The Download Estimation Task on KDD Cup 2003

Janez Brank and Jure Leskovec

Jožef Stefan Institute
Jamova 39, Ljubljana, Slovenia

janez.brank@ijs.si, jure.leskovec@ijs.si
http://ai.ijs.si/kddcup03/

October 28, 2003

Abstract

This paper describes our work on the Download Estimation task for KDD Cup 2003. The task requires us to estimate how many times certain papers have been downloaded in the first 60 days after they have been published on *arXiv.org*, a preprint server for papers on physics and related areas. The training data consists of approx. 29000 papers (both full text and some metadata were available), the graph of citations among those papers, and information about the downloads of a subset of these papers (not only the number of downloads, but timestamps of individual downloads as well). Our approach is based on an extension of the bag-of-words model, with linear SVM regression as the learning algorithm. We describe our experiments with various kinds of features, both helpful and useless ones. We focus particularly on issues of feature weighting, which turns out to be quite important for this task.

1 Introduction

KDD Cup is an annual data mining competition organized by ACM SIGKDD (the special interest group on knowledge discovery and data mining). A dataset and problem specification are usually published a few months before each year's KDD conference, teams who wish to participate have to submit their entries by a certain deadline, and winners are announced at the conference.

The dataset on KDD Cup 2003 (http://www.cs.cornell.edu/projects/kddcup/) was a collection of scientific papers from *arXiv.org*, a well-known preprint server, together with some metadata and a graph of citations between those papers. Several tasks were defined: (1) predicting how many new downloads some papers will receive from papers added to the archive within some period of time; (2) reconstructing a citation graph for a set of papers for which it was not provided by the competition organizers; (3) estimating how many times certain papers have been downloaded in the first 60 days since they were added to the archive; and (4) an "open task" where one could define and investigate any interesting problem related to the dataset. In this paper we focus on the third of these tasks (download estimation) and describe the methodology and experiments that led to our winning entry for this task.

1.1 Task description and available data

We are given a set of 29014 documents from the "hep-th" (high-energy physics — theory) area of *arXiv.org*, a preprint repository for physics and related areas. They cover the period from January 1992 through February 2003.[1] Each paper has a seven-digit identifier of the form *yymmnnn* (year, month, sequential number within the month).

The following information was available:

[1] Subsequently, data concerning 541 new papers from March and April 2003 was also made available by the KDD Cup 2003 organizers; however, we did not make use of this extra data, and its relevance to the download estimation task is probably small anyway.

- The full text of all the papers (in the TeX format, i. e. plain text cluttered with a heavy dose of TeX commands).

- A separate (and much tidier) small file for each paper, containing the abstract of the paper and some metadata: title, author names, journal name, subject classification, date, etc. (see Section 1.3 for details).

- A citation graph, i. e. a set of pairs of paper IDs, stating that a certain paper references a certain other paper. References pointing out of the dataset were probably simply ignored when preparing the graph. Likewise, this graph does not contain information about citations of papers from the dataset by papers outside the dataset.

- For each paper submitted in February and March of 2000, February and April of 2001, and March and April of 2002 (we will refer to these six months as the "training period"), there is a list of downloads of this paper from the *arXiv.org* servers in the first 60 days since it was included in the archive. The exact date and time of each download is provided, not just the number of downloads.

The Download Estimation task consists of predicting the number of downloads (in the first 60 days since their inclusion in the archive) of the papers submitted during April 2000, March 2001, and February 2002. We refer to these months as the "test period".

1.2 Evaluation

The KDD Cup rules state that only the 50 most frequently downloaded papers from each of three test months would be used for evaluation. The L_1 distance measure would be used, i. e.

$$Score = \sum_p |DlCount(p) - Prediction(p)|,$$

where the sum goes over the set of 150 papers consisting of the 50 most frequently downloaded papers of each test month. Of course, since one doesn't know in advance which papers will be the most frequently downloaded, it makes sense to submit predictions for all the papers from the test period. The predictions on the less frequent papers will be simply ignored during the evaluation, so it doesn't matter how incorrect they are.

In effect, this means that it makes sense to treat each test document as if it belonged to the 50 most frequently downloaded papers of the month; if it actually does, that's fine, and if it doesn't, we'll predict too many downloads but this error will be ignored.

To achieve this, it might make sense to focus on frequently downloaded papers during training, and thus encourage the learner to produce a model that will always predict as if the document were a fairly frequently downloaded one.

1.3 Contents of the abstract files

There is one abstract file per paper. Apart from the abstract itself, all of these files contain the "Title", "Date" and "From" fields. "From" is formatted much like the "From" header in e-mail messages, and probably refers to the person (practically always one of the authors) who submitted the paper to the archive. It contains a name, an e-mail address, and often both. Likewise, "Date" probably refers to the date when the paper was submitted to the archive. A variety of formats and local time zones are used in this field. We ignored this date field because we only really need the month and year, which can easily be obtained from the paper ID. Some papers' abstract files contain additional "Date (revised vn)" fields (going up to $n = 15$ in one case!), which we also ignored. These fields are present in cases when the paper was revised after being added to the archive.

Every paper has an "Authors" field listing all the authors (in 4357 papers, it's called "Author", but is otherwise equivalent). These author names are not given in any particular normalized or standardized form, and often include TeX commands for accent marks or other special characters. We normalized each name by keeping only the surname and the initial of the first word of the name; thus "Foo Baz", "Foo Bar Baz", "F.Baz", "Foo B. Baz", etc., would all become "baz-f".

Almost all papers have a "Comments" field, but it usually contains just the number of pages and so doesn't appear to be very useful.

About 72 % of the papers have a "Journal-ref" field containing a reference to a journal where the paper was (or will be?) published. Occasionally this field refers to conference proceedings or other publications, but in most cases it is a standard reference to a journal. That is, the same journal is always referred to in the same way, e. g. "JHEP", "Phys. Lett.", "Int. J. Mod. Phys.", and so on.

If the paper appeared as a technical report, it might include a "Report-no" field stating the number or code of the report. About 53 % of the papers have this field. We ignored this field because (1) it is relatively rare (only present in half the papers), thus there would be lots of missing values if it were included in the learning process; (2) it doesn't seem too promising; there are probably many different series of technical reports and it doesn't seem likely that being published in a particular technical report series would encourage people to download the paper; (3) the technical report series where a paper is published is probably strongly correlated to the institution where the authors work, and we already capture this information elsewhere (see Section 3.1).

Some papers have a "Subj-class" field, which contains standardized names of one or more subject classes such as "quantum algebra", "mathematical physics", and so on. (It always includes "high energy physics — theory".) This is nice, but unfortunately only approx. 8 % of the papers have this field. (A further two papers contain a "MSC-class" field, which we ignored.) 55 different subject classes occur in this field, but only six of them occur in more than 100 papers.

Ten papers contain a "Notes" field, with contents that would usually be in "Comments"; one paper contains a "Proxy" field. These fields were ignored.

1.4 The number of papers

The number of papers per year has been growing steadily during the nineties, but has stabilized somewhat since 2000.

Year	1992	1993	1994	1995	1996	1997
No. of papers	1385	2091	2441	2347	2626	2685
Year	1998	1999	2000	2001	2002	2003
No. of papers	2567	2825	3144	2890	3333	2916*

*For 2003, we only have papers from the first two months. There are 486 of them, hence the estimate $2916 = 6 \times 486$ for the entire year.

The number of papers from the training and test period is as follows:

Year	2000	2001	2002		Total
February	256	208	*210*	Training period:	1566
March	300	*265*	275	Test period:	*678*
April	*203*	267	260	(Entire dataset:	29014)

Remember that only 50 most frequently downloaded papers from each test month will be used for evaluation. As this table shows, this is only approximately 22 % (= 150/678) of all the papers from the test months.

1.5 Most downloads occur in the first few days

For each paper from the training period, we have a list of the times when it was downloaded within the first two months after it has been published in the archive. It turns out that the vast majority of these downloads occur within the first few days (typically, about 70 to 75 % of the downloads occur within the first week).[2] (See Figure 1.)

Because most downloads occur in the first few days after a paper had been published in the archive, it is reasonable to assume that they are mostly caused by people who notice the paper on the "new" and "recent" pages. The "recent" page (`http://www.arxiv.org/list/hep-th/recent`) shows, for each paper submitted in the last few days (a week or so), the authors, title, and the "Comments" field (see Section 1.3). The "new" page (`http://www.arxiv.org/list/hep-th/new`) shows the abstracts in addition to the above-mentioned fields, but it only covers the papers submitted during the current day.

[2] The most frequently downloaded paper (# 0203101, "The Holographic Principle"), with 2912 downloads, is a curious anomaly to this general rule. Originally submitted on March 11, it was downloaded "only" 1334 times up to the end of March 19, after which there were less than 20 downloads per day for a while. However, there was then a burst of downloads starting late in April 29, resulting in 51 downloads that day, 719 the following day, 108 on May 1, and 334 more until the end of May 10, after which no more data is available. Thus there were 1212 downloads in the last 12 days, but only 76 in the preceding 12 days! Without this flurry of downloads in the last 12 days, the paper would have between 1700 and 1800 downloads and would thus be less of an outlier. (The next few most frequently downloaded papers have 1540, 1351, 1351, 1345, and 1308 downloads.) We were unable to determine the cause of this late mass of downloads.

	2000		2001		2002		All
	Feb.	Mar.	Feb.	Apr.	Mar.	Apr.	papers
All papers	170.2	174.4	173.9	182.5	221.4	236.8	193.6
Top 50 only	411.6	479.6	350.7	436.5	528.7	567.0	462.3

Table 1: Average number of downloads for papers that were added to the archive in a particular month.

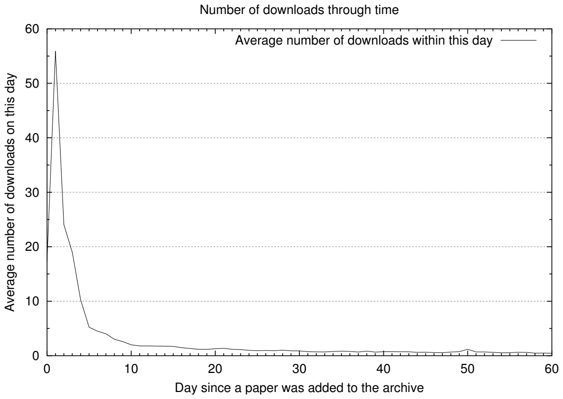


Figure 1: This chart shows the average number of times that a paper has been downloaded on its n-th day in the archive, for n up to 60. These averages have been computed over all 1566 papers from the training period (i. e. all for which the download times are known). The small bump on day 50 has been caused by an unusual flurry of downloads of paper 0203101 on its fiftieth day in the archive.

Thus, information not present on these pages presumably does not strongly and directly influence a reader's decision whether to download a paper or not. In particular, these downloads are probably not much influenced by citations, because a paper usually only obtains citations some time after publication, not in the first few days. Of course this does not mean that citations are completely useless for predicting the number of downloads: interesting and relevant papers are probably downloaded more frequently; readers can recognize such papers from the abstract, and while our learning algorithms might not be so successful at that, they could use citations as an indicator whether a paper is relevant.

1.6 Number of downloads from year to year

Table 1 shows the average number of downloads for papers published in each month of the training period. The average number of downloads has grown between 2001 and 2002, probably because more users were becoming aware of *arXiv.org*. Thus, it might be promising to treat each year separately, learning a model on training data from that year and using it to obtain predictions for the test data from that year (or, at least treat year 2002 spearately from the previous two years). However, each of these models would then be based on fewer training papers, which would likely decrease its accuracy. Our preliminary experiments with separate predictors for each year did not prove promising, so we decided to always work with all the papers regardless of the year in which they were published in the archive.[3]

1.7 Distribution of the number of downloads

We can regard the download counts of our training papers as samples from an underlying probability distribution. One might expect this distribution, like that of so many other measurable aspects of social networks, to be "Zipfian" or power-law (where the probability density function has the form $f(x) \propto x^{-c}$ for some constant c). However, this would require probability density to strictly decrease as the number of number of downloads. The very low download counts should be the most common.

However, it turns out that this is not the case. The distribution of download counts is actually log-normal; that is, the logarithm of the download count is distributed approximately normally. Thus, the most frequent download counts are in the range

[3] We also tried adding the year of publication as an additional attribute, but it didn't lead to more accurate models. See Section 6.

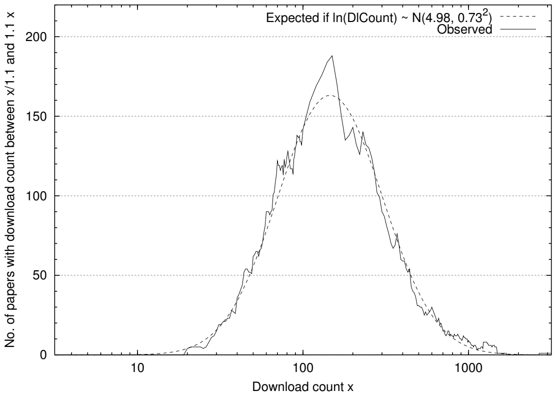


Figure 2: We can estimate the probability distribution of the download counts by counting, for each x, the documents whose download count falls within the range $[x/1.1, 1.1x]$ (this definition ensures that the ranges are all equally wide on the logarithmic scale). Note how well this empirical distribution coincides with the results we would get for a perfect lognormal distribution.

$[100, 200]$; smaller ones are just as rare as the larger ones.

The average value of $\ln(\textit{download count})$ on the training set is approx. 4.98, with a standard deviation of 0.73. Figure 2 compares the shape of the probability density function as estimated empirically from the training data with the shape it would have if $\ln(\textit{download count})$ were actually distributed as $N(4.98, 0.73^2)$. To illustrate how nicely this distribution fits our data, recall that the probability that a normally distributed variable falls less than one standard deviation away from its mean is approximately 68.3 %; in our data set, 1079 out of 1566 papers have a download count between $e^{4.98-0.73} \approx 70.4$ and $e^{4.98+0.73} \approx 301.9$, and $1079/1566 = 68.9\,\%$.

2 Our approach

2.1 Representing text

In fields such as information retrieval and text categorization, the traditional way of working with textual documents is by treating them as "bags of words". A *bag* is a mathematical concept similar to that of a set in that it implies no particular ordering of its elements, but differs from a set in that an element may occur in the bag more than once. Thus, a *bag of words* representation of a document means that the order of words in the document is completely ignored, but the number of occurrences of each word is preserved.[4]

Under the bag of words model, a document can be represented efficiently by a vector. Let d be a document, and $\mathbf{x}$ a vector representing it. Then $\mathbf{x}$ has one component, x_w, for each possible word w (i. e. each word that occurs in at least one of the documents we are dealing with). The value x_w is usually based on the *term frequency* $\mathit{TF}(w, d)$, i. e. the number of occurrences of the word w in the document d. Typically, it has the form

$$x_w = f_1(\mathit{TF}(w, d)) f_2(w) f_3(d),$$

and there are several possibilities for each of the functions $f_{1,2,3}$. f_1 is often the identity function, though it can be based on log to dampen excessively large TF values of some words. f_2 can be used to increase or decrease the influence of certain words. f_3 can be used for normalization, typically to ensure that $\mathbf{x}$ has unit length ($||\mathbf{x}||_2 = 1$).

Our representation is one of the variants of the well-known TF-IDF scheme:

$$x_w = \mathit{TF}(w, d)\mathit{IDF}(w) \cdot \textit{normalization}$$

where

$$\mathit{IDF}(w) = \ln \frac{\text{total number of documents}}{\text{number of documents containing } w}$$

is known as the *inverse document frequency*. The normalization value is used to ensure that $\mathbf{x}$ has unit length, i. e. $||\mathbf{x}||_2 = 1$.

The role of IDF is to reduce the influence of common words that occur in very many documents, because they are less likely to be useful for distinguishing classes of documents (in classification) or predicting some real-valued property of

[4] The bag of words model can also be extended to take word order into account, e. g. by recording pairs of adjacent words in addition to individual words. Longer sequences of adjacent words, known as *n-grams*, can also be included, though their usefulness is typically limited, and n-grams for $n > 5$ are rarely used. n-grams allow one to treat phrases as additional atomic parts of text and could thus potentially be helpful on our dataset, which contains scientific papers that often use specific (and hopefully unambiguous) phrases with a clearly defined meaning. However, we did not perform any experiments involving n-grams. This would be an interesting avenue for further work.

the documents (in regression). On the other hand, rarer words have greater IDF. Extremely rare words probably can't be useful for generalization from training to test data, and it is customary to discard them. We discarded words that occur in less than two documents.

Since there are usually tens of thousands of possible words but a document d typically contains just a few hundreds or perhaps thousands of them, $\mathit{TF}(w,d)$ equals 0 for most w. Thus most of the components of $\mathbf{x}$ are 0. The vector $\mathbf{x}$ can be stored and manipulated quite efficiently if only the nonzero components are actually stored: $\mathbf{x} = (x_1, \ldots, x_d)$ would be represented by a sequence of pairs (i, x_i) for all those i where $x_i \neq 0$. This is a convenient representation for operations such as linear combinations and dot products.

When dealing with textual data which has a very high number of features, it is often desirable to discard some features entirely. Various heuristics, such as information gain or odds ratio, are often used to decide which words are to be kept and which should be discarded. However, in text categorization it has been found that SVM does not really profit much from feature selection [BGMM02]. Therefore, the only form of feature selection that we use is discarding all features that occur in less than two documents on the entire dataset; after all, such a feature cannot possibly appear in *both* the training and the test set (and it's quite possible that it does not appear in either of them, for the training and test set are only a small part of the entire dataset), and so shouldn't be in any way useful for learning prediction. We apply this feature selection criterion to all features, not only those based on textual data. Our preliminary experiments with the "Abstract" representation (see Section 3.1) suggested that raising this threshold to more than two, even as many as ten, documents (which would cause more features to be discarded before learning) it not really helpful; the average test set error would increase (although very slightly).

2.2 Extending the representation with other features

We extend the bag-of-words idea by introducing additional features that will hopefully help us to obtain more accurate download predictions. One can view a paper as being a bag containing not just words but other kinds of items as well, e. g. authors, citations, a journal, and so on. Equivalently, this can be seen as extending the vectors representing the documents by adding new components. Different schemes of normalization and weighting can then be applied to different parts of the vectors. Most of our work focuses on experimenting with various sets of features to see which of them can lead to more successful models.

To actually obtain predictors, we use the regression SVM algorithm (support vector regression) [SS98]. This is a well-known algorithm that is quite efficient and has been found to be successful in many problem domains. Its output is a linear predictor, i. e. one of the form

$$\mathit{prediction}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b,$$

where $\mathbf{w}$ and b are selected by the learning algorithm. Given a sequence of training vectors and the corresponding labels, $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, l$, SVM regression finds the $\mathbf{w}$ and b that minimize

$$f(\mathbf{w}, b) := \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \max\{0, |\mathbf{w}^T\mathbf{x}_i + b - y_i| - \varepsilon\}.$$

Apart from minimizing the sum of absolute errors on the training set (which is just the sort of measure that would be used to evaluate our predictions on the KDD Cup), this criterion function attempts to avoid overfitting by ignoring small errors (smaller than ε) and by adding the regularization term $\mathbf{w}^T\mathbf{w}$.[5] We set the value of ε at 20 downloads (see Section 3.3), and the error cost parameter C at 1.

SVM can also work with nonlinear models; however, in text categorization it has often been found [Joa98, ZI02] that nonlinear SVM models are only very slightly more accurate than linear ones, while learning and using them is slower and the models

[5] This term encourages the learner to find a model that is a "flat" (i. e. not too steep) linear function. This is particularly helpful in high-dimensional spaces where linear models could otherwise easily overfit the training data. For example, in our experiments we will have at most approx. 1500 training examples, but more than 10000 dimensions. In one experiment we tried using traditional least-squares regression (which minimizes $g(\mathbf{w}, b) := \sum_i (\mathbf{w}^T\mathbf{x}_i + b - y_i)^2$), which resulted in models with an average error of around 10^{-13} on the training set and 394 on the test set. This is much worse than even the trivial models of Section 2.4: these models, which output a constant prediction (i. e. they have $\mathbf{w} = \mathbf{0}$), can achieve test set errors around 152.

are also less easy to understand. What is more, SVMs achieve nonlinearity using "kernels", a technique that introduces additional learning parameters; this makes exploring the parameter space and tuning the parameters still more complex and time-consuming. Therefore, although our problem is that of regression rather than classification, we decided to limit ourselves to linear SVMs and focus on choosing the features and their weights instead.

We used a well-known SVM library, *libSVM* [CL01], to train our SVM models.

2.3 The structure of our experiments

We were given the correct number of downloads for 1566 papers. We used ten-fold cross-validation on this set to compare different models and help us choose the features and weights that seem to work best. At the end, after we made our final choice of features and their corresponding weights, we trained the final model on the entire training set of 1566 papers. We then used this final model to obtain predictions for the 678 test papers. These predictions were then submitted to the KDD Cup.

For cross-validation, the set of 1566 papers was partitioned randomly into 10 disjoint groups or "folds". Nine of the folds were used for training and the resulting model was used to predict download counts for documents from the remaining fold. This was repeated ten times, the test fold being a different one every time, and the average error over all ten runs was used as the "score" of the current set of features and their weights.

Thus, in principle, we have approximately 1409 documents for training and 157 for testing. However, as has been mentioned in Section 1.2, in the end only our predictions on approximately 20 % of the papers, the most frequently downloaded ones, will be used in evaluation. It doesn't matter if our predictions for the other papers are wildly inaccurate. Thus we can hope to make our learning task somewhat easier by focusing on just the 20 % of the most frequently downloaded papers and ignoring the rest.

