Improving css-KNN Classification Performance by Shifts in Training Data

Karol Draszawka[1(✉)], Julian Szymański[1], and Francesco Guerra[2]

[1] Gdańsk University of Technology, Gdańsk, Poland
{kadr,julian.szymanski}@eti.pg.da.pl
[2] Universita' di Modena e Reggio Emilia, Modena, Italy
francesco.guerra@unimore.it

Abstract. This paper presents a new approach to improve the performance of a css-k-NN classifier for categorization of text documents. The css-k-NN classifier (i.e., a threshold-based variation of a standard k-NN classifier we proposed in [1]) is a lazy-learning instance-based classifier. It does not have parameters associated with features and/or classes of objects, that would be optimized during off-line learning. In this paper we propose a training data preprocessing phase that tries to alleviate the lack of learning. The idea is to compute training data modifications, such that class representative instances are optimized before the actual k-NN algorithm is employed. The empirical text classification experiments using mid-size Wikipedia data sets show that carefully cross-validated settings of such preprocessing yields significant improvements in k-NN performance compared to classification without this step. The proposed approach can be useful for improving the effectivenes of other classifiers as well as it can find applications in domain of recommendation systems and keyword-based search.

Keywords: KNN classifier · Wikipedia · Documents classification · Missing data imputation

1 Introduction

Text classification finds many applications e.g., in the library systems where it aggregates thematically related resources into a one category, in e-commerce, where it serves for profiling users on the basis of their behaviours, and in the organization of the web, where categories of pages have been proposed (e.g., see the DMOZ directory).

Classification is a task that only for small repositories can be performed manually, in almost all the cases the size of the data requires the development and exploitation of the automatic techniques. The text classification problem is widely studied and approached using variety of methods such as: SVM [2], Bayesian [3] and others [4,5]. Due to their simplicity, k-Nearest Neighbours algorithms [6] have been frequently used for this task, achieving also good results as

J. Cardoso et al. (Eds.): KEYWORD 2015, LNCS 9398, pp. 51–63, 2015.
DOI: 10.1007/978-3-319-27932-9_5

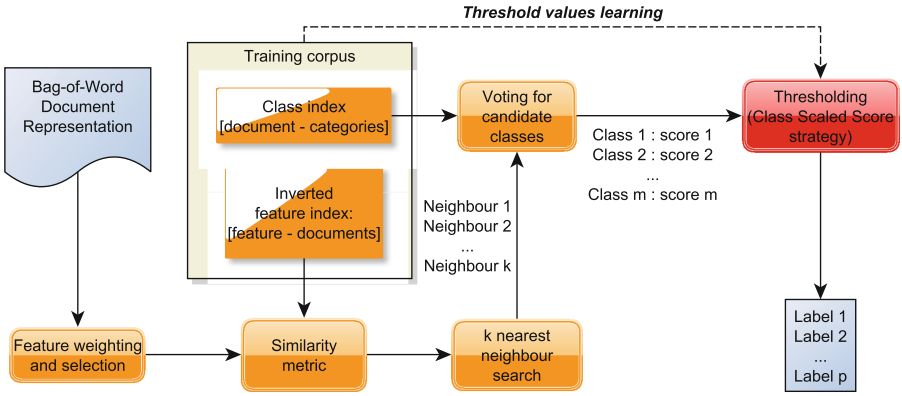


Fig. 1. The architecture of css-KNN-classifier used in experiments (details in [1]).

in [7,8]. Modifications in k-Nearest Neighbours algorithms also enables multi-label classification [9,10] that can be introduced with relatively low computational cost, in comparison to prominent SVM classifiers that are known to have troubles with scalability and multi-label tasks [11,12].

The main disadvantage of the typical k-NN approaches is that they do not provide ability for improving the results using the learning by the examples. Success and popularity of k-NN approach to multi-label large-scale text classification encouraged us to undertake research in this field. In [1], we proposed an approach based on k-NN to text classification that introduced class-specific parameters that can be cross-validated to generate best results. Figure 1 shows the architecture of our classifier: it is based on a typical k-NN classifier, with cosine metric, and k parameter set to 30. The nearest neighbors contribute to the calculation of each class's score, according to scoring formula described in [7]. Then, a thresholding phase using so called Class-specific Scaled Score strategy (CSS) determines the output of the classifier: if a score of class exceeds a threshold specific for this class it is added to the final prediction. The values of class-specific thresholds are tuned using training set subsampling and validation.

