Using Feature Conjunctions across Examples for Learning Pairwise Classifiers

Satoshi Oyama[1] and Christopher D. Manning[2]

[1] Department of Social Informatics, Kyoto University,
Kyoto 606-8501, Japan
oyama@kuis.kyoto-u.ac.jp
http://www.dl.kuis.kyoto-u.ac.jp/~oyama/
[2] Department of Computer Science, Stanford University,
Stanford, CA 94305-9040, USA
manning@cs.stanford.edu
http://www-nlp.stanford.edu/~manning/

Abstract. We propose a kernel method for using combinations of features across example pairs in learning pairwise classifiers. Identifying two instances in the same class is an important technique in duplicate detection, entity matching, and other clustering problems. However, it is a difficult problem when instances have few discriminative features. One typical example is to check whether two abbreviated author names in different papers refer to the same person or not. While using combinations of different features from each instance may improve the classification accuracy, doing this straightforwardly is computationally intensive. Our method uses interaction between different features without high computational cost using a kernel. At medium recall levels, this method can give a precision 4 to 8 times higher than that of previous methods in author matching problems.

1 Introduction

Pairwise classifiers, which identify whether two instances belong to the same class or not, are important components in duplicate detection, entity matching, and other clustering applications. For example, in citation matching [8], two citations are compared and determined whether they refer to the same paper or not (Figure 1).

In early work, these classifiers were based on fixed or manually tuned distance metrics. In recent years, there have been several attempts to make pairwise classifiers automatically from labeled training data using machine learning techniques [1, 2, 11, 14]. Most of them are based on string edit distance or common features between two examples.

These methods are effective when two instances from the same class have many common features like two variant citations to the *same* paper. However, if two instances from the same class have few common features, these methods have difficulties in finding these pairs and achieving high recall. An instance of this is trying to identify the same author across *different* papers.

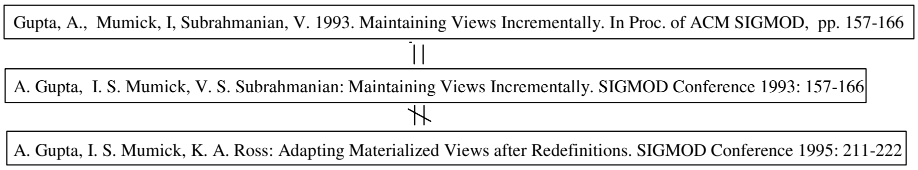

Fig. 1. Matching citations

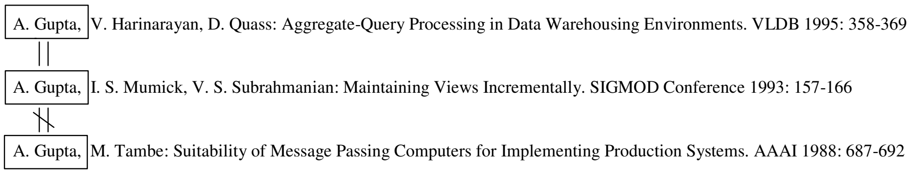

Fig. 2. Matching authors

First names of authors are abbreviated to initials in many citations. As shown in Figure 2, identifying the same authors among abbreviated names is another important problem in citation analysis or evaluating researchers. However, fielded citation databases such as ISI Citation Index[3] or "Most Cited Authors in Computer Science" in CiteSeer[4] cannot distinguish different authors with the same first initial and the same last name. Distinguishing these authors is important for treating people as first class entities in citation databases.

Matching authors is a harder problem than matching citations. As we can see in Figure 1, two citations to the same paper have many common keywords. Conversely, if two citation strings are the same or have many common keywords, we can suspect the two citations refer to the same paper. On the other hand, even if two strings of author names are exactly same, we cannot conclude these names refer to the same person in the real world. To disambiguate author names, we have to look into other fields in citations than author names themselves.

However, there is another difficulty in this problem. The first two records in Figure 2 have no common words other than the names of the first authors even though these two authors are the same person. Humans can somehow infer the identity of these two persons by considering the strong connection between the two conferences and the topical similarity between words in the titles. However, in such a case, where pairs from the same class have few common features, it is difficult to automatically determine these pairs using pairwise classifiers based on string similarity or common features.

