# KNN and Re-ranking Models for English Patent Mining at NTCIR-7

Tong Xiao, Feifei Cao, Tianning Li,
Guolong Song, Ke Zhou, Jingbo Zhu, Huizhen Wang
*Natural Language Processing Laboratory*
*Northeastern University*
*Shenyang, Liaoning, P.R. China 110004*
*{xiaotong,zhujingbo,wanghuizhen}@mail.neu.edu.cn*
*{caoff,litn,songgl,zhouke}@ics.neu.edu.cn*

## Abstract

*This paper describes our English patent mining system for NTCIR-7 patent mining task which maps a research paper abstract into IPC taxonomy. Our system is basically under the k-Nearest Neighboring framework, in which various similarity calculation and ranking methods are used. We employ two re-ranking techniques to improve the performance by the use of richer features. Our systems performed well on the NTCIR-7 patent mining task (English sub-task) and obtained the best MAP-measure among all the participations.*

**Keywords***: patent mining, IPC, KNN, re-ranking*

## 1. Introduction

Classifying research papers and patents is important for real-world patent processing applications such as invalidity search and technical trend analysis/mining [7]. Along this research line, we developed a patent mining system for NTCIR-7 English patent mining sub-task, in which research papers (or abstracts) can be automatically mapped into patent classification taxonomy, namely International Patent Classification (IPC).

Our system is developed under the k-Nearest Neighboring (KNN) framework, in which various similarity calculation and ranking methods are used. To achieve further improvement, two different re-ranking methods are proposed to learn a better ranking from multiple rankers, in which *rank combination* and *RankSVM* are used respectively. Our experimental results on NTCIR-7 English patent mining data sets show that our system performed well on this task. Both of the re-ranking-based systems outperform the basic system significantly. In the following parts of this paper, we would present a detailed description of our system.

## 2. Problem description

### 2.1. Problem of patent classification

To make the patent documents accessible to anyone who needs them, patent classification is required to allow patent documents related to any technology field can be retrieved and identified. International Patent Classification is such a classification system. The IPC taxonomy consists of over 60,000 fields or groups. Each group is described by a "classification symbol" called IPC code. A patent is generally assigned to one or multiple IPC codes that indicate the related technical field or fields.

In NTCIR-7 English patent mining sub-task, our system aims to assign one or multiple appropriate IPC codes to a non-patent document. To achieve such goal, NTCIR-7 provides us with such a task that categorizes research papers into IPC based on titles and abstracts. That is, the input to our system is the title and abstract of a published research paper. The output of our system is a list in which IPC codes are ranked in similarity scores. If an IPC code is more strongly related to the input document, it should be assigned a larger score and ranked in an earlier position. The following Figure 1 illustrates an example of the classification procedure.

### 2.2. Available resources

In NTCIR-7 English patent mining task, two training data sets are provided

1. USPTO patent data. This data set contains 889,113 US patents published by United States Patent and Trademark Office in 1993-2000. Each patent is made up of several sections. But only the information of 7 sections (<PRIMARY IPC>, <TITEL>, <ABSTRACT>, <SPECIFICATION>, <CLAIM>, <DOCUMENT> and < DOC IDENTIFIER>) is allowed to be used for this task. It should be noted that only the primary IPC is assigned to each patent in this data set, which is selected from the original patent IPC codes that the patent is associated with. In other words, it is a training data set for single label classification task.

2. Patent Abstracts of Japan (PAJ). The PAJ data set contains translations of titles and abstracts of 2,382,595 Japanese patents in 1993-2002. It is a multi-label training set, and each document in PAJ is labeled with one or more IPC codes.

