Algorithm 1 are the same as lines 10, 11 and 12 of Algorithm 2).

Therefore, the two algorithms return the same edit distance for two unweighted strings. □

Proposition 2. *Given the edit cost function c, Algorithm 2 returns a true distance function between* $\mathbf{x}(T)$ *and* $\mathbf{y}(V)$ *computable in* $\mathcal{O}(|T| \times |V|)$.

Proof. The edit distance computed from an edit cost function c is a metric if c fulfills the following two conditions, $\forall \mathbf{x}, \mathbf{y}, \mathbf{z} \in (\Sigma \times \mathbb{R}_+^*) \cup (\{\lambda\} \times \{0\})$:

1. $c(\mathbf{x}, \mathbf{y}) = 0$ if and only if $\mathbf{x} = \mathbf{y}$ (associated with the fact that $w_x \in \mathbb{R}_+^*$, this defines the *positive definiteness*). This is true because $c(\mathbf{x}, \mathbf{y}) = 0$ only if $|w_x - w_y| = 0$, *i.e.* when $x = y$ and $w_x = w_y$, so when $\mathbf{x} = \mathbf{y}$. On the other hand, this can not occur when $\mathbf{x} \neq \mathbf{y}$ because, except for λ, the weight of a symbol belongs to $\mathbb{R}_+^*$, so $max(w_x, w_y)$ cannot be equal to 0 in this case.

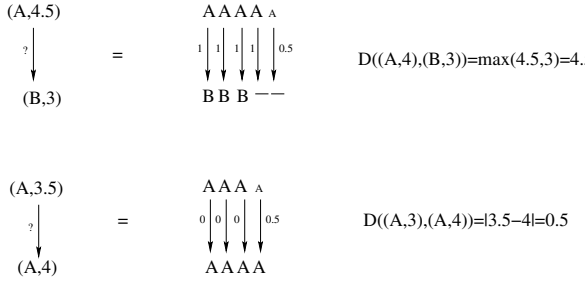

Fig. 1. Intuitive idea of the calculation of the edit distance between two weighted symbols

2. $c(\mathbf{x}, \mathbf{y}) = c(\mathbf{y}, \mathbf{x})$ (*symmetry*). This is always true because the two functions $max(w_x, w_y)$ and $|w_x - w_y|$ fulfill the symmetry condition.

Moreover, the edit distance is computable in $\mathcal{O}(T \times V)$ by the Algorithm 2 if c also satisfies the triangle inequality $c(\mathbf{x}, \mathbf{y}) \leq c(\mathbf{x}, \mathbf{z}) + c(\mathbf{z}, \mathbf{y})$ (see (17)). Since the output of c depends on the nature of the input symbols, let us study the different possible configurations:

1. If the symbols x, y, z are the same, the function c always returns $|n - m|$, that is the Manhattan function. Therefore, the property holds because the Manhattan function is a metric (d_1).
2. If the symbols x, y, z are different, the function c always returns $max(n, m)$, that is a specific case of the Tchebychev function. Therefore, the property holds because the Tchebychev function is a metric (d_∞).
3. If $x = y$ and $x \neq z$, one must satisfy $max(w_x, w_z) + max(w_z, w_y) \geq |w_x - w_y|$. This holds because $max(w_x, w_z) + max(w_z, w_y) \geq max(w_x, w_y)$ (cf case 2) and $max(w_x, w_y) \geq |w_x - w_y|$, $\forall w_x, w_y \in \mathbb{R}_+^*$.
4. If $y = z$ and $x \neq y$, one must satisfy $max(w_x, w_z) + |w_z - w_y| \geq max(w_x, w_y)$. 6 subcases must be studied:
   - If $w_z \in [w_x, w_y]$ we must prove that $w_z + w_y - w_z \geq w_y \Rightarrow w_y \geq w_y$, that is always true.
   - If $w_z \leq w_x \leq w_y$ we must prove that $w_x + (w_y - w_z) \geq w_y$, that is always true because $w_x - w_z \geq 0$.
   - If $w_x \leq w_y \leq w_z$ we must prove that $w_z + w_z - w_y \geq w_y$. Since $w_z + w_z - w_y = 2w_z - w_y \geq 2w_z - w_z = w_z \geq w_y$, the property holds.
   - If $w_z \in [w_y, w_x]$ we must prove that $w_x + (w_z - w_y) \geq w_x$, that is always true because $w_z - w_y \geq 0$.
   - If $w_z \leq w_y \leq w_x$ we must prove that $w_x + (w_y - w_z) \geq w_x$, that is always true because $w_y - w_z \geq 0$.
   - If $w_y \leq w_x \leq w_z$ we must prove that $w_z + w_z - w_y \geq w_x$. Since $w_z + w_z - w_y = 2w_z - w_y \geq 2w_z - w_z = w_z \geq w_x$, the property holds.
5. If $x = z$ and $x \neq y$, one must satisfy $|w_x - w_z| + max(w_z, w_y) \geq max(w_x, w_y)$. By symmetry, this case is proved as for case 4 □

