Similarity Measures for Smooth Web Page Classification

Nikolay Jetchev

August 22, 2007

Diploma Thesis at the Darmstadt Technical University

Supervisor: Professor Dr. Johannes Fürnkanz

Second Supervisor: Professor Dr. Stefan Ulbrich

Acknowledgements

First of all, I would like to thank Prof. Dr. Fürnkranz for supervising this thesis and the many useful discussions we have had about web page classification and machine learning. He has supported me greatly throughout my studies in TU Darmstadt.

I would also like to thank Prof. Dr. Stefan Ulbrich for being a second supervisor of this thesis and for the discussions we had about optimization.

Thanks go to Moritz Türk, Todor Dobrikov and Boyana Ivanova for proof-reading my thesis.

Nikolay Jetchev

Contents

List of Figures

List of Tables

Abstract

This thesis examines the application of consistency learning techniques for the classification of hyperlinked web pages. Different data similarity measures between the web pages are defined, using local features like textual content and features of the linked pages as a graph. The pairwise object similarities are gathered in similarity matrices, each of which can be used together with methods from consistency learning to make classification smooth with respect to the data structure revealed by these similarity matrices and improve the accuracy of a simple text classifier. To achieve even better performance in the primary task of web page classification, a secondary machine learning problem is defined as finding the optimal similarity matrix combination. The results on several hyperlinked text collections, including the well known WebKB collection show significantly better accuracy of the smooth learning methods over the plain text classification. The main novel contribution of this thesis is the definition and testing of various similarity measures between web pages and the construction of a locally flexible similarity measure from heterogeneous data sources that improves classification accuracy on each of them. These ideas may also be used with little modification in other domains besides web page classification, like bioinformatics and citation graph classification.

1 Introduction

The motivation of this thesis is to combine the approach to use different information sources for web pages with some advanced machine learning algorithms with graphs, which are good in employing different knowledge sources for a better classification. Automatic classification of text documents based on their text is one of the most basic tasks of machine learning. The classification algorithm should read the text of the document and predict its class label. Categorizing web pages is an interesting application of text classification. It may be used to create automatically structured web page directories, like the ones from `http://dir.yahoo.com/`, or assist other data mining tasks. Machine learning with hyperlinked text collections presents an interesting opportunity to combine the local text content with the network connectivity properties of the whole document database, which often leads to better categorization accuracy. The world's best search engine, Google, and its algorithm Page Rank, use hyperlinks to better assess web page importance, as described in Brin and Page [3]. The gains from connectivity analysis are shown in many papers dealing with web page classification in the last several years. Some of them are using properties of the hyperlinks between documents, which may be considered as explicitly defined links between documents from their authors. Another approach is to classify using properties of some similarity matrix between documents, which may be considered as implicitly defined links between the data points.

1.1 Overview of Hyperlinked Web Page Classification Methods

Hyperlinks in web page collections present a ready source of additional information. The assumption that these links carry important information about the classes of the web pages they connect is shown to be true in many papers in the last years.

1.1.1 Using Linked Neighbour Page Classes

In Chakrabarti et al. [4] the class labels of web page neighbours are used, preclassified by some text-based learner, to derive "semantic clues" and better classify documents. Clues about correct classes come from both neighbour class labels and neighbour features. The clues from neighbour classes are less noisy and bring more information for better classification accuracy. The classification itself is performed using iterative relaxation labelling. In order to predict the class of a web page the classes of its neighbours are taken by a learner function as input and some class predictions are given as output. Afterwards the new predictions are used again as input to the learner, until a certain number of iterations, or until the class predictions converge. For his experiments he uses pages from Yahoo directories and pages from US patent database.

Lu and Getoor [10] model probabilistically the link distribution and use the statistical learning method of logistical regression to combine features from the text of web pages themselves, and the class count histograms of web page neighbours. The web page neighbour concept is enhanced and each page has three different types of neighbours - pages it points to, pages that point to it, and pages which are co-cited by the current page. The neigh-

bours may give a hint for the class of the document and the probabilities for links between different classes are modelled. An iterative relaxation labelling algorithm, similar to the one in Chakrabarti et al. [4] is used, where first the neighbour class statistics are computed and used to predict web page classes, and then the neighbour class statistics are recomputed. The authors report a significant improvement in classification accuracy when testing on the citation graph databases of Cora and Citeseer, and a modest improvement with the WebKB database.

In Angelova and Weikum [1] web page classification is formulated as a problem of classifying the vertices of a graph. The hyperlinks are edges in this graph, and they are assigned different weights according to web page text similarities. Similar to Lu and Getoor [10], an iterative labelling technique is used, and performance is improved over the baseline text classifier on web page subsets from `www.wikipedia.org` and `www.imdb.com`.