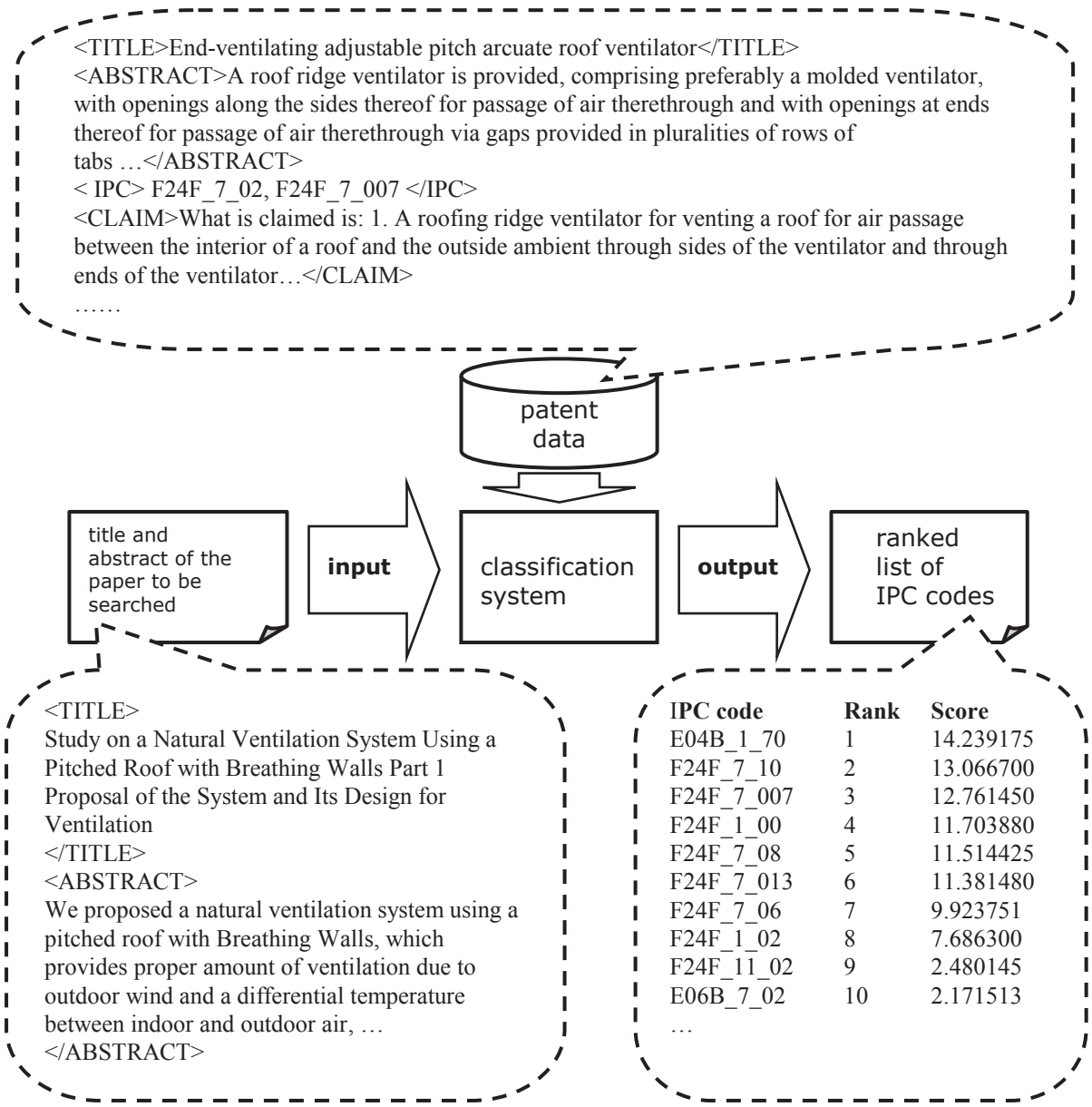


**Figure 1. Procedure of the categorization of research papers into IPC codes**

There are two test collections for this task, which are provided within the dry-run and formal-run tasks respectively. Each test instance (topic) only consists of the title and abstract of an English research paper and is labeled with one or more appropriate IPC codes. The test collection of dry-run contains 97 topics, and the formal-run data contains 879 topics. The official result is evaluated on the formal-run test set.

## 2.3. Evaluation method

In this task Mean Average Precision (MAP) is adopted to measure how well the ranked list matches the gold standard [1]. The MAP is calculated in the following way,

$$MAPvalue = \frac{\sum_{i=1}^{n} AverP(t_i)}{n}$$

$$AverP(t_i) = \frac{\sum_{r=1}^{r=N} P_i(r) \times rel_i(r)}{m}$$

Where $t_i$ is the i-th topic in test collection, $n$ is the total number of the topics. For a given topic $t_i$, $r$ is its rank in the list, $P_i(r)$ is the precision at $r$ which is obtained by dividing the number of correct IPC codes in top-r of the ranked list by $r$. $rel_i(r)$ is a 0-1 function that indicates the relevance of rank r. If the IPC code at rank r is correct, $rel_i(r)=1$; otherwise $rel_i(r)=0$. $m$ is the number of correct IPC codes for $t_i$, and $N$ is the maximal rank that is evaluated. In this task, $N$ is set to 1000.