One approach to solving this problem is using conjunctions of features across examples. In the case of Figure 2, we could give similarities to different words across examples like "SIGMOD" and "VLDB", and compute the overall similarity based on them. This helps avoiding zero similarity and breaking orthogonal-

[3] http://isiknowledge.com/
[4] http://citeseer.ist.psu.edu/mostcited.html

ity. If there are many pairs where one paper is published in VLDB and the other paper is published in SIGMOD in labeled positive data (pairs of papers authored by the same person), we can expect that the learning algorithm incorporates this feature into the classifier. However, if we straightforwardly make all pairs from original features, the dimension of the resulting feature space become large and we cannot apply learning algorithms to real problems with many features.

In this paper, we propose a method for using feature conjunctions across examples in learning pairwise classifiers without causing excessive computational cost by using kernel methods [12, 13]. By using our kernel, learning algorithms can use feature conjunctions across examples without actually computing them. This results in high classification accuracy for problems with few common features, which are difficult for existing methods.

2 Pairwise Classification

2.1 Problem Definition

Pairwise classification is the problem of determining whether a pair of instances, $\boldsymbol{x}^\alpha$ and $\boldsymbol{x}^\beta$, belong to the same class or not. In a binary classification case, we look for the following function:

$$f(\boldsymbol{x}^\alpha, \boldsymbol{x}^\beta) = \begin{cases} 1 & (\text{if } \boldsymbol{x}^\alpha \text{ and } \boldsymbol{x}^\beta \text{ belong to the same class}), \\ -1 & (\text{otherwise}). \end{cases}$$

Pairwise classification and pairwise similarity have a close relation. We can also consider a problem where the function f outputs continuous values such as $f(\boldsymbol{x}^\alpha, \boldsymbol{x}^\beta) \in [0, 1]$, which give similarities between instances. We can change this into a binary classifier by introducing a certain threshold. On the other hand, many binary classifiers can be converted to a classifier that outputs continuous values [2]. Therefore, we will sometimes use the terms pairwise classification and pairwise similarity interchangeably.

Pairwise classification is an important component in duplicate detection, identity matching, and many other clustering problems. We make a global clustering decision based on pairwise classifications, for instance, by making the transitive closure of guessed positive pairs [2].

2.2 Making Pair Instances from the Original Data

It is a difficult problem to define accurate pairwise classifiers by hand. Thus there have been many works on inducing classifiers automatically from data using machine learning techniques. Many earlier methods first sample pair instances from the data and have humans label them according to whether they belong to the same class or not. Then these training examples are fed to binary classifier learning algorithms such as Support Vector Machines (SVMs) [15].

For example, Bilenko and Mooney [2] represent an original instance by a feature vector $\boldsymbol{x}^\alpha = (x_1^\alpha, x_2^\alpha, \ldots, x_n^\alpha)$, where each feature corresponds to whether a word in a vocabulary appears in the string representation of the instance and the dimension of feature vectors n is the number of words in the vocabulary. From two original instances $\boldsymbol{x}^\alpha = (x_1^\alpha, x_2^\alpha, \ldots, x_n^\alpha)$ and $\boldsymbol{x}^\beta = (x_1^\beta, x_2^\beta, \ldots, x_n^\beta)$, they make a pair instance $\hat{\boldsymbol{x}} = (\boldsymbol{x}^\alpha, \boldsymbol{x}^\beta)$ and represent it as a vector in an n dimensional feature space:

$$\hat{\boldsymbol{x}}_{common} = (x_1^\alpha x_1^\beta, x_2^\alpha x_2^\beta, \ldots, x_n^\alpha x_n^\beta) \ . \tag{1}$$

(They also do normalization by dividing the value of each feature by $|\boldsymbol{x}^\alpha||\boldsymbol{x}^\beta|$.)

This method is effective for a problem like citation matching, where two instances from the same class have many common features. However, in the problem where two instances from the same class have few common features, this method cannot achieve high classification accuracy. For example, in Figure 2, the first and the second papers have no common words other than "A. Gupta" even though they are actually written by the same person. The representation of this pair instance by Equation (1) becomes a zero vector. This phenomenon is not rare in papers written by the same author, and the method based on common features cannot distinguish these pairs from the many negative examples that also have zero vectors as their representation.