1.1.2 Using Information from Hyperlinks

Fürnkranz [5] creates features from the textual structure of the hyperlinks themselves, e.g. the link anchor text, the text in a window around the link in the web page, the multi word phrases around the link, etc. These features are combined in ensembles, with different voting methods, and each hyperlink votes for the class of the webpage it points to. The hyperlink ensembles achieve better classification accuracy than plain text features on the WebKB database. One advantage of using features from links is that a correct classification may be made even with misleading or missing readable text, for example when the text information in a webpage is present as image instead of string, the links pointing to it may still have useful textual information.

1.2 Overview of Learning Methods using Data Similarity Matrices

One may also consider using as additional information for classification implicitly defined pairwise relations or links in the document collections. This is a more general scenario, because one may not always have enough hyperlinks in document collections. The different methods introduced here may work on any data with abstract vector representation, not only web pages. The algorithms differ from each other by how exactly they use the similarity matrix for classification.

1.2.1 Consistency Learning

Consistency learning methods, as presented classically in Zhou et al. [22] , are a major inspiration for this thesis, and are described in further detail in Section 4. They use a similarity measure between data points to define an affinity matrix between them. Classification is done by spreading label activation between the data points proportional to their similarity. The results presented in Zhou et al. [22] are on both synthetic data point sets and real world text collections like the 20 Newsgroup dataset. The consistency learning methods successfully improve the classification accuracy of a text-only SVM learner, and also show excellent results as a stand alone method to classify data with few labelled examples.

1.2.2 Random Walk Classification

There are many semi-supervised learning methods in the literature, like in the work of Xu et al. [21] , where similarity matrices are interpreted as probabilities and used for random walk text classification. The unlabeled

data points jump randomly between all data points. They are absorbed by some labelled points at the equilibrium state of the random walk. Denoising and pruning isolated edges is crucial for achieving better accuracy on the classical 20 Newsgroup and WebKB collections.

The work of Szummer and Jaakkola [16] examines the influence of parameters for partially labelled classification and random walks. They start with a similarity measure, a RBF kernel $exp(\frac{-|a-b|}{\sigma})$, and adjust its width parameter σ and notice how this has the practical effect of emphasizing longer or shorter walks in the transition matrix. They also experiment with changing the size of local neighbourhoods, and how this influences the random walk.

1.2.3 Similarity Data Fusion

Affinity matrices and their applications for data classification are also popular for other forms of supervised and unsupervised learning, where information about the data is available as multiple different relations for each pair of data points. Each of them alone may not be enough for an accurate classification, but using all of the data together may be beneficial. Many authors in the fields of machine learning have recognized the importance of using all of the available information to construct a more accurate similarity measure.

Argyriou et al. [2] achieve good results in optical character recognition by finding an optimal data affinity matrix and using this to learn a smooth function on the graph vertices. The optimal matrix consists of a combination of similarity matrices of different image distance functions, for example those arising from rotating and translating the image information, and their modifications obtained by constructing the nearest neighbour graphs. The optimal convex combination of matrices is found by a method similar to Zhou et al. [22].

Bioinformatics is another area where there are multiple rich and heterogeneous data sources, for example protein sequence and gene expression measurements. Classification accuracy for protein function may be greatly improved by careful selection and convex combination of the available data similarity matrices, as in the paper of Lanckriet et al. [8].

In Stahl and Gabel [15] a similar approach is examined. A variety of similarity functions are combined. The combination coefficients depend on the data attributes. They are optimized and learned via an evolutionary algorithm to make a local distance function as a function of data features. This bears some similarity in spirit to this thesis, because here different similarity measures are also combined with data features for an optimal similarity function.

1.3 Structure of the Thesis

A brief overview of the content of this thesis follows. In Section 2 it is shown how we define our classification problems, the ODP and WebKB databases are examined, and the performance measures are introduced. In Section 3 a brief overview of graphs and similarity measures is shown. In Section 4 the concrete learning method with graphs we will use, consistency learning, is explained in detail. In Section 5 the connection between the similarity matrix used and the success of consistency learning is explored. Various similarity measures for web pages are defined and tested. In Section 6 a novel method for constructing a combined similarity matrix is presented. Afterwards in concluding Section 7 possible enhancements and further work is examined. At the end, there are several appendixes with mathematical proofs required in the thesis.