# 3. Basic idea

## 3.1. Challenges

To use machine learning algorithm for this patent mining task, there are some challenging issues we should consider as follows:

(1) Huge amount of training samples. As mentioned in section 2.2, the number of patent samples for training is over 3 million. In real-world applications, how to train a supervised classifier on such a large scale of data set is a critical issue. For example, training SVM-based classifier on such large corpus is also intractable due to the high computation cost.

(2) Huge IPC code set and multi-label classification task. IPC taxonomy is a large and hierarchical classification system which consists of more than 60,000 IPC codes. Training a supervised classifier on such large taxonomy is a challenging task. Besides, how to learn good supervised classifier from a large multi-label corpus is another key issue.

(3) Class imbalance issue of IPC taxonomy. The distribution of IPC codes is much skewed. Seen from training corpus, most patents are concentrated on a small set of IPC codes, and the number of patents of most IPC codes is smaller than the average value. For example, in PAJ data set, the total number of IPC codes is 30,885, and the average patent number for an IPC code is over 80. The IPC codes containing less than 10 and 50 patents take about 25% and 57% of all the IPC codes respectively. This situation generally causes a bottleneck in performance of the system trained on such imbalanced class distribution data [9].

(4) Different writing styles between research papers and patents. Research papers and patents are different in many aspects, such as words, mode of expression and formats. Even discussing the same topic, patents and research papers are expressed quite differently. This phenomenon gives rise to a problem that the research paper data and patent data do not follow the same or similar distribution, which conflicts with the foundational hypothesis of supervised document classification theory. Thus a supervised classifier based on this hypothesis is somewhat questionable when applied to patent mining task.

## 3.2. Motivation

The issues mentioned above make it difficult to apply sophisticated machine learning methods such as maximum entropy methods [2] and support vector machines [5] on patent mining task. Roughly speaking, a great deal of memory space and time cost is required by using these methods in large scale classification environment; for another, all of these methods are generally designed to solve the single label classification problem. Though there is some work on transforming multi-label classification into single label classification or designing much more complicated models to support the multi-label classification problem, there are still no good solutions to multi-label classification on large class set.

In contrast, using k-nearest neighboring method is a comparatively easy solution to deal with large amount of data, because the classification is only based on extracting similar examples and no training process is required. Besides, KNN is itself a ranking, which can be applied on IPC codes ranking directly. Therefore, the KNN-based classification is used as our basic work frame.

# 4. KNN-based method

## 4.1. Architecture

The system consists of two major parts – KNN-based classification module and re-ranking module. In the first stage, the list of candidate IPC codes is generated by a KNN classifier. Then the re-ranking technique aims to refine the ranked list. In this section, we describe the KNN-based classification module in detail.

## 4.2. Similarity calculation

In the design of our KNN classifier, five methods are used to calculate the similarity between an input document and each document in the training data set.

(1) Cosine + tfidf

Cosine is the most commonly used technique for similarity calculation in vector space model, which is regarded as a measurement of the angle between vectors. Given two document vectors $v_1$ and $v_2$, the Cosine-based similarity is computed as,

$$Sim_{\cos ine}(\mathbf{v}_1, \mathbf{v}_2) = \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{\|\mathbf{v}_1\| \ \|\mathbf{v}_2\|}$$

In our system, each feature term $t_j$ of a document vector $v_i=(w_{1,i}, w_{2,i}, \ldots, w_{m,i})$ is weighted by term frequency-inverse document frequency (tfidf) weighting scheme, as follows,

$$w_{j,i} = (\log(tf_j) + 1) \log \frac{N}{df_i}$$

Where $tf_j$ is the term frequency of $t_j$ in document $i$, $df_i$ is the number of documents containing term $t_j$, $N$ is the total number of documents in the training data set. If $tf_j = 0$, we set $w_{j,i}=0$.

(2) BM25

BM25 is a widely adopted weighting method in information retrieval [10]. In the BM25 weighting scheme, the input document is viewed as a query $q$ containing a serious of terms $\{t_1, t_2, .., t_m\}$. The System collects all the documents in which at least one term

occurs. Then the system calculates the BM25-based similarity between the query $q$ and each document $d$ in training data set by the following equation,

$$Sim_{BM25}(q,d) = \sum_{t \in q} \frac{tf_t^d \cdot (k_1+1) \cdot w_t}{tf_t^d + k_1 \cdot (1 - b + b \cdot \frac{dl}{avgdl})}$$

Where $tf_t^d$ is the frequency of term $t$ in $d$. $dl$ is the length of document $d$. $avgdl$ is the average document length in the training data set. $k_1$ and $b$ are two parameters, namely term-frequency influence parameter and document normalization influence parameter respectively. In our system, $k_1$ and $b$ are set to 1.5 and 1.0 by default. $w_t$ is the inverse document frequency (idf) of term $t$, which is calculated as follows,

$$w_t = \log(\frac{N - df_t + 0.5}{df_t + 0.5})$$

Where $df_t$ is the number of documents containing $t$. It should be noted that $w_t$ can be negative in some cases, which may lead to negative score of $Sim_{BM25}$. In our system, $Sim_{BM25}$ is set to 0 if $Sim_{BM25} < 0$.