3 Image Representation as a String of Weighted Symbols

As seen in the introduction, the representation of an image as a *bag-of-visual-words* has become the reference method in the field of image classification, inspired by the *bag-of-words* representation in text. Given a vocabulary defined by $\mathcal{V} = \{v_1, \ldots, v_j, \ldots, v_{|\mathcal{V}|}\}$, an image a_i is represented as a vector of weights $\mathbf{u_i} = (u_{i,1}, \ldots, u_{i,j}, \ldots, u_{i,|\mathcal{V}|})$. Each component $u_{i,j}$ of $\mathbf{u_i}$ gives the weight of the visual word v_j in the image a_i.

Unlike a text document, an image does not have a natural visual vocabulary. In the *bag-of-visual-words* approach, visual words represent small patches with a characteristic shape, texture or color that often appear in a reference image collection. Consequently, a visual vocabulary depends on the reference image collection and on the three following elements: the patch detector, the local descriptor and the patch classification method. The way of calculating $u_{i,j}$ weights also plays an important role in the image representation. The natural weighting is the number of occurrences of a visual word v_j in the image. This representation as an occurrence histogram is not the most efficient one and other strategies have been proposed (9). We choose to use a TF-IDF weighting scheme which has been shown to be relevant for image classification (10; 11). The method consists in multiplying a term $tf_{i,j}$ giving the representativeness of word v_j in image a_i to a term idf_j giving the discriminative power of this word in the reference collection. Thus, the weight $u_{i,j} = tf_{i,j}idf_j$ is high if the word appears frequently in the image and rarely in the reference collection.

Based on this vector *bag-of-visual-words* representation, we propose a new representation of an image as a string of weighted symbols. We first expose this new representation. Then we present a way to implement it.

3.1 The Proposed Representation

The principle of our representation is to consider the visual words as symbols whose weight is equal to the TF-IDF weight used in the vector representation. The question is how to structure the symbols as an ordered string.

Let us first note that visual words with zero TF-IDF weight are not taken into account in the string construction. Indeed, this happens when the term is not present in the image or when it appears in most of the documents which means it is not discriminative. This is consistent with the weighted distance previously defined which admits only one zero weighted symbol (the empty symbol λ). Then, the string associated with an image will have a length T equal to the number of visual words appearing at least once in this image.

Let us come to the question of symbol order which provides structural information and so can enhance the image representation. For that purpose, we must choose an order that takes into account visual words characteristics and possibly their distribution in the image. This is a difficult choice because of the high variability of the visual content of an image: two images of similar visual content can have different visual words spatial distributions. To circumvent this difficulty, we propose to use only the global characteristics of the visual vocabulary. For example, we can use the principle that if some visual words (*i.e.* symbols) are similar,

it may be interesting to allow substitutions between them in the string. Such substitutions may occur if the corresponding symbols are close in the strings. Thus, the chosen order must group similar visual words. We finally propose to sort the symbols in descending order of their IDF discriminative power. The most discriminative symbols (those that are rare in the collection) will be at the beginning of the string while the less discriminative ones (those that are most common in the collection) will be at the end. This choice is based on the (quite likely) assumption that two similar symbols have a similar IDF discriminative power. Note that such an order is global to all strings since the discriminative power is defined from the entire collection.

Thus, the string $\mathbf{x_1} \dots \mathbf{x_r} \dots \mathbf{x_T}$ associated with a given image a_i is composed of T symbols $\mathbf{x_r} = (x_r, w_{x_r})$ with:

$$\begin{cases} j = ord_{idf}(r) \\ x_r = v_j \\ w_{x_r} = u_{i,j} \end{cases} \tag{1}$$

where $ord_{idf}(r)$ represents the position j of symbol x_r in the original vector $\mathbf{u_i}$.