In this paper we propose a new approach for improving the results of k-NN classifier. Our idea is to introduce a training phase where we "modify the training dataset" to generate better results in terms of classification accuracy. In particular, in Sect. 3 we propose four methods in which training data examples are *translated* or *shifted* in the feature space, i.e. their feature values are modified, so that they become more similar to other examples. In geometrical interpretation, the points in the high dimensional feature space, representing training data examples, are translated towards other points. In particular, for attributes that a given example do not have any value, these approaches induce values, which relates the approaches to "missing value management" methods, largely studied in the literature. The k-NN classification on such enhanced training dataset shows better performance than without modifications. The results obtained by our experiments, computed with large and sparse datasets, as it is typical in text classification, encourage us to proceed in this direction.

Machine learning techniques have been widely used for supporting keyword search. k-NN classification techniques have been in particular adopted in the field of textual documents, images or videos retrieval, allowing users to "query by example" and obtaining items close to the ones given as input. Nevertheless, machine learning can support other tasks in keyword search: k-NN classification can be particularly useful in the keyword search field, in two areas: disambiguation and expansion of the users' keywords. Disambiguation task is necessary in solving queries, where the keywords have several meanings according to the reference context. To be able to select among the possible meanings the one suitable for the keywords, in the specific query, different techniques are typically applied. One of them is to make a reduction of the document search space, thus providing a large benefit in the accuracy of the results. The classification of the keyword query obtained by the exploitation of the k-NN classification on a feature vector representing the user's keyword query can be exploited for this purpose, by identifying the possible categories where the query belongs to, and in consequence the set of the interesting documents.

Query expansion is another approach that aims at improving the retrieved results accuracy. The typical approaches employ dictionaries of different but semantically related terms. Query expansion provides an additional number of keywords to be included in the original set formulated by the user. In this case, the technique for completing missing values introduced in the paper can be directly adopted for modification of the feature vector representing the user's keyword query. In particular, the technique can be used for changing the values of terms not provided in the original query (i.e., having a 0 value in the query feature vector), with new estimated values. This enhancement of the feature vector can subsequently be adopted for selection of similar documents or identification of the possible reference categories as described below.

The paper is constructed as follows: Sect. 2 introduces some Related Work with the emphasis on dealing with missing values, Sect. 3 describes our *shifting* methods, which are evaluated by the results of the experiments presented in Sect. 4. Finally in Sect. 5 we sketch out some conclusions and future work.

2 Related Work

Our proposal computes changes in the dataset for improving the k-NN classifier. Techniques for modifying values in the datasets have been developed in the literature as related to the management for missing values.

All the existing solutions can be usually classified in one of the following categories [13]: (1) Ignoring and discarding incomplete records and attributes; (2) Managing missing data as special values; and (3) Parameter estimation in the presence of missing data.

The elimination of all samples with missing values and the consequent reduction of the dataset is the simplest solution for managing the issue. Nevertheless, this solution is not always suitable due to the discard of significant fragments of the dataset with potentially useful informative power.

Approaches falling in the second category treat "unknown" itself as a new value to be treat in the same way as other values [14].

Finally, the problem of missing values can be handled by various imputation methods. Working with these approaches, we make the assumptions the "missingness mechanism" can be analyzed and patterns of missing values (if any) can be applied for imputation. As stated in [15], there are three different mechanisms for induction of missing values: 1. Missing completely at random (MCAR), when the distribution of an example having a missing value for an attribute does not depend on either the observed data or the missing values; 2. Missing at random (MAR), when the distribution of an example having a missing value for an attribute depends on the observed data, but does not depend on the missing values; and 3. Not missing at random (NMAR), when the distribution of an example having a missing value for an attribute depends on the missing values. Unfortunately missing values imputation methods are suitable only for missing values caused by MCAR and some of them for MAR mechanisms. If missing values are caused by NMAR, it must be handled by going back to the source of data and obtaining more information or the appropriate model for the missing data mechanism have to be taken into account [13]. Techniques in this category include: replacing missing value with mean, mean or median for the given class, most common attribute value, with values computed with machine learning techniques, association rules (see [13,14,16] for some surveys). Our approach can also be interpreted as a data smoothing technique, a preprocessing commonly used especially in connection with Bayes classifiers [17].