(3) SMART

This method is adopted in the well known information retrieval system SMART [3]. As in BM25, the input document is viewed as a query that contains a bag of terms. In the first step, the system extracts the documents in which at least one term occurs. Then the similarity between a document $d$ and the input query $q$ can be calculated as follows,

$$Sim_{SMART}(q,d) = \sum_{t \in q} \frac{(1 + \log(tf_t^d)) \cdot w_t}{(1 + \log(avgtf_t)) \cdot (1 - r + r \cdot \frac{utf_d}{pivot})}$$

$$w_t = (1 + \log(tf_t^q)) \cdot \log(\frac{N-1}{df_t})$$

Where $tf_t^d$, $df_t$ and $N$ are the same as in BM25. $argtf_t$ is the average number of occurrences of term $t$ in the document set extracted. $utf_d$ is the number of unique terms in document $d$. $pivot$ is the average number of unique terms per document in the training data set. $f_t^q$ is the frequency of the term $t$ in the query $q$. There is only one parameter r in this method, which is set to 0.2 by default.

(4) Pivoted document length normalization

Pivoted document length normalization is another weighting scheme in information retrieval [11]. In this paper we call it PIV for short. The scoring function of PIV is similar to BM25 and SMART,

$$Sim_{PIV}(q,d) = \sum_{t \in q} \frac{(1 + \log((1 + \log(tf_t^d))) \cdot tf_t^q \cdot w_t}{(1-s) + s \cdot \frac{dl}{avgdl}}$$

$$w_t = \log(\frac{N-1}{df_t})$$

Where $tf_t^d$, $tf_t^q$, $dl$, $avgdl$, $df_t$ and $N$ have the same meaning as in BM25. $s$ is the normalization parameter referred to as the slope and has a default value of 0.2.

(5) Log-linear method

Generally speaking, there are many basic similarity calculation methods which treat the problem in different views. It is natural to explore methods that can make benefits from the various existing scoring methods together. In our system, a log-linear way [4] is used to combine multiple scores (features) generated by basic similarity calculators together. The scoring function of this method is given as follows,

$$Score_{\text{log-linear}}(c) = \frac{\exp(\sum_{m=1}^{M} \lambda_m \cdot Score_m(c))}{\sum_c \exp(\sum_{m=1}^{M} \lambda_m \cdot Score_m(c))}$$

Where $M$ is the number of basic similarity calculators that are combined. $Score_m(c)$ is the normalized score of the class $c$ in the $m$-th similarity calculator. The default features of our system are shown in Table 1. $\lambda_m$ is the weight of each basic similarity calculator. In our system, the optimization of $\lambda$ is performed on a development data set using the grid search method (hill-climbing). The development data set is made up of 3000 randomly selected documents from PAJ data.

**Table 1. Default features of log-linear method**

| Feature | Description | Weight |
|---|---|---|
| feature 1 | $Sim_{Cosine}$ | 0.41 |
| feature 2 | $Sim_{BM25}$ | 0.30 |
| feature 3 | $Sim_{f3} = \log(\frac{Nc}{size(c)} + 1)$ | 0.20 |
| feature 4 | $Sim_{f4} = \log(tf_t^d \cdot \frac{N}{df_t} \cdot \frac{1}{dl} + 1)$ | 0.09 |

## 4.3. Ranking

In the first step of ranking, the system extracts the top-k documents $\{d_1, d_2, ..., d_k\}$ with the highest similarities (k-nearest neighbors). Next, the system calculates a score $Score(c)$ for each IPC code $c$ in the extracted documents. Here $Score(c)$ can be regarded as a measure of the likelihood that the input document has the label $c$. At last these IPC codes are sorted by their scores and outputted as the final ranked list. In our system, the following state-of-the-art scoring methods are used,

(1) Original KNN ranking method

The system scores each $c$ by the number of its occurrence in the extracted top-k documents, as follows,

$$Score_{Original}(c) = \sum_{i=1}^{k} occur(c, d_i)$$

Where $occur(c, d_i)$ is a 0-1 function that indicates whether $c$ occurs in $d_i$ ($occur(c, d_i)$=1) or not ($occur(c, d_i)$=0).