A summary of the proposed representation is given in Figure 2.

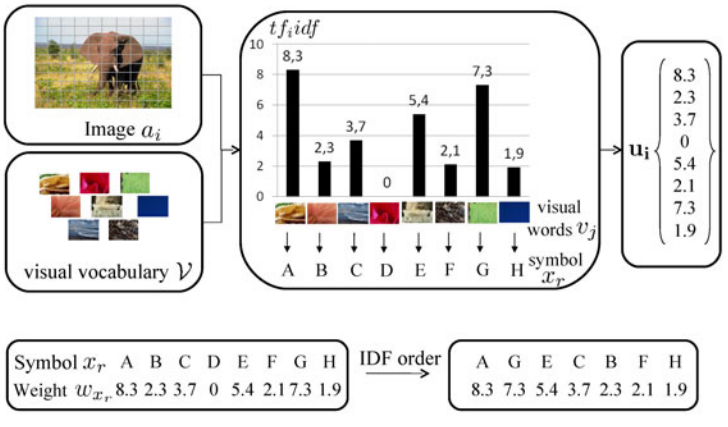

Fig. 2. Image representation as a string of weighted symbols

3.2 A Practical Implementation

In practice, to obtain the string associated with an image, we must choose the three different elements required to build the visual vocabulary (the patch detector, the local descriptor and the patch classification method) plus the TF-IDF weighting. Of course these choices depend on the nature of the images and the task to achieve. The four chosen elements are presented below.

The patch detector. We extract patches sampled in a 10×10 cells regular grid so as to extract 100 patches per image. This dense sampling was found to be

more efficient than salient point detection for image classification, since salient points do not cover all the image spatial domain and lead to an incomplete description (4; 9). Moreover, dense sampling may produce a very large number of features.

The local descriptor. We choose to describe patches using a color descriptor. We transform the RGB components of the patch into three normalized components defined as $\frac{R}{R+G+B}$, $\frac{G}{R+G+B}$ and $\frac{R+G+B}{3\times255}$. This color space presents two main advantages. First, it makes the first two variables independent of the third one representing the luminance. Second, it is very easy to compute. From the normalized components of a patch, we compute 6 features equal to the mean and the standard deviation of the three values.

The patch classification method. We learn a visual vocabulary $\mathcal{V}$ applying a k-means algorithm over all the computed patches on training images. We get k clusters of features whose centers represent k visual words, k being the size of the visual vocabulary. Local patches of images are mapped to their closest visual words using the euclidean distance. In our experiments, we extracted roughly 100000 patches and worked with different vocabulary sizes.

The TF-IDF weighting. Several formulations exist to calculate $tf_{i,j}$ and idf_j, but the okapi one proposed by Robertson *et al.*(23) is often reported superior to others in classification problems. We apply a modified version implemented in the lemur software[2] proposed by (24):

$$tf_{i,j} = \frac{k_1 n_{i,j}}{n_{i,j} + k_2(1 - b + b\frac{|a_i|}{a_{avg}})}$$

where $n_{i,j}$ is the occurrence of the word v_j in the image a_i, $|a_i|$ the number of visual words used to represent image a_i, a_{avg} the average number of visual words per image in the collection A. k_1, k_2 and b are three constants.

$$idf_j = \log \frac{|A| - |\{a_i | v_j \in a_i\}| + 0.5}{|\{a_i | v_j \in a_i\}| + 0.5}.$$

A main disadvantage of this formula is that it can be possibly negative, which has been discussed in (25). This happens when a term appears in more than half of the documents. Thus, we choose to floor the IDF values to 0.

4 Experiments

4.1 Experimental Protocol

To assess the relevance of the new edit distance computed with our TF-IDF-based weighting scheme, we carried out a series of experiments in image classification. The reference image database is the SIMPLIcity collection[3] (26), containing

[2] http://www.lemurproject.com

[3] http://wang.ist.psu.edu/~jwang/test1.zip

Fig. 3. Examples extracted from the SIMPLIcity collection

1000 images extracted from the COREL database. Each image (384×256 pixels) belongs to one of 10 meaningful categories: *African people, beaches, buildings, buses, dinosaurs, elephants, flowers, food, horses and mountains* (see Figure 3). Note that the different categories are equally distributed, each of them composed of 100 images.