3 Shifting Methods of Training Examples

This section describes four types of training data modifications used in our experiments. A given training dataset D_{train} consists of N training pairs (examples):

$$D_{\text{train}} = \left\{ (\boldsymbol{x}, \mathcal{Y}) : \boldsymbol{x} \in \mathbf{R}^M, \mathcal{Y} \subseteq \mathcal{L} \right\},$$

where $\boldsymbol{x}$ is a M-dimensional feature vector and $\mathcal{Y}$ is a set of objects' labels, which is some subset of the set of all labels/classes $\mathcal{L}$ in the data. Each of the following methods can be treated as a function that modifies a training feature vector $\boldsymbol{x}$ according to its label set $\mathcal{Y}$ and its relation to the rest of training examples in D_{train}:

$$\boldsymbol{x}' = shift\left(\boldsymbol{x}, \mathcal{Y}, D_{\text{train}}\right).$$

The whole modified training data set D'_{train} is then composed of $(\boldsymbol{x}', \mathcal{Y})$ pairs.

It is important to notice that such modifications of the training dataset should be applied to test examples, because label sets of test examples are assumed to be unknown. Therefore, transformations are employed only to training data, the modified training examples are given to a classifier, but during testing the classifier is fed with test examples not affected by any such transformation. The following approach then breaks the usual scheme, that the processing pipeline from an input data feature vector to an output prediction vector is the same in the training and testing phases.

The four proposed modifications of the training dataset have a common form, given by Eq. (1):

$$\boldsymbol{x}'^{(i)} = \boldsymbol{x}^{(i)} + \alpha \cdot \left(\boldsymbol{d}^{(i)} - \boldsymbol{x}^{(i)} \right) = (1 - \alpha) \cdot \boldsymbol{x}^{(i)} + \alpha \cdot \boldsymbol{d}^{(i)}. \tag{1}$$

They are vector translations (shifts) of an i-th data point $\boldsymbol{x}^{(i)}$ towards some destination point $\boldsymbol{d}^{(i)}$ by some *step size* $\alpha \in [0, 1]$. The knowledge of $\mathcal{Y}^{(i)}$ as well as the rest of D_{train} is necessary to compute $\boldsymbol{d}^{(i)}$.

How exactly calculating the destination point of a given training dataset is the first differentiating factor of the following methods. $\boldsymbol{d}^{(i)}$ can be determined by global or local feature space examination. The second differentiating factor specifies the way of multi-labeled training examples treatment. These two factors taken together generate the following four methods of shifts in training data.

3.1 Shifts Towards Globally Defined Destinations

In the first two methods the direction towards which a data point is pushed is determined by examining the data set globally. Particularly, the shift destination point is the center of the class to which an object belongs. In this paper, a class center is chosen to be an average of all examples in the class (i.e. a simple *centroid* of the class), although a medoid representation of a class center should also work. The centroid $\boldsymbol{c}_i$ of a class C_i is given by:

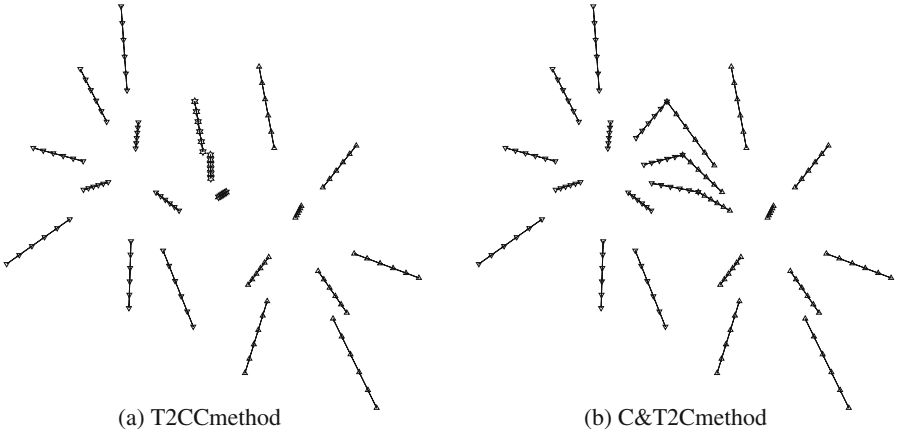
$$\boldsymbol{c}_i = \frac{\sum_{\boldsymbol{x} \in C_i} \boldsymbol{x}}{|C_i|} \tag{2}$$

Because of multi-label character of the problem, the arising question is how to move objects that belong to more than one class. Moving a point towards all of its classes' centers sequentially, i.e. in the direction of a centroid of one class, then towards other centroid of an affiliated class and then towards another and so on, is not a correct strategy. In effect, this would place the point nearest to the last class and farthest from the first class to which it was pushed. Therefore the object's classes would not be treated equally and this is unjustified. Instead, there are two other ways of translating multi-labeled objects having known the centers of their affiliated classes.