One approach to avoiding the problem of zero vectors is using conjunctions of different features across examples $x_i^\alpha x_j^\beta$ and representing a pair instance as

$$\hat{\boldsymbol{x}}_{Cartesian} = (x_1^\alpha x_1^\beta, \ldots, x_1^\alpha x_n^\beta, x_2^\alpha x_1^\beta, \ldots, x_2^\alpha x_n^\beta, \ldots, x_n^\alpha x_1^\beta, \ldots, x_n^\alpha x_n^\beta) \ . \tag{2}$$

That is, the set of mapped features, $\{x_i^\alpha x_j^\beta | i = 1, \ldots, n; j = 1, \ldots, n\}$, is a Cartesian product between the sets of original features of $\boldsymbol{x}^\alpha$ and $\boldsymbol{x}^\beta$. In this feature space, a pair instance does not become a zero vector unless one of the original instances is a zero vector. If there are many positive pairs in which "VLDB" appears in one citation and "SIGMOD" appears in the other, we can expect that a learning algorithm incorporates the conjunction of these two features into the learned classifier, and it successfully classifies the case of Figure 2.

However, implementing this idea straightforwardly causes the following problems. One is that the dimension of the feature space becomes n^2 and the computational cost becomes prohibitive for practical problems with many features. Moreover, learning in a high dimensional feature space is in danger of overfitting, that is, the "curse of dimensionality."

3 Kernel Methods for Using Feature Conjunctions across Examples

3.1 Kernel Methods

Some learning algorithms such as SVMs can be written in forms where examples always appear as inner products $\langle \boldsymbol{x} \cdot \boldsymbol{z} \rangle$ of two examples and never appear individually [12, 13]. Kernel methods enable classification in higher dimensional space by substituting kernel functions $K(\boldsymbol{x}, \boldsymbol{z})$ for inner products $\langle \boldsymbol{x} \cdot \boldsymbol{z} \rangle$ in these algorithms.

Let us consider the following kernel function:

$$K(\boldsymbol{x}, \boldsymbol{z}) = \langle \boldsymbol{x} \cdot \boldsymbol{z} \rangle^2 \ . \tag{3}$$

Learning with this kernel function is equivalent to mapping examples into the following higher dimensional feature space,

$$\boldsymbol{\phi}(\boldsymbol{x}) = (x_1 x_1, \ldots, x_1 x_n, x_2 x_1, \ldots, x_2 x_n, \ldots, x_n x_1, \ldots, x_n x_n) \ ,$$

and then applying the learning algorithm. We can show this as follows:

$$\begin{aligned} \langle \boldsymbol{x} \cdot \boldsymbol{z} \rangle^2 &= \left(\sum_{i=1}^{n} x_i z_i \right) \left(\sum_{j=1}^{n} x_j z_j \right) = \sum_{i=1}^{n} \sum_{j=1}^{n} x_i z_i x_j z_j \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} (x_i x_j)(z_i z_j) = \langle \boldsymbol{\phi}(\boldsymbol{x}) \cdot \boldsymbol{\phi}(\boldsymbol{z}) \rangle \ . \end{aligned}$$

The kernel above is called a quadratic polynomial kernel. Previous work has also used another popular kernel, the Gaussian kernel, $K(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{z}\|^2}{2\sigma^2} \right)$, which corresponds to a feature mapping into an infinite dimensional space. Using kernels, the algorithms can learn classifiers in a high dimensional feature space without actually doing feature mappings, which are computationally expensive. Moreover, SVMs with kernels are known to be robust against the overfitting problem of learning in a high dimensional feature space.

3.2 Using Kernels for Feature Conjunctions

A straightforward way to conjoin different features across examples is using the polynomial kernel mentioned above. We represent a pair instance as a vector with $2n$ dimensions,

$$\hat{\boldsymbol{x}} = (\boldsymbol{x}^{\alpha}, \boldsymbol{x}^{\beta}) = (x_1^{\alpha}, \ldots, x_n^{\alpha}, x_1^{\beta}, \ldots, x_n^{\beta}) \ , \tag{4}$$

and then apply the kernel of Equation (3) on this feature space.