Classification was performed using the 1-Nearest Neighbor (1-NN) rule. We conducted a 10-fold cross-validation to estimate the classifier accuracy. In order to evaluate our weighted edit distance (WED), we divided the experiments into two parts. The first series of experiments aims to analyze how the WED behaves compared to the standard edit distance (ED) and how weighting the symbols contributes to improve the image classification results. The second set of experiments compares the WED with the common metrics usually applied on feature vectors, *i.e.* the normalized dot product and different Minkowski distances such as the L_1, L_2, and L_∞. In this case, images are no more represented in the form of strings but rather by (unordered) vectors whose components are the TF-IDF weights of the visual vocabulary. To assess the impact of the visual vocabulary size, we carried out the experiments with an increasing number of visual words, from 20 to 80.

4.2 Weighted Edit Distance versus Standard Edit Distance

To compare the WED with the standard ED, we performed two series of experiments. First, the ED was applied on the same strings of symbols as for WED without considering their corresponding weights. Figure 4 clearly shows that our WED outperforms the classic ED. Whatever the alphabet size, the difference is statistically significant using a Student paired-t test (note that the highest p-value is equal to 0.02 for an alphabet size of 60 symbols). These results are not so surprising since by discarding the weights, we removed a part of the information about the symbols. Even if the standard ED is not able to deal with such real-valued weights, a fairer comparison would consist in converting each weighted symbol into a substring where the considered symbol would be repeated a number of times corresponding to its weight. But since weights are real valued, we set the number of repetitions to the integer value of the corresponding weight. As we can see on Figure 4, once again, WED performs better than the ED with repetitions.

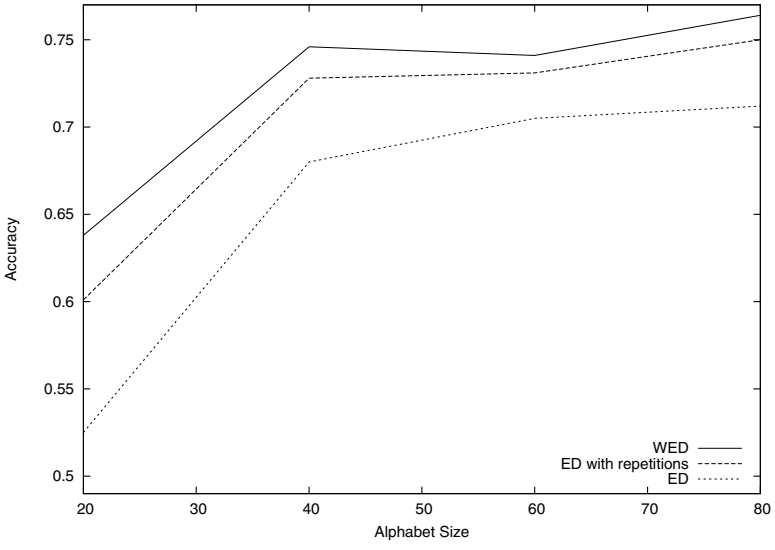

Fig. 4. Comparison between WED and the standard edit distance, computed either omitting the real weights (ED) or by repeating the symbols according to their corresponding weights (ED with repetitions).

Moreover, it is important to note that repeating the symbols generates larger string sizes that leads to a dramatic increase of the algorithmic complexity of the edit distance calculation. Indeed, the complexity becomes $\mathcal{O}(\overline{\lfloor w_x \rfloor} \times \overline{\lfloor w_y \rfloor} \times T \times V)$, where T and V still denote the lengths of the original strings $\mathbf{x}(T)$ and $\mathbf{y}(V)$, $\overline{\lfloor w_x \rfloor}$ and $\overline{\lfloor w_y \rfloor}$ being the average of the integer values of the weights.

4.3 Weighted Edit Distance versus Vector Space Metrics

Figure 5 compares the performance of the WED with that of classical vector space metrics, *i.e.* Minkowski distances (L_1, L_2, L_∞) and the normalized dot product. Several remarks can be made. First, WED is very efficient compared to the other metrics, particularly for small vocabularies. Indeed, it provides the best global behavior by returning the highest classification accuracy for vocabularies smaller than 60, and remaining competitive after. We can also note that for small vocabulary sizes WED significantly outperforms the normalized dot product, which is most often used to measure similarities between bags-of-words. Second, we can observe that the L_1 returns the closest results to WED ones. This behavior can be easily explained by the fact that when symbols are ordered (that is the case in our string-based representation), the L_1 is equivalent to the edit distance if the substitutions between two different symbols are not allowed. Therefore, the difference between the two curves WED and L_1 directly comes from the possibility to substitute symbols in the structured strings.