2 Basics and Experimental Setup

2.1 Machine Learning Basics

2.1.1 Learning and Classification

A description of the general supervised classification problem follows. A set of n d-dimensional data points $\chi = \{ x_1, ..., x_n \} \subset \Re^d$ and a label set $\gamma = \{1, ..., c\}$ are given. Each of the d dimensions are a different feature or attribute of the elements of χ . A classifier function is defined as $f : \chi \longmapsto \gamma$. Some of the data points $x \in \chi$ have an assigned label $y \in \gamma$, and they are called the training set. The test set consists of those points $x \in \chi$ that have no label assigned. The supervised classification task consists of learning the true labels of the test data by training a classifier function, called the learner on the available training data. This is illustrated graphically in figure 1 and figure 2 below, where a learner first trains on two different data with known labels and then classifies an instance with unknown label. The semi-supervised learning scenario is quite similar, but fewer labelled examples are available and the data distribution of unlabeled points is usually quite helpful for classification and augmenting the scarce label information available. In this work, web page, data point and object may all be used in the same sense. Label and class will also be used interchangeably.

2.1.2 Base Text Classifier

The easiest and most straightforward way to classify web pages is just to train some classifier function on their text content. This is not always the best solution, but it is useful as a first step in the framework described here. For the experiments in this thesis, a Naive Bayes text classifier was taken, since it is very fast and usually performs quite well on text data. It

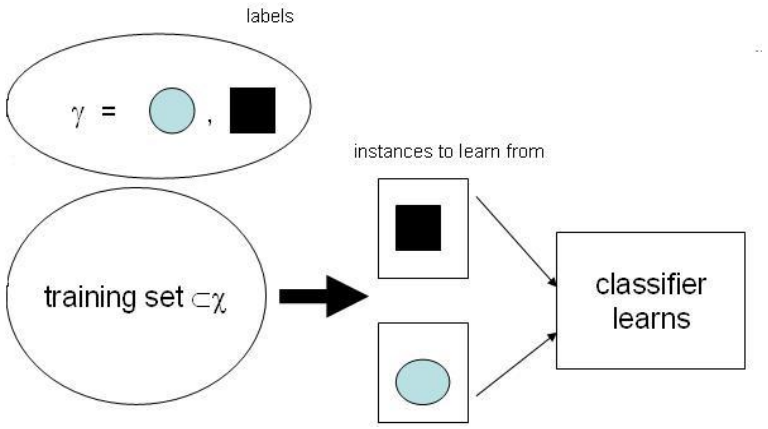

Figure 1: Training in supervised learning

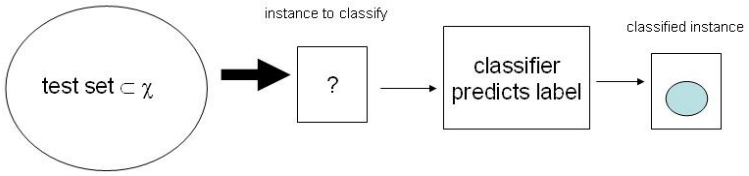

Figure 2: Testing in supervised learning

is described in many artificial intelligence and machine learning books, for example in Russell and Norvig [12]. The main mathematical rule behind the Naive Bayes classifier is the Bayes theorem connecting the prior and posterior probabilities for two hypotheses A and B:

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}$$

If x is a web page belonging to χ and c_i for i = 1...c are the different labels, then

$$p(c_i|x) = \frac{p(x|c_i)p(c_i)}{p(x)}$$

denotes the probability of the data point x having true label c_i. To obtain a prediction for the label using the Naive Bayes classifier, one needs to cal-

culate the most probable class label: $\text{argmax}_i\, p(x|c_i)p(c_i)$, ignoring the term $p(x)$ which is the same for all labels. To make the prediction, we need to find first the terms involved in the equation. $p(c_i)$ is just the percentage of all training data points which have label c_i, assuming that the data we have is a representative sample of the domain. $p(x|c_i)$ is the probability of data instance x to be with label c_i. The Naive Bayes Classifier is naive, because it assumes that all data attributes are independent from each other given the class. The probability of a data point with attribute values a_j for $j = 1...\text{d}$ to be in a certain class is just $\prod_j p(a_j|c_i)$. Each of the terms $p(a_j|c_i)$ can be easily estimated by looking at what percentage of data points with class label c_i have their attribute j equal to a_j . Whenever in this thesis a plain or simple text classifier is mentioned, a Naive Bayes Classifier is meant.

2.2 Data Sets

The experiments in this thesis use two different databases of web pages with known categories. The first one is WebKB, a popular database of the web structure of four American universities and their computer science departments, and is available online from `http://www.cs.umd.edu/~sen/lbc-proj/data/WebKB.tgz`. It has four subsets with different hyperlink structure - the Wisconsin, Washington, Cornell and Texas university domains. The different labels, like faculty and students, have clear meaning for university web domains. The class distribution is shown in table 1.