The set of conjoined features resulting from the corresponding feature mapping is $\{x_i^{\alpha} x_j^{\alpha}\} \cup \{x_i^{\alpha} x_j^{\beta}\} \cup \{x_i^{\beta} x_j^{\alpha}\} \cup \{x_i^{\beta} x_j^{\beta}\}$ and it includes the set of features in Equation (2). However, it also includes features from the same example, $\{x_i^{\alpha} x_j^{\alpha}\}$ and $\{x_i^{\beta} x_j^{\beta}\}$. These features are clearly irrelevant to pairwise classification because they are related to only one party of the pair. When the frequencies of original instances are different between the set of positive pairs and the set of negative pairs, there is a possibility that a learning algorithm give weight to joint features from single parties and the generalization performance deteriorates.

What we want is the following feature mapping, which generates conjunctions of features only across the two original instances:

$$\begin{aligned}\boldsymbol{\phi}(\hat{\boldsymbol{x}}) &= \boldsymbol{\phi}((\boldsymbol{x}^\alpha, \boldsymbol{x}^\beta)) \\ &= (x_1^\alpha x_1^\beta, \ldots, x_1^\alpha x_n^\beta, x_2^\alpha x_1^\beta, \ldots, x_2^\alpha x_n^\beta, \ldots, x_n^\alpha x_1^\beta, \ldots, x_n^\alpha x_n^\beta) \ .\end{aligned} \tag{5}$$

So we propose using the following kernel for pair instances:

$$K(\hat{\boldsymbol{x}}, \hat{\boldsymbol{z}}) = K((\boldsymbol{x}^\alpha, \boldsymbol{x}^\beta), (\boldsymbol{z}^\alpha, \boldsymbol{z}^\beta)) = \langle \boldsymbol{x}^\alpha \cdot \boldsymbol{z}^\alpha \rangle \langle \boldsymbol{x}^\beta \cdot \boldsymbol{z}^\beta \rangle \ . \tag{6}$$

This kernel first computes the inner product of $\boldsymbol{x}^\alpha$ and $\boldsymbol{z}^\alpha$ and that of $\boldsymbol{x}^\beta$ and $\boldsymbol{z}^\beta$ respectively, then computes the product of these two real values.

We can show that this kernel does the feature mapping of Equation (5):

$$\begin{aligned}\langle \boldsymbol{x}^\alpha \cdot \boldsymbol{z}^\alpha \rangle \langle \boldsymbol{x}^\beta \cdot \boldsymbol{z}^\beta \rangle &= \left(\sum_{i=1}^{n} x_i^\alpha z_i^\alpha \right) \left(\sum_{j=1}^{n} x_j^\beta z_j^\beta \right) = \sum_{i=1}^{n} \sum_{j=1}^{n} x_i^\alpha z_i^\alpha x_j^\beta z_j^\beta \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} (x_i^\alpha x_j^\beta)(z_i^\alpha z_j^\beta) = \langle \boldsymbol{\phi}((\boldsymbol{x}^\alpha, \boldsymbol{x}^\beta)) \cdot \boldsymbol{\phi}((\boldsymbol{z}^\alpha, \boldsymbol{z}^\beta)) \rangle = \langle \boldsymbol{\phi}(\hat{\boldsymbol{x}}) \cdot \boldsymbol{\phi}(\hat{\boldsymbol{z}}) \rangle \ .\end{aligned}$$

This kernel is a tensor product [6] of two linear kernels (inner products) on the original feature space. An intuitive explanation of this kernel is the following. A kernel defines similarity in an input space. In our case, the input space is the space of pair instances. The kernel of Equation (6) defines the similarity between pair instances so that it yields a high value only if each of the original instances in one pair has high similarity (a large value for the inner product) with the corresponding original instance in the other pair. For example, if the value of $\langle \boldsymbol{x}^\alpha \cdot \boldsymbol{z}^\alpha \rangle$ is 0, the overall value of Equation (6) always becomes 0 even if the value of $\langle \boldsymbol{x}^\beta \cdot \boldsymbol{z}^\beta \rangle$ is large. This is a desirable property because a pairwise classification decision should be based on both examples in a pair.