Translation to Centroid of Centroids (T2CC). In this approach, we average the centroids of the classes to which object $x^{(i)}$ belongs:

$$\boldsymbol{cc}^{(i)} = \frac{\sum_{j \in \mathcal{Y}^{(i)}} \boldsymbol{c}_j}{|\mathcal{Y}^{(i)}|} \tag{3}$$

After obtaining such *centroid of centroids* for each object, the object can be shifted in accordance to Eq. (1), with $\boldsymbol{d}^{(j)} = \boldsymbol{cc}^{(j)}$.



(a) T2CCmethod (b) C&T2Cmethod

Fig. 2. Comparison of two methods using global dataset investigation to calculate destination points and then translate towards them using α values ranging from 0.1 to 0.5. The difference is visible in the way in which multi-labeled objects are treated. Examples in the middle, depicted as stars, are assigned to both upside and downside triangle labels. T2CC calculates the centroid out of both labels centroids and moves the objects towards that single point, whereas C&T2C splits the objects into two single-labeled ones and translates them to respective class centers.

Copy and Translate (C&T2C). Here, we begin by decomposing each multi-labeled object in the dataset into single-labeled copies of it. That is, an object associated with p classes (i.e. $|\mathcal{Y}^{(i)}| = p$) is replaced by p copies of itself and each of these copies is marked with a different class from the set of original object's labels:

$$(\boldsymbol{x}, \{l_1, l_2, \ldots, l_p\}) \Longrightarrow (\boldsymbol{x}, \{l_1\}), (\boldsymbol{x}, \{l_2\}), \ldots, (\boldsymbol{x}, \{l_p\}) \tag{4}$$

After such operation, in C&T2C method each object is pushed according to (1), having $\boldsymbol{d}^{(j)}$ set to a suitable class centroid $\boldsymbol{c}$. Figure 2 illustrates the difference between T2CC and C&T2C translation methods in a simple 2D example.

It should be noticed that this dataset transformation, objects' *forking*, significantly increases the number of training objects. This may be a potential drawback for some applications, because much more memory space is needed to store the data. It also changes the character of the training data, which is not multi-labeled any more. In connection with the fact that the classification problem to solve is still multi-labeled, this means that the approach can be used only when working with classifiers capable of dealing with such situations.

3.2 Shifts Towards Locally Defined Destinations

In contrast to the above global methods, the local approach to calculate destination points for data shifts is presented here. The idea is that instead of choosing a class center as the point ($\boldsymbol{d}^{(i)}$) towards which executing a movement of a

feature vector $\boldsymbol{x}^{(i)}$, it is determined by m closest examples to $\boldsymbol{x}^{(i)}$ (with respect to some distance metric, e.g. cosine) among those that belong to the same class as $\boldsymbol{x}^{(i)}$ does (assuming for a while that it is single labeled). Therefore, they can be called m nearest *in-class* neighbors (*nin*-s) of the example.

When m *nin*-s of an $\boldsymbol{x}^{(i)}$ are found, constituting a set $\mathcal{N}_{\boldsymbol{x}^{(i)}}$, then $\boldsymbol{d}^{(i)}$ is simply the average of these points, analogously to class centroid calculation:

$$\boldsymbol{d}^{(i)} = \frac{\sum_{\mathcal{N}_{\boldsymbol{x}^{(i)}}} \boldsymbol{n}}{|\mathcal{N}_{\boldsymbol{x}^{(i)}}|} = \frac{\sum_{\mathcal{N}_{\boldsymbol{x}^{(i)}}} \boldsymbol{n}}{m} \tag{5}$$

For single-labeled datasets, the parameter m can be fixed to a constant value for all classes or it could be proportional to the size of the class to which $\boldsymbol{x}^{(i)}$ is assigned. For multi-labeled cases, the following two methods have been developed.