The second database is taken as a subset from Open Directory Project `http://www.dmoz.org/`, and is called ODP500, or ODP for short. It was compiled by the author in Seiermann [13]. The classes are different themes of pages, like arts or business, and the distribution can be seen below in table 2. Note that both databases have a class label called "other". Those are web

object	absolute count	fraction of total links/pages
all pages	3804	1.0
course label	227	0.059
faculty label	146	0.038
staff label	45	0.012
student label	502	0.132
project label	73	0.019
department label	4	0.001
other label	2807	0.738
all links	2864	1.0
links w.o. O	1007	0.352

Table 1: WebKB database statistics

pages without a single clear category, but which don't belong to any other class labels. They have various topics themselves and are not very appropriate for text classification. We are not interested in classifying them, but they provide useful information and context with their hyperlinks pointing to web pages to the classes of interest, and having enough links is important for successful web page classification. In the experiments done here, only performance on classes different from "other" was taken in consideration for performance reasons. The "other" classes" were implicitly present with the information of their hyperlinks, especially the hyperlink anchors and other link information used in Fürnkranz [5] to improve classification . Such a setup is quite appropriate for the Internet, where we have a subset of web pages of interest, and many more web pages pointing to them which are not interesting by themselves, but provide useful information, as depicted in figure 3. The number of links without the "other" web pages, i.e., links

between web pages with label different than "other", is also shown below, and is abbreviated "links w.o. O".

object	absolute count	fraction of total links/pages
all pages	3244	1.0
arts label	99	0.031
business label	97	0.030
computers label	98	0.030
science label	90	0.028
sports label	99	0.031
other label	2761	0.851
all links	9237	1.0
links w.o. O	40	0.004

Table 2: ODP500 database statistics

All experimental results were done using 4-fold crossvalidation, i.e. three quarters of the web pages were taken as training set and the remaining one quarter was the test set. Afterwards results are averaged over all four different test folds. Such a setup makes perfect sense for the WebKB database, where the web pages come from four distinct university domains, and the four universities were used as different folds, even though they are not exactly one quarter each, but only approximately. For ODP, random sampling was used to create folds of size one quarter of the total database each.

2.3 Implementation Notes

The implementation of the above classification framework was done in Java. It is convenient to use and provides the ability to easily use additional li-

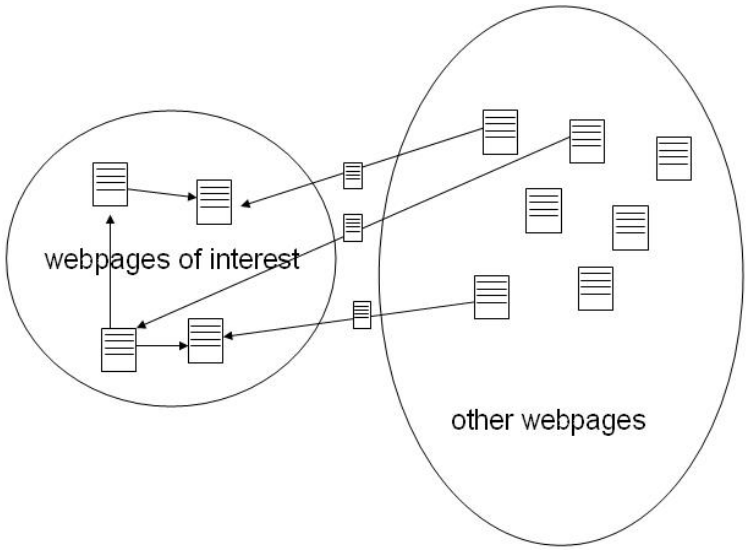

Figure 3: Pages of interest in the World Wide Web

braries. The consistency learning employed in this thesis requires a plain text classifier, for class predictions on web pages and link anchor texts. A Naive Bayes text classifier was used, taken from the WEKA machine learning library, available at `http://www.cs.waikato.ac.nz/ml/weka/`. It is fast and reliable, and does perfectly the job of a plain text classifier required as first step in our framework here. The JAMA matrix library, from `http://math.nist.gov/javanumerics/jama/`, and the JUNG graph library, available from `http://jung.sourceforge.net/`, were also necessary for the implementation of the classification algorithm. To use the Matlab optimization toolbox for quadratic optimization with side conditions for some of the experiments in Section 6, the JMatlink java library, available from `http://www.held-mueller.de/JMatLink/`, was required to allow calls of Matlab functions inside Java code. For the processing of texts, Porter stemming (changing words to base forms "playing" to "play") and stop word removal (removing words with too general meaning like "which") were done using

Lucene from `http://lucene.apache.org/`. The software was run on a machine with Pentium Mobile 1.5 gigahertz processor and 1 gigabyte of RAM.