(2) Naïve method

In this method, the order of IPC codes follows the order of their first occurrences in the documents extracted. The scoring function is given as follows,

$$Score_{Naive}(c) = \frac{1}{firstrank(c, \{d_1, d_2, ..., d_k\})}$$

Where *firstrank(c, {$d_1$, $d_2$, ..., $d_k$})* indicates the rank of the first occurrence of c in {$d_1$, $d_2$, ..., $d_k$}. If the first occurrence is in $d_i$, then *firstrank(c, {$d_1$, $d_2$, ..., $d_k$})=i*.

(3) Sum/SumAver

In this method, score is calculated by summing up the similarities of all the extracted documents containing c, as follows,

$$Score_{sum}(c) = \sum_{i=1}^{k} occur(c, d_i) \cdot Sim(q, d_i)$$

Where $q$ is the input document, *Sim(q, $d_i$)* is the similarity between $q$ and $d_i$ which has been obtained in the step of similarity calculation.

Generally a patent document has more than one IPC codes. In the Sum scoring method, the patents labeled with multiple IPC codes make more contribution to ranking, since their similarities are added more than one time. To limit the influence of the patents containing multiple IPC codes, a modified version of Sum is used, namely SumAver. Supposing the IPC codes in one patent have equal contribution to the similarity, we have,

$$Score_{SumAver}(c) = \sum_{i=1}^{k} occur(c, d_i) \cdot SimAver(q, d_i)$$

$$SimAver(q, d_i) = \frac{Sim(q, d_i)}{nubmer\ of\ IPC\ codes\ in\ d_i}$$

(4) Listweak/ListweakAver

To emphasize the patents ranked in the frontier part of the list, another scoring method based-on Sum is used, namely Listweak.

$$Score_{listweak}(c) = \sum_{i=1}^{k} occur(c, d_i) \cdot Sim(q, d_i) \cdot r_1^{\,i}$$

Where $r_1$ is a parameter ranging in (0,1). $r_1^{\,i}$ can be regarded as a penalty that punishes the patents that have lower ranks. In our system, $r_1$ is set to 0.95 by default. Like the Sum scoring method, Listweak has a modified version named ListweakAver, which is obtained just by replacing *Sim(q, $d_i$)* in the above equation with *SimAver(q, $d_i$)*.

Besides the basic methods above, a variant is also used in our system.

(5) Weak/WeakAver

A drawback of KNN is the prediction of the input document tends to be dominated by the classes with the more frequent examples, as they are more likely in the k-nearest neighbors when the neighbors are computed due to their large number. It is a serious problem in this task because of the class imbalance of IPC. To alleviate the problem, a new scoring method is used, namely Weak.

$$Score_{weak}(c) = \sum_{i=1}^{k} occur(c, d_i) \cdot Sim(q, d_i) \cdot r_2^{\left(crank(c,i) \times \frac{size(c)}{k}\right)}$$

Where *size(c)* is the number of training documents in $c$, *crank(c, i)* is the number of occurrence of $c$ in top *i-1* documents. $r_2$ is a parameter ranging in (0,1) and is set to 0.9 by default. The factor $r_2^{\left(crank(c,i) \times \frac{size(c)}{k}\right)}$ is a penalty that punishes the class having large size. If $c$ contains more documents in the training data set and occurs more frequently in {$d_1$, $d_2$,..., $d_{i-1}$}, the penalty factor will punish more on *Sim(q, $d_i$)*. Actually this factor reflects both the density of class $c$ on the whole training data set (factor $r_2^{\left(\frac{size(c)}{k}\right)}$) and the density of class $c$ in the set of top-i extracted documents (factor $r_2^{crank(c,i)}$). Like the Sum scoring method, the method of Weak has a modified version named WeakAver, which is obtained just by replacing *Sim(q, $d_i$)* in the above equation with *SimAver(q, $d_i$)*.

### 4.4. Other settings

In our system, each document is represented as a bag-of-words. The terms are extracted from the sections of title and abstract, as some parts of training documents (PAJ data) have the information of these two sections only. And stemming is performed on all of the words.

For parameter setting, k is set to 100 in all the ranking methods.

## 5. Re-ranking

In our KNN-based system different combinations of similarity calculation method and ranking method generate different results, since the problem is treated in different views. If the system could learn a better ranking from individual ranked lists, the performance will probably be improved. Thus in the second stage, we aim at learning better ranking from multiple ranked lists. Two approaches are used in our system.