Translation to Centroid of Nearest In-Class Neighbors (T2CNIN). In this approach, the local destination point calculation given above is adapted to multi-labeled objects by changing the notion of an in-class object. Now, an example is considered as in-class with respect to some $\boldsymbol{x}^{(i)}$, if and only if *at least* one of this example's labels is in $\mathcal{Y}^{(i)}$ or, in other words, if the intersection between this example's $\mathcal{Y}$ and $\mathcal{Y}^{(i)}$ is not empty. With this adaptation, the rest of the procedure is the same as for a single-labeled case: the m nearest in-class neighbors have to be found, $\boldsymbol{d}^{(i)}$ is determined by applying Eq. 5, and the change of $\boldsymbol{x}^{(i)}$ by shift given in Eq. 1.

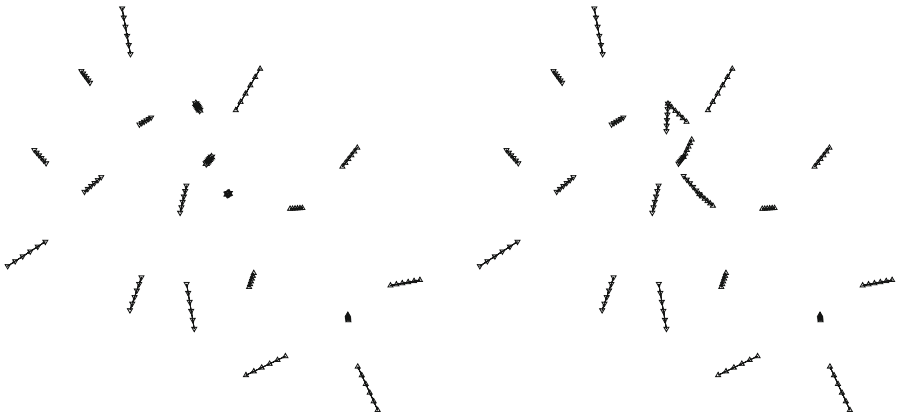
Of course, this method can have many other variants. For example, one can be more demanding in treating examples as an in-class *peers* to a given $\boldsymbol{x}^{(i)}$ by requiring the respective $\mathcal{Y}$ and $\mathcal{Y}^{(i)}$ to be the same (i.e. a label power set approach). Or, one can incorporate a weighted average, instead of Eq. 5, to vary the importance of m nearest neighbors, so that those with many common labels would pull stronger than those with, say, only one common label. In the experiments, these variants were not used.

Copy and Translate to Nearest In-Class Neighbors (C&T2NIN). The last proposed method is a result of applying local destination points calculation (Eq. 5) with previous multi- to single-label data transformation, presented by 4. After the object forking transformation, each example has exactly one label, so the notion of an in-class peer is not problematic, and Eq. 5 can be applied directly. Of course, all restrictions about the consequences of using the object forking transformation, described previously, apply here also.

The difference between the two local shift methods on a toy 2D example is shown in Fig. 3.

3.3 Pre- and Post-shift Data Pruning

For sparse data, the effect of points shifts in the feature space is the addition of many non-zero values, i.e. the sparseness of data is reduced. For example, the



(a) T2CNIN method (b) C&T2NIN method

Fig. 3. Comparison of two methods using local neighborhood investigation to calculate destination points and then translate towards them using α values ranging from 0.1 to 0.5, $m = 3$. Here also the difference lies in the way multi-labeled objects are treated. Objects in the middle, e.g., stars, are assigned to both upside and downside triangle labels. T2CNIN calculates centroids of in-class neighbors and moves the objects owards those points, whereas C&T2NIN splits the objects into two single-labeled ones and translates them to suitable in-class neighbors' centers.

average number of nonzero elements of a class centroid in our dataset (described in Sect. 4.1) is ca. 700, whereas the average number of nonzero elements of a single data example is 53. This means, that on average a shifted data example to a class centroid has over 1200 % nonzero elements of the original data example.

On the one hand, this is a massive data imputation, which was one of the aims of the methods and is beneficial for better classification. On the other hand, however, it significantly increases the space requirements for the transformed data which can cause memory problems. Therefore, in practice a deletion of small nonzero elements is needed to restrict the reduction of data sparsity. This is obtained in a process called *data pruning*. A pruning can be done by setting an *absolute threshold* value and zeroing feature values less than this threshold. The threshold can also be *relative*, in which case a feature is removed from the example, if its value *scaled* by example's highest feature value is below the threshold.