2.4 Measures for the Experiments

2.4.1 Classification Quality

The goal of this thesis is to obtain a better web page classification. To qualify what "better" means, three different well-known measures from the machine learning and data mining communities are shown in the result tables. Precision for each label is defined as the ratio of the web pages with this label correctly classified, divided by all web pages classified with this label. Recall for each label is defined as the ratio of the web pages with this label correctly classified, divided by all web pages that really have this label. There is a precision vs. recall trade-off. If the classifier labels all examples with a certain label, the recall for it will be great, but the precision will suffer. If the classifier labels just a few examples in which it is most certain, the precision will be good, but the recall very bad. Since precision and recall are defined for each different class label, the values shown in the tables here are averaged over all different classes. Larger values are better for both recall and precision. The error rate is another simple and useful measure. It is the ratio of all web pages classified incorrectly divided by all web pages in the database. It is closely related to accuracy, which is the proportion of web pages classified correctly. If we have a total of 10 web pages, and 8 of them are classified correctly, than the error rate is 0.2 and the accuracy is 0.8. To avoid redundant information, only error rates are shown on the tables here, the accuracy is just (1 - error rate). A small error rate, respectively a large accuracy rate, is good.

2.4.2 Comparing Experimental Results with Previous Works

Even though WebKB is a well known database, the experimental setups of different authors vary greatly. The work of Lu and Getoor [10] used a subset of WebKB with 750 web pages, and their base text classifier had an error of 0.159, which is quite small compared to the value of 0.254 for our Naive Bayes text classifier. The work of Fürnkranz [5] has a variant of WebKB with 1050 web pages, and base text classifier error of 0.48. The improvement of the hyperlink ensembles in this work has an error rate of 0.26, which shows their utility, but the numbers can't be directly compared with the setup in this thesis. Seiermann [13] used all class labels from WebKB, including those from "other", so his database had around 3000 web pages and a direct comparison could not be made. The matrix learning methods used in this thesis would have had memory problems if we used all the 3000 other web pages and thus needed matrices with more than 9 million entries. The database of ODP also could not be evaluated fairly. Seierman reports a 0.11 error rate with a Naive Bayes classifier, which is much better than the Naive Bayes implementation in this thesis, with error of 0.246 on the same data set, probably due to some setup differences, like folds of crossvalidation, preprocessing and use of the "other" label. For this reason, the results in this thesis are best interpreted relative to the plain text Naive Bayes classifier. The reported gains from smooth classification with similarity matrices will most probably also transfer to other setups, so the exact values of the initial classification are not too crucial to assess the utility of consistency learning.

2.4.3 Similarity Matrix Quality

To illustrate how the similarity matrix used in the consistency method influences the classification accuracy, we need a measure of the quality of the

matrix. This can be illustrated with the ratio of the average similarity within the same class label, divided by the average similarity between web pages with different labels. The ratio is calculated for all labels at the same time, i.e. all edges in the graph are considered. The assumption is that the higher the ratio, the more activation flows between data points with same labels and leads to a better accuracy. The intuitive reasons behind this assumption are described in more detail in sections 4 and 5.2.1 . One may also consider not the average values, but the sum of all edge values within and between classes, which will give the same information up to a constant scaling factor, the numbers of edges. A possible problem with this ratio is that classes with more instances will skew the numbers, and classes with few instances will not be represented enough.

3 Learning with Graphs

3.1 Similarity Functions and Matrices

A similarity matrix on a set of n objects using similarity function s is a matrix with dimension $n \times n$, where entry in row i and column j corresponds to the similarity between object i and object j $s(i,j)$. A similarity function has the intuitive meaning that it has a large value for similar objects and small value for dissimilar ones, i.e. the opposite of a distance function. A matrix W is positive semidefinite (p.s.d.) iff $v^t W v \geq 0$ for any vector v. The formal definition of a similarity function $s : \chi \times \chi \rightarrow \Re$, giving rise to similarity matrix W, using the notation of von Luxburg [19], is:

(S1) $s(x,x) > 0$

(S2) $s(x,y) = s(y,x)$

(S3) $s(x,y) \geq 0$

(S4) $W p.s.d.$

In this thesis, we use a small relaxation of these conditions. For the consistency learning algorithm the similarity of an object to itself often does not matter, so we may also have $s(x,x) = 0$. We also don't need to have the property of positive semidefiniteness of the similarity matrix W, which is sometimes difficult to obtain and prove, and our methods work without it. It is required for some algorithms because p.s.d. similarity matrices correspond to similarity measures in Euclidean spaces, see von Luxburg [19].