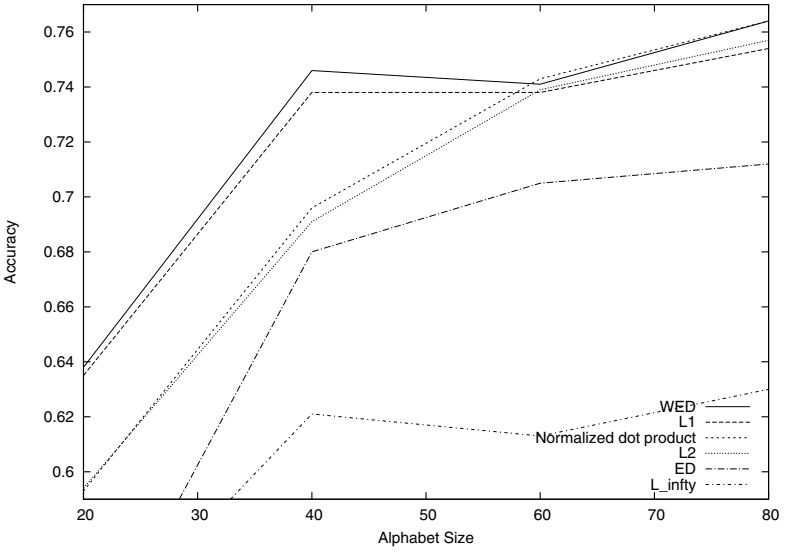

Fig. 5. Comparison between WED, the normalized dot product and some Minkowski metrics

4.4 Plugging Learned Edit Costs in the WED

So far, we mainly concentrated our efforts to take into account real-valued symbols in a new weighted edit distance. In this context, we paid little attention to the cost of the edit operations (*insertion*, *deletion* and *substitution*) that were set to 1 for all the previous experiments. However, as we explained in the introduction of this paper, there exists a huge literature about how to learn those edit costs often in the form of the parameters of stochastic state machines (18; 19; 20; 21; 27). The main drawback of these probabilistic approaches is that the returned parameters "only" satisfy constraints of statistical distribution, and so often do not fulfill the properties of a distance (the symmetry and the triangle inequality are often not ensured). Therefore, the resulting models provide a learned stochastic *edit similarity* rather than a true *edit distance*. To keep a distance function and also to allow the algorithm to compute the WED in quadratic time, we propose in the following a simple solution to determine efficient edit costs.

As explained in Section 3, the visual symbols used for the string comparison correspond to the centers of the clusters computed by a k-means algorithm. These centers are described by some coordinates in a 6-dimensional color space. Intuitively, it means that two centers are close in this space if their corresponding colors are close. When comparing (with an edit distance) two strings, it seems more relevant to favor substitutions of such symbols. This means that the substitution cost should vary according to the distance between one symbol and another.