The feature mapping of Equation (5) depends on the order of instances, that is, $\boldsymbol{\phi}((\boldsymbol{x}^\alpha, \boldsymbol{x}^\beta)) \neq \boldsymbol{\phi}((\boldsymbol{x}^\beta, \boldsymbol{x}^\alpha))$. We can also make a product between two pair instances by making inner products between instances with different superscripts as $\langle \boldsymbol{x}^\alpha \cdot \boldsymbol{z}^\beta \rangle \langle \boldsymbol{x}^\beta \cdot \boldsymbol{z}^\alpha \rangle$. However, if we have both $(\boldsymbol{x}^\alpha, \boldsymbol{x}^\beta)$ and $(\boldsymbol{x}^\beta, \boldsymbol{x}^\alpha)$ in the training set, both definitions of kernels are equivalent in terms of learned classifiers.

4 Experiments

4.1 Datasets and Code

We show experimental results on the following two datasets. One is the DBLP dataset which is a bibliography of more than 400,000 computer science papers.[5] The data is publicly available in XML format. We used journal papers and

[5] http://dblp.uni-trier.de/

conference papers as the data for our experiments. Bibliographic entries in DBLP were entered by humans and many author names are given as their full names. To make a training set and a test set for the author matching problem, we abbreviated first names into initials and removed middle names. We used words in titles, journal names, and names of coauthors as features.

The other is the Cora Citation Matching dataset provided by Andrew McCallum.[6] We used these data for citation matching problems. They are also used in [2] and [3]. The dataset is composed of 1,879 citations to 191 papers in Cora, a computer science research paper search engine. We used each word appearing in citations as a feature.

We used SVM^{light}, an implementation of an SVM learning algorithm developed by Thorsten Joachims [7]. SVM^{light} provides basic kernels such as polynomial and Gaussian, and allow use of user-defined kernels. We implemented the kernel of Equation (6) for our experiments.

4.2 Results

From among the top 20 "Most Cited Authors in Computer Science,"[7] we selected four cases of first-initial-plus-surname names which involve a collapsing of many distinct authors (that is, we select names like *J. Smith* but not *J. Ullman*). To make a training set and a test set for each abbreviated name, we retrieved papers written by authors with the same last name and the same first initial from the DBLP data. If we make all pairs of instances, the number of negative examples becomes much larger than that of positive examples because the number of pairs from different classes is larger than that of pairs from same classes. To assess the effect of the imbalance between the numbers of positive and negative data, we prepared two different datasets. One is the *imbalanced data sets*, for which we generated pair instances from all combinations of two papers. The other is the *balanced data sets*, for which we first generated positive examples by making all combinations of papers in same classes, and then we generated negative examples by randomly sampling pairs from different classes. We evaluated classifiers learned from (im)balanced training sets by (im)balanced test sets respectively.

We trained classifiers on the training sets and evaluated them on the test sets in terms of precision and recall. In [2], the precision and recall are evaluated after making the transitive closure of guessed positive pairs. To make evaluation focused on the accuracy of pairwise classifiers, we calculated the precision and recall simply based on how many positive pairs classifiers can find. As suggested in [3], we drew precision-recall curves by shifting the decision boundary induced by SVMs and evaluating precision values on 20 different recall levels.

We evaluated the performance of classifiers with three different kernels.

Common The Gaussian kernel applied to pair instances of Equation (1), which uses only common features across examples. The parameter of the Gaussian kernel is set to $\sigma = 10$, according to the preceding work [2].