Kernel functions, for example the polynomial kernel $(x_i^t x_j + 1)^p$, fulfill all definitions of similarity functions and are often a natural choice for many machine learning algorithms where nonlinearity is required, see Russell and Norvig [12] for more details. With a little abuse of definitions, similarity matrix, affinity matrix and kernel matrix will be used interchangeably here.

3.2 Graphs

The definition of supervised learning is general and flexible enough to allow any features that define the data points. Features may be local for the data points, like the local text of a web page, which constitutes the most straightforward classification method - classical flat text classification. More global features may also be considered, for example the text of neighbours of a webpage in a hyperlinked web graph. It will be useful to look at web pages as a complete graph, i.e. a graph having edges between every two vertices, with some edges coming from explicitly defined hyperlinks, and others from implicitly defined similarity measures. Thus, a similarity matrix between objects corresponds directly to the adjacency matrix of a full graph, and the matrix value in column i and row j corresponds to the weight of the edge between i and j, as can be seen in figure 4. Finding a single similarity matrix combining information from all different matrices corresponds to merging the multiple edges between data points to a single edge between every pair of data points. We consider the pairwise similarity between two web pages as an object in itself. When we have similarity measures between data points, we may gather them in a matrix, where the entry in column j and row i corresponds to the pairwise similarity between objects j and i. We will work with undirected graphs here, where the edge from i to j is the same as the edge from j to i, so the corresponding matrix entries are the same and we have symmetrical similarity matrices, fulfilling definition (S2) from the previous section. A future modification might experiment with asymmetrical matrices for the consistency algorithm in Section 4. Since hyperlinks between web pages are by their nature directed, they are not used directly in our graph representations, but are used as features to give rise to symmetric similarity measures, as described in Section 5. As a convention, we will also set the

diagonal elements of similarity matrices to 0, to avoid self reinforcement in the graph learning processes in the consistency learning.

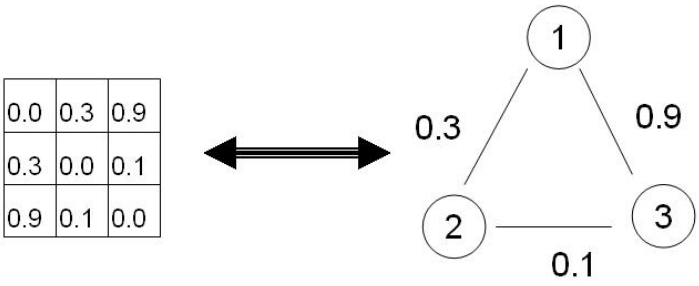

Figure 4: Graph and similarity matrix connection

3.3 Manifolds

Manifolds are important objects in mathematics and physics. They are defined as an abstract mathematical space in which every point has a neighbourhood which resembles Euclidean space locally, but in which the global structure may be more complicated. They are useful in machine learning when dealing with complex topological spaces, because they allow working with simpler lower dimensional spaces that retain the properties of the original high dimensional space. A simple and intuitive example of a manifold is the 3-dimensional sphere and its surface. If one looks at the surface locally, in a small enough neighbourhood, a 2-dimensional Euclidean plane captures its properties quite well. Classifying objects on the 3-dimensional curvature may take place on a simplified 2-d plane instead. In figure 5 the two moons dataset is depicted, a popular synthetic example for machine learning and pattern recognition, used for example in Xu et al. [21] and Zhou et al. [22]. Manifold learning may help much for the correct classification of the objects

from the two moon structures. If we look at the global distance function between points, point A is closer to point B from the other moon than to point C from the same moon. Constructing carefully a local similarity measure and a manifold consisting of highly connected data points, we will obtain a local neighbour structure where point A and C will be closer than A and B. In the scenario of web page classification, we assume that the graph has some simple structure, a manifold, on which web pages with the same labels are close, and those with different labels far away. Removing noisy edges from the whole graph will make the underlying structure of the data points clear, and avoid potentially ambiguous states. The assumption is that the manifold structure is preserved when trimming the graph. In figure 6, the manifolds are well formed and points from the same moons are connected. However, points D and E have "false" edges and may be misclassified. On the other hand, removing too many edges may destroy the manifold and the data points from the same moon won't be connected anymore, as in figure 7.

In this thesis the manifolds are created through a careful selection of the underlying data similarity measure. The manifold is constructed by changing the values of the pairwise similarities between web pages, and then removing edges from a local neighbourhood of the resulting graph. Different similarity measures correspond to graphs with different edge weights and thus give rise to different manifolds.