In effect, our model may treat every paper as if it were among the 20 % most frequently downloaded ones. A simple way of achieving this is to use only 20 % of the most frequently downloaded training papers for learning; then the learner, and the resulting model, will be almost unaffected by the existence of the other papers.

Our early experiments suggested, however (see Section 3.2), that the performance of our predictors is likely to be slightly better if more than 20 % of the training papers are used (see Section 3.2); thus we use 30 % of the training set for training, or approximately $1409 \cdot 30\,\% \approx 423$ papers. For evaluation, we use 20 % of the most frequently downloaded test papers, i. e. about $157 \cdot 20\,\% \approx 31$ papers.

2.4 A trivial model

As a baseline against which more sophisticated models can be compared, we considered a very simple model that always outputs some constant c, i. e. the same prediction for all papers. Let the true download counts be $y_1, \ldots, y_n$, and assume without loss of generality that $y_1 \leq y_2 \leq \ldots \leq y_n$. The error measure that we would like to minimize is $f(c) := \sum_{i=1}^{n} |y_i - c|$. If $y_j \leq c \leq y_{j+1}$, this is equivalent to $\sum_{i=1}^{j}(c - y_i) + \sum_{i=j+1}^{n}(y_i - c)$, and its derivative is $f'(c) = j - (n - j) = 2j - n$. Thus, if $c > y_{\lfloor n/2 \rfloor + 1}$, the derivative is positive and decreasing c would decrease the error. Similarly, if $c < y_{\lfloor (n+1)/2 \rfloor}$, the derivative is negative and increasing c would decrease the error. Thus, if n is even, the optimal value of c is any from the range $[y_{n/2}, y_{n/2+1}]$. If n is odd, the optimal value is $c = y_{\lceil n/2 \rceil}$. Therefore, if we decide to use a constant model of this sort, it makes sense to use the median download count (computed from the training papers) as our prediction. Alternatively, one could also use the average download count.

As described in the previous section, only the most frequently downloaded training papers are used for training the model. In later experiments we will fix this proportion at 30 % of the training papers, but here we also consider other perecentages. The average number of downloads over these training documents (shown in the second column) is then used as the prediction of the model on all test papers. To compute the average error, we always look at the prediction errors on 20 % of the most frequently downloaded test documents.

The results of these experiments are shown in Table 2 and Figure 3. The variance of the number of downloads is rather large and thus, as the table shows, the trivial models are woefully inaccurate.

	Training set statistics		Avg. err. on the test set when predicting with the	
Proportion of training set used	Average no. of downloads	Median no. of downloads	average	median
10 %	608.8	501.2	254.3	192.5
20 %	458.8	372.3	172.0	152.3
30 %	384.2	303.2	152.5	166.9
40 %	335.1	262.4	155.6	194.0

Table 2: Average errors of trivial predictors based on the median or average number of downloads computed on the training set.

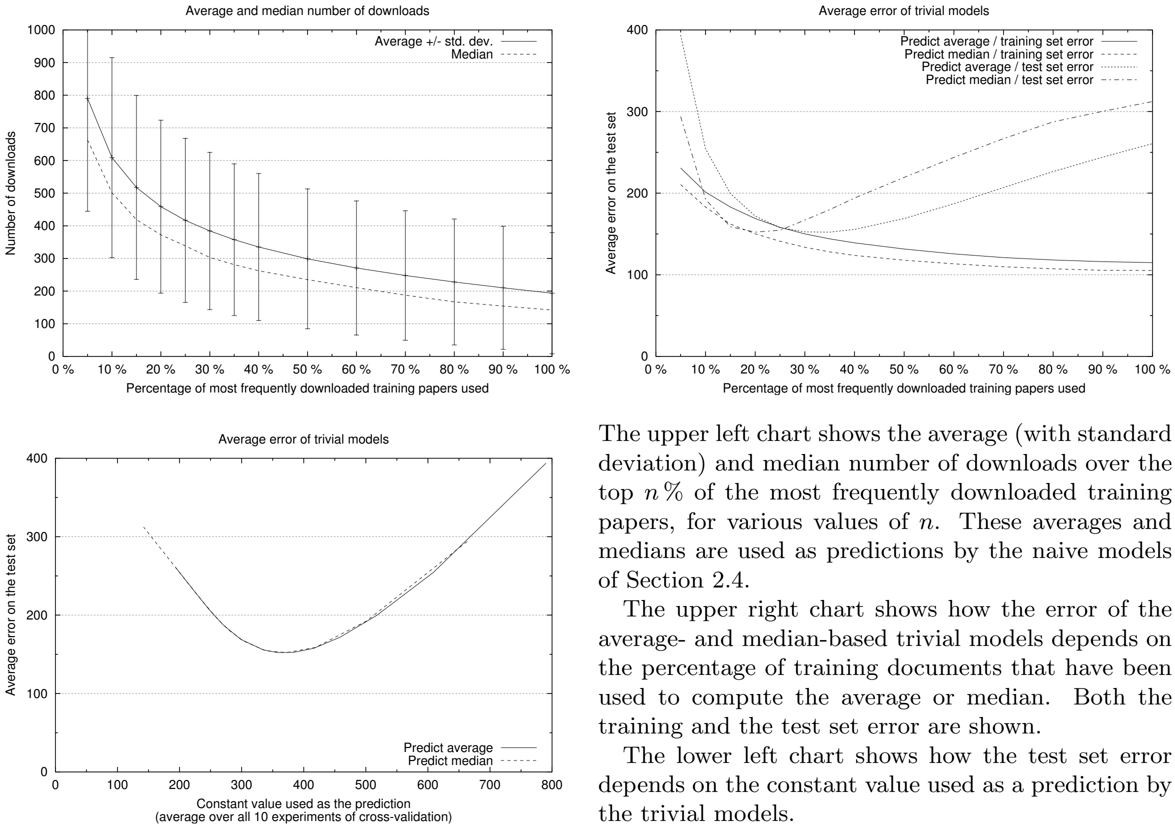
When predicting with the median, using 20 % of the training data gives the smallest error (152.2). This is, of course, reasonable because it is then that the training median is the closest to the test median (because 20 % of the test papers are used for evaluation). The average is usually larger than the median, so using just 20 % of the most frequently downloaded training papers to estimate the average results in too optimistic predictions (avg. error: 172.0), and it is better to estimate the average using 30 % of the training papers: this includes some of the less frequent ones, the average is therefore smaller and so is the error on the test set (152.5). The best that could be done by a predictor of this type would be, of course, to predict the median download count of the test set; this would result in an average error of 147.9.

2.5 Further nearly trivial models

The trivial model described in the previous sections could be extended by choosing a constant prediction based on some property of the paper, rather than having just one prediction for all the papers. For example, in Section 1.6 we've seen that there were on average more downloads in 2000 than in 2001, and more in 2002 than in 2000. So we might have three constants and use the **year** when a paper was published to choose one of them as our prediction for that paper. However, the benefits of this are very small. The median of the entire training set is, on average (across all 10 experiments in the cross-validation), 372.3; the medians for individual years are 377.6 (year 2000), 353.7 (2001), and 379.8 (2002). Thus having separate predictions for each year only makes a real difference for the year 2001, and the average error on papers from that year really decreases a little (from 138.3 to 134.8). (See also the lower left chart of Figure 3; the flat area of the curve around the minimum shows that if the constant value used as the prediction by our trivial models changes by, say, 20 or so, the change in average error will be very small.) The average error over all the test papers is now 151.3, as opposed to 152.3 of the model that uses the median of the entire training set. Using the test set to compute the medians would now result in an average error of 144.0 (as opposed to 147.9 if the median for the entire test set had been used).

A similar approach would be to have one constant prediction for each possible **number of authors**. (Each paper has at least one and at most 10 authors; from 3 authors upwards, the average number of downloads decreases as the number of authors increases.) That is, our prediction for a paper d is the median number of downloads over all the training papers with the same number of authors as d. It turns out that this causes the average error on the training set to decrease from 150.2 to 148.6, but on the test set to increase from 152.3 to 154.7. This means that overfitting is ocurring and our trivial predictor cannot be easily improved along such simple lines.

As yet another alternative, we could have a separate prediction for each **journal**. The prediction for a paper d is then simply the median number of downloads over all training papers that belong to the same journal as p. However, it turns out that this leads to overfitting: the error on the training set decreases from 150.2 to 131.8, but on the test set it increases from 152.2 to 165.4. (If we could compute the per-journal medians on the test set and use these as our predictions, the test set error would decrease to a mere 107.4. Of course, this model has many parameters, one for each journal, so it is unsurprising that if they are chosen optimally the performance can be really good — but, as we have seen, it is so much more difficult to choose



The upper left chart shows the average (with standard deviation) and median number of downloads over the top $n\,\%$ of the most frequently downloaded training papers, for various values of n. These averages and medians are used as predictions by the naive models of Section 2.4.

The upper right chart shows how the error of the average- and median-based trivial models depends on the percentage of training documents that have been used to compute the average or median. Both the training and the test set error are shown.

The lower left chart shows how the test set error depends on the constant value used as a prediction by the trivial models.

Figure 3: Experiments with trivial predictors (see Section 2.4).

them even remotely well, let alone optimally.)

Another possibility is using the median number of downloads over all training papers from the same **cluster** as the paper whose download count we want to predict. We used a recursive version of 2-means (see Section 7) and partitioned the papers into 26 clusters, or 7 "super-clusters" if the cutoff criterion is set differently. Using the median over all training papers from the same cluster results in mild overfitting (training set error 138.8, test set error 154.4), but using the superclusters actually achieves the best result of this section (training set error 147.0, test set error 153.0). Of course, this still not any better than the error of the original model, which is 152.2.

Summary. In this section, we attempted to extend the trivial predictor from the preceding section (which outputs a constant value) into a **regression stump** (a one-level regression tree) with a constant prediction in each leaf. We have seen that performance cannot be easily improved in this way, and in particular it is difficult to avoid overfitting. Obviously, one could try to go still further in this direction and train larger, full-blown regression trees. The enticing aspect of this approach is that the models might be (relatively) easily understood and interpreted. However, since these first steps were so un-promising in terms of prediction accuracy, we will not pursue this direction any further and will instead focus on SVM regression for the remainder of this paper.

3 Abstract, author, address

3.1 The basic features

An obvious thing to try is to represent each paper by its **abstract and title**. Both are seen by the readers on the "What's new" page. For the purposes of processing, both are easily obtained from the separate abstract file that is available for each paper. Each paper is treated as a bag containing the words from the title and abstract. We use the TF-IDF weights and normalize the resulting vector to unit length, as has been described in Section 2.1. Note that normalization is not strictly necessary, but if it is omitted, the components of the vectors are much larger and, to avoid a rather bad case of overfitting, we must either change SVM's error cost parameter (C, which we will always set to 1) or multiply all the vectors by some constant weight to reduce the values of the components. But all this weighting is tedious, and there will be plenty of it later anyway, so we prefer to simply normalize each vector to have unit Euclidean length.

Author names are another obvious source of features. These are available in the abstract files, but are meant to be human-readable rather than strictly formal and consistent. First names are sometimes given in full, sometimes the initial only; middle initials are sometimes included and sometimes omitted. The names sometimes include TEX commands for special letters and diacritic marks. When there are several authors, their names are sometimes separated by commas, sometimes by *and*, sometimes by both.

We tried to normalize the names somewhat by removing diacritic marks (e. g. `\v{c}`, which represents the letter *č*, is replaced by `c`), replacing TEX commands for special letters with ASCII approximations of those letters (e. g. `\ss`, which represents the letter *ß*, is replaced by `ss`), retaining just the initial letter of the first name and discarding any middle initials. Text surrounded by parentheses is also removed from the author descriptions during this preprocessing, as it usually gives information about the institution rather then the author's name. This does not clean the data perfectly, but in most cases it is quite sufficient. Examples:

```
Alfredo Mac\'{\i}as, Abel Camacho,
Eckehard W. Mielke (UAM-I, Mexico)
and Tonatiuh Matos (CINVESTAV, Mexico)
        ⟶ macias-a camacho-a mielke-e matos-t

Chiang-Mei Chen, T. Harko, W. F. Kao and
M. K. Mak          ⟶ chen-c harko-t kao-w mak-m
```

n	1	2	3	4	5	6	7	8	9	10
No. of papers with n authors	11313	10039	5339	1814	354	120	28	4	2	1

We then define one feature for each author. In the vector $\mathbf{x}$ representing a document d with $n(d)$ authors, the component x_a has the value $1/\sqrt{n(d)}$ if a is one of the authors of d, and 0 otherwise. Thus all vectors have unit length, regardless of the number of authors.

One unpleasant aspect of author names as attributes is that, in the relatively short time periods from which our training and test papers are taken, few people have managed to write more than one paper. If we take 30 % of the most frequently downloaded papers from the training period and 20 % from the test period, we find that only 84 authors have written at least one paper from each of these two groups. And unless an author is present both in the training and the test set, it will be either impossible to learn about him or her from the training set, or whatever we will learn about him or her will be useless on the test set.

Incidentally, after our preprocessing there are a total of 8599 distinct authors left in the dataset (of course it is possible that several authors with the same last name and first initial have been confused into one "author"). As there are 29014 papers, this means that the average author has written about 3.37 papers in this dataset.

Another potentially interesting source of information is the **address** of the institution(s) where the authors work. It seems plausible that, other things being equal, a paper is more likely to be downloaded if its authors are from a reputable and well-known institution. Before downloading and reading a paper, the reader wants to know that the paper will be sufficiently relevant, informative, etc., and in the absence of more direct information about this, reputation of the author(s) and institution(s) may influence the reader's decision.

Admittedly, the "new" and "recent" pages (whence most downloads are believed to originate) do not show institution information (unless this is included in the "Author" field, which does happen but not very often[6]). Thus one might expect that the direct influence of the institution address on a reader's decision whether to download a paper or not is relatively small. There may be an indirect influence, however, insofar as "better" institutions

[6] When names institutions are mentioned in the "Author" field, they are usually enclosed by parentheses. It turns out that only 190 of the 29014 papers have any "(" or ")" characters in their "Author" fields.

tend to employ “better” authors who write “better” papers, and readers may recognize better papers as being such even without knowing the author’s institution. In other words, even though the institution address does not directly influece the number of downloads (because the readers don’t see this information when deciding about the download), it might still be sufficiently well correlated with it to be useful for predicting it.

The abstract files do not contain institution information (except if included in the “Author” field), so they had to be extracted from the TeX files containing the full text of the papers. Some TeX files use the `\address` command, but many only use formatting commands such as `\large`, `\bf`, vertical spacing and centering to set the (author’s name and) address off from the surrounding text. It would be too time-consuming to extract the addresses manually; therefore, an automatic but not completely reliable procedure had to be devised.

1. First we remove the abstract and everything following it from the TeX file. After all, the addresses practically always appear before the abstract.[7]

2. If a bibliography appears at the top of the paper (before the abstract), it is also removed.

3. If any `\address{...}` commands occur in the text, extract the contents of their arguments. If there are no such, try `\def\addresses{...}` as well.

4. If no addresses have been found in the previous step, try looking for certain words (or parts of words) such as “univer”, “institu”, “depar”, etc.; if any such word is found, try to determine a suitable block of text containing this word and declare this to be an institutional address. The block of text is defined based on matching braces (`{` and `}`), or the `\vspace` and `\vskip` command (which are often used to insert vertical spacing around addresses), or `\center`, or a matching pair of `\begin` and `\end`.

Representation	Average error on the training	test set
Author	63.66	146.38
Abstract	62.46	149.28
Address	80.60	154.06
Abstract + Address	42.20	142.89
1.4 Abstract + Address	32.47	141.75
Abstract + Author	37.62	**135.80**
Address + Author	49.19	143.38
Address + 1.2 Author	44.75	142.96
Abstract + Address + Author	32.29	136.64
1.2 Abstract + 0.6 Address + Author	31.56	**134.70**

Table 3: The performance of various representations based on the author, abstract, and address features described in Section 3.1.

Once the addresses have been extracted in this way, each paper can be represented by the bag of words from the address (or addresses) found in this paper. The resulting vector is again based on TF-IDF weighting and normalized to unit length.

The results of experiments with these three representations are shown in Table 3. The “Author” representation is the best of the three, with an average error of 146.4. Unfortunately, this is a rather small improvement over 152.3, which is the error of the best trivial model from Section 2.4.

An obvious thing to try now is **combining several representations**. In this case vectors resulting from two or more representations are simply concatenated together; that is, if one representation of the document d is the vector $\mathbf{x} = (x_1, \ldots, x_d)$ and the other is $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_r)$, the combined representation of d would be the vector $(x_1, \ldots, x_d, \hat{x}_1, \ldots, \hat{x}_r)$. Additionally, one might also apply different weights to different parts of the combined representation, to adjust their influence on the learning process and the resulting predictions. Thus combining $\mathbf{x}$ and $\hat{\mathbf{x}}$ would, in general, result in the vector $(\alpha x_1, \ldots, \alpha x_d, \beta \hat{x}_1, \ldots, \beta \hat{x}_r)$, where α and β are two constants, typically cho-

[7] To find the abstract, we can use the text of the abstract that is available in the abstract file for the current paper. In the TeX file, the abstract usually appears with the same text, but may be interspersed with additional TeX commands, line breaks, etc. For greater robustness, we use the following approach to find the abstract in the TeX file: for each word from the text of the abstract, we mark all occurrences of this word in the TeX file; then we move through the TeX file with a “window” whose length (in characters) is the same as that of the abstract text. We find the position of the window where it contains the largest number of marked characters; this is considered to be the location of the abstract in the TeX file. This algorithm is simple and quite reliable; it occasionally misses the first few words of the abstract, but (what is more important to is) it usually doesn’t place it too early in the file (which, if it happened, could cause some useful text to be removed together with the abstract and the subsequent contents of the file).

sen through cross-validation. We will use this same principle to extend our representations throughout the rest of this paper.

When experimenting with using two out of the three groups of features described here (i. e. author, abstract, and address), we always fixed the weight of one of these two groups at 1 and varied the weight of the other group. This was done chiefly in order to limit the number of combinations of weights that would otherwise have to be tried. As it turns out, keeping the other group's weight at 1 is usually the best choice, or at least not much worse than the best choice (e. g. "1.4 Abstract + Address" is slightly better than "Abstract + Address", and "1.2 Author + Address" is slightly better than "Author + Address"). Using excessively large weights always leads into overfitting; training error decreases to around 30 or even around 20, while test error grows to around 160 or even 180.

The best performance obtained in this way (for each pair of representations) is shown in Table 3. "Abstract + author" is the most successful combination, with an average error of 135.8.

Combining all three groups of features poses an even larger problem of how to find as good a weighting as possible without having to explore too many groups of weights. We tried two approaches. (1) In the experiments with combining two of the three groups of features, it has been found that the best weights are usually close to 1. Therefore we tried all 5^3 combinations of weights from the set $\{0.6, 0.8, 1, 1.2, 1.4\}$. The best representation found in this way is "1.2 Abstract + 0.6 Address + Author", with an average error of 134.70. The "naive" combination where all three groups have a weight of 1 has average error 136.64.

(2) Experiments with combining two groups of features such that the weight of one of these two was fixed at 1 have resulted in five promising representations (the second group in Table 3). For each of these, we hold the weights of the existing two groups of features constant and then vary the weight of the third group (which was previously not present in the representation, i. e. it had a weight of 0). These experiments show that tuning the weight of the third feature in this way is not much better than simply setting it to 1; either way, the resulting error is between 135 and 137. As before, excessively large weights still lead to much higher errors. None of the representations obtained in this way is better than the best representation from the preceding paragraph.

We decided to adopt the "Abstract + Author" representation as the baseline for further experiments. (In the remainder of this paper, this representation will often be referred to simply as "AA".) Its performance is only slightly worse than that of the best representation found in this section, which is the considerably more complex "1.2 Abstract + 0.6 Address + Author". In the spirit of Ockham's razor, we believe that the small reduction of error (from 135.80 to 134.70) is not worth the additional complexity of the model.

3.2 Influence of the amount of training data

In Section 2.3, we argue that because our predictor will only be evaluated on the 50 most frequently downloaded papers of each month, or about 20 % of the test papers, it might also be beneficial if the predictor is trained on only the most frequently downloaded 20 % of the training papers. However, experiments have shown that using somewhat more than that can actually result in a more accurate predictor.

Figure 4 shows how the average error varies with the percentage of the training set that has been used for training. The "Author + Abstract" representation from Section 3.1 has been used in these experiments. We note that the best results are achieved when approximately 30 % of the training papers are used to train the model.

3.3 Influence of the ε parameter

Recall from Section 2.2 that regression SVM tries to minimize the "ε-insensitive loss function", $\sum_i \max\{0, |\mathbf{w}^T\mathbf{x} + b - y| - \varepsilon\}$. This means that all training set errors are decreased by ε, and errors that were less than ε to begin with are completely ignored.

The SVM library which we used, *libSVM*, has a default value of $\varepsilon = 0.1$. Since we divide all download counts by 200 before training (to bring them from the range of approx. [0, 2000] to a range of smaller values, such as [0, 10]), $\varepsilon = 0.1$ corresponds to 20 downloads. We used this default value in all the experiments presented elsewhere in this report.