[6] http://www.cs.umass.edu/~mccallum/code-data.html
[7] http://citeseer.ist.psu.edu/mostcited.html

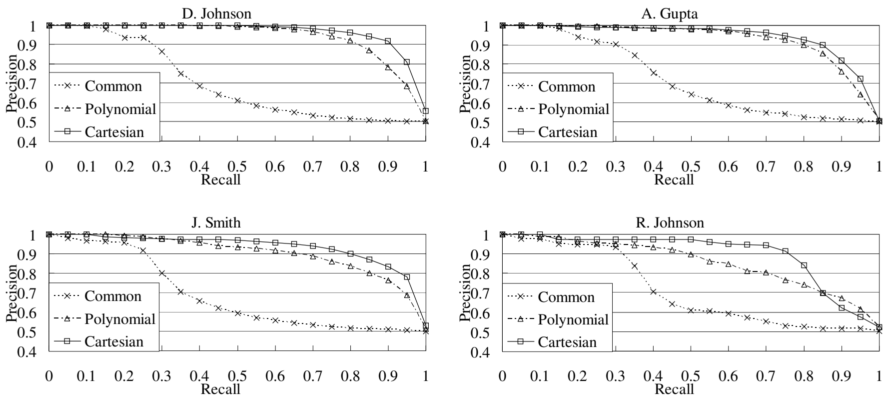

Fig. 3. Results of author matching problems with the balanced data sets

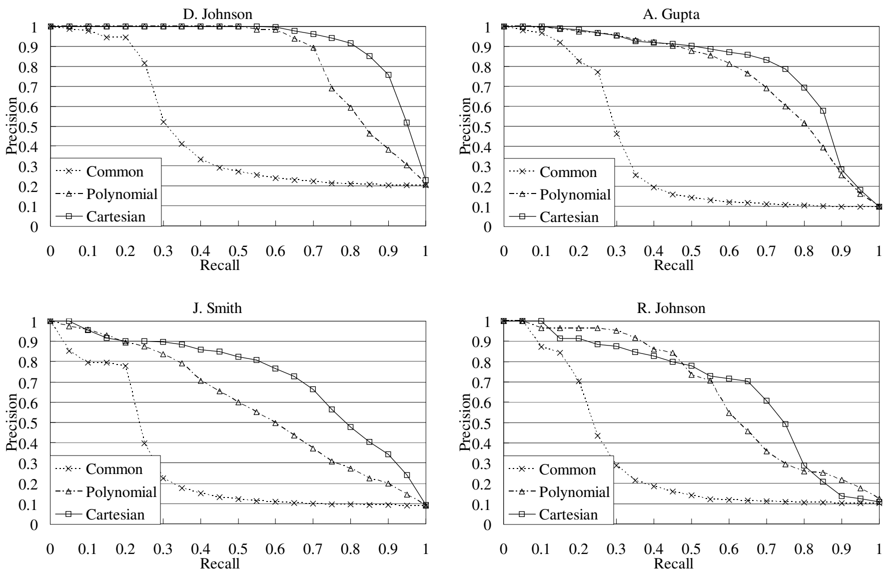

Fig. 4. Results of author matching problems with the imbalanced data sets

Polynomial The quadratic polynomial kernel applied to pair instances of Equation (4)
Cartesian Our kernel of Equation (6)

Figure 3 shows the results with the balanced data sets. For low recall levels, **Common** yields high precision values. However, when recall levels become larger than a certain threshold, the precision start to decrease drastically. This