### 5.1. Rank combination

The motivation behind this method is the ranks of class c in different lists can give us more evidence to rank it at an appropriate place. Given $h$ ranked lists {$l_1$, $l_2$, ..., $l_h$} outputted by different individual classifiers {$C_1$, $C_2$, ..., $C_h$}, the system re-score each $c$ in the lists with the following equations.

$$Score_{rank\text{-}combination}(c) = \frac{1}{\sum_{i=1}^{h} \lambda_i \cdot rankinlist(c, l_i)}$$

Where *rankinlist(c, $l_i$)* is the rank of $c$ in list $l_i$. If $l_i$ does not contain $c$, *rankinlist(c, $l_i$)* is set to the length of $l_i$. $\lambda_i$ is the weight of $l_i$. In our system, the optimization of $\lambda$

is performed on a development data set with the gird search method.

Actually this method models the re-ranking problem as a linear combination of individual classifiers. The features of a class *c* are its ranks in different lists. Here we cannot use the conventional regression techniques (e.g. least squares linear regression) for the optimization of parameter $\lambda$ , because there is never a gold-standard to tell the system what the correct value of $Score_{rank\text{-}combination}(c)$ is. In this task, the target function is actually the loss of MAP value, but not the sum of the squared residuals.

### 5.2. Re-ranking using learning to rank techniques

The re-ranking problem here can be regarded as learning a ranking over instances in terms of a series of features. Actually it is an issue in machine learning techniques, namely learning to rank. And there have been many approaches proposed to solve this problem, such as RankSVM [8] and RankBoost [6]. Here we also treat the re-ranking as a learning problem. For model training, we can employ any existing learning method. In our system, we choose RankSVM.

Given a set of ranked list $\{l_1, l_2, ..., l_h\}$ , each class c can be viewed as an instance denoted by $x_c=\{f_1(c), f_2(c), ..., f_h(c)\}$, where $f_i(c)$ is the feature function that returns the feature value of the i-th dimension. In our system, we define

$$f_i(c) = \begin{cases} Score_i(c) & \text{if c is included in list}_i \\ 0 & \text{otherwise} \end{cases}$$

Where $Score_i(c)$ is the score of *c* in $l_i$. For ranking problem, the important thing is to induce the pairwise instance preference $x_{c1} \succ x_{c2}$ , which means the instant $x_{c1}$ should be ranked higher than $x_{c2}$. Models can be trained on a series of pairwise instance preferences that are extracted from the training data, and then predict the relation between the unseen instances. The final ranking list is generated by the set of pairwise preference relations on the input instances (test data). In our system, since the parts such as model training and ranking is done by well-developed toolkits SVM-light[1], the key point is the extraction of pairwise instance preferences for training. Actually there is no data for training. We just randomly selected a data set containing 3000 documents from PAJ data. For each document *d* in this data set, we obtained *h* ranked lists $\{l_1, l_2, ..., l_h\}$ of IPC code based on *h* basic classification systems. Suppose $C_d$ is the set of IPC codes in these lists, and the correct IPC code of *d* is $C_{correct}=\{ipc_1,ipc_2,...,ipc_3\}$, we define,

$$S_{prefer} = \{x_{c1} \succ x_{c2} : c_1 \in C_{correct} \wedge c_2 \in C_d - C_{correct}\}$$

The final training data is the union set of $S_{prefer}$ of each *d*.

[1] http://svmlight.joachims.org/

## 6. Experiment

### 6.1. Settings

We used the default settings of our system for all the experiments.

The training data is the PAJ data only. The USPTO data is not used, since it seems not useful in our experiments.

### 6.2. Evaluation of our system

The first set of experiments is carried out to evaluate the performance of KNN-based classifiers on dry-run data set. We evaluate the performance of the system with various combinations of similarity calculation and ranking. The experimental results are shown is table 2 (the columns represent the methods of similarity calculation while the rows represent the methods of ranking.).

We also evaluated the performance of re-ranking methods on the same data set. The basic classifiers are selected empirically. They are "BM25+ListweakAver", "BM25+Listweak", "BM25+Naïve", "SMART+Listweak", "SMART+ListweakAver", and "PIV+Naïve". For the re-ranking using RankSVM, the same basic classifiers are used expecting "PIV+Naïve". The experimental results are shown in table 3.

The second set of experiments is carried out on the formal-run data set which is the final test data set of NTCIR-7 patent mining task. Here we present the performance of systems that we entered in NTCIR-7. The three systems got the top-3 places among all the systems in NTCIR-7 English patent mining task.