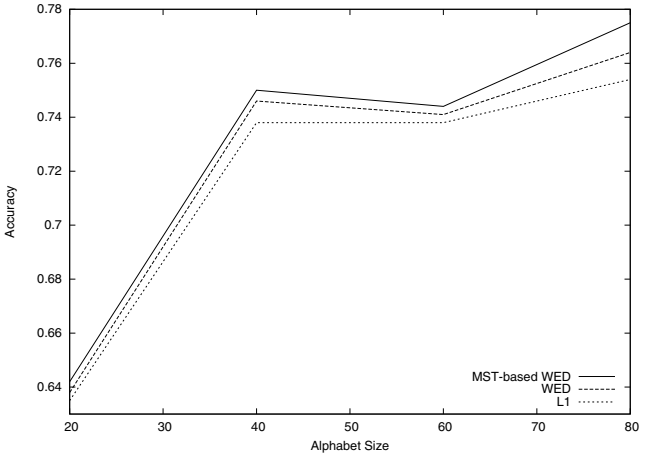

Fig. 6. Effect of the MST-based substitution costs on the performance of WED

To represent these distances, we decided to compute a Minimum Spanning Tree (MST) between the symbols (*i.e.* the center of the clusters) in the 6D space. We considered the complete simple undirected graph $G = (\mathcal{V}, E, \gamma)$ where $\mathcal{V}$ is the set of visual words (vertices), E is the set of all edges in G and $\gamma : e \in E \rightarrow \mathbb{R}$ is a function which assigns a weight to each edge of the graph. We considered here that $\gamma(v_1, v_2) = L_2(v_1, v_2)$ where $L_2(v_1, v_2)$ is the Euclidean distance between the visual words v_1 and v_2. Let us recall that a spanning tree of G is a subgraph of G (a tree) which connects all the vertices together. A *minimum* spanning tree is a spanning tree with weight (the sum of the weights of all the edges in the tree) less than or equal to the weight of every other spanning tree.

To set the cost of the substitution between two given symbols x_r and y_k, noted $\sigma(x_r, y_k)$, we used the number of edges between x_r and y_k in the MST. The cost of the insertion and deletion operations is left to 1. To be integrated into the edit distance calculation, $\sigma(x_r, y_k)$ is just multiplied to the edit cost function $c(\mathbf{x_r}, \mathbf{y_k})$ in line 10 of Algorithm 2. To ensure that this extended version of WED remains a true distance computable in quadratic time, we must prove that the substitution cost function σ provided by the MST is also a distance.

Proposition 3. *The substitution cost function σ is a metric.*

Proof. Let us prove the three properties of a distance:

- Since the MST is an undirected graph, $\sigma(x, y) = \sigma(y, x)$, and so the *symmetry* property holds.
- If $x \neq y$, by construction the number of edge in the MST between x and y is at least equal to 1. On the other hand, if $x = y$ then $\sigma(x, y) = 0$. Therefore, the *positive definiteness* holds.
- Since $\sigma(x, y)$ is associated to a path of the MST, this path is minimal. Therefore, $\sigma(x, y) \leq \sigma(x, z) + \sigma(z, y)$ and so the *triangle inequality* holds. □

The results of the classification task using the cost function σ in our WED are given in Figure 6. The figure shows that using these substitution costs instead of the naive ones used in the previous section allows us to always improve the classification accuracy and to increase the difference with the L_1 distance, in favor of WED.

5 Conclusion

In this paper, we have presented a new string edit distance in which symbols are allowed to be weighted. This metric opens the door to a better use of *bag-of-visual-words*-based image representations which are often constrained to be compared by vector space metrics. By organizing the visual words in the form of a string, it allows us to take into account additional structural information in the description of images. In this paper, we showed that our new edit distance is very competitive with the state of the art vector space metrics on an image classification task when the symbols are weighted by a TF-IDF measure inspired from information retrieval. We also proposed an extension of our approach by automatically determining the substitution costs from a minimum spannig tree built on the alphabet of *visual-words*. Even if we used our weighted distance with a nearest neighbor classifier, note that it can be easily integrated in *edit kernels* for which there exists a huge literature (see (22) for instance). Moreover, we claim that this distance could be applied to other fields (for example in molecular biology) where one might want to ease or make more difficult some operations on some specific parts of the string or of more complex structured data (*e.g.* trees or graphs). To be able to manage these more general cases, our approach should be extended to the tree or graph edit distances.

References

[1] Chapelle, O., Haffner, P., Vapnik, V.: SVMs for histogram-based image classification. IEEE transactions on Neural Networks 10(5), 1055 (1999)
[2] Vailaya, A., Figueiredo, M., Jain, A., Zhang, H.: Image classification for content-based indexing. IEEE Transactions on Image Processing 10(1), 117–130 (2001)
[3] Salton, G., Wong, A., Yang, C.S.: A vector space model for automatic indexing. Commun. ACM 18(11), 613–620 (1975)
[4] Jurie, F., Triggs, B.: Creating efficient codebooks for visual recognition. In: ICCV 2005: Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV 2005), Washington, DC, USA, vol. 1, pp. 604–610. IEEE Computer Society, Los Alamitos (2005)
[5] Vidal-Naquet, M., Ullman, S.: Object recognition with informative features and linear classification. In: ICCV 2003: Proceedings of the Ninth IEEE International Conference on Computer Vision, Washington, DC, USA, p. 281. IEEE Computer Society, Los Alamitos (2003)
[6] Mikolajczyk, K., Schmid, C.: Scale & affine invariant interest point detectors. Int. J. Comput. Vision 60(1), 63–86 (2004)
[7] Lowe, D.G.: Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vision 60(2), 91–110 (2004)
[8] Bisgin, H.: Parallel clustering algorithms with application to climatology. Technical report, Informatics Institute, Istanbul Technical University, Turkey (2008)