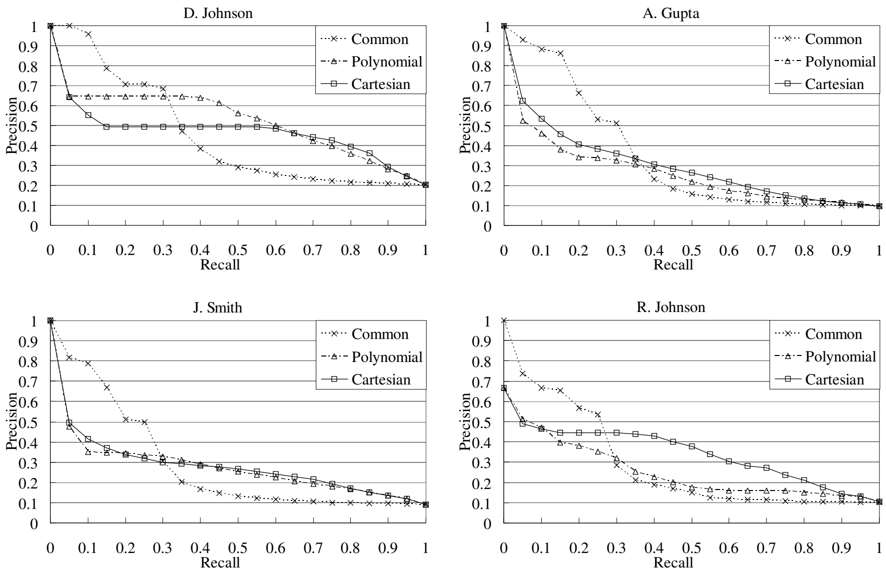

Fig. 5. Results of author matching problems by a general classifier

seems to be because there are many (nearly) zero vectors among positive pairs generated by Equation (1) and these positive examples cannot be distinguished from negative pairs. On the other hand, **Polynomial** and **Cartesian** keep high precision in higher recall levels. Among the two kernels, **Cartesian** generally yields higher precision than **Polynomial**. Figure 4 shows the results with the imbalanced data sets. As in the case of the balanced data sets, the methods using feature conjunctions are much superior to the method using only common features. **Cartesian** can give a precision 4 to 8 times higher than that of **Common** at medium recall levels.

In the above experiments, we trained different classifiers for each abbreviated author name. A general classifier, which can identify papers written by the same author, given any pair of papers, is preferable because we need not train many classifiers. It could also classify papers by new authors, for which we do not have enough training data. We trained a general classifier in the following steps. First, we listed 50,000 authors who have more than one paper in the DBLP dataset, according to their alphabetical order. For each author, we chose two papers randomly and use the pair as a positive training example. Then we made pairs of papers written by different authors and prepared the same number of negative training examples. We used the same test data that was used in the experiments in Figure 4. We present the results by a general classifier in Figure 5. In the region where recall is smaller than 0.3, **Common** get better results than the others. For the higher recall levels, however, it gives worse results than **Cartesian** and **Common**, among which **Cartesian** generally yields better

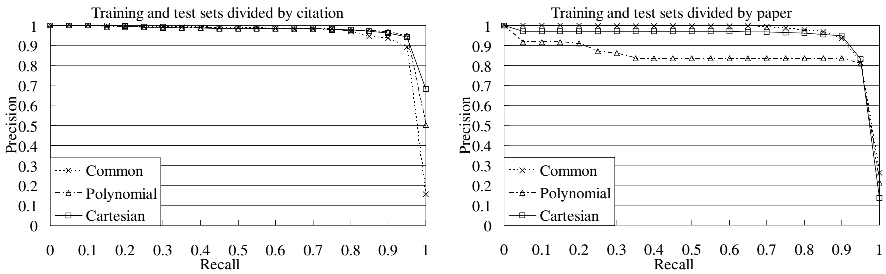

Fig. 6. Results of citation matching problems with the Cora dataset

precision. The overall results are not as good as those of specific classifiers. This seems to be because the word distributions in the test sets for the specific abbreviated names are different from the distribution in the training set collected using many different names. Similar phenomena are also reported in [2].

Figure 6 shows the results of *citation* matching (as opposed to *author* matching) with the Cora dataset. We tried two different methods for splitting the data into a training set and a test set according to [3]. One method simply assigns each citation into the training set or the test set without considering the paper it refers to. The other method first assigns each paper into the training or the test set, then assigns all citations to the paper to the same set. For the Cora dataset, **Common** works as well as **Cartesian**. In citation matching, two citations to the same paper usually have many common words and the learning algorithm can find clues to identify these examples by only using their common features.

5 Related Work

A general definition of a tensor product between two kernels can be found in [6]. A tensor product between kernels on input space and output space is used in [16] for inducing metrics both Euclidean and Fisher separable spaces, on which the triangle inequality is satisfied and a distance between examples from different classes is always larger than a distance between examples from the same class. On the other hand, our work uses a tensor product between two kernels both on input space for solving the problem of zero similarities between examples, which has not been addressed in the preceding work.

Duplicate detection and entity matching have been studied for a long time in the database community. Recently these problems have attracted interest from machine learning researchers. We refer to only recent literature using machine learning technologies. Other conventional approaches are summarized by Bilenko *et al.* [1].