In the presented data shifts methods, pruning can be executed two times. First, after the calculation of the centroids (for global methods), the centroids can be immediately pruned, so that they contain only important features, that characterize classes. This can be called *pre-shift* pruning. For example, pruning the centroids of the above example with relative pruning $tr_{rel} = 0.05$, reduces their average number of nonzero elements from 700 to 311. After data movements, the obtained modified train dataset can also be pruned using the same technique. This process can be referenced to as *post-shift* pruning.

4 Experiments

4.1 The Dataset

The evaluation was performed using the dataset constructed from the corpus of articles from Simple English Wikipedia. It is a well known, frequently used dataset that has characteristic properties, such as: the data is highly multi-labeled (in some cases more than 5 labels), has many irregularities, contains noise and missing values, the articles are of different lengths, are not necessarily correctly assigned to categories (or not assigned to all categories to which they would fit well), and often have some less important content while missing crucial information. Simple English Wikipedia corpus is also suited well to test-bed experiments, because it is middle sized – big enough to exhibit most important properties of real problems, but computations upon it are relatively not much time consuming (all the data can fit at once into a typical computer memory).

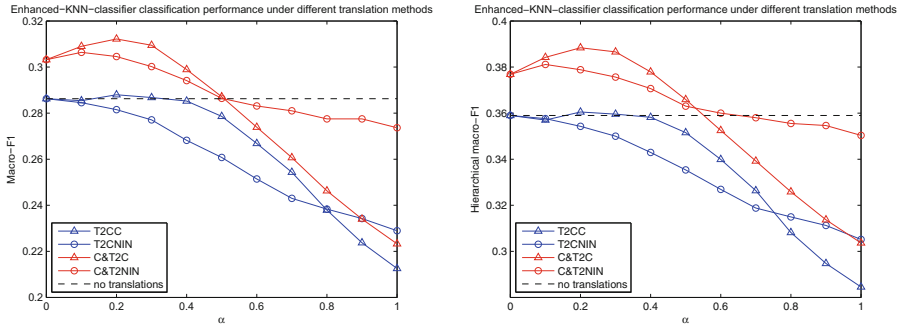
The corpus was preprocessed using Matrix'u software [18] to obtain a bag-of-words representation of the articles. The *confidence weighting* scheme [19] was used, which gives slightly better classification accuracy than typical TF-IDF feature weighting. From the original corpus, the articles with less than 5 features as well as the categories with less than 5 articles were removed. After such filtering, the dataset has 55637 articles, connected in a multi-label fashion with 5679 categories.

The dataset was divided into train and test sets in proportion 3:1, so that at least one article from every category is included in the test set. The problem of stratified partitioning a multi-labeled dataset is NP-hard, therefore a version of simulated annealing was used to obtain a satisfactory partition.

4.2 Results

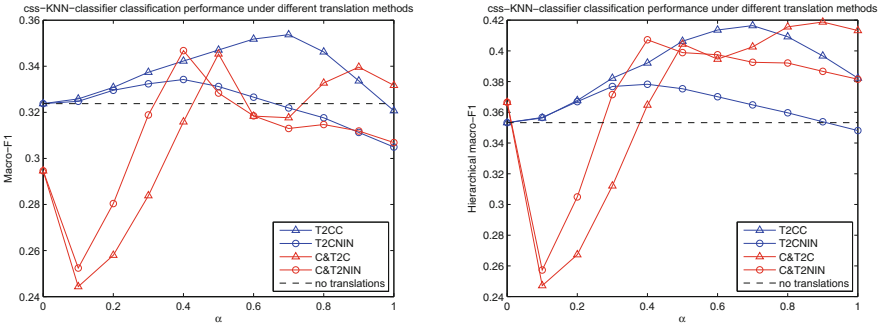
Figures 5 and 4 presents the quality of the test set examples classification using two KNN-based classifiers under train data examples shift. For comparison, the baseline (classification performance on raw training data) is also presented. For each method, the results were obtained by computation of destination points first and then gradually moving training data examples towards them (changing α in Eq.(1) from 0 to 1 by 0.1) each time checking the performance of classifiers on the same test set examples. The performance is expressed by a popular macro-averaged F1-score (abbrev. *maF*) [20]. Because Wikipedia category structure has a pseudo-hierarchical structure, besides typical "flat" maF, the measurements using a hierarchical version of it, abbreviated as *hMaF*, introduced in [21], are also presented (Fig. 5b and 4b).