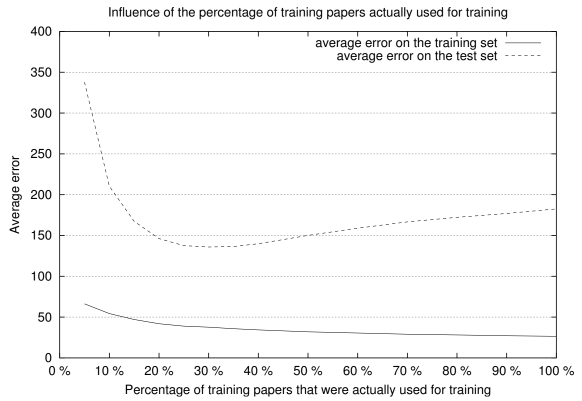


Figure 4: This chart shows how the training and test error change depending on what percentage of the most highly downloaded training papers have actually been used to train the regression SVM predictor. The performance of the model is then evaluated both on the set of papers on which it was trained as well as on 20 % of the most highly downloaded papers from the test set. Note how the best performance is achieved when 30 % rather than just 20 % of the training set has been used. The "author + abstract" representation has been used in these experiments.

In this section, however, we present the results of experiments with the "Author + Abstract" representation (see Section 3.1) and with various values of ε. As expected, the training error increases if ε is increased. Surprisingly, however, the increase of ε has only a very small effect on the test error. Test error decreases a little as ε increases, from 135.8 when $\varepsilon = 0.1$ to 134.6 when $\varepsilon = 0.75$. Note that this value of ε corresponds to ignoring all training errors less than 150 downloads; nevertheless, the average training error is still just 58.4 (up from 37.6 at $\varepsilon = 0.1$). For larger values of ε, test set error begins to grow relatively quickly, reaching 137.0 at $\varepsilon = 1$ and 140.8 at $\varepsilon = 1.2$.

3.4 Principal component analysis

Principal component analysis, or PCA for short, is a well-known data analysis technique. Given a random vector variable $\mathbf{X}$ and a direction vector $\mathbf{z}$ (assuming, without loss of generality, that $||\mathbf{z}|| = 1$), $\mathbf{z}^T\mathbf{X}$ is the projection of $\mathbf{X}$ in the direction of $\mathbf{z}$. The *principal component* of $\mathbf{X}$ is the direction $\mathbf{z}$ that maximizes the variance of the projection $\mathbf{z}^T\mathbf{X}$. If this direction is then projected out of the data, leaving $\hat{\mathbf{X}} := \mathbf{X} - (\mathbf{z}^T\mathbf{X})\mathbf{z}$, the second principal component of $\mathbf{X}$ is the direction $\hat{\mathbf{z}}$ that maximizes the variance of $\hat{\mathbf{z}}^T\hat{\mathbf{X}}$, etc.

It turns out that the principal components are simply the eigenvectors of the covariance matrix of $\mathbf{X}$.[8] Thus the first few principal components can be found using the simple iterative methods for computing eivenvalues. However, this requires the ability to premultiply an arbitrary vector $\mathbf{y}$ by the covariance matrix of $\mathbf{X}$.

Our data vectors $\mathbf{x}^1, \ldots, \mathbf{x}^n$, representing the n training documents, can be seen as samples of a random variable $\mathbf{X}$. The mean of $\mathbf{X}$ can then be estimated as $\mathbf{m} = \frac{1}{n}\sum_{k=1}^n \mathbf{x}^k$ and the covariance matrix of $\mathbf{X}$ as $C = (c_{ij})$ for $c_{ij} = \frac{1}{n}\sum_{k=1}^n (x_i^k - m_i)(x_j^k - m_j)$. Of course, if our data is d-dimensional, the covariance matrix is a $d \times d$ matrix, and it is usually not particularly sparse, so working with can be difficult if d is large. If $X = [\mathbf{x}^1|\ldots|\mathbf{x}^n]$ is a $d \times n$ matrix whose columns are our data vectors, we can write C as $C = \frac{1}{n}XX^T - \mathbf{m}\mathbf{m}^T$. Because our data vectors are sparse, multiplying by C is quite easy now: $C\mathbf{y} = \frac{1}{n}XX^T\mathbf{y} - \mathbf{m}\mathbf{m}^T\mathbf{y} = \frac{1}{n}\sum_{k=1}^n(\mathbf{y}^T\mathbf{x}^k)\mathbf{x}^k - (\mathbf{y}^T\mathbf{m})\mathbf{m}$. The dot products $\mathbf{y}^T\mathbf{x}^k$ can be computed efficiently because the $\mathbf{x}^k$ are sparse. There is no need to store the covariance matrix C explicitly.

After determining the first few principal components of our data, say $\mathbf{z}^1, \ldots, \mathbf{z}^k$ (all of unit length), we can use them as a new coordinate system, approximately representing any vector $\mathbf{x}$ by its projections $(\mathbf{x}^T\mathbf{z}^1, \ldots, \mathbf{x}^T\mathbf{z}^k)$ onto the directions of the principal components. This could be useful if only a small number of components (small k, much less than the original number of dimensions d) is sufficient to explain most of the variance in the data. In this case the k principal components form a more "natural" coordinate system for the given data, and limiting ourselves to the subspace spanned by these components could be seen as removing uninteresting noise from the data.

However, as it turns out, the vectors we're working with are scattered in too many directions. Even keeping as many as $k = 100$ principal components only preserves a small amount of variance and dis-

[8] The eigenvalue corresponding to an eigenvector $\mathbf{z}$ is then equal to the variance of the data in the direction of $\mathbf{z}$. The sum of all the eigenvalues is equal to the total variance in the data, i. e. to $E[||\mathbf{X} - E[\mathbf{X}]||^2]$.

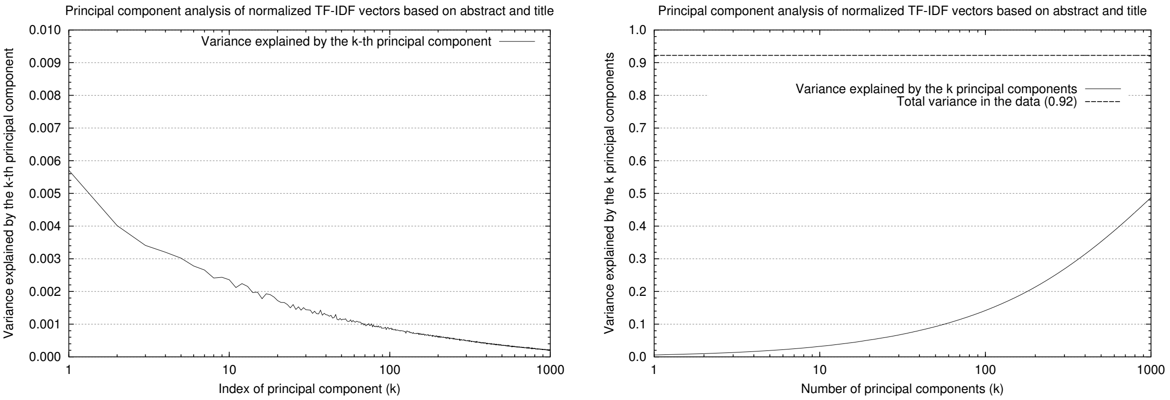


Figure 5: We represented each of the 29014 papers by a normalized TF-IDF vector based on text of the title and abstract. Then we tried to find 1000 principal components of this set of vectors.

The left chart shows the amount of variance along each of these primary components. Since the k-th primary component is really the k-th principal eigenvector of the covariance matrix of our data, and the variance along this direction is the corresponding value, this variance should always be smaller for larger values of k. The graph shows that this is not strictly the case here, which means that numerical inaccuracies are seriously affecting our computation except for the first few principal components. However, the main message of this graph is that the first few principal components explain very little variance in the data, so we do not expect that PCA could be used to efficiently map our data into a low-dimensional space.

The right chart shows the total amount of variance explained by the first k principal components. This is in effect a cumulative version of the left chart. Note that even after taking 1000 principal components, nearly half of the original variance remains unaccounted for.

cards too much useful information. Thus the accuracy of models based on the vectors in the new coordinate system (the one implied by the k principal components) is worse than with the original representation. What is more, computing many principal components is computationally quite expensive compared to the other operations in our data preparation and learning process. When computing the k-th eigenvector of the covariance matrix C, after each multiplication of the current approximation of $\mathbf{z}^k$ by the matrix C, the contributions of earlier eigenvectors must be projected out to make sure that the computation does not converge to one of them (which would surely happen because of numerical inaccuracies and the fact that the earlier eigenvectors have larger eigenvalues). That is, $\mathbf{z}^k$ must be replaced by $\mathbf{z}^k - \sum_{i=1}^{k-1}((\mathbf{z}^i)^T\mathbf{z}^k)\mathbf{z}^i$ and normalized. The cost of this operation grows (linearly) with k, i.e. computing each additional principal component is a little more expensive than computing the previous one.

For example, suppose that each of our 29014 papers is represented by a normalized TF-IDF vector based on the contents of that paper's title and abstract (see Figure 5). It turns out that the total variance of this set of vectors, i.e. the average squared distance from a vector to the average of all these vectors, is approx. 0.92. The variance along the first principal component, however, is only 0.0057! The variance left if we keep just the first 10 principal components is only 0.032; if we keep the first 100 components, the variance is 0.14; if we keep the first 1000, the variance is 0.49. Thus we see that it takes a very large number of principal components to preserve a substantial amount of variance. Computing so many principal components is slow and numerically unreliable. Representing each vector $\mathbf{x}$ by its projections on the k principal components and using this as the input to the SVM learner resulted in larger errors than using the original title-and-abstract vectors. The errors decrease as k increases, but even at $k = 100$ the test set error is still 157.97, much larger than 149.28, which is achieved by the original "Abstract" representation of Section 3.1.

3.5 Author-based statistics

One hopes that papers by some authors tend to be downloaded more often than those by some others, either because their research deals with topics that are currently attracting more interest, or because that author's papers are on average of a higher quality, or because the author's name and reputation are recognized and attract more downloads by themselves.

Therefore, we tried introducing, as additional features of each paper, some statistical information about the download counts of the training papers written by the same authors. Let $DL_{avg}(a)$ and $DL_{max}(a)$ be the average and the maximum number of downloads over all the training papers co-authored by the author a. Then, in the vector $\mathbf{x}$ representing a paper d, we have the following four new features: $\text{avg}_a DL_{avg}(a)$, $\text{avg}_a DL_{max}(a)$, $\max_a DL_{avg}(a)$, $\max_a DL_{max}(a)$. These averages and maxima are always computed over all the authors a of the paper d.[9]

However, it turns out that adding these features to the basic "author + abstract" representation does not improve the accuracy of our predictions. Instead, it leads into overfitting.[10] This is probably caused by the fact that most authors have written only a few papers during the six months with which we are dealing. In each of the 10 training/test runs of our cross-validation procedure, there are on average 423.2 training papers (30 % of the nine training folds) and 31.8 test papers (20 % of the test fold). Of the latter, only 21.1 have at least one author that has also (co-)written a training paper, and only 10.1 have only such authors. On average (over all test papers), the number of training papers that have at least one author in common with a given test paper is only 1.59! The test set mentions, on average, 67.1 different authors, of which only 34.1 have also (co-)written one of the training papers.

These statistics show that there is not very much overlap between the authors of the training set and those of the test set. Thus it is not surprising that the features proposed in this section have not turned out to be useful. Of course, in the end, when preparing predictions for evaluation on the KDD Cup, we would have a larger test set than during these cross-validation experiments (150 documents rather than 31 or 32), so these features could perhaps be more helpful; however, since we wouldn't be able to tune their weights using cross-validation, we decided not to use them after all.

4 Using the citation graph

The citation graph that has been provided by the KDD Cup organizers covers all the citations within our dataset of 29014 papers. If paper i cites paper j, the graph contains a directed edge from i to j. If a paper cites, or is cited by, a paper outside the dataset, the citation graph does not contain any information about this.[11]

Given that we are interested in downloads within the first two months of a paper's presence in the archive, and a paper probably does not accumulate many citations in the first two months, it seems reasonable to suppose that the number of downloads is not directly influenced by the citations. People download these papers because they found them interesting on the "new" or "recent" listings, rather

[9] If an author a of a test paper d has not (co-)authored any of the training papers, the values $DL_{max}(a)$ and $DL_{avg}(a)$ cannot be defined meaningfully and this author would be ignored in the computation of avg_a and $\max_a$. If this problem occurs with all the authors of d, these four features cannot be computed at all; what should be done in that case? Note that this won't ever happen if d is a training paper, so it won't affect the learning; the model will come to rely at least partly on these features, and will get into trouble on the test set when they ocasionally don't have meaningful values. Leaving them at some irrelevant value such as 0 would be horrible. It is necessary to supply some sensible number of downloads to be used when the author-based statistics cannot be computed. We decided to use simply the average number of downloads over all training papers in such cases.

[10] For example, adding the $\max DL_{max}(a)$ attribute reduces the training set error from 37.6 to 28.3, but increases the test set error from 136.0 to 155.2! The other three attributes mentioned here give similar (in fact even slightly worse) results, as does including two or more of these attributes at the same time, so we will omit further details.

[11] According to the KDD Cup organizers (`http://www.cs.cornell.edu/projects/kddcup/download/KDDCup-Overview.pdf`), this graph is based on data from SPIRES (Stanford Public Information REtrieval System). The SPIRES web site (`http://www.slac.stanford.edu/spires/hep/references.html`) states that the citation information is extracted automatically from the text of each paper and is then checked by a human and corrected as necessary. Thus we believe that the citation graph is quite reliable and we simply use it as it was given, without attempting to further investigate its accuracy or reliability.

than because another paper has referenced them.

However, in the long term, citations are an indication of the quality and relevance of the paper; and better and more relevant papers are probably more likely to be downloaded. Additionally, a heavily downloaded paper is likely to be more widely read, and thus more likely to influence other researchers and be cited by them. Because of these indirect relationships between citations and downloads, citations may be helpful for predicting downloads. Although the readers do not see any citation information when deciding whether to download a paper (one that has been in the archive for less than two months), they may be able to recognize promising papers from the title and abstract. Our learning algorithm probably couldn't do this, but perhaps the citations could give it a hint as to which papers are popular.

4.1 Measures and weights implied by the graph

Using the citation graph, we can obtain various numerical features for each paper. The **in-degree** (the number of edges pointing into a node, i. e. the number of times this paper was cited by other papers) and the **out-degree** (the number of citations this paper makes to other papers) are perhaps the most obvious. The in-degree is, of course, famously regarded as an indicator of the quality and relevance of papers, and we expect it to be useful in our task as well. On the other hand, the out-degree is simply the number of references to other papers that occur at the end of a paper, and there doesn't seem to be any obvious reason why this should be related to the number of downloads (except, perhaps, in the case of survey papers which typically have lots of references and are also read (and thus perhaps downloaded) by many people).

The citation graph contains 342437 edges, which, since there are 29014 papers, means that the average in-degree and the average out-degree are approximately 11.8. However (as is reasonable), the standard deviation of the in-degree is 38.3, more than twice as large as that of the out-degree 15.7. In other words, there is a lot more inequality in the number of times a paper has been cited by others than in the number of citations that a paper makes to other papers.

The **hub value** and **authority value**, as computed by Kleinberg's HITS algorithm [Kle99], are another well-known pair of measures of relevance of a node within a network. The iterative algorithm, originally developed for use on (carefully constructed subgraphs of) the WWW-graph, is based on the intuition that a page is a good hub if it points to pages with authoritative content, and a page is a good authority on a topic if it is pointed to by good hubs. A hub value and an authority value are associated with each page; the algorithm then iterates two steps: first, each page divides its hub value among the pages to which it points, and the resulting values become the new authority values; then, each page divides its authority value among the pages which point to it, and the resulting values become the new hub values:

```
1 (Initialization.) For each $i$, set $a[i]$ and $h[i]$ to 1.
2 (A page is a good authority if it is cited by good
  hubs.) For each $i$, set $a[i]$ to
      $c + (1 - c) \sum_{j:j \text{ points to } i} h[j]/\mathit{OutDeg}(j)$.
3 (Normalization.) Compute the average value of $a[i]$
  and divide each $a[i]$ by it.
4 (A page is a good hub if it cites good authorities.)
  For each $i$, set $h[i]$ to
      $c + (1 - c) \sum_{j:i \text{ points to } j} a[j]/\mathit{InDeg}(j)$.
5 (Normalization.) Compute the average value of $h[i]$
  and divide each $h[i]$ by it.
6 If the values $a[\cdot]$ and $h[\cdot]$ haven't changed enough,
  stop. Otherwise, go back to step 2.
```

The parameter c encourages numerical stability and quicker convergence. Since each page gets this much authority and hub weight "for free", it means that the larger the c, the smaller the differences between weights of different nodes (the set of weights becomes more and more "egalitarian"). We used $c = 0.02$. The values usually converge fairly quickly.

When applied to our citation graph, one expects highly cited papers, papers that stimulated a lot of further research, to have high authority values. On the other hand, a survey paper that references many authoritative papers on a certain topic is likely to have a high hub value. However, since practically every paper, even if it is of poor quality or relevance, references several authoritative papers from its field, the hub values are less likely to be helpful for discriminating between more and less relevant papers. Besides, similarly to the relationship between the out-degrees and in-degrees, there is less diversity in hub values than in authority values; the standard deviation of the of the authority values is 2.72, while the standard deviation of the hub values is only 1.10.

PageRank [BP98, PBMW98], originally developed for use on the WWW-graph in the Google search engine, is another well-known weighting scheme, based on ideas similar to HITS. Here, each page has a single weight, called the PageRank. In each iteration of the algorithm, the weight of each page is distributed evenly among the pages to which it points. For greater stability and faster convergence, each page also gets a small constant amount of weight in each iteration, regardless of its position in the graph.

The same algorithm can also be applied to our citation graph. However, its usefulness for our task is limited by the fact that all citations point backward in time, which is not the case with web pages (many good web pages are updated every now and then, which allows them to point to pages that were added to the web later). Thus, on our citation graph, most of the PageRank would accumulate in the earliest papers in the dataset (actually in the ones that do not reference any other papers in the dataset). The chart in Figure 6 shows that later papers really tend to have much lower PageRank than earlier ones.

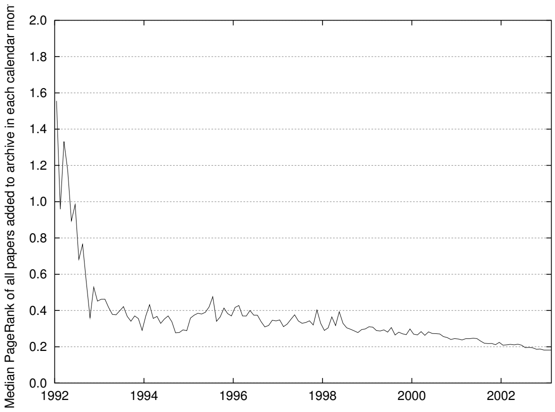


Figure 6: This chart shows, for each calendar month, the median PageRank over all papers added to the archive in that month. Notice how most of the PageRank has accumulated in the earliest papers, which have no outgoing links and thus act lie "sinks" of PageRank from the entire dataset.

Note that HITS does not have this problem because it moves the weights both forwards and backwards through the graph, not just in one direction.

We also computed a version of PageRank with the directions of all the edges reversed (or, in other words, a paper's weight is distributed among those papers which point to it, rather than those to which it points). For lack of a better term, we refer to this measure as **PageKnar**. Needless to say, it isn't very useful, and it tends to accumulate among the later papers, just like PageRank accumulates among the earlier ones.

4.2 Statistical properties of these attributes

Correlations between these attributes. As Table 6 shows, in-degree, authority weight, and PageRank are strongly correlated (particularly the first two of these three). Similarly, out-degree and hub weight are strongly correlated, and PageRank is somewhat less strongly correlated to these two. These two groups of features are practically independent of each other (correlation coefficients close to 0). None of the features is really strongly correlated to the number of downloads (i. e. the value that we wish to predict); authority has the strongest correlation with the number of downloads (0.26), followed by in-degree (0.24) and PageRank (0.21), followed by out-degree and hub weight (0.15), followed finally by PageKnar (0.06). This order is just what one might have expected from the explanation of how these various features have been obtained, but it is somewhat disappointing that the correlation coefficients are so low.

4.3 Using these features for download estimation

The results of these experiments are sumarized on the charts in Figure 7 for the results of these experiments.

In-degree. The in-degree is certainly a helpful attribute, but it requires us to consider the issues of scaling. The attributes in our baseline "author + abstract" representation have values in the range $[0, 1]$, due to normalization. If we simply add the in-degree as a new attribute, with an average value of 11.8, it will appear much more important to

Paper number	Hub value	Pages	Beginning of the abstract
9905111	39.08	261	We review the holographic correspondence between field theories and...
9710046	24.85	240	We review various aspects of classical solutions in string theories.
0110055	21.02	238	This review is devoted to strings and branes.
0210157	19.58	84	We review various aspects of configurations of intersecting branes...
0101126	19.07	56	A self-contained review is given of the matrix model of M-theory.

Table 4: The papers with the highest hub values. Incidentally, the first of these also has a large authority value (54.02); others have smaller authority values. As another indicator of the correlation between out-degree and the hub value, we observe that there are 71 papers with an out-degree of 100 or more, and all of them occur among the 82 strongest hubs.