[9] Nowak, E., Jurie, F., Triggs, B.: Sampling strategies for bag-of-features image classification. In: Leonardis, A., Bischof, H., Pinz, A. (eds.) ECCV 2006. LNCS, vol. 3954, pp. 490–503. Springer, Heidelberg (2006)
[10] Moulin, C., Barat, C., Géry, M., Ducottet, C., Largeron, C.: UJM at ImageCLEFwiki 2008. In: Peters, C., Deselaers, T., Ferro, N., Gonzalo, J., Jones, G.J.F., Kurimo, M., Mandl, T., Peñas, A., Petras, V. (eds.) Evaluating Systems for Multilingual and Multimodal Information Access. LNCS, vol. 5706, pp. 779–786. Springer, Heidelberg (2009)
[11] Yang, J., Jiang, Y.G., Hauptmann, A.G., Ngo, C.W.: Evaluating bag-of-visual-words representations in scene classification. In: MIR 2007: Proceedings of the international workshop on Workshop on multimedia information retrieval, pp. 197–206. ACM, New York (2007)
[12] Chang, S.K., Shi, Q.Y., Yan, C.W.: Iconic indexing by 2-d strings. IEEE Trans. Pattern Anal. Mach. Intell. 9(3), 413–428 (1987)
[13] Punitha, P., Guru, D.S.: Symbolic image indexing and retrieval by spatial similarity: An approach based on b-tree. Pattern Recogn. 41(6), 2068–2085 (2008)
[14] Hsieh, S.M., Hsu, C.C.: Retrieval of images by spatial and object similarities. Inf. Process. Manage. 44(3), 1214–1233 (2008)
[15] Freeman, H.: Computer processing of line-drawing images. ACM Comput. Surv. 6(1), 57–97 (1974)
[16] Daliri, M.R., Torre, V.: Robust symbolic representation for shape recognition and retrieval. Pattern Recognition 41, 1782–1798 (2008)
[17] Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. J. ACM 21(1), 168–173 (1974)
[18] Durbin, R., Eddy, S., Krogh, A., Mitchison, G.: Biological sequence analysis. Cambridge University Press, Cambridge (1998)
[19] Ristad, E.S., Yianilos, P.N.: Learning string-edit distance. IEEE Trans. Pattern Anal. Mach. Intell. 20(5), 522–532 (1998)
[20] McCallum, A., Bellare, K., Pereira, P.: A conditional random field for discriminatively-trained finite-state string edit distance. In: Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI 2005), pp. 388–400 (2005)
[21] Oncina, J., Sebban, M.: Learning stochastic edit distance: application in handwritten character recognition. Pattern Recogn. 39(9), 1575–1587 (2006)
[22] Bellet, A., Bernard, M., Murgue, T., Sebban, M.: Learning state machine-based string edit kernels. Pattern Recogn. 43(6), 2330–2339 (2010)
[23] Robertson, S.E., Walker, S., Hancock-Beaulieu, M., Gull, A., Lau, M.: Okapi at trec-3. In: Text REtrieval Conference, pp. 21–30 (1994)
[24] Zhai, C.: Notes on the lemur tfidf model. Technical report, Carnegie Mellon University (2001)
[25] Robertson, S.E., Walker, S.: On relevance weights with little relevance information. In: SIGIR 1997: Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 16–24. ACM, New York (1997)
[26] Wang, J.Z., Li, J., Wiederhold, G.: Simplicity: Semantics-sensitive integrated matching for picture libraries. In: Laurini, R. (ed.) VISUAL 2000. LNCS, vol. 1929, pp. 360–371. Springer, Heidelberg (2000)
[27] Boyer, L., Esposito, Y., Habrard, A., Oncina, J., Sebban, J.: Sedil: Software for edit distance learning. In: Daelemans, W., Goethals, B., Morik, K. (eds.) ECML PKDD 2008, Part II. LNCS (LNAI), vol. 5212, pp. 672–677. Springer, Heidelberg (2008)