From Table 2, 3 and 4, we can draw the following main conclusions:

1. BM25 and log-linear method outperform other similarity calculation methods.
2. Among all the ranking methods, Listweak and ListweakAver perform the best. Thus we choose the system with the combination of "BM25+listweak" for NTCIR-7 formal run evaluation[2]. Surprisingly, the simplest ranking method "Naïve" works well in some cases, which indicates that the simple ranking methods are also helpful to our system.
3. Ranking is a key factor that affects the performance of the basic KNN-based classification system. The system performs quite differently among various ranking methods.
4. Re-ranking can improve the performance of the basic KNN-based system significantly.

[2] Actually log-linear similarity calculation + ListweakAver ranking is the most effective combination on the dry-run data set. But we did not submit its result for the formal- run evaluation due to the time cost of the log-linear similarity calculation method.

**Table 2. Performance (MAP) of the KNN-based classifiers on dry-run data set.**

| Ranking \ Sim | Cosine + tfidf | BM25 | SMART | PIV | Log-linear |
|---|---|---|---|---|---|
| Original KNN | 35.16 | 34.79 | 35.78 | 34.51 | 35.05 |
| Naïve | 32.41 | 38.57 | 33.55 | 37.23 | 40.02 |
| Sum | 35.97 | 35.78 | 36.83 | 35.58 | 38.33 |
| SumAver | 35.05 | 35.92 | 36.46 | 34.13 | 38.05 |
| Listweak | 36.63 | 40.52 | 37.42 | 36.85 | 40.37 |
| ListweakAver | 34.85 | 40.88 | 37.65 | 36.79 | 41.11 |
| Weak | 36.25 | 36.53 | 37.11 | 35.91 | 38.24 |
| WeakAver | 33.42 | 36.15 | 34.90 | 33.01 | 38.38 |

**Table 3. Performance (MAP) of re-ranking systems on dry-run data set**

| System | Performance (MAP) |
|---|---|
| Rank combination | 45.31 |
| Re-Ranking using RankSVM | 43.02 |

**Table 4. Performance (MAP) of our systems on formal-run data set.**

| System | Performance (MAP) |
|---|---|
| BM25+listweak | 44.53 |
| Rank combination | 48.86 |
| Re-ranking using RankSVM | 47.21 |

# 7. Discussion

## 7.1. Classification vs. retrieval

The categorization of research papers into IPC codes is not a standard multi-label classification problem, since the system just generates the score or possibility for each class instead of a set of classes associated with the input document. The problem here is more likely a task of relevant IPC codes retrieval. Of course, there are many ways to transform a ranked list of labels into a set of labels for an input document. For example, the target labels can be extracted by using a threshold. However, we cannot expect the system will perform well if we evaluate it from the viewpoint of multi-label classification due to a large number of IPC codes used. In practical applications such as invalidity search a ranked list can provide more information rather than an imprecise set of labels. Therefore viewing the systems as information retrieval systems may be more appropriate than regarding it as multi-label classification systems.

## 7.2. Single label vs. multi-label

As mentioned above, the training data of single label (i.e. USPTO data set) and multi-label (i.e. PAJ data set) are provided within this competition task. However we found that the data of USPTO shows harmful to our system. The performance degrades when we trained the system on USPTO data solely or a mixed data set of "USPTO+PAJ", comparing to training on PAJ data. It is the major reason that why we only use PAJ as training data set.

Actually we still believe that the system can benefit from the USPTO data. But it needs our further study. Beside the problem of training on both single labeled data and multi-labeled data, another key issue is how to train our system on heterogeneous data. Both of the problems are worth studying in our future work.

## 7.3. Rank-based ranking vs. similarity-based ranking

The KNN ranking and re-ranking in our system can be divided into two types of methods, namely rank-based ranking and similarity-based ranking respectively. In rank-based ranking, the system calculates the score of each IPC code (i.e. class) using the document ranks, such as the Naïve ranking method and rank combination method while similarity-based ranking scores each class with the document similarities, such as the Sum ranking method and re-ranking using RankSVM. Theoretically, similarity-based ranking is more sophisticated than rank-based ranking, since the document similarities can provide us with more evidence for classification. However, in our experiments the similarity-based ranking did not show obvious advantages as we expected. The rank-based ranking outperforms similarity-based ranking in some cases, for instance, the Naïve ranker achieved the best performance among all the rankers based on the similarity calculation method of PIV on the dry-run data set, and the system based on rank combination is the most effective system on both the dry-run and formal-run data sets. It indicates that the rank-based ranking methods are still effective for this task, though they are very simple. For similarity-based ranking, the problem is that the scores are calculated in terms of the sum of document similarities. However the similarities do not have the property of additivity. It means that the similarity 1.0 of a single document and the similarity 1.0 generated by 0.5 + 0.5 are incomparable. It is a possible reason that similarity-based ranking methods do not work well sometimes. If a more reasonable way is used to generate scores for ranking,

the performance of similarity-based ranking may be improved.