It can be seen that all four methods can improve the training set leading to better classification performance. However, having chosen a method, it is very important to find a proper step size (α) value. The optimal α has to be large enough to reduce variance in the training data associated with noise, but small enough to maintain the variance resulting from class distribution in the feature space. Too small α may not increase the performance significantly and too big



(a) Macro F1-score (b) Hierarchical macro F1-score

Fig. 4. The results of Enhanced-KNN classifier [7] performance on a held out test set under train data translations using proposed methods. Simple English Wikipedia dataset, centroid pruning $th_{rel} = 0.05$, shifted data pruning $th_{rel} = 0.05$, neighborhood size for local methods: $m = 3$.



(a) Macro F1-score (b) Hierarchical macro F1-score

Fig. 5. The results of css-KNN classifier performance on a held out test set under train data translations using proposed methods. Simple English Wikipedia dataset, centroid pruning $th_{rel} = 0.05$, shifted data pruning $th_{rel} = 0.05$, neighborhood size for local methods: $m = 3$.

value may even cause the performance to be worse than the baseline. Indeed, for methods with globally calculated destination points, shifting too much towards class centroids makes KNN to behave similarly to centroids-based classifiers (like nearest centroid / Rocchio classifier), which is known to be inferior than KNN in most cases.

The degree of data smoothing in methods incorporating nearest in-class neighbors (nin) is controlled not only by the step size, but also by the number of nins involved in destination points calculation, i.e. the m parameter. The presented results were obtained with m set arbitrarily to 3 for all classes, without attempts to optimize it. Figure 5 shows that, css-KNN classifier is vulnerable

to multi-label examples forking (red lines, performance even without shifts, i.e. for $\alpha = 0$) and therefore initially exhibits worse quality than the baseline. However, it becomes better after increasing the step size to 0.4–0.5. Interestingly, for enhanced-KNN classifier, the multi-label examples splitting instantly improves the outcome quality. The methods' usefulness is then closely connected with a chosen classifier.

It should also be noted, that the presented results are obtained using pre- and post-shift data pruning. In both the relative thresholds were used with set to 0.05. For selected α values, we also tested the classification performance on less pruned data (and even without pre-translation pruning), but the obtained results were almost identical. However, there is much difference in memory space requirements, which is presented in Table 1. With our simple, *all-data-in-memory* implementation, some data pruning was needed to avoid out of memory problems.

Table 1. The impact of pre- and post-translation data pruning on the size of the modified training dataset (in MB). Simple English Wikipedia train data. Original train data size is 25.

		post-shift pruning				
		no pruning	relative 0.01	absolute 0.01	relative 0.05	absolute 0.05
pre-shift	no pruning	-	-	-	-	34
	relative 0.01	1060	149	78	40	34
	absolute 0.01	430	143	78	40	34
	relative 0.05	304	127	75	40	34
	absolute 0.05	90	74	60	40	34

5 Conclusions and Future Work

In our paper we present an approach for shifts in the training data for text classification based on KNN that allowed us to improve generalization of the categorization process. We proposed four variants of train data points shifts and shown that all of them can improve classification quality. The modifications perform a data smoothing, and, especially, can be seen as a type of missing data imputation.

The proposed modification lies before main classification phase and as a general approach can be used with other classification tools, thus improve their effectiveness. In future we plan to investigate the impact of training data modifications on results of the SVM and neural network models. For these models however, it may be necessary to *learn* a function that will perform a similar transformation, but in an unsupervised manner, i.e. without the knowledge of class labels, so that the transformations could be performed also on the test examples.

Besides this, future work will also be oriented in two main directions. Firstly we will investigate whether the approach for dealing with missing data can be applied to keyword search over databases that do not offer direct access to their instances (e.g. deep web databases, which can be typically accessed by query forms). In this case, the proposed technique can be used for estimating the assignment of a keyword to a particular database schema element, thus completing and extending our previous work [22]. Then we will adopt the technique in a recommender system, where we have to discover the users' preference rates for unrated items. In this case, unrated preference rates have to be thought of as missing value. We plan to perform this experimentation in the field of recommendation of web pages, where new pages can be proposed to a user on the basis of the analysis of the pages visited in the current session.

Acknowledgement. The authors would like to acknowledge networking support by the ICT COST Action IC1302 KEYSTONE - Semantic keyword-based search on structured data sources (www.keystone-cost.eu).