Paper number	Auth. value	Pages	Authors, title
9711200	164.48	20	J. M. Maldacena: *The Large N Limit of Superconformal Field Theories and Supergravity*
9802150	119.89	40	E. Witten: *Anti De Sitter Space and Holography*
9802109	110.19	15	S. S. Gubser, I. R. Klebanov, A. M. Polyakov: *Gauge Theory Correlators from Non-Critical String Theory*
9407087	97.18	45	N. Seiberg, E. Witten: *Electric-Magnetic Duality, Monopole Condensation, and Confinement in $N = 2$ Supersymmetric Yang-Mills Theory*
9610043	85.46	41	T. Banks, W. Fischler, S. H. Shenker, L. Susskind: *M Theory as a Matrix Model: A Conjecture*

Table 5: The papers with the highest authority values. It's hard to relate these authority values to the contents of the papers (without asking a high-energy physicist at least). A look at the WWW suggests at least that the authors of the papers listed above have impressive careers at distinguished institutions. Maldacena, Seiberg and Witten are currently at the Institute for Advanced Study (Princeton); Gubser, Klebanov and Polyakov are at the Physics Dept., Princeton Univ.; Shenker and Susskind are at the Physics Department, Stanford Univ.; Banks is at the Santa Cruz Institute for Particle Physics; Fischler is at the Physics Dept., Univ. of Texas at Austin.

the learner than the other attributes. Fortunately, SVM manages to avoid pathological overfitting in this case, but training time increases considerably. For example, without the in-degree attribute, the "author + abstract" experiment spends about 2.9 s for SVM training (the entire experiment takes about 21 s). After adding the in-degree attribute, training time increases to 148 seconds! However, the average error has decreased from 135.7 to 128.4, which is clearly a nice improvement. (The average error on the training set has also decreased, but only very slightly: from 37.6 to 35.7. Thus we can see that no serious overfitting is occurring.)

It makes sense to try multiplying the in-degree attribute by some constant weight in order to bring it into the range $[0, 1]$ and thus prevent it from predominating over other attributes. Of course, the weight mustn't be too small, or the learner will practically ignore the new attribute. It turns out that as soon as the weight exceeds 0.003 or so, the test set error will be close to 128. The lowest error, 127.6, is achieved when the weight is 0.004. If the weight is increased further, the error slowly grows to approx. 128.4 and doesn't change much from then on. Until the weight exceeds 0.3 or so, the training time is not noticeably different than if the in-degree attribute had not been added at all; after that it grows nearly exponentially with the weight.

Out-degree. As may be expected, this attribute is not really useful. It reduces the average error on the test set from 135.7 to a little below 135, depending on the weight of this new attribute. The lowest error is 134.7 (at weight 0.1). The training set error is reduced from 37.6 to approx 37.1. As with the in-degree, the training time begins to grow quickly if the weight increases beyond a certain point. In short, this attribute is quite useless and we decided not to use it in further experiments.

Hub value. While computing the hub and authority values, we multiply the values, after each iteration of the HITS algorithm, by a constant such that the average of all hub weights (over all 29014 papers) will equal 1, and similarly for the author-

	in-deg.	auth.	PR	out-deg.	hub	PK	# downl.
in-degree	**1.00**	**0.97**	**0.90**	*0.15*	*0.04*	*−0.08*	0.24
authority weight	**0.97**	**1.00**	**0.92**	*0.13*	*0.07*	*−0.06*	0.26
PageRank	**0.90**	**0.92**	**1.00**	*0.02*	*−0.02*	*−0.12*	0.21
out-degree	*0.15*	*0.13*	*0.02*	**1.00**	**0.92**	**0.78**	*0.15*
hub weight	*0.04*	*0.07*	*−0.02*	**0.92**	**1.00**	**0.86**	*0.15*
PageKnar	*−0.08*	*−0.06*	*−0.12*	**0.78**	**0.86**	**1.00**	*0.06*
no. of downloads	0.24	0.26	0.21	*0.15*	*0.15*	*0.06*	**1.00**

Table 6: Correlation coefficients between various measures obtained from the citation graph. Very strong correlations are shown in bold type, and very weak ones are in italics.

	Average error on the		
Representation	training set	test set	Comment
AA	37.623	135.695	baseline result from Section 3.1
AA + 0.004 in-degree	36.022	**127.624**	best result in this table
AA + 0.055 authority	36.120	128.041	almost as good as in-degree
AA + 0.2 PageRank	36.765	130.503	slightly worse
AA + 0.1 out-degree	37.143	134.688	useless
AA + 0.09 hub	37.113	134.965	useless
AA + 0 PageKnar	37.623	135.695	useless, even slightly harmful

Table 7: This table summarizes the results of Section 4.3. The smallest error achieved by each of the six attributes considered here is shown. In-degree was the most successful; authority was almost equally good; PageRank was slightly worse; the remaining three (out-degree, hub and PageKnar) are useless.

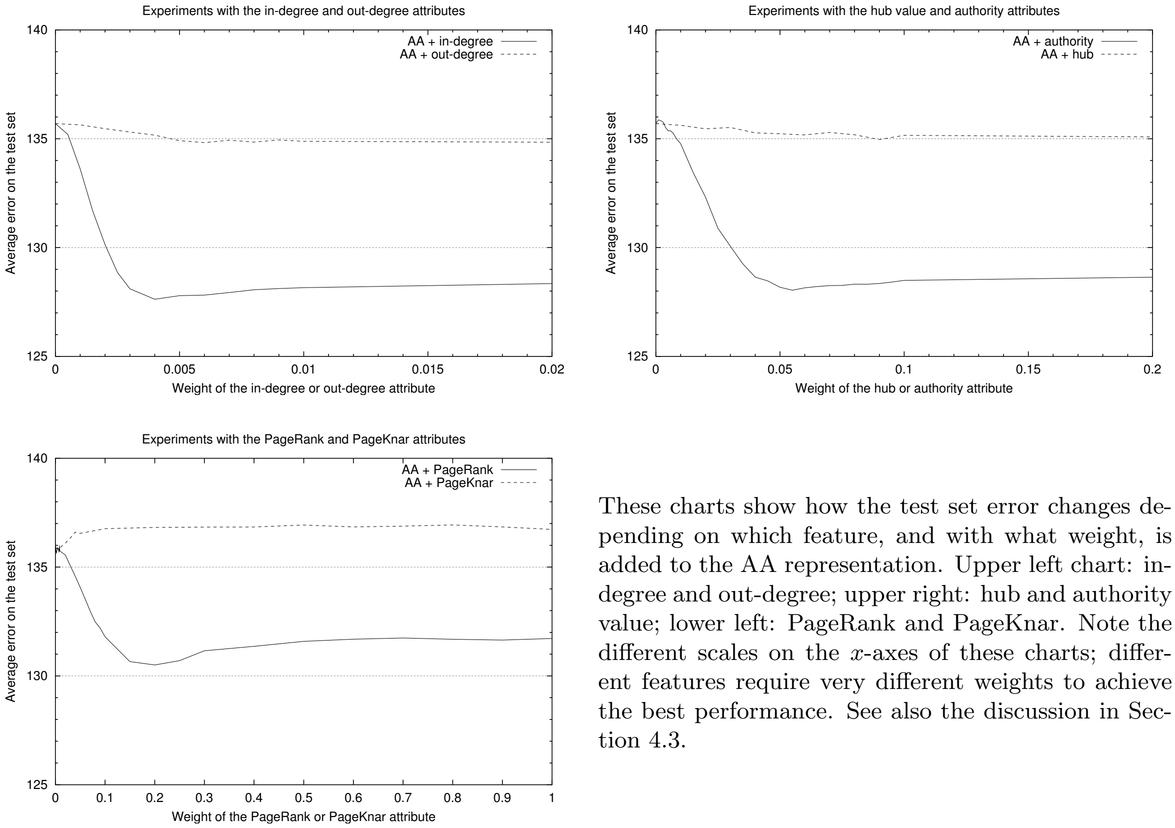
ity value. Thus, the hub and authority attributes already have rather small values and if any further weighting is to be used, it will probably be with weights greater than 1, to prevent these two attributes from being neglected.

As expected given its close relationship to the out-degree, the hub attribute turns out to be quite useless. As the weight of this attribute grows, average error quickly decreases from its original value of 135.7 towards approximately 135.0, and doesn't change much thereafter. Training time begins to grow noticeably as soon as the weight of the hub attribute is increased above the original value of 1; by the time it reaches 10, training time increases to nearly 50 seconds. Thus, we did not pursue this experiment any further.

Authority value. The effects of introducing this attribute into our representation are very similar to those of introducing the in-degree (not surprising given their strong correlation). As before with the other attributes, multiplying the the authority value by some constant weight can affect the average error and the training time. For weights above 0.04, the average error is between 128 and 129. The lowest error we found was 128.0 at weight 0.055; if the weight increases further, error increases a little and then remains around 128.8 or 128.9. Training time begins to grow noticeably if the weight is increased above 1.

PageRank. This feature is not as useful as in-degree or the authority value, but it is not quite useless either. By default, normalization is applied during the computation of PageRank in the same way as mentioned above for hubs and authorities, i. e. to make sure that the average PageRank is 1. Adding the PageRank attribute decreases the average error to 131.7. Multiplying all the PageRank values by a constant weight can improve this a little; the best result we found was at weight 0.2, where the average error is 130.5. Weights larger than 1 are not helpful, though they do not harm performance either, and training time increases very slowly.

PageKnar. We have seen (Section 4.2) that the correlation coefficient between PageKnar and the



These charts show how the test set error changes depending on which feature, and with what weight, is added to the AA representation. Upper left chart: in-degree and out-degree; upper right: hub and authority value; lower left: PageRank and PageKnar. Note the different scales on the x-axes of these charts; different features require very different weights to achieve the best performance. See also the discussion in Section 4.3.

Figure 7: Experiments with various attributes based on the citation graph.

number of downloads is very close to 0. Thus it is not surprising that, if PageKnar is added as a new attribute to our representation, it is basically ignored by the learner. Insofar as it is not ignored, it is in fact harmful. As the weight of PageKnar increases, the average error over the test set increases from the original value of 135.7 to approximately 136.8, while the average error on the training set increases by a minuscule amount, from 37.62 to approximately 37.68. If the weight of PageKnar grows above 1, the training time begins to grow as well, and faster than in the case of PageRank. In short, this attribute is completely useless.

Combinations of several attributes. We could also use several of the above-mentioned attributes at the same time. Combining in-degree, authority value, and PageRank did not result in any improvements over the smallest error achieved by in-degree alone (127.6). This is not really surprising, as these three attributes are closely correlated and give basically the same information about a paper.

We also tried combining in-degree and out-degree (the latter being the most successful of the unsuccesful features considered in this section). This is, in fact, not completely useless, because after all in-degree and out-degree describe two different aspects of the paper. However, since out-degree is not very successful by itself, it does not help very much in combination with in-degree either. The average error of “AA + 0.004 in-degree + 0.08 out-degree” is 127.075, i. e. slightly better than that of in-degree itself. However, since this improvement is so small, we decided to use the simpler representation “AA + 0.004 in-degree” as the baseline for subsequent experiment.

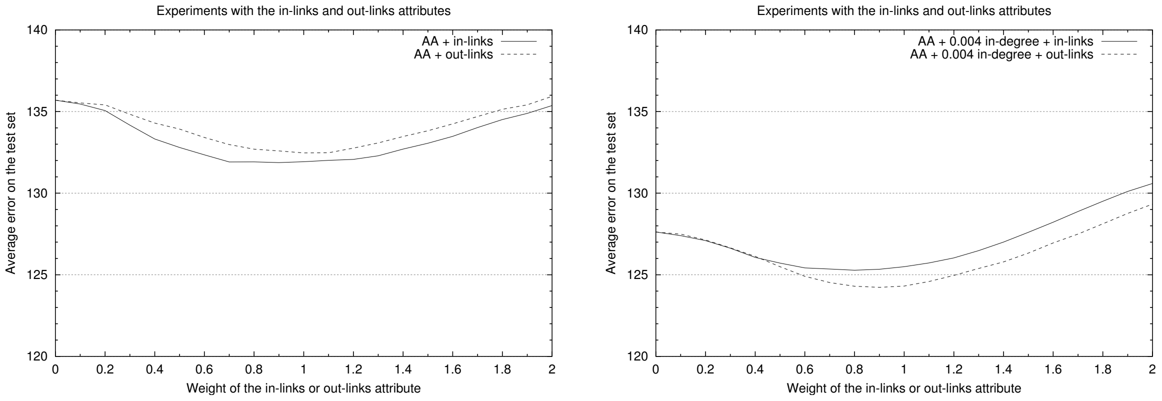


Figure 8: Experiments with the in-links and out-links attributes (see Section 4.4). These charts show how the average test set error varies depending on which group of features is used and with what weight. In the left chart, in-links and out-links were added to the basic “author + abstract” (AA) representation; in the right chart, they were added to “AA + 0.004 in-degree’.

4.4 Using the connectivity information directly

Another way of using the citation graph is to introduce as many new features as there are documents. Then, in the vector $\mathbf{x}$ representing a document d, let the component x_i equal $1/\sqrt{InDeg(d)}$ if the i-th document references document d, and zero otherwise. This gives us a very sparse vector $\mathbf{x}$, normalized to unit length. We refer to this representation as **in-links**. The results of these experiments are sumarized in Figure 8 and Table 4.4.

Adding in-links to the basic “author + abstract” (AA) representation decreases the test set error from 135.70 to 131.93 (training set error decreases from 37.62 to 30.19). Weighting the in-links attributes is not particularly helpful; the best weight is 0.9, with an error of 131.87 instead of 131.93. If we increase the weight above 1, overfitting eventually occurs; for example, at weight = 2, the training set error is only 23.43 but the test set error is 135.37. For large enough weights, training set error decreases to approximately 20 and the test set error increases to approximately 150.

Of course, rather than adding in-links to the plain AA representation, it is better to add them to the best representation so far, i. e. to “AA + 0.004 in-degree”, found in Section 4.3. In this case, the best weight is 0.8, which brings the test set error to 125.28 (down from 127.62). In fact, the default weight of 1 is not much worse (125.50). As before, increasing the weight beyond 1 leads into overfitting and larger errors on the test set.

Analogously to in-links, we can define **out-links**, with a new set of as many features as there are documents, and x_i set to $1/\sqrt{OutDeg(d)}$ if d references the i-th documents, and to zero otherwise.

Out-links are slightly less helpful than in-links. Adding them to AA reduces the error from 135.70 to 132.47; multiplying the out-links attributes by a weight does not improve this in any way. By increasing the weight of out-links, overfitting occurs, just as in the case of in-links.

Interestingly, adding out-links to the “AA + 0.004 in-degree” representation from Section 4.3 works better than adding in-links. This reduces the test-error to 124.31 (compared to 125.28 when using in-links instead, and to 127.62 when using neither).

The natural next step is to try adding **both,** i. e. to have a representation of the form “AA + 0.004 in-degree + in-links + out-links”. However, it takes some tuning of the weights to achieve errors smaller than 124.31 (which is what can be done by using out-links alone), and even then the improvement is not particularly impressive. The best combination we managed to find is “AA + 0.004 in-degree + 0.4 in-links + 0.7 out-links”, with an average error of 123.79. By increasing the weight of in-degree, one can also get “AA + 0.005 in-degree + 0.5 in-links + 0.8 out-links”, with an average error of 123.72.

		Normalization type		
Representation		Euclidean	Manhattan	None
AA	135.70			
AA + in-links		131.87	135.21	128.16
AA + out-links		132.47	135.34	132.34
AA + in-degree	127.62			
AA + in-degree + in-links		125.28	127.26	124.13
AA + in-degree + out-links		124.23	126.94	125.25
AA + in-degree + both		**123.72**	126.74	123.72

Table 8: This table summarizes the results of Section 4.4. It shows the smallest error that can be achieved by each representation (by selecting a suitable weight for the in-links or out-links attributes). For comparison, the errors achieved without these attributes are also shown.

Importance of normalization. As the formulas above show, the values of the in-link and out-link attributes have so far been normalized so that the Euclidean norm of the corresponding part of the vector is 1. (Weighting may then be applied by multiplying all the components in that part of the vector by some constant value.) Surprisingly, if Manhattan norm is used instead of the Euclidean norm (i. e. the nonzero components of in-links are $1/\mathit{InDeg}(d)$ instead of $1/\sqrt{\mathit{InDeg}(d)}$, and similarly for out-links), these attributes become completely useless. For example, even after trying a wide range of weights, the average error of "AA + in-links" never falls outside the range $[135.4, 136]$, until the weight becomes large enough (around 4) and overfitting begins to occur (the training error can eventually decrease to 20, while the test error increases to 180). Representations that include in-degree, or that use out-links instead of in-links, similarly show no benefit from the in/out-link attributes.

Another possibility is to avoid normalization altogether; that is, the nonzero components of the in-links and out-links attributes are all equal to some constant weight which is the same for all papers, regardless of their in- or out-degree. It turns out that this constant value should be around 0.1 or 0.15 to achieve the best results.[12] This works very nicely for "AA + 0.15 in-links", which has an average error of 128.16 (much better than 131.93, which we achieved with Euclidean normalization). However, what we are really interested in is improving "AA + in-degree", and here the non-normalized representation is not any better than the one based on Euclidean normalization. The smallest error found here is 124.13 for "AA + 0.004 in-degree + 0.1 in-links". Using both in-links and out-links, we find that "AA + 0.004 in-degree + 0.1 in-links + 0.12 out-links" has an average error of 123.78, and "AA + 0.004 in-degree + 0.1 in-links + 0.12 out-links" has an average error of 123.73. Thus, in the end, we are not any better off than with Euclidean normalization.

As an alternative to these normalization schemes, we also tried having the nonzero values of the in-link attributes correspond to the HITS authority value of the paper from which the in-link originates. However, after preliminary experiments which suggested that this would not lead to smaller average errors than the simpler normalization schemes mentioned above, we decided not to experiment with this any further.

Attempts to use PCA. The in-links attributes contain 29014 attributes, one for each paper in the dataset. For each of the training and test paper, the in-links part of its vector is a sparse 29014-dimensional vector in its own right. We tried running PCA on these vectors; let $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(k)}$ be the k principal components. Since these are also 29014-dimensional vectors, there corresponds, to each document d_i, a k-tuple $(z_i^{(1)}, \dots, z_i^{(k)})$, which we can use as the values of k additional features in the vector that represents the document d_i. Alternatively, we can use, as k new features, the projections of the in-links vector of document i onto

[12] This can, in fact, also be seen as a kind of Manhattan normalization. Multiplying by 0.1 is equivalent to dividing by 10, and 10 is approximately the average in-degree (or average out-degree, which is of course the same thing) over all the papers. Thus, the in-links (or out-links) parts of the vectors may now have different Manhattan norms, but the average norm will be close to 1.

		Statistics for the number of downloads					
	Total	on all 1566 training papers			on 50 top papers per month		
Journal	# papers	#	Avg. ± st. dev.	Med.	#	Avg. ± st. dev.	Med.
(missing)	8052	375	194.3 ± 203.8	130	62	553.7 ± 274.7	481
JHEP	1947	243	248.0 ± 177.9	200	74	442.9 ± 203.0	387
Phys. Rev.	3261	204	201.4 ± 165.3	155	47	421.3 ± 205.6	353
Phys. Lett.	3995	203	162.3 ± 115.5	132	28	386.9 ± 115.4	358
Nucl. Phys.	3850	190	186.0 ± 131.8	160	35	373.5 ± 189.2	340
(all papers)	29014	1566	193.6 ± 185.9	142	300	462.3 ± 271.4	379

Table 9: This table shows the number of papers and the average (with standard deviation) and median number of downloads for five most commonly occurring journals (including "missing"). These statistics are shown both for the entire training set of 1566 papers for which download counts were available, and for the set obtained by taking the 50 most frequently downloaded papers from each of the 6 training months (for a total of 300 papers). The total number of papers that refer to each journal (over the entire dataset of 29014 papers) is also shown. The journals shown here account for about 77 % of the 1566 papers of the training period. The next most common journal after them has only 41 papers in the training period. The total number of journals mentioned at least once in the entire dataset is approximately 300 (the exact number of real journals is not completely obvious because some of the abstract files use the "Journal" field to refer to conference proceedings), but only about 60 of them have at least one paper in the training period.

all the k principal components. We tried $k = 10$ and $k = 50$, as well as varying the weight of these features, but didn't succeed in really reducing the error (the lowest error achieved in this way was 123.20).

5 Journal information

As mentioned in Section 1.3, approximately 72 % of the papers have a "Journal" field in their abstract file, usually stating the journal where the paper was or will be published. (In a very few cases, this field refers to conference proceedings.)

The argument for using this information is similar as in the case of citation data: although the readers do not see the journal name when downloading the paper (and, indeed, if the paper has only been in the archive for 60 days or less, it probably hasn't really been published in a journal — more likely it was just submitted, and perhaps accepted for publication), the fact that a paper has appeared in a reputable journal suggests that it might be an interesting and relevant one, characteristics which, assuming that the readers are able to recognize them from the abstract and title, would certainly encourage a larger number of downloads.

The training data certainly shows that papers from some journals are downloaded more frequently than those from others, although the variance within each journal is still quite large (see Table 9).

5.1 Binary journal membership attributes

We will introduce one feature for each journal (papers without a "Journal" field are taken to have appeared in a journal called "missing"). In the vector $\mathbf{x}$ representing a document d, the component x_j will be nonzero if and only if d has appeared in the journal j. Note that the "Journal" field, if present, mentions exactly one journal; this, coupled with the introduction of the additional feature for papers without a journal, means that exactly one of these components will be nonzero in each vector $\mathbf{x}$. Thus there is no further need for normalization, but there may be a need for weighting. These experiments are summarized on the left chart of Figure 9.