The work by Bilenko and Mooney [2] is most relevant to our research. They proposed using learnable domain-specific string similarities for duplicate detection and showed the advantage of their approach against fixed general-purpose similarity measures. They trained a different similarity metric for each database field and calculated similarities between records composed of multiple fields by combining these field level similarities. In our work, we treated examples as single field datasets and focused on improving single level similarity measures. Sarawagi and Bhamidipaty [11] and Tejada *et al.* [14] used active learning for interactively finding informative training examples when learning classifiers that distinguish between duplicated pairs and non-duplicated ones. Their method need only a small number of training examples to obtain high accuracy classifiers. Bilenko and Mooney [3] advocated using precision-recall curves in evaluation of duplicate detection methods, and they also discussed different methods of collecting training examples for learnable duplicate detection systems.

6 Future Work

In this paper, we presented entity matching in citation databases as an application of our method. Entity matching has been also studied in natural language processing under the name of coreference resolution, where coreferent terms in texts are to be identified [9, 10]. Our future work includes application of the proposed method for this problem.

Our approach is applicable to the problem of matching different kinds of object. For example, English texts and Japanese texts have few common words. However, our method could learn a matching function between them without using external information sources like dictionaries. Our method can learn similarities between objects of different kinds based on similarities between objects of the same kind. This indicates great potential of our approach because there is no straightforward way to define similarity between completely different kinds of objects like texts and images while defining similarities between two texts and similarities between two images is much easier.

As mentioned in Section 4, learning pairwise classifiers faces the problem of imbalanced data. Employing techniques for handling imbalanced data [4] could improve the accuracy. We will also compare our supervised approach with unsupervised dimension reduction approaches such as LSI [5] for sparse data.

7 Conclusion

Pairwise classification is an important technique in entity matching. Preceding methods have difficulty in learning precise classifiers for problems where examples from the same class have few common features. Since similarities between examples from the same class become small, classifiers fail to distinguish positive pairs from negative pairs. To solve this problem, we proposed using conjunctions of features across examples in learning pairwise classifiers. Using a kernel on pair instances, our method can use feature conjunctions without causing a large computational cost. Our experiments on the author matching problem show that the new kernel introduced here yields higher precision than existing methods at middle to high recall levels.

Acknowledgements

This research was partially supported by the Informatics Research Center for Development of Knowledge Society Infrastructure (Kyoto University 21st Century COE Program) and by a Grants-in-Aid for Scientific Research (16700097) from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

References

1. M. Bilenko, W. W. Cohen, S. Fienberg, R. J. Mooney, and P. Ravikumar. Adaptive name-matching in information integration. *IEEE Intell. Syst.*, 18(5):16–23, 2003.
2. M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proc. KDD-2003*, pages 39–48, 2003.
3. M. Bilenko and R. J. Mooney. On evaluation and training-set construction for duplicate detection. In *Proc. KDD-2003 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, pages 7–12, 2003.
4. N. Chawla, N. Japkowicz, and A. Kolcz, editors. *Special Issue on Learning from Imbalanced Datasets, SIGKDD Explorations*, 6(1), 2004.
5. S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *J. Am. Soc. Inf. Sci.*, 41(6):391–407, 1990.
6. D. Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, Baskin School of Engineering, University of California, Santa Cruz, 1999.
7. T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT-Press, 1999.
8. S. Lawrence, K. Bollacker, and C. L. Giles. Autonomous citation matching. In *Proc. Third International Conference on Autonomous Agents*, 1999.
9. A. McCallum and B. Wellner. Toward conditional models of identity uncertainty with application to proper noun coreference. In *Proc. IJCAI Workshop on Information Integration on the Web*, pages 79–84, 2003.
10. T. S. Morton. Coreference for NLP applications. In *Proc. ACL-2000*, 2000.
11. S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proc. KDD-2002*, pages 269–278, 2002.
12. B. Schölkopf and A. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*. MIT Press, 2002.
13. J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.
14. S. Tejada, C. A. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *Proc. KDD-2002*, pages 350–359, 2002.
15. V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 2nd edition, 1999.
16. Z. Zhang. Learning metrics via discriminant kernels and multidimensional scaling: Toward expected euclidean representation. In *Proc. ICML-2003*, pages 872–879, 2003.