### 7.4. Does patent structure really help

Generally speaking, patent structure is very useful for people to understand the content of patents. And we hope it can also play an important role in tasks of patent mining. To evaluate the effectiveness of the use of patent structure, we did a set of experiments in which the words appearing in different sections are considered as different features. The experimental results on dry-run data set show that patent structure specified features are not helpful for the system performance. To use the features in some sections such as claims, we also perform the same experiment on a test set consisting of the full document of patents. Experimental results show that the patent structure still seems not helpful. It suggests that patent structure does not benefit to our system in which the documents are represented as bag-of-words vectors. The study of more effective ways of document representation is valuable in the future work.

### 7.5. Why re-ranking works

The re-ranking worked very well on this task. One major reason is that it makes much benefit from richer features. Another possible reason is that the re-ranking is performed on the frontier part of the list and has great potential to make performance improvement, since the evaluation method (MAP) emphasizes the return of more correct IPC codes earlier.

## 8. Conclusions

We focus on the categorization of documents into IPC by participating in NTCIR-7 patent mining task. Our system is basically under the k-nearest neighboring framework, in which various similarity calculation and ranking methods are used. Two re-ranking methods are further used to improve the performance of the basic system. The experimental results on dry-run and formal-run data sets show that the "BM25+listweak" is one of the most effective methods for this task. And the re-ranking techniques can improve the basic system significantly. In future, we will use more patent specified features such as the patent structure to improve the performance. And we will apply our techniques on the practical patent processing applications such as invalidity search and technical trend analysis.

## 9. Acknowledgments

The authors wish to acknowledge Chunliang Zhang for his comments on an early draft of this document, and Dr. Matthew Y. Ma for his kind help to our work of patent mining. This work was supported in part by the National 863 High-tech Project (2006AA01Z154), the Program for New Century Excellent Talents in University (NCET-05-0287), MSRA research grant (FY08-RES-THEME-227), and National Science Foundation of China (60873091).

## References

[1] Baeza-Yates, R. A. and B. A. Ribeiro-Neto. Modern Information Retrieval. ACM Press/Addison-Wesley, 1999.

[2] Berger, Adam L., Stephen A. Della Pietra and Vincent J. Della Pietra. A Maximum Entropy Approach to Natural Language Processing. *Computational Linguistics*, 22(1):39-71. 1996.

[3] Buckley, Chris, Amit Singhal, Mandar Mitra, and Gerard Salton. New Retrieval Approaches Using SMART: TREC 4. In the Fourth Test REtrieval Conference. Pages 25-48. USA. 1996.

[4] Christensen, R. Log-Linear Models and Logistic Regression. Springer-Verlag Inc. New York, USA. 1997.

[5] Cristianini, Nello and John Shawe Taylor. An Introduction to Support Vector Machines and other kernel-based learning methods. Cambridge University Press. 2000.

[6] Freund, Y. , R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. Journal Machine Learning Research. 4:933–969. 2003.

[7] Fujii, A., M. Iwayama, and N. Kando. The patent retrieval task in the fourth NTCIR workshop. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. Pages 560–561. Sheffield, UK. 2004.

[8] Herbrich, R. , T. Graepel, and K. Obermayer. Support vector learning for ordinal regression. In *the ninth International Conference on Artificial Neural Networks*. Pages 97–102. London, UK. 1999.

[9] Japkowicz, Nathalie, and Shaju Stephen. The class imbalance problem: A systematic study. In *Intelligent Data Analysis*, 6(5): 429-449. 2002.

[10] Robertson, Stephen E., Steve Walker, and Micheline Hancock-Beaulieu. Okapi at TREC-7: automatic ad hoc, filtering, VLC and interactive track. In *Proceedings of the Seventh Text REtrieval Conference*. Pages 253-264. Gaithersburg, USA. 1998.

[11] Singhal, Amit, Chris Buckley and Mandar Mitra. Pivoted document length normalization, In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*. Pages 21-29. Zurich, Switzerland. 1996.