Experiments show that the journal attribute is not terribly useful, but it does decrease the error a little. Weighting is important because if the nonzero values in the journal attributes are too large, the learner will overfit. When the journal attributes are added to the basic "author + abstract" (AA) representation, the smallest error is achieved if the weight of the journal attributes is around 0.2; the error is then achieved is 134.9. The improvement in comparison to 135.7, which is the error without the journal attribute, is really very

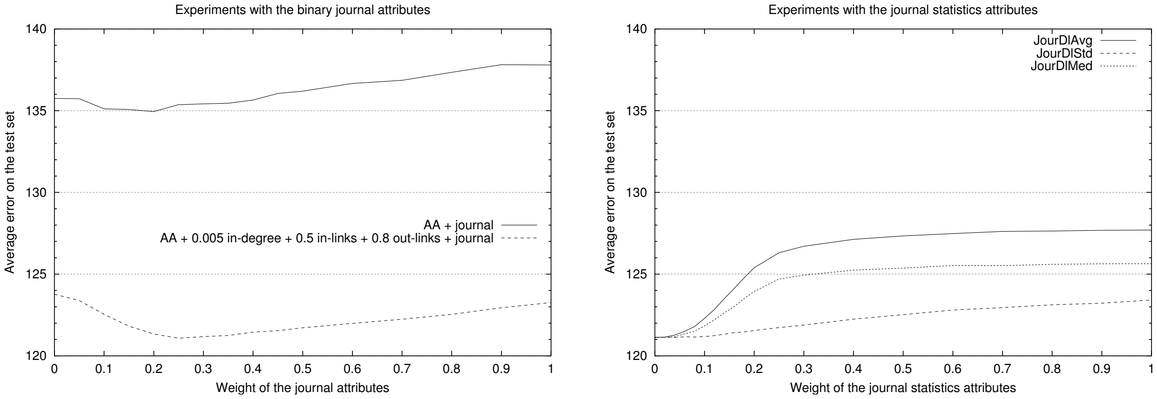


Figure 9: Experiments with the journal-related attributes.

The left chart shows the influence of the binary journal attributes, depending on their weight. We considered adding them both to the basic "author + abstract" representation and to the best representation from Section 4.4.

The right chart shows the influence of the journal statistic attributes. These are statistical characteristics of the number of downloads over all the training papers from the same journal. These attributes were added one at a time to the best representation from the left chart. See Section 5.2 for a precise definition of these features.

small.

However, it is more interesting to try adding the journal attributes to one of the best representations from the previous section, e. g. to "AA + 0.005 in-degree + 0.5 in-links + 0.8 out-links". Interestingly, the journal attributes make a larger contribution now: the error decreases from 123.72 to 121.08. This smallest error is achieved if the weight of the journal attributes is 0.25. For weights larger than 1, overfitting occurs; training time begins to grow, training set error decreases from about 28 to about 23, and the test set error increases to approximately 130.

5.2 Journal statistics

Another way of using the journal information may be to compute, for each journal j, the average number of downloads over all the training papers which belong to this journal. We denote this by $\mathit{JourDlAvg}(j)$. We can then use this as an additional feature in each document. Let $J(d)$ be the journal to which the document d belongs; then in the vector $\mathbf{x}$ representing d, the component $x_{\mathit{JourDlAvg}}$ equals $\mathit{JourDlAvg}(J(d))$. We can also use the standard deviation (→ *JourDlStd*) or the median (→ *JourDlMed*) instead of the average. The hope is that, since this is a regression model, values that actually represent some number of downloads (rather than, say, the TF-IDF weight of some word) might be useful for predicting the target value, which is also a number of downloads.

However, experiments show that these new features aren't useful. *JourDlStd* is practically ignored by the learner; adding it to "AA + 0.005 in-degree + 0.5 in-links + 0.8 out-links" decreases the training set error from 29.9 to 29.7 and increases the test set error from 123.7 to 125.7. The other two attributes, *JourDlAvg* and *JourDlMed*, have a very similar effect; both lead into serious overfitting, reducing the training set error to around 25 and increasing the test set error to around 131. Weighting doesn't help either; weights below 1 just reduce overfitting, but do not decrease the test set error below that achieved when these attributes are not used at all.

Adding these features to the best representation from Section 5.1, "AA + 0.005 in-degree + 0.5 in-links + 0.8 out-links + 0.25 journal", is not successful either. This group of experiments is illustrated on the second chart of Figure 9. All three attributes cause overfitting and increase the test set error, though not all equally badly. *JourDlAvg* is the most harmful, *JourDlMed* somewhat less, and *JourDlStd* still less.

ID	Download count	Title	Title length (in characters)
0203101	2927	The holographic principle	25
0204027	1540	Twenty Years of Debate with Stephen	35
0003052	1351	Brane New World	15
0204051	1351	A semi-classical limit of the gauge/string correspondence	57
0204131	1351	A tentative theory of large distance physics	44
0104005	1343	(De)Constructing Dimensions	27
0204035	1308	Lectures on supergravity	24
0003006	1246	A Short Survey of Noncommutative Geometry	41
0003004	1193	The Holographic Principle	25
0204143	1180	Field Theory of Tachyon Matter	30

Table 10: The ten most frequently downloaded papers from the entire training period. Notice how their titles are relatively short (average: 32.3 characters). If we take the 50 most frequently downloaded papers of each training month and compute the average length of the title over these 300 papers, the result is approximately 51.6. The average over all 1566 papers from the training period is even larger: 59.0.

ID	Download count	Title	Title length (in characters)
0204069	$896^\star$	Large N	7
0204062	217	Supercurves	11
0204089	$746^\star$	Open Strings	12
0104010	$559^\star$	D(NA)-Branes	12
0104221	98	Mind The Gap	12
0203265	$1018^\star$	Tachyon Matter	14
0203073	87	RSOS revisited	14
0003052	$1351^\star$	Brane New World	15
0203211	$985^\star$	Rolling Tachyon	15
0002222	$767^\star$	Mirror Symmetry	15

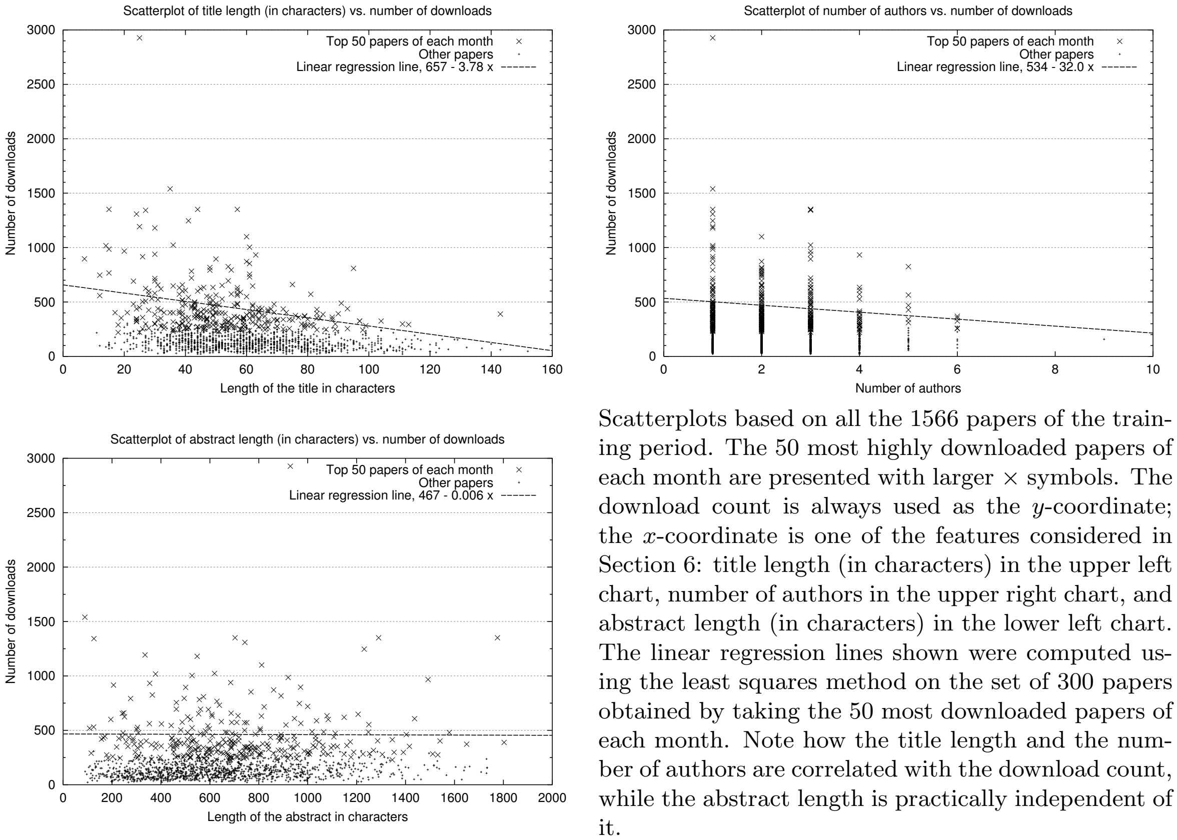
Table 11: The ten papers with the shortest titles. Those marked * (seven out of these ten papers) are also among the 50 most frequently downloaded papers among all the papers that were added in the archive during the same calendar month.

6 Miscellaneous statistics

The attributes considered in this section are somewhat far-fetched, though one of them turns out to be useful anyway. The baseline for all experiments in this section is the "AA + 0.005 in-degree + 0.5 in-links + 0.8 out-links + 0.25 journal" representation, with an average error of 29.4 on the training set and 121.1 on the test set. The results of these experiments are summarized in Table 14 and Figure 11.

Title and abstract length. We looked at a few of the most frequently downloaded papers, hoping to see what sets them apart from the others and how they may be recognized (see Table 10). One particularly conspicuous characterstic is that they have relatively short titles. (The converse is also close to the truth — of the papers with the shortest titles, many have a high number of downloads; see Table 11.) Perhaps there is something about these bold, plain, self-confident titles that attracts the curiosity of readers and encourages more downloads. On the other hand, long titles of many less frequently downloaded papers perhaps imply that the paper is a tiny investigation of some extremely narrowly limited topic, and is thus less likely to attract as many readers. Surely "The holographic principle" sounds much more attractive than "A condition on the chiral symmetry breaking solution of the Dyson-Schwinger equation in three-dimensional QED" (paper 0102205, 57 downloads).

This observation suggests that the **length of the title** might be a useful feature for download esti-



Scatterplots based on all the 1566 papers of the training period. The 50 most highly downloaded papers of each month are presented with larger × symbols. The download count is always used as the y-coordinate; the x-coordinate is one of the features considered in Section 6: title length (in characters) in the upper left chart, number of authors in the upper right chart, and abstract length (in characters) in the lower left chart. The linear regression lines shown were computed using the least squares method on the set of 300 papers obtained by taking the 50 most downloaded papers of each month. Note how the title length and the number of authors are correlated with the download count, while the abstract length is practically independent of it.

Figure 10: Scatterplots showing how three of the attributes from Section 6 relate to the download count.

mation. We considered counting the length both in characters (which includes spaces and punctuation symbols) and in words. Similarly, we tried using the **length of the abstract** (in characters and in words) as a feature.

The title length in characters is the most useful of these features. It must, of course, be weighted carefully to prevent its large values from overshadowing everything else in the document vectors. The smallest test set error (119.59) is achieved when this attribute is multiplied by 0.004. The coefficient corresponding to this attribute in the resulting linear model suggests that lengthening the title by one character reduces the number of downloads by 0.948. If the weight of this attribute is increased, a little overfitting occurs, but not much. The test set error eventually stabilizes around approx. 120.7.

Interestingly, the title length in words is less helpful than the title length in characters. Perhaps it is not just the number of words but also the length of the words that influences the users who are trying to decide whether to download a given paper (perhaps long words sound too technical or too boring and discourage them?). The smallest error achieved by this attribute is approx. 120.35 (at weight 0.03). Larger weights eventually lead to mild overfitting again; the error eventually stabilizes around 121.6. Using both the title length in characters and the title length in words at the same time does not reduce the error any further.

The observation in the preceding paragraph suggests that the **average length of title words** might also be a useful attribute. It turns out that, by itself, it reduces the test set error from 121.1 to 120.5, and when used in addition to the title length in characters, it reduces the error from 119.6 to 118.9. This attribute is not much affected by weighting, except if the weight is extremely small, in which case the result is as if the attribute had not been present at all.

The abstract length, regardless of whether it is expressed in characters or in words, turns out to be completely useless. The error, on the training as well as on the test set, remains practically unchanged regardless of how one alters the weight of these two attributes. However, if the weight is too large, the SVM training time begins to grow (approximately with the square of the weight). Apparently, although there is considerable variance in the length of abstracts from one paper to another, these variations are not correlated with the download count in any particular way. (This is also confirmed by the fact that the correlation coefficient between the abstract length and the number of downloads is close to zero; see Table 13.)

Number of authors. As Table 12 shows, there is a slight tendency for papers with a larger number of authors to be downloaded less frequently; or, if we are looking at a set containing only the most frequently downloaded papers, the relative amount of papers with a larger number of authors will be smaller than in a set of papers with a "natural" distribution of download counts. Thus, it might be interesting to add the number of authors as an additional feature.

However, apparently the correlation between the number of authors and the number of downloads is too weak for this attribute to be of any use. Weighting doesn't help either; the average error, on the training se as well as on the test set, hardly changes, and insofar as it does, it becomes slightly larger.

A related attribute with which we also experimented is the average number of downloads over all the training papers with the same number of authors as the current paper. This isn't helpful either. It causes the test set error to increase slightly, from 121.1 to 121.6; weighting this attribute can make this increase larger or smaller but doesn't change the fact that performance is slowly deteriorating.

Year of inclusion in the arXiv. The feature will actually be year − 2000, so that its value will be in the set $\{0, 1, 2\}$. As we have already seen in Section 2.5, the number of downloads in 2001 was slightly smaller than in the other two years, so an attribute telling the year of publication could perhaps be helpful. However, since the average number of downloads in 2000 and 2002 is greater than in 2001, it could be difficult to model this by a linear predictor of the form $\mathit{downloads}(d) = \mathit{const} \cdot (\mathit{year}(d) - 2000) + (\mathit{other\ terms})$. Therefore, we also introduce three binary features, one for each year (2000, 2001, and 2002), telling whether the current paper was added to the archive in that year or not.

Both of these approaches lead to a very small decrease in the test set error. The year − 2000 attribute, when multiplied by the constant weight 0.18, reduces the error from 121.14 to 120.75; larger weights increase the error again to around 121.0. Similarly, the three binary attributes (one per year) achieve an average error of 120.93 if multiplied by the weight 0.1. Combining both year − 2000 and the three binary attributes resulted in no further improvement of accuracy.

Combinations of several features. We only tried combining the title-related features and the year, because the other features presented in this section have been found so unpromising. The smallest error we found was 118.77, achieved by a suitably weighted combination of title length (in characters), average title word length, and year − 2000. This is not much of an improvement from 118.90, which was achieved by combining just the title length and the average title word length.

Summary. Some statistical properties of the features considered in this section are shown in Table 13. The best performance achieved by each of these features, as well as by some combinations of several features, is shown in Table 14. The charts in Figure 11 show how the average error changes with the weight of these features.

7 Clustering

We tried clustering the hep-th papers and using information about cluster membership and cluster statistics as additional features, in the hope that they will be useful in the download estimation task. As it turns out, none of the features considered in this section were really helpful.

	Total no. of	Statistics for the			
	papers with	top 50 papers per month		entire training period	
n	n authors	Count	Avg. downloads	Count	Avg. downloads
1	11313	541	197.6	94	520.1
2	10039	563	175.3	90	437.3
3	5339	304	219.3	81	453.0
4	1814	120	188.3	24	385.8
5	354	26	217.4	6	489.3
6	120	11	231.6	5	312.4
7	28				
8	4				
9	2	1	157.0		
10	1				
Total	29014	1566	193.6	300	462.3

Table 12: For each number n of authors, this table shows the number of papers with n authors, as well as their average download count. The rightmost two columns are based on taking the 50 most frequently downloaded papers from each of the 6 training months, for a total of 300 papers.

	Statistics: on all training papers,			on top 50 papers per month		
Feature	Avg. ± st. dev.	Med.	Correl.	Avg. ± st. dev.	Med.	Correl.
Characters in title	59.0 ± 22.0	57	−0.240	51.6 ± 20.1	49.5	−0.280
Words in title	8.00 ± 3.05	8	−0.207	7.08 ± 2.89	7	−0.227
Avg. title word length	7.52 ± 1.36	7.4	−0.052	7.49 ± 1.56	7.32	−0.084
Characters in abstract	657.7 ± 313.2	621.5	0.095	711.6 ± 324.5	682.5	−0.007
Words in abstract	99.4 ± 47.9	94	0.097	107.4 ± 48.9	102.5	−0.001
Number of authors	2.08 ± 1.06	2	0.024	2.24 ± 1.14	2	−0.134
Year − 2000	0.99 ± 0.83	1	0.126	1.00 ± 0.82	1	0.154
Number of downloads	193.6 ± 185.9	142	1.000	462.33 ± 271.8	379	1.000

Table 13: Some statistical properties of the features considered in Section 6. The left part of the table shows the statistics for the entire training set (of 1566 papers), and the right part shows statistics computed on the set of 300 papers obtained by taking the 50 most frequently downloaded papers from each of the six months for which training data is available. The "Correl." columns contain correlation coefficients between these features and the number of downloads.

	Correlation with	Best performance	
Feature	download count	achieved at weight	Test set error
Characters in title	−0.280	0.004	119.59
Words in title	−0.227	0.03	120.36
Avg. title word length	−0.084	1.1	120.55
Characters in abstract	−0.007	0	121.14
Words in abstract	−0.001	0	121.14
Number of authors	−0.134	0	121.14
Year − 2000	0.154	0.18	120.75
0.004 title chars. + 1.3 title word length			118.90
0.004 title chars. + 0.9 title word length + 0.1 (year − 2000)			118.77

Table 14: This table shows the correlation coefficients between the features considered in Section 6 and the download count. These coefficients have been computed on the set of 300 papers obtained by taking the 50 most frequently downloaded papers from each of the 6 months covered by the training period. Additionally, the table shows the best performance achieved by adding each of these features to "AA + 0.005 in-degree + 0.5 in-links + 0.8 out-links + 0.25 journal". Reasonably enough, features with stronger correlation to the download count achieve a larger reduction of error. The last two rows show the best performance of certain combinations of two or three features.

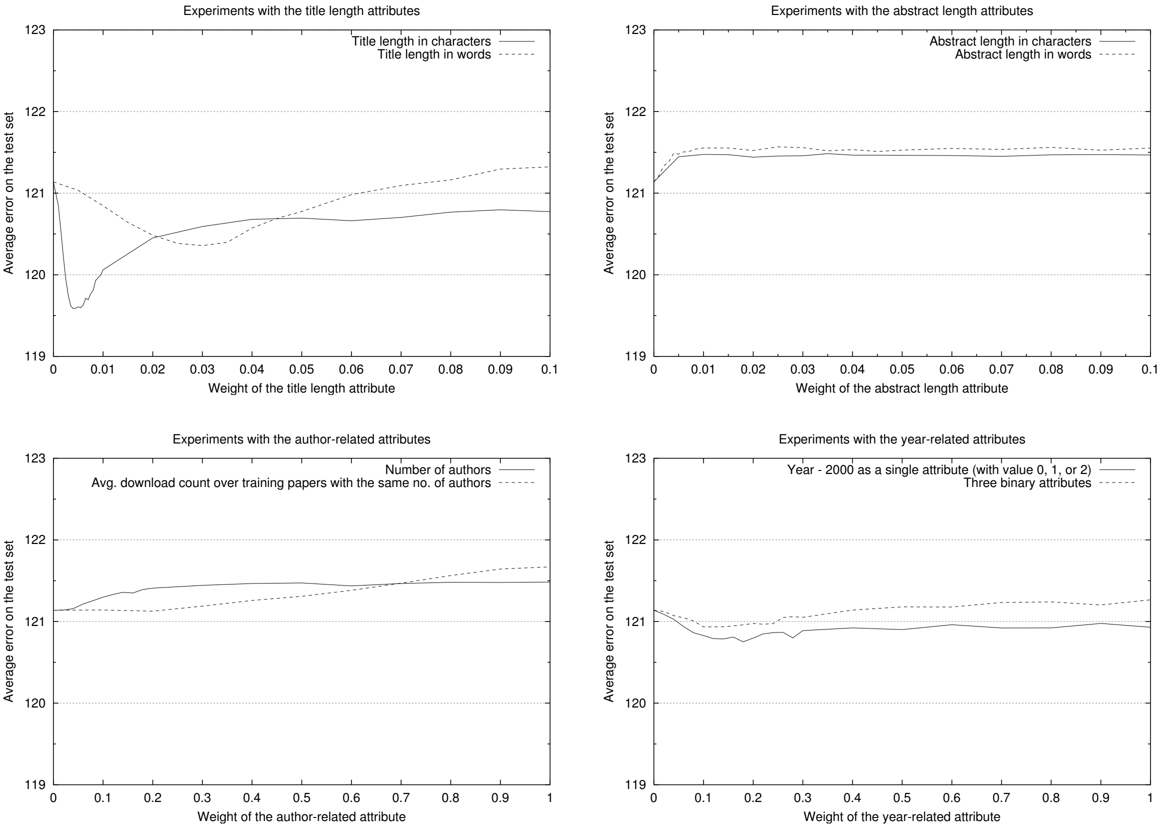


Figure 11: These charts show how the average error changes if one of the attributes considered in Section 6 is added to the "AA + 0.005 in-degree + 0.5 in-links + 0.8 out-links + 0.25 journal" representation. Top left: title length; top right: abstract length; bottom left: number of authors and the average download count over all training papers with the same number of authors; bottom right: year of inclusion in the archive, either as one three-valued or as three binary attributes.