References

1. Draszawka, K., Szymanski, J.: Thresholding strategies for large scale multi-label text classifier. In: The 6th International Conference on Human System Interaction (HSI), 2013, pp. 350–355. IEEE (2013)
2. Joachims, T.: Transductive inference for text classification using support vector machines. In: ICML, vol. 99, pp. 200–209 (1999)
3. McCallum, A., Nigam, K., et al.: A comparison of event models for naive bayes text classification. In: AAAI-98 Workshop on Learning for Text Categorization, vol. 752, pp. 41–48. Citeseer (1998)
4. Sebastiani, F.: Machine learning in automated text categorization. ACM Comput. Surv. (CSUR) **34**, 1–47 (2002)
5. Westa, M., Szymański, J., Krawczyk, H.: Text classifiers for automatic articles categorization. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2012, Part II. LNCS, vol. 7268, pp. 196–204. Springer, Heidelberg (2012)
6. Tan, S.: Neighbor-weighted k-nearest neighbor for unbalanced text corpus. Expert Syst. Appl. **28**, 667–671 (2005)
7. Wang, X., Zhao, H., Lu, B.: Enhanced k-nearest neighbour algorithm for largescale hierarchical multi-label classification. In: Proceedings of the Joint ECML/PKDD PASCAL Workshop on Large-Scale Hierarchical Classification, Athens, Greece, vol. 5 (2011)
8. Zhou, Y., Li, Y., Xia, S.: An improved knn text classification algorithm based on clustering. J. Comput. **4**, 230–237 (2009)
9. Zhang, M.L., Zhou, Z.H.: Ml-knn: a lazy learning approach to multi-label learning. Pattern Recogn. **40**, 2038–2048 (2007)
10. Read, J.: Scalable multi-label classification. Ph.D. thesis, University of Waikato (2010)
11. Yu, H., Yang, J., Han, J., Li, X.: Making svms scalable to large data sets using hierarchical cluster indexing. Data Min. Knowl. Disc. **11**, 295–321 (2005)

12. Tang, L., Rajan, S., Narayanan, V.K.: Large scale multi-label classification via metalabeler. In: Proceedings of the 18th International Conference on World Wide Web, pp. 211–220. ACM (2009)
13. Kaiser, J.: Dealing with missing values in data. J. Syst. Integr. **5**, 42–51 (2014)
14. Grzymała-Busse, J.W., Hu, M.: A comparison of several approaches to missing attribute values in data mining. In: Ziarko, W.P., Yao, Y. (eds.) RSCTC 2000. LNCS (LNAI), vol. 2005, p. 378. Springer, Heidelberg (2001)
15. Little, R.J.A., Rubin, D.B.: Statistical Analysis with Missing Data, 2nd edn. Wiley, New York (2002)
16. Farhangfar, A., Kurgan, L.A., Dy, J.G.: Impact of imputation of missing values on classification error for discrete data. Pattern Recogn. **41**, 3692–3705 (2008)
17. Juan, A., Ney, H.: Reversing and smoothing the multinomial naive bayes text classifier. In: PRIS, pp. 200–212. Citeseer (2002)
18. Szymanski, J.: Comparative analysis of text representation methods using classification. Cybern. Syst. **45**, 180–199 (2014)
19. Soucy, P., Mineau, G.W.: Beyond tfidf weighting for text categorization in the vector space model. IJCAI. **5**, 1130–1135 (2005)
20. Tsoumakas, G., Vlahavas, I.P.: Random k-Labelsets: an ensemble method for multilabel classification. In: Kok, J.N., Koronacki, J., Lopez de Mantaras, R., Matwin, S., Mladenič, D., Skowron, A. (eds.) ECML 2007. LNCS (LNAI), vol. 4701, pp. 406–417. Springer, Heidelberg (2007)
21. Kiritchenko, S., Matwin, S., Nock, R., Famili, A.F.: Learning and evaluation in the presence of class hierarchies: application to text categorization. In: Lamontagne, L., Marchand, M. (eds.) Canadian AI 2006. LNCS (LNAI), vol. 4013, pp. 395–406. Springer, Heidelberg (2006)
22. Bergamaschi, S., Domnori, E., Guerra, F., Trillo-Lado, R., Velegrakis, Y.: Keyword search over relational databases: a metadata approach. In: SIGMOD, pp. 565–576. ACM (2011)