7.1 The algorithm

To cluster the papers, we used a simple top-down approach: begin by separating the papers into two clusters using 2-means (i. e. k-means [DHS00] for $k = 2$), then call the same algorithm recursively on each of the two resulting clusters. We start with the entire set of 29014 papers; the recursive splitting stops if one of the resulting clusters would have less than 600 papers (in this case, the split is aborted).

Each paper was represented by a normalized TF-IDF vector (see Section 2.1) based on the text of the abstract and the title of the paper. The cosine measure is used as a similarity measure between these vectors: the cosine of the angle between $\mathbf{x}$ and $\mathbf{y}$ is $(\mathbf{x}^T\mathbf{y})/(||\mathbf{x}||\,||\mathbf{y}||)$; if the vectors are normalized, this simplifies to the dot product $\mathbf{x}^T\mathbf{y}$. A larger cosine means a smaller angle between the vectors, and thus suggests that the documents are more similar. The value $1 - \cos$ can be used as a distance.

The 2-means algorithm starts by choosing several random pairs of documents and retains the one with largest distance between the two documents in the pair. These two documents are used as the "seeds", and each of the remaining documents is assigned to the closer of these two documents. Thus we obtain a partition of the document set into two clusters; the centroids of the two clusters are then used as seeds in the next iteration of 2-means. This produces a new partition of the documents into two clusters and the whole process then repeats for several more iterations. We stop after 30 iterations, or sooner if less than 5 documents have switched to a different cluster in the last iteration.

The results of this clustering algorithm are shown on Figure 12. This shows the sizes of the clusters, their compactness, as well as download count statistics for each cluster. In addition, some of the larger "super-clusters" are presented with a few keywords as well. Since we are not familiar with the subject matter of the papers in our dataset, it is hard to say whether the clusters are meaningful, though a look at these keywords may suggest that the papers in each super-cluster do have some sort of common theme that sets them apart from the other papers.

7.2 Features based on clustering

An obvious way of using the partition of documents into clusters is to introduce one **binary feature** for each cluster. In the vector representing a particular document, each of these features will be nonzero if and only if the document belongs to that cluster. However, experiments show that these features lead into a curious kind of overfitting: the average training set error doesn't decrease much (from 29.2 to 28.5), but the test set error increases considerably (from 118.9 to over 125, depending on the weight of the clustering features).

By terminating the recursive clustering sooner, one can reduce the number of clusters. In this way we obtained 8 "super-clusters" (denoted A–F in Figure 12), which can again be used to define a set of binary attributes. These attributes cause overfitting similar to that described in the previous paragraph, except that both the decrease in training error and the increase in test error are smaller.

As an alternative to these useless binary features, one might introduce **cluster statistics** analogous to the journal statistics described in Section 5.2. Let $\mathit{ClusDlAvg}(c)$ be the average number of downloads over all training papers from cluster c; let $C(d)$ be the cluster to which the paper d belongs; then, in the vector $\mathbf{x}$ representing d, the component $x_{\mathit{ClusDlAvg}}$ equals $\mathit{ClusDlAvg}(C(d))$. We can use standard deviations (→ *ClusDlStd* and medians (→ *ClusDlMed*) in the same way.

None of these attributes is particularly helpful. The *ClusDlMed* attribute is the most successful of them; when added to the best model from Section 6, it reduces the average test set error from 118.8 to 118.31. This attribute is not particularly influenced by weighting; any weight from 0.4 to 1.0 produces an error between 118.3 and 118.4. Larger weights, even up to 10, only increase the error to 118.44, so we can say that there is practically no overfitting.

ClusDlAvg is approximately equally successful, but is more sensitive to weighting. It achieves the smallest error, 118.31, at a weight of 0.3. Larger weights lead to larger errors, reaching 118.66 at weight 1 and 118.76 at weight 10.

The *ClusDlStd* attribute is even less useful; the smallest error it achieves is 118.66, at weight 0.15. With larger weights, the error quickly increases and eventually grows to 119.6 at weight 1 and even 119.8 at weight 10.

We also tried using both *ClusDlMed* and *ClusDlStd*, but found no combination of weights that would improve performance over that of *ClusDlMed* alone. The results of the experiments in this section are shown on the graphs in Figure 13.

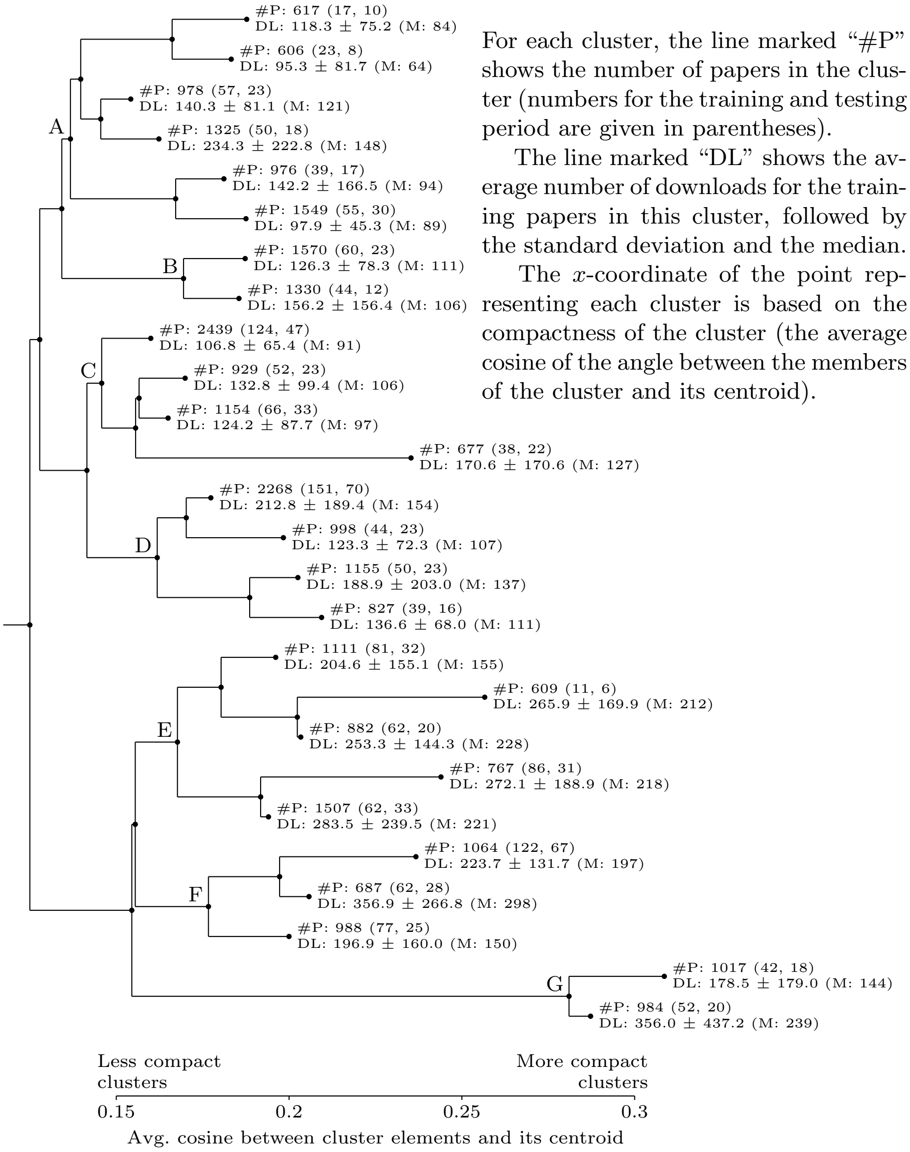
Another clustering-related feature we considered is *ClusCentDist*, the Euclidean distance from the vector that represented the document during clustering document to the centroid of the cluster to which the document belongs. Surprisingly enough, this feature turns out to be more useful than *ClusDlMed*. It is, however, more sensitive to the choice of weight. It achieves the smallest error, 117.96, when multiplied by the weight 0.35. With larger weights, error begins to grow and reaches 119.69 if the weight grows to 10. In the linear model found by the SVM learner, the component corresponding to the *ClusCentDist* attribute is always negative, i. e. the predictor works as if documents that are farther away from the centroid of their cluster are downloaded less frequently. If this is more than just coincidence, it perhaps suggests that readers dislike papers that lack a clear focus (we hope that it is not a sign of a general unpopularity of interdisciplinary research).

ClusDlMed and *ClusCentDist* can also be **combined**. A good weighting is for example "0.7 *ClusDlMed* + 0.4 *ClusCentDist*", with an average test set error of 117.23. In fact any larger weight of *ClusDlMed* works just as well; the test error doesn't grow if larger weights are used.

For each cluster, the line marked "#P" shows the number of papers in the cluster (numbers for the training and testing period are given in parentheses).

The line marked "DL" shows the average number of downloads for the training papers in this cluster, followed by the standard deviation and the median.

The x-coordinate of the point representing each cluster is based on the compactness of the cluster (the average cosine of the angle between the members of the cluster and its centroid).



	No. of elements			Avg. cosine	No. of downloads		Words with greatest weight
	Total	Training	Test	to centroid	Avg. ± st. dev.	Med.	in the centroid vector
A	6051	241	106	0.137	144.6 ± 142.5	104	quantum, models, boundary
B	2900	104	35	0.169	138.9 ± 118.8	110	algebra, algebras, lie
C	5199	280	125	0.146	124.4 ± 99.6	97	model, field, renormalization
D	5248	284	132	0.162	184.3 ± 170.6	137	gauge, theories, yang-mills
E	4876	302	122	0.168	252.2 ± 186.1	202	string, type, theory
F	2739	261	120	0.177	247.5 ± 190.9	196	brane, cosmological, solutions
G	2001	94	38	0.281	276.7 ± 357.5	189	black, hole, holes

Figure 12: A dendrogram of the partition of the dataset into clusters used in the experiments in Section 7. The table below presents some statistics of the larger clusters ("super-clusters"), marked in the dendrogram with the letters A–G.

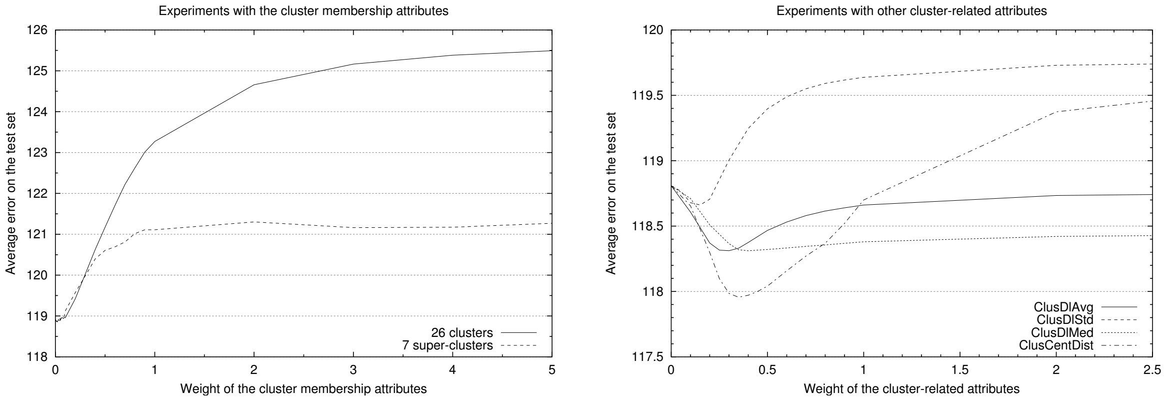


Figure 13: Experiments with the attributes based on clustering. These attributes were added to the best representation from Section 6 ("AA + 0.005 in-degree + 0.5 in-links + 0.8 out-links + 0.25 journal + 0.004 title-characters + 0.9 title-word-length + 0.1 (year − 2000)").

The left chart shows the effect of the binary cluster membership attributes, depending on the weight of these attributes and on how many clusters we use. The right chart shows the effect of the cluster statistic attributes, which are statistical properties of the number of downloads over all training papers from the same cluster. See Section 7.2 for exact definitions. Note the different scales on the y-axes.

8 Other learning algorithms

We have already described our early experiments with some really simple predictors: a constant predictor (Section 2.4) and several "regression stumps" (Section 2.5). In this section, we will describe a few additional approaches that we considered using instead of the straightforward use of regression SVM that was described in Section 2.2 and then used throughout the paper.

8.1 Nearest neighbors

We used a simple version of the nearest-neighbor method [CH67] (also known as "instance-based learning" [AKA91]). One has to define a measure of distance between papers. Then, to predict the number of downloads for some paper d, one looks at the number of downloads for the k "nearest neighbors" of d, i. e. those training papers that are closest to d under the chosen distance measure.

We used the "Author + Abstract" representation and the cosine measure as the distance measure between vectors, i. e. $d(\mathbf{x}, \hat{\mathbf{x}}) := 1 - \mathbf{x}^T\hat{\mathbf{x}}/(||\mathbf{x}||\,||\hat{\mathbf{x}}||)$. To combine the download counts of the k nearest neighbors into a prediction, we considered four functions: (1) average; (2) weighted average: if the vector of the paper whose download count we want to predict is $\mathbf{x}$, the influence of each neighbor $\hat{\mathbf{x}}$ on this prediction is proportional to the cosine of the angle between $\mathbf{x}$ and $\hat{\mathbf{x}}$, i. e. to $\mathbf{x}^T\hat{\mathbf{x}}/(||\mathbf{x}||\,||\hat{\mathbf{x}}||)$; (3) maximum; (4) median.

The results of this section are summarized in Table 15 and on the graphs in Figure 14. At $k = 1$, all three variants return the same predictions and have an average error of 180.7 on the training set and 204.2 on the test set. The maximum-based predictor is completely useless; the larger the k, the more optimistic predictions we get, and the error grows quickly. On the other hand, the other three predictors (average, weighted average, and median) benefit from a larger k; $k = 30$ is enough to get an error that is very close to the smallest error over all k.

If k is too large, the predictions of both the average- and the median-based predictors converge towards the average/median download count over the entire training set and the error therefore tends towards the error of the trivial models from Section 2.4, which is approx. 152 for the average-based and approx. 167 for the median-based model. On the other hand, the accuracy of the weighted average predictor is not much affected if k is too large, as the distant "neighbors" get very small weights and do not influence the predictions very much.

	20 % of the training set used					30 % of the training set used				
	Avg. test set error				cos to	Avg. test set error				cos to
k	Avg	WgtA	Med	Max	k-th NN	Avg	WgtA	Med	Max	k-th NN
1	213.40	213.40	213.40	213.40	0.260	204.18	204.18	204.18	204.18	0.293
5	170.10	172.48	153.23	406.18	0.075	154.83	156.18	152.15	356.90	0.090
10	174.47	170.67	148.48	589.65	0.051	148.67	147.83	149.62	487.04	0.059
20	169.76	165.03	146.77	785.79	0.035	147.68	142.00	155.72	676.64	0.042
30	167.80	162.74	147.35	909.81	0.027	146.54	139.69	157.35	794.21	0.034
all	171.90	162.16	152.15	2342.80	0.000	152.44	141.69	166.77	2342.80	0.000
best	165.87	160.28	145.16	213.40		146.12	**138.48**	144.43	204.18	
(best k)	68*	64*	8	1		39*	80*	8*	1	

Average error of SVM regression on this representation (from Section 3.1): **135.80**

*A wide range of other values of k results in comparable performance.

Table 15: Comparison of several variants of the nearest-neighbor algorithm. The "Author + Abstract" representation was used, and nearest neighbors were always chosen based on the cosine measure.

The first four columns in each group show the average test set error of the four variants of the nearest-neighbor predictor (average, weighted average, median, and maximum). The fifth column shows the average cosine between a test document and its k-th nearest neighbor. This gives one a vague idea of the distribution of documents in space. Additionally, this cosine is used as the weight by the weighted-average nearest neighbor predictor.

The row marked "all" shows results if k is set to the size of the training set (because we use cross-validation and not all the folds have exactly the same number of elements, the training set as seen by the learner is not always the same size; it has 282 or 283 papers if 20 % of the papers from the training folds are used, and 423 or 424 if 30 % are used).

The last two rows show the best performance achieved by each variant, and the k where it is achieved. In some cases (marked with an asterisk*) a wide range of other values of k results in comparable performance, meaning that the particular value of k shown here is should not be seen as extremely significant.

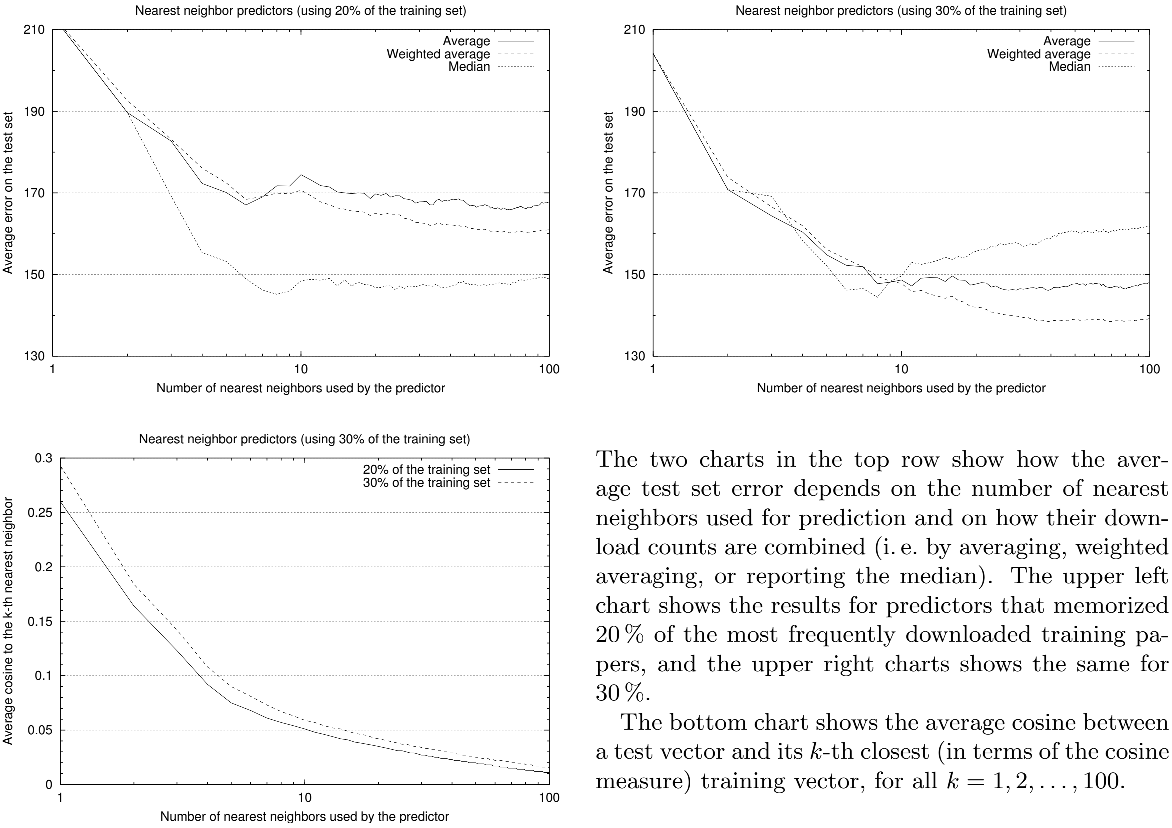
We also tried basing these predictors on just 20 % of the most frequently downloaded papers from the training folds (rather than using 30 % as usually), but this increased the error of all the predictors, except for the median-based variant (which is reasonable, for the same reasons as already discussed in the case of trivial models in Section 2.4). The smallest error of the median-based variant is now approximately the same as before, but it is now achieved over a much larger range of values of k.

In Section 3.1, we have seen that the model learned by the regression SVM on this representation ("Author + Abstract") has an average error of 135.80. On the other hand, the best nearest-neighbor model found in this section has an average error of 138.48. Thus, we see that k-NN is doing remarkably well on this combination of dataset and representation. Perhaps its performance could be improved further by using some different distance function to choose the nearest neighbors and by using a different weighting for the weighted average predictor.

It would be interesting to see how much the performance of nearest-neighbor methods can be improved by adding more features to our representation of papers. However, we did not carry out any systematic experiments in this direction. It is certainly the case that a representation that works well for SVM does not necessarily work particularly well for nearest neighbors; we tried running k-NN on the best representation from Section 7.2 (where SVM has an average test set error of only 117.2), and the smallest error achieved was 142.4. Probably some of the features had too large values and therefore distorted the optics of the cosine measure. In short, the nearest neighbor method has its own specific view of the data and would probably again require careful selection and tuning of weights (and perhaps distance measures as well) to achieve its best performance.

8.2 Two-level models

Consider the 50 most frequently downloaded papers from each of the six months of our training period. Of these 300 papers, 14 have 1000 or more



The two charts in the top row show how the average test set error depends on the number of nearest neighbors used for prediction and on how their download counts are combined (i. e. by averaging, weighted averaging, or reporting the median). The upper left chart shows the results for predictors that memorized 20 % of the most frequently downloaded training papers, and the upper right charts shows the same for 30 %.

The bottom chart shows the average cosine between a test vector and its k-th closest (in terms of the cosine measure) training vector, for all $k = 1, 2, \ldots, 100$.

Figure 14: Charts showing the performance of the nearest neighbor predictors, and the average distance to the k-th nearest neighbor. See Section 8.1 for more details. Note the logarithmic scale on the x-axis of all the charts, which we use to show the results for smaller k more clearly.

downloads; 123 have from 400 to 1000 downloads; 163 have less than 400 downloads. If we could accurately predict into which of these three groups a paper belongs, and then use the median download count of that group as our prediction, the average error would be just 81.9, a fabulous result in the light of the poor performance of all the predictors considered so far.

This is a simple kind of two-level model: a three-class classifier (= the first level) followed by a trivial predictor for each class (= the second level). Of course one could define many variations on this theme, by altering the definition of the classes and the second-level predictors. As an alternative to the three-class scheme described in the previous paragraph, we also tried dividing the papers into just two classes (those with less than 600 and those with 600 or more downloads). For the second-level predictors, we considered the following: (1) predict the average download count (computed on the training papers from the current class); (2) predict the median download count; (3) a linear predictor trained using regression SVM. The second-level predictors are always trained on only those training examples that belong to the class for which the predictor will be used.

The classifier that forms the first level of our two-level models will be a linear SVM classifier [CV95, Bur98]. Since the original formulation of the SVM algorithm produces only binary (two-class) classifiers, we will use the "one against others" approach for the three-class scheme. We will train three binary SVM classifiers, each of which will try to separate one class from the other two. A paper is then predicted to belong to the class whose classifier had the highest confidence that

% of	Classification			Avg. error if		Actual	
training	accuracy		2nd level	acc. were 100 %		average error	
set used	training	test	predictor	training	test	training	test
Two classes (< 600 and ≥ 600)							
20%	100.0%	84.9%	Average	99.67	102.43	99.67	*136.66*
20%	100.0%	84.9%	Median	94.98	97.85	94.98	*139.37*
20%	100.0%	84.9%	SVR	27.88	98.63	27.88	**133.76**
30%	100.0%	84.3%	Average	94.59	103.53	94.59	148.45
30%	100.0%	84.3%	Median	89.52	110.44	89.52	163.16
30%	100.0%	84.3%	SVR	25.13	97.72	25.13	142.80
Three classes (< 400, 400..999, and ≥ 1000)							
20%	99.8%	55.3%	Average	78.08	80.87	78.03	163.84
20%	99.8%	55.3%	Median	75.25	79.10	75.30	159.65
20%	99.8%	55.3%	SVR	23.03	77.73	23.19	154.73
30%	99.7%	57.5%	Average	69.74	84.61	69.90	171.99
30%	99.7%	57.5%	Median	67.42	87.12	67.73	176.65
30%	99.7%	57.5%	SVR	21.27	80.43	21.71	161.40
Performance of the simple one-level SVR predictor (from Section 3.1)						37.62	**135.80**

	Predicted class	
True class	< 600	≥ 600
< 600	258	4
≥ 600	44	12

	Predicted class		
True class	< 400	400..999	≥ 1000
< 400	139	42	0
400..999	83	37	3
≥ 1000	5	9	0

Table 16: Experiments with two-level models (see Section 8.2) on the simple "Author + Abstract" representation. The upper table shows average errors achieved depending on the number of classes, the percentage of training set used (how many most frequently downloaded papers are actually used for training), and the type of the second-level model: average, median, or support vector regression (SVR). Apart from the actual average errors achieved in the experiments, the table also shows the average error that would be achieved if the classifier on the first level were 100 % accurate.

Below the upper table are the contingency tables for the case when 20 % of the training papers have been used (the tables for 30 % are quite similar). These tables show, for each pair of classes, how many test papers from one class were predicted as belonging to the other class. The contingency matrix of an ideal classifier would have nonzero entries only on the main diagonal.

the paper belongs to it. There are many ways of combining binary SVM models into multiclass [ASS00, HL01, WW98], but the more sophisticated approaches are rarely much better than the simple one-against-others method described here.

Unfortunately, as experiments show, training good classifiers for the problem introduced here is far from easy.

Experiments with the "Author + Abstract" representation. This simple representation was introduced in Section 3.1. The results of these experiments are shown in Table 16.

Interestingly, using 20 % of the most frequently downloaded training papers now leads to smaller errors than using 30 %. (For one-level models that we considered in earlier sections, it was the other way around; see 3.2.)

The comparison of the three kinds of second-level predictors shows that the trivial average- and median-based predictors perform somewhat worse than linear regression models trained using SVM, but the differences are perhaps surprisingly small.

The major conclusion of these experiments, however, has to be that the classification problem that forms the first level of our two-level models is unfortunately quite difficult. In particular, many papers with a large number of downloads are incorrectly classified into the low-download class (see the contingency tables in Table 16). This means, of course,

% of training set used	Classification accuracy training	Classification accuracy test	2nd level predictor	Avg. error if acc. were 100 % training	Avg. error if acc. were 100 % test	Actual average error training	Actual average error test
Two classes (< 600 and ≥ 600)							
30%	100.0%	86.2%	Average	94.59	103.53	94.59	140.67
30%	100.0%	86.2%	Median	89.52	110.44	89.52	155.06
30%	100.0%	86.2%	SVR	22.30	93.45	22.30	**126.69**
Performance of the simple one-level SVR predictor (from Section 7.2)						28.82	**117.23**

Table 17: Experiments with two-level models (see Section 8.2) on the best representation from Section 7.2. See the caption of Table 16 for a detailed explanation of the individual columns.

that the wrong predictor will be applied to them on the second level and the prediction is likely to be badly wrong. These errors are sufficiently common and sufficiently large that they ruin more or less all of the advantage that the two-level scheme would otherwise have in comparison to the simpler one-level predictors that we have considered in all earlier sections. This is also the reason why the two-class scheme works better than the three-class scheme — the latter's classification task is more difficult and the accuracy of the resulting classifiers is much lower than in the two-class scheme.

The best two-level model we have found has an average error of 133.76, which is a disappointingly small improvement over the best one-level model for this representation (whose error, as we have seen in Section 3.1, is 135.80). This small improvement is probably not worth all the extra complexity.

As the contingency tables in Table 16 show, the major problem of our two-level models is that many papers from the high-download class are misclassified into the low-download class. This is perhaps not surprising as the high-download class is much smaller and is thus somewhat neglected by the learner. We tried changing the threshold used by the SVM classifier so as to increase the number of papers that would be classified into the high-download class. However, this also causes many low-download papers to be (incorrectly) classified into the high-download class, and the net result is an even larger average error.

The very high (100 % or nearly 100 %) training set accuracy of the first-level classifier suggests that the classifier may be overfitting, but our attempts to reduce this supposed overfitting by reducing SVM's error cost parameter (usually denoted "*C*") were not successful. Smaller values of *C* just encourage the learner to ignore the smaller class (i. e. the highly downloaded papers) even more than otherwise. Perhaps assigning a larger error cost to members of the smaller class than to those of the larger class would be more helpful. However, it should be borne in mind that such approaches usually encourage the classifier to be more accurate on instances of the smaller class but less accurate on instances of the larger one. This is fine if one is inclined to view the smaller class as interesting and important and the larger one as just trash that needs to be filtered out. On the other hand, for us a low-download paper misclassified into the high-download class is just as much of a problem as a highly-downloaded paper misclassified into the low-download class; both contribute approximately equally much to the average prediction error. Thus, cost-based approaches do not sound too promising for our situation, as they do not focus on increasing accuracy but rather other measures such as precision and recall.

Experiments with the best representation. Seeing that the two-level approach did achieve a slightly better performance than the standard one-level predictor on the "author + abstract" representation, we decide to try the two-level approach on the best representation (i.e. the representation with the lowest average error among the standard one-level regression models) found so far. This is "author + abstract + 0.005 in-degree + 0.5 in-links + 0.8 out-links + 0.25 journal + 0.004 title-length + 0.9 title-word-length + 0.1 (year − 2000) + 0.7 *ClusDlMed* + 0.35 *ClusCentDist*" from Section 7.2. The regression model based on this representation had an average error of just 117.23.

However, trying to use this representation for

Representation	No. of features	Average error on the training set	test set	Wgt. of DL-count in the model
author + abstract (from Section 3.1)	31 582	37.62	135.86	
author + abstract + download count	31 583	16.56	15.28	0.973
best representation (from Section 7.2)	66 867	28.82	117.23	
best representation + download count	66 868	16.21	14.85	0.968

Table 18: This table shows what happens if the true download count, i. e. the value that we would like to predict, is added to the representation as an additional feature. SVM makes good use of it, even though tens of thousands of other features are also present in the data. It finds predictors of the form *prediction*(d) := $0.97 \cdot$ *download-count*(d) + (*other terms*), where the other terms are of course small compared to the first term.

two-level models was somewhat disappointing (see Table 17). It apparently does not make classification any easier (the classsification accuracy for the two-class scheme was 86.2 %, as opposed to 84.9 % of the much simpler "author + abstract" representation). The problem of wrongly classifying highly downloaded papers into the low-download class is still present. The second-level models are not much more accurate than before. The net result nevertheless is that the errors are noticeably smaller than for the simple "author + abstract" representation. Now the best model is the one that uses the two-class scheme, 30 % of the training data, and has SVR models as second-level predictors; its average error is 126.69. This is certainly an improvement over the 133.76 of the "author + abstract" representation, but it is also a very far cry from 117.23, which has been achieved by a one-level SVM regression model.

Of course, this does not conclusively show that the two-level models cannot achieve errors comparable to or even smaller than those of the straightforward one-level models. It is likely that with some more experimentation one could find a combination of features and weights where two-level models would achieve smaller errors than the 126.69, which is the best result of this section. Perhaps they could even do better than 117.23, which is the best result we have found altogether. However, what we have seen so far suggests that improvements are not likely to be large and aren't easily achieved. At the same time, they must be paid for by a considerable increase in complexity of the model, as well as of the learning algorithm.

8.3 Bagging

Bagging [Bre96] is a simple and elegant ensemble learning technique. Suppose that we are given n training examples. We prepare a new training set by sampling with replacement: n times, we choose an example at random (all examples having probability $1/n$ to be chosen). Of course it is possible for some examples to be chosen several times (and around $(1 - 1/n)^n \approx 1/e \approx 37\,\%$ of the training examples will not be chosen even once), so that the new training "set" is actually a training bag. A model can then be trained on this new training "set". This process can be repeated many times, resulting in an ensemble of several models. When testing the ensemble, the prediction of the ensemble on a new instance is defined as the average of the predictions of the individual models in the ensemble on this instance.

Unfortunately, our experiments (with the "Author + Abstract" representation) suggest that bagging does not help in our situation. We tried ensembles of 10, 20, ..., 100 models, and the test set error of all of them is around 136.8; thus they are all slightly worse than the simple non-bagged model of Section 3.1, which has a test set error of 135.8.

This unhelpfulness of bagging for our problem is not really suprising. As Breiman [Bre96] pointed out, bagging is useful if the learning algorithm is not very stable with respect to changes in the training set. For a stable algorithm, the models learned on individual bags will not differ much from the model learned on the entire training set, and therefore neither will the aggregate. It is certainly reasonable to think of SVM regression as a stable algorithm: if instances that are not support vectors are removed from the training set, the model found by SVM will not change at all; if a few support vectors are removed, the model might change, but probably not very much (otherwise it would suggest that the original model had been overfitting the original training set, and as we know SVM is relatively successful at avoiding overfitting). Thus, since SVM is a rather stable algorithm, it is unsuprising that it doesn't profit from bagging.

We also tried training the individual models of the ensemble on larger bags, containing $2n$ rather than n items. This resulted in average errors comparable to, and occasionally slightly better, than the original non-bagged model. The smallest error found in this way was 135.0, achieved by the ensemble of 20 models. However, the total time spent on training individual SVM regression models is now approx. 29 times larger than for the original non-bagged model. We believe that this tiny decrease of prediction error is not worth the great increase in time complexity, and we decided not to perform any further bagging experiments (e. g. with other representations).

9 The target value as a feature

Our representations have tens of thousands of features. When considering whether it may be worth adding some new feature, one may be tempted to think that, however informative its values may be, they will surely be overlooked by the learner in the huge mass of features that are already present in the data. To test whether this is true, we tried adding a feature containing the download count, i. e. the very value that we want to be able to predict. A perfect learner would notice that this feature is always equal to the target value and would produce a model that always predicts using this feature and thus has an average error of 0.

The results of these experiments (see Table 18) show that the error after adding the download count attribute to the representation is indeed very small, and is approximately the same on both the training and the test set. This shows that even one single attribute, if its contents are genuinely valuable, can make a huge impact on the average error of a representation.

10 A look at the model

Since we are using the linear support vector regression algorithm to train our models, they are simply linear functions of the data vectors: $\mathit{prediction}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}+b$, where the values of $\mathbf{w}$ and b have been found by the learning algorithm. Thus we can try looking at the components in the normal $\mathbf{w}$ to see which features have a strong influence on the predictions of the model. Note that while performing our experiments, we divided all download counts by 200 so that the values that have to be predicted our models are relatively small, e. g. mostly in the range $[0, 5]$ or so. (Otherwise we would have to change the error cost parameter C from its default value of 1; C would have to be smaller to prevent the learner from taking the training set errors too seriously.) Thus the values of $\mathbf{w}$ and b discussed in this section would have to be multiplied by 200 to obtain a model that really predicts download counts. We will examine the model obtained by training on the best representation found in Section 7.2. The features available to the learner were: author, abstract, in-degree, in-links, out-links, journal, title length in characters, average length of title words, year (− 2000), and *ClusDlMed*. The learner used 30 % of the most frequently downloaded papers from the entire training period (30 % of 1566 papers = 470 papers). The scalar constant b obtained durign learning was 1.70; that is, one could say that each paper is assigned 340 downloads by default, and then its attributes can cause this value to change depending on their weights in the normal.

We could look directly at the components of the normal $\mathbf{w}$ to see which features have the largest weights (largest in terms of absolute value). However, this can give us a somewhat skewed view because some feature may have a large weight in the normal $\mathbf{w}$ but only occur in a handful of documents and thus does not really have an important role in the prediction. Many of these largest weights belong to features that represent individual authors, and thus aren't likely to occur in many documents.[13] Similarly, if a feature has a smaller

[13] For example, the second heaviest feature is *hooft-g*. Gerard 't Hooft (one of the winners of the 1999 Nobel prize in physics) is the author or co-author of 16 papers in our dataset, of which five fall within the training period. They have 853, 896, 916, 1004, and 1193 downloads, so the high

Feature	DF	Weight
in-degree	470	1.855
AU:hooft-g	5	1.808
AU:sen-a	7	1.295
title-characters	470	−1.221
AU:hawking-s	4	1.040
AU:susskind-l	5	1.029
AU:bousso-r	1	1.000
AU:friedan-d	1	1.000
AU:polyakov-a	2	0.944
AU:connes-a	2	0.943
T:lectures	11	0.832
OUT-L:9306069	9	0.782
AU:brandt-f	2	0.712
AU:spector-d	1	0.707
AU:efthimiou-c	1	0.707
J:*missing*	96	0.707
T:introduction	13	0.696
T:stephen	1	0.693
AU:hertog-t	3	0.687
AU:witten-e	1	0.668
AU:beasley-c	1	0.668
T:de)constructing	1	0.644
J:nucl-phys	60	−0.627
AU:maldacena-j	2	0.624
AU:fabinger-m	3	−0.585
AU:gubser-s	3	0.580
AU:linde-a	1	0.577
AU:arkani-hamed-n	1	0.577
AU:cohen-a	1	0.577
AU:georgi-h	1	0.577
AU:kofman-l	2	0.577
AU:moore-g	1	0.573
T:g-flux	1	0.570
T:physics	39	0.557
AU:lindstrom-u	2	0.548
AU:seiberg-n	2	0.545
OUT-L:0102077	7	0.533
OUT-L:0001197	18	0.520
T:review	36	0.506
T:idempotent	1	0.505
T:gauge	129	0.504
IN-L:0008222	1	0.500
OUT-L:9705117	2	0.497
AU:vafa-c	2	0.487
AU:shaposhnikov-m	1	0.480
AU:tinyakov-p	1	0.480
AU:medved-a	1	−0.480
AU:ooguri-h	1	0.480
J:phys-lett	53	−0.477
AU:garcia-compean-h	1	0.470

Table 19: The 50 heaviest components of the normal (i. e. those with the largest absolute value). DF is the document frequency, i. e. the number of training documents in which the feature was present (the total number of training documents is 470). The prefixes specify the group to which the feature belongs: AUthor, Journal, Text of title and abstract, IN-Links, OUT-Links. See Section10 for a discussion of these features.

weight in $\mathbf{w}$, this might mean simply that the values of this feature in the document vectors tend to be relatively large, and a smaller weight was necessary to tone them down; this does not mean that the feature is not useful or influential.

Table 19 shows the 50 largest weights in the normal. Many of these are quite logical, e. g. the large positive weight of in-degree (highly cited papers are also frequently downloaded, even in their first days when they don't have any citations yet — probably because they are simply good papers, and readers recognize that quickly), and the strong negative weight of the title length (we've seen in Section 6 that highly downloaded papers tend to have shorter titles). Similarly, the negative weights of *Nucl. Phys.* and *Phys. Lett.* can be explained by looking at Table 9: papers from these two journals have considerably less than the average number of downloads. The high weights of *introduction*, *lectures*, and *review* are also reasonable; people like to download survey papers, although considering their length one wonders how many of these they have the time to read. We have no idea why *gauge* should be so important; apparently, something called "gauge theories" is currently quite popular in theoretical physics. The paper 9306069 is referenced by nine training papers, of which several have very high download counts (average: 1010.8, median: 782), which explains the large weight of *out-links:*9306069.

As an alternative to just looking at the largest weights of the normal $\mathbf{w}$, we computed the average training vector, $\mathbf{m}$, and looked at the absolute values of $w_i m_i$. These should give us an idea of how large an effect the i-th feature has, on average, on the predictions produced by our model. As Table 20 shows, this ranking places greater emphasis on more common features; there are much fewer authors among the top 50 features, but more statistical features (including journals) and common keywords from titles or abstracts. The negative weight of *out-links:*9908142 is reasonable,

weight of this feature is certainly understandable. However, none of 't Hooft's papers appear in the test period, so the weight of this feature has absolutely no effect on the test set predictions.

Feature	DF	w_i	m_i	$w_i m_i$	Feature	DF	w_i	m_i	$w_i m_i$
ClusDlMed	470	0.373	0.765	0.2851	OUT-L:0001197	18	0.520	0.008	0.0043
title-characters	470	−1.221	0.210	−0.2559	T:noncommutative	59	0.144	0.030	0.0043
in-degree	470	1.855	0.113	0.2091	OUT-L:23190	92	0.138	0.031	0.0043
title-word-length	470	−0.025	6.739	−0.1691	T:universe	33	0.420	0.010	0.0041
year − 2000	470	0.391	0.114	0.0446	T:show	141	−0.298	0.014	−0.0040
J:*missing*	96	0.707	0.051	0.0361	T:spacetime	51	0.266	0.014	0.0037
J:nucl-phys	60	−0.627	0.032	−0.0200	T:branes	59	−0.219	0.016	−0.0034
AU:hooft-g	5	1.808	0.011	0.0192	AU:connes-a	2	0.943	0.004	0.0034
AU:sen-a	7	1.295	0.012	0.0153	OUT-L:9603167	23	0.412	0.008	0.0034
J:phys-lett	53	−0.477	0.028	−0.0135	T:m-theory	43	0.281	0.012	0.0033
OUT-L:9908142	77	−0.390	0.027	−0.0105	T:d-brane	32	−0.305	0.010	−0.0031
T:gauge	129	0.504	0.020	0.0102	T:holographic	23	0.271	0.011	0.0031
T:string	148	0.360	0.027	0.0098	OUT-L:9306069	9	0.782	0.004	0.0031
T:brane	92	−0.332	0.029	−0.0096	T:matter	36	0.408	0.007	0.0030
T:theory	275	0.335	0.024	0.0080	T:strings	49	0.236	0.012	0.0029
AU:susskind-l	5	1.029	0.007	0.0075	T:non-commutative	19	−0.298	0.010	−0.0028
J:jhep	110	−0.113	0.059	−0.0066	AU:harvey-j	4	0.459	0.006	0.0028
T:physics	39	0.557	0.010	0.0057	AU:fabinger-m	3	−0.585	0.005	−0.0027
AU:hawking-s	4	1.040	0.005	0.0054	AU:hertog-t	3	0.687	0.004	0.0027
J:phys-rev	70	−0.143	0.037	−0.0053	T:spectrum	46	−0.272	0.010	−0.0027
T:boundary	51	−0.442	0.012	−0.0053	AU:polyakov-a	2	0.944	0.003	0.0026
T:bulk	40	−0.434	0.012	−0.0052	T:distance	20	0.429	0.006	0.0026
T:review	36	0.506	0.009	0.0047	AU:gubser-s	3	0.580	0.004	0.0026
T:theories	117	0.260	0.018	0.0047	T:introduction	13	0.696	0.004	0.0025
T:lectures	11	0.832	0.005	0.0044	T:black	37	−0.210	0.012	−0.0025

Table 20: The 50 "most influential" features, as measured by the value of $|w_i m_i|$. Here w_i is the weight of feature i in our linear model, and m_i is the average value of this feature over the 470 training vectors. DF is the document frequency, i. e. the number of training vectors that contain this feature.

because this paper is referenced mostly by infrequently downloaded papers.

We can also try to get a feeling for the total influence of a group of features by looking at the sum $\sum_i |w_i m_i|$ over all features i from this group. (One might also look at $|\sum_i w_i m_i|$, but if the group contains some features with negative and some with positive weights, they will cancel each other out and the sum will be close to zero, making the group of features appear less important than it really is.) The feature groups, ordered by this statistic, are shown in Table 21, along with some other statistics of this type. It is interesting that $\sum_i w_i m_i$ for the group of binary journal attributes is negative; but this is in fact reasonable, if we look back at Table 9, which shows that the average download counts for most journals (except for "missing") are in fact below the global average download count. Either the reviewers of those journals are accepting the wrong papers, or they are in fact rightfully rejecting papers whose title and abstract consists of attractive hype that led to many downloads although the results of the paper were perhaps not really that solid.

11 Comparison of various models on real test data

So far, the average errors reported in this paper have always been based on cross-validation on the 1566 papers for which the correct number of downloads was known before the end of the KDD Cup 2003. However, after the contest, the correct target values for 150 test papers (the 50 most frequently downloaded papers of each test month) have also been published by the organizers. We can now compare the performance of various models on this previously unseen test data.

Interestingly, it turns out that prediction on the 150 test papers is for the most part noticeably easier than for the test papers during our cross-validation experiments. The only exception for this is paper # 0103239 (J. Khoury *et al.*: "The Ekpyrotic Universe: Colliding Branes and the Origin of the Hot Big Bang", 67 pages), which has 7160 downloads. This is far more than any other paper whose download count we know, either in the training or the test period. (The runner-up is # 0203101

Set of features, F	$\|F\|$	$\sum_{i\in F} w_i m_i$	$\sum_{i\in F} w_i^2$	$\sum_{i\in F} m_i^2$	$\sum_{i\in F} \|w_i m_i\|$
Text (title and abstract)	4085	0.022	30.110	0.023	0.509
ClusDlMed	1	0.285	0.139	0.585	0.285
title-characters	1	−0.256	1.490	0.044	0.256
out-links	4305	0.003	18.253	0.011	0.250
authors	641	0.044	30.215	0.004	0.226
in-degree	1	0.209	3.439	0.013	0.209
title-word-length	1	−0.169	0.001	45.415	0.169
journal	29	−0.009	1.564	0.009	0.087
in-links	3660	0.020	7.432	0.002	0.081
year − 2000	1	0.045	0.153	0.013	0.045

Table 21: Aggregated statistics for various sets of features. $\mathbf{w}$ is the vector of weights used by the linear model that is being examined in Section 10. $\mathbf{m}$ is the average of all 470 vectors that were used to train the model. The sets of features are ordered by decreasing value of $\sum_i |w_i m_i|$, which is an attempt to assess the total influence of a set of features on the predictions.

from the training period, with 2927 downloads.) Inevitably, all our predictors fail to notice that this paper is anything special. Suppose that our prediction is around 1000; thus there is an error of around 6000, which, since there are 150 papers in the test set, increases the average error by about 40. Thus, as our best average errors are around 140, it means that the average error would really be around 100, considerably below the cross-validation error on the training period, if the outlier had not been present.

Table 22 shows the performance of models obtained from richer and richer representations, from the basic "author + abstract" representation of Section 3.1 to the best representation found in Section 7.2. Figure 15 shows a comparison of the true download counts and the values predicted by our best model.

One might have been worried that our relentless tuning of the features and their weights in the representation might have overfit the training data. It is true that we used cross-validation all the time, but with a sufficiently rich space of parameters one might overfit even that. Therefore it is nice to see that not much overfitting is actually taking place. The model that has been found best during cross-validation is also one of the best on the real test data. Only in the last step, when using both *ClusDlMed* and *ClusCentDist*, did overfitting occur (cross-validation error decreased in comparison to the model that uses just *ClusCentDist*, while the average error on the final 150-paper test set increased). There was also a small amount of overfitting in the model that we actually submitted, which was slightly better during cross-validation than the simpler model denoted R_3 in Figure 22, but was slightly worse on the final test set.

The predictions that we actually submitted to the KDD Cup were based on a slightly different model than the best one found in Section 7.2. At that time, we had not yet thought of the average title word length attribute (see Section 6), so the model did not include that. We hadn't thought of *ClusDlMed* either, so it uses *ClusDlAvg* instead (see Section 7.2); in addition, owing to a bug in our earlier implementation of the clustering algorithm, it used a different partition of papers into clusters, with 18 clusters instead of 26. Finally, there is a difference in the C parameter of the SVM regression learner. C defines the importance of training set errors during training. Throughout all the experiments presented in this paper, we used the default value of $C = 1$. However, during the preparation of our KDD Cup submission we noticed that a smaller value of C can sometimes be beneficial, and the model that we actually used in the end has $C = 0.7$, and the weights of many attributes had to be adjusted accordingly. The statistics for this model are also shown in Table 22.

In addition, Table 22 shows the average test set error achieved by the second and third best entry on KDD Cup. This is based on the results published on the KDD Cup web site (the scores of the three best entries at `http://www.cs.cornell.edu/projects/`

		Average cross-		Full		
		validation error		training	Actual test set error	
	Representation	Training	Test	set error	150 docs.	149 docs.
Trivial model (predict median of training set)		150.18	**152.26**	150.15	**181.11**	136.77
(AA)	**author + abstract**	37.62	135.86	37.77	155.38	111.31
	AA + 0.004 in-degree	35.99	127.69	35.90	146.77	103.62
(R_1)	AA + 0.005 in-deg. + 0.5 in-links + 0.8 out-links	29.90	123.74	29.96	143.06	100.25
(R_2)	R_1 + 0.25 journal	29.42	121.12	29.45	143.38	100.60
(R_3)	R_2 + 0.004 title-chars.	29.14	119.58	29.22	140.30	97.27
	R_3 + 1.3 title-word-length	29.21	118.94	29.33	139.75	96.79
(R_4)	R_3 + 0.9 title-word-length + 0.1 (year − 2000)	29.17	118.81	29.30	138.69	95.74
	R_4 + 0.4 *ClusDlMed*	29.10	118.31	29.22	138.02	95.09
	R_4 + 0.35 *ClusCentDist*	28.99	117.96	28.97	138.43	95.45
	R_4 + 0.7 *ClusDlMed* + 0.35 *ClusCentDist*	28.81	**117.23**	28.77	**137.81**	94.88
Our submission on KDD Cup 2003: SVM C parameter = 0.7, AA + 0.006 in-degree + 0.7 in-links + 0.85 out-links + 0.35 journal + 0.006 title-chars. + 0.3 *ClusDlAvg*		31.80	118.89	31.77	141.60	98.72
Second best entry on KDD Cup 2003					146.34	
Third best entry on KDD Cup 2003					158.39	
Representations we obtained by **automated tuning** of parameters:						
— Minimal cross-validation test error: SVM C = 0.7, 35 % training set used, 1.2 author + 1.1 abstract + 0.005 address + 0.003 in-degree + 0.03 out-degree + 0.04 authority + 0.1 PageRank + 0.6 PageKnar + 0.9 in-links + 0.03 out-links + 0.3 journal + 0.075 title-words + 0.7 title-word-length + 0.3 (year − 2000) + 0.045 author-count + 0.01 *ClusDlAvg* + 0.14 *ClusDlStd* + 0.9 *ClusDlMed* + 0.45 *ClusCentDist*		29.24	**114.72**	29.01	141.39	98.62
— Minimal error on the 150-document test set: SVM C = 1.2, 35 % training set used, 1.1 author + 0.65 abstract + 0.085 in-degree + 0.09 out-degree + 0.005 hub + 1.7 authority + 0.55 PageKnar + 0.85 in-links + 0.005 out-links + 0.055 journal + 0.003 title-chars. + 0.075 title-word-length + 0.25 (year − 2000) + 0.2 year-binary + 0.02 abstract-chars. + 0.13 *ClusDlStd*		28.50	124.66	28.48	**128.70**	86.57

Table 22: A comparison of the performance of increasingly complex representations. Apart from the training and test errors obtained during cross-validation (on papers from the training period), the table also shows the training and test error when all papers of the training period are available for training while testing is done on the same 150 papers from the test period that have been used for evaluation on the KDD Cup. The final column shows the performance on this test set without paper 0103239, which has an extremely and unexpectedly high number of downloads (7160). As usually, only 30 % of the most frequently downloaded training papers are actually used for training the regression SVM model. The the trivial model uses the median computed on 20 % of the training papers (see Section 2.4).

`kddcup/results.html` and the rough histogram of all scores at `http://www.cs.cornell.edu/projects/kddcup/download/KDDCup-Overview.pdf`). Interestingly, if we had submitted the results of the very simple “author + abstract” representation instead of our best model, we would still achieve the second place in the KDD Cup. If we had submitted the predictions of the trivial model instead, we would achieve the eighth or ninth place (of 18 teams).

Automated overfitting. Why tune the parameters manually (and risk overfitting) when the computer can do it so much better by itself (and overfitting is practically guaranteed)?

We wrote a wrapper around our training/test cycle that tries to tweak the weights of various feature groups so as to minimize the average error of the resulting model. For each weight, the wrapper tries first to increase it by some unit; if that fails to decrease the error, it tries to decrease this weight; if even that fails, it moves on to the next weight. If the error cannot be decreased by changes of this type, we have reached a local minimum in the parameter space and the algorithm stops. (For more sophisticated parameter-tuning approaches along these lines, see [Mla95].)

Unsurprisingly, this automated tuning leads into overfitting. Recall that the best manually tuned

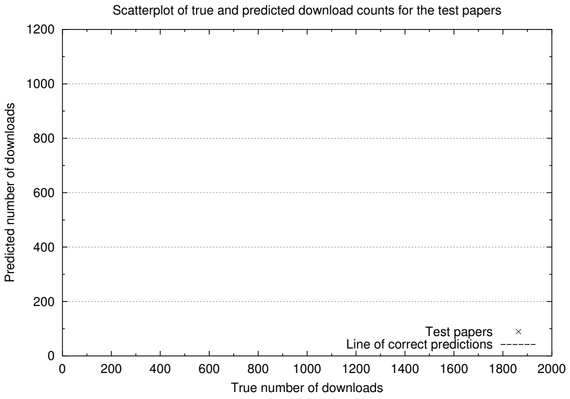


Figure 15: A scatterplot of true and predicted download counts for the 150 papers that were used for evaluation on the KDD Cup. Actually, only 149 ×-marks are shown here; one paper had 7160 downloads (our prediction was 625.7) and could not fit on the chart without distorting the scale. The predictions shown here are those of the best model found by cross-validation on the training set; its average error on the test set is 139.03.

model, i. e. the one from Section 7.2 has an average cross-validation error of 117.23, and its average error on the 150 test documents that were used on KDD Cup is 137.81. The wrapper can start with this model's parameters and try to improve them, or it can start with a simpler representation, e. g. the simple "Author + Abstract" of Section 3.1, and work from there.

If the wrapper tries to minimize the average cross-validation test error, it can reduce it to 114.72, but the average error on the 150 test documents grows to 141.39. Apparently it was a good decision on our part to tune the parameters manually rather than automatically when preparing our entry for the KDD Cup; the limit of our patience when tuning also imposed a very welcome limit on overfitting.

To reduce overfitting in the case of mechanical tuning of parameters, a more thorough evaluation method than simple 10-fold cross-validation is necessary. For example, we tried evaluating each model by performing five cross-validation experiments, each time with a different split of the training set into ten folds. With this evaluation method, the automated tuning process reached an average cross-validation error of 115.48; when the parameter settings tuned in this way were used on the true test set of 150 papers, the average error was 138.30. This is still slightly worse than 137.81, which is the error of the best hand-tuned model, but not much. This result suggests that automated tuning can be used to select appropriate parameter values, but special care must be taken to prevent overfitting. Another interesting alternative to the multiple cross-validation runs used here might be to use a bootstrapping-based approach [Efr79, Koh95].

On the other hand, if the wrapper tries to minimize the error on the final test set of 150 papers (after using the entire training period for training), it can decrease this error to 128.70, but the cross-validation error then grows to 124.66. Here overfitting is worse than in the previous paragraph, where we minimized cross-validation error. This is, of course, reasonable as it is easier to overfit one ⟨training set, test set⟩ pair than the whole ten-fold cross-validation scenario.

Both models obtained in this way are shown at the bottom of Table 22.

12 Conclusions and future work

Clearly this download estimation task was not a very simple one. Although our predictions were the most accurate among the 18 entries that participated in this task on the KDD Cup 2003, the average error is still quite large compared to the average number of downloads. On the set of 150 papers that were used for evaluation on the KDD Cup, the average number of downloads is approx. 454.5, and the average error of the predictions that we had submitted to the Cup is 141.5.

As the course of experiments in this paper shows, we are facing a sort of law of diminishing returns: more and more new features have to be tried before one is found that decreases the error, and these decreases in error are getting smaller and smaller. Besides, features need to be carefully weighted, for, as our experiments show (see e. g. Figures 7, 8, 9, and 11), if a feature is given an inappropriate weight, much or even all of the error reduction that the feature could otherwise provide will be lost. This weighting grows more and more tedious and time-consuming as the number of different features and feature sets (and hence the number of training parameters) grows, and the risk of overfitting is correspondingly increased (see the discussion of automated tuning in Section 11).

Thus, although the approach used here could be extended in various ways and extended with more features, one is not particularly hopeful that this would lead to errors much smaller than the ones achieved here.

One such extension would be to augment the abstract and address attributes by adding n-grams, i. e. pairs, triples, etc. of adjacent words (usually after "stopwords" have been removed from the text). This could introduce as features many useful phrases such as multi-word technical terms and names of institutions. More sophisticated feature selection methods could also be applied to the textual attributes, though it is doubtful whether this would be of much use.

It would also be interesting to try improving the institution address information to make it more useful. It is hardly believable that this information is really as useless as our experiments have indicated. Perhaps some more sophisticated information extraction or cleaning technique should be applied to clean, filter, and normalize the institution names and addresses. As long as we are only interested in papers from the training and test periods, which is a total of 2244 papers, one could even prepare all this information manually.

The idea of multilevel models, which has been briefly explored here in Section 8.2, could perhaps be explored more thoroughly. It has shown some promise on the "author + abstract" representation and could perhaps be made to work well on the more complex representations as well if more care were taken to select the features and tune their weights. It could be combined with the regression stump or regression tree based approaches from Section 2.5. So far class membership was defined based on the number of downloads, but perhaps other characteristics (such as cluster membership, year of publication, journal, etc.) could also be used. The difficulty of this approach is that there is not very much training data to begin with, and as documents are divided into classes, there is still less data for each class. This means that the per-class predictors will be less reliable and the global error may increase.

Another possible improvement to the multilevel predictor approach would be to redefine the borders between classes so as to obtain classes which are roughly equal in size. This might reduce the error rate of the classifier on the first level (which, as we saw, often misclassifies members of the smaller class as if they belonged to the smaller class, resulting in large prediction errors on such instances).

Some papers contain a "Subj-class" field in the abstract file. We didn't use this in any way, because it is only present in about 8 % of the papers. However, perhaps this field could be used for clustering: initially, there would be one cluster for each subject class, containing the papers with a "Subj-class" field that refers to this class. We could then use these clusters as a starting point for k-means, or assign the remaining papers to clusters based on some similarity measure. Unfortunately, most of the subject classes are small, so the resulting partition of documents might contain a few large and many small clusters, which is perhaps not really useful for our download estimation problem.

However, all of these extensions are still based on the same underlying thinking. They are only small steps from the representations considered in this paper. To achieve larger improvements, it would probably be necessary to introduce features that say something genuinely new about the documents — something that hasn't yet been expressed by the features currently used. As Section 9 has shown, even just one new feature, if it is truly useful, can decrease the error immensely.

Perhaps it would be necessary to investigate what is it that influences a reader's decision, when looking at the "What's New" page, whether to download a particular paper or not. There are probably questions such as "Is this an interesting paper?" and, more specifically, "Will this paper help me with my own research? Is it something I need to know to keep up with my field?" Of course the most highly downloaded papers are probably so highly downloaded because they are in some sense exceptional, and because there is something about them that excites the curiosity of readers. But these are in a sense outliers that we perhaps need not expect to be able to reliably recognize and predict them accordingly. The questions listed earlier are, however, probably relevant for most of the papers that have achieved at least an average number of downloads.

Of course we cannot tell what exactly is going on inside a particular reader's mind at such a moment, but the aggregate answer over all readers should be reflected in concepts such as the popularity of individual research areas, sub-areas, topics, etc.; perhaps one could make use of bibliographic databases to estimate such things. In fact our dataset itself is perhaps already a good source of such information, but we aren't using it in the right way at the moment.

One imagines that the reputation of the authors also affects the number of times a paper will be downloaded. Other things being equal, people are more likely to give the paper a try if the authors are known for the quality of their previous work. We tried to take these observations into account by introducing the author statistics features (Section 3.5), but found that they just cause overfitting. Other features along the same lines could be introduced (e. g. the average or total number of citations received by earlier papers of the same authors), though it is of course questionable whether they would be any more successful. It might also be interesting to try to determine and use each author's "network" or even "clique" — his or her coworkers, people who work in the same lab or on the same projects, people who publish on the same conferences; the more extensive this network is, the more people are likely to be interested in the paper at least partly for the author's sake, thereby increasing the number of downloads.

And besides, it might very well turn out that there is great variation in the number of downloads even among papers by the same author; we probably do not know the download counts for a sufficiently large set of papers to be able to reliably say anything about that.

Perhaps part of the difficulty of our task is that we have to estimate downloads that, for the most part, occur within the first few days after a paper has been added to the archive. Thus many readers' decisions whether to download a paper or not might be, as it were, made on the spur of the moment and are thus not entirely predictable, especially since a download in those early days does not necessarily imply that the paper will be read or cited and thus the number of downloads is not necessarily particularly closely related to citations or other kinds of recorded, measurable characteristics of the papers.

To conclude, we might reiterate that: this is not an easy task; it is not clear (to us at least) how best to proceed from here; and it is indeed not particularly obvious whether one can get very far at all.

Acknowledgments. Thanks to Marko Grobelnik and Dunja Mladenić for their encouragement and support, and to Marko in particular for an interesting discussion of the clustering algorithm.

References

[AKA91] David W. Aha, Dennis Kibler, Marc K. Albert: *Instance-based learning algorithms.* Machine Learning, 6(1):37–66, January 1991.

[ASS00] Erin L. Allwein, Robert E. Schapire, Yoram Singer: *Reducing multiclass to binary: a unifying approach for margin classifiers.* J. of Machine Learning Research, 1:113–141, Dec. 2000.

[BGMM02] Janez Brank, Marko Grobelnik, Nataša Milić-Frayling, Dunja Mladenić: *Feature selection using support vector machines.* Proc. 3rd Int'l. Conf. on Data Mining Methods and Databases for Engineering, Finance and Other Fields, Bologna, Italy, 25–27 September 2002, pp. 261–274.

[BP98] Sergey Brin, Lawrence Page: *The anatomy of a large-scale hypertextual web search engine.* Proc. 7th Int'l. World Wide Web Conf. (WWW7), Brisbane, Australia, 14–18 April 1998. Computer Networks, 30(1–7):107–117, 1 April 1998.

[Bre96] Leo Breiman: *Bagging predictors.* Machine Learning, 24(2):123–140, August 1996.

[Bur98] Christopher J. C. Burges: *A tutorial on support vector machines for pattern recognition.* Data Mining and Knowl. Disc., 2(2):121–67, June 1998.

[CH67] Thomas M. Cover, Peter E. Hart: *Nearest neighbor pattern classification.* IEEE Trans. on Information Theory, IT-13(1):21–27, January 1967.

[CL01] Chih-Chung Chang, Chih-Jen Lin: *libSVM: A library for support vector machines* (version 2.3). Dept. of Comp. Science and Information Engineering, National Taiwan University, April 2001. `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

[CV95] Corinna Cortes, Vladimir Vapnik: *Support-vector networks.* Machine Learning, 20(3):273–297, September 1995.

[DHS00] Richard O. Duda, Peter E. Hart, David G. Stork: *Pattern classification.* Wiley, 2000.

[Efr79] Bradley Efron: *Bootstrap methods: another look at the jackknife.* Ann. Statistics, 7:1–26, 1979.

[HL01] Chih-Wei Hsu, Chih-Jen Lin: *A comparison of methods for multi-class support vector machines.* IEEE Transactions on Neural Networks, 13(2):415–425, March 2002.

[Joa98] Thorsten Joachims: *Text categorization with support vector machines: Learning with many relevant features.* Proc. of the 10th European Conference on Machine Learning (ECML-98), Chemnitz, Germany, April 21–23, 1998, pp. 137-42.

[Kle99] Jon M. Kleinberg: *Authoritative sources in a hyperlinked environment.* Journal of the ACM, 46(5):604–632, September 1999.

[Koh95] Ron Kohavi: *A study of cross-validation and bootstrap for accuracy estimation and model selection.* Proc. 14th Int'l. Joint Conf. on Artificial Intelligence, IJCAI 95, Montréal, Québec, Canada, August 20–25, 1995. Vol. 2, pp. 1137–1145.

[Mla95] Dunja Mladenić: *Domain-tailored machine learning.* M. Sc. Thesis, University of Ljubljana, Slovenia, September 1995.

[PBMW98] Lawrence Page, Sergey Brin, Rajeev Motwani, Terry Winograd: *The PageRank citation ranking: Bringing order to the web.* January 29, 1998. Published 11th November 1999, Digital Libraries Project report SIDL-WP-1999-0120, Stanford University.

[SS98] Alex J. Smola, Bernhard Schölkopf: *A tutorial on support vector regression.* NeuroCOLT2 Technical Report, NC2-TR-1998-030, October 1998.

[WW98] Jason Weston, Chris Watkins: *Support vector machines for multiclass pattern recognition.* Proc. of the 7th European Symposium On Artificial Neural Networks (ESANN 1999), Bruges, Belgium, April 21–23, 1999.

[ZI02] Tong Zhang, Vijay S. Iyengar: *Recommender systems using linear classifiers.* J. of Machine Learning Research, 2(Feb):313–334, Feb. 2002.