3.4 Matrix and Graph Modifications

3.4.1 Neighbourhood Modification and Edge Pruning

The data points and their affinity matrix W make a complete graph, where each data point is a vertex and is connected to all other vertices with a

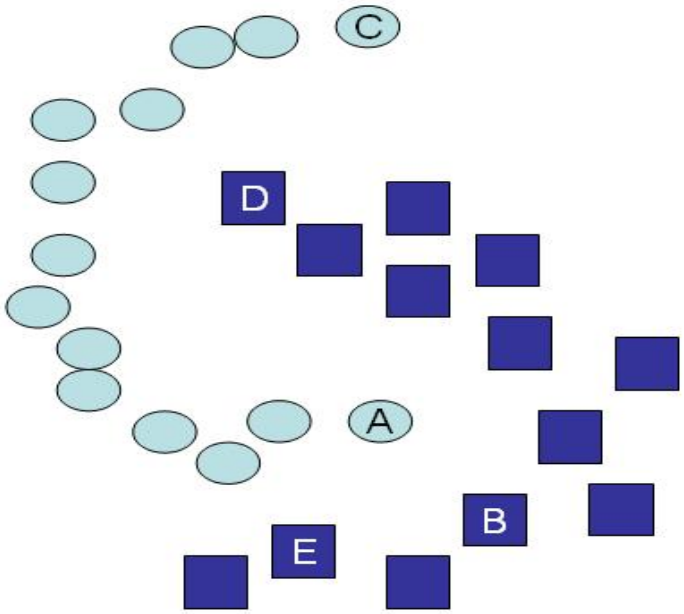

Figure 5: Two moons dataset

weighted edge with weight equal to W_{ij}. We assume that data points with different labels have some simple structure in this graph, a manifold. It should be connected with enough edges with large enough values and it should remain connected even we remove some edges from the whole graph. Edges between far away data points are removed, and preserved only in a local neighbourhood. $i \in kNN(j)$ is defined to mean that data point i is among the k nearest neighbours to j, i.e. W_{ij} is among k largest entries in the $W_{.j}$ column. By controlling the parameters of the number of local neighbours k in the local neighbourhood kNN and thus removing various amount of noisy, low-value edges, the newly modified matrix W_{nb} improves performance. A good similarity measure on the two moons dataset will give rise to a manifold connecting only points belonging to the same moon, and the activations will spread between data points with same labels. The exact parameter

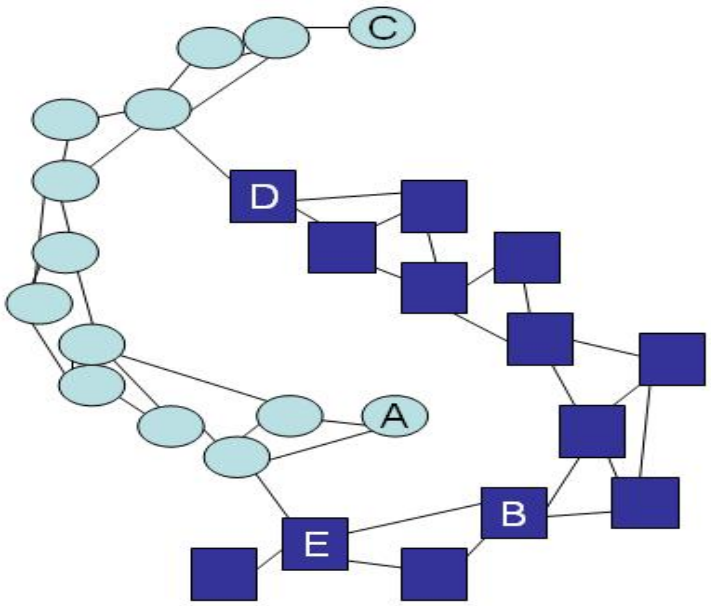

Figure 6: Neighbour graph of two moons dataset with noise edges

values which maximize performance depend on the graph and the data point structure. In this work it was tuned by a simple heuristic: some training data is held out and different parameter values for the nearest neighbour graph are tested on it. The best performing parameters from the held-out subset are usually the best for the other data points, which confirms the assumption that there is some underlying graph structure, and so we may present as results the best performing nearest neighbourhood parameters. The k parameter changes the classification accuracy and an ideal neighbourhood parameter will prune noisy edges, but also leave enough edges intact to allow activation of data points to lead to better classification. The NNB procedure that takes a similarity matrix W and modifies it via its mutual neighbourhood graph with parameter k is the following:

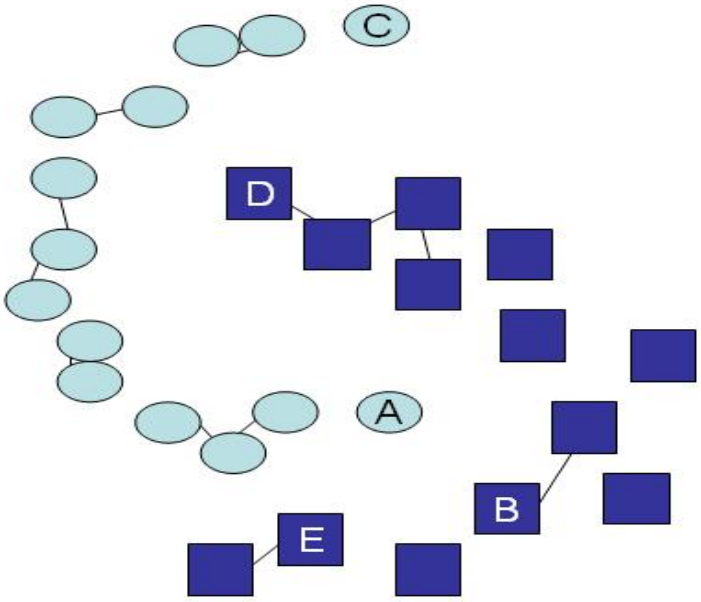

Figure 7: Manifolds with not enough edges for two moons dataset

for $i = 1...n$ find $kNN(i)$

for $i = 1...n$ and $j = 1...n$ if($i \notin kNN(j)$ or $j \notin kNN(i)$ set $W_{ij} = 0$

When mentioned in the experiments later, k for nearest neighbour graphs will also be called kNN. It was preferred to the threshold pruning variant, where edges below a certain weight are set to 0, because kNN was easier to tune with discrete k parameter than a real valued threshold. Taking the most connected neighbours is always well defined and robust to different setups, while the correct threshold depends on the data and similarity measure characteristics.

3.4.2 Normalizing Similarity Matrix

It is a popular convention in graph theory to define the degree matrix D of a graph with corresponding adjacency matrix W as a matrix having on

its main diagonal $D_{ii} = \sum_{j=1}^{n} W_{ij}$ and 0 on all other entries. The normalized affinity matrix S is defined to be $D^{-\frac{1}{2}}\mathrm{W}D^{-\frac{1}{2}}$, i.e. $S_{ij} = \frac{W_{ij}}{\sqrt{D_{ii}D_{jj}}}$. The intuitive meaning of the equation is that each vertex is treated relatively equally and vertices connected to too many other vertices do not influence excessively the matrix. This normalization also makes all entries of S to be ≤ 1, which is necessary for the calculations of the consistency learning algorithm and other semi supervised methods to converge, see Appendix C. In the experiments in this thesis each similarity matrix is both modified by its mutual neighbourhood graph and normalized afterwards. The whole process may be summarized like this, if we denote by W any starting similarity matrix, by S the final matrix used in the end in the consistency learning iterative equation in Section 4 to improve accuracy, and by NNB(W) the modification of a matrix by taking out edges not belonging to the nearest neighbours :

$$W \longrightarrow W = NNB(W) \longrightarrow S = D^{-\frac{1}{2}}WD^{-\frac{1}{2}} \longrightarrow S_{ii} = 0 \; for \; i = 1...n \quad (1)$$

4 Consistency Learning Framework

This section will describe the consistency learning algorithm in details and follows closely Zhou et al. [22].

4.1 Definition of the Consistency Framework

The consistency assumptions from semi-supervised learning, as defined in the work of Zhou et al. [22], are that nearby points in the feature space are likely to have the same labels and points on the same manifold are likely to have the same labels. Traditional supervised learning algorithms utilize only the first assumption and small changes in the input lead to small changes in the output. Such a learner will not be able to classify the two moons dataset well, because the class labels are mixed up in a more complex way. Many semi supervised learning methods use the second assumption and employ some measure of data similarity to improve further classification performance. The algorithm used in this thesis to improve data classification using similarity matrix information is the consistency learning algorithm from the work of Zhou et al. [22]. The setup in the original paper is the classical semi-supervised scenario. There are a limited number of training data points preclassified with true labels, sometimes only a very small fraction of the overall data. The test set has value of 0 for every label, which means there is no information to choose any label over some other. We want to infer labels for the test set. Both the test and training set are lumped together in n data points $\{x_1, ..., x_n\} = \chi \subset \Re^d$. We also have labels $\{y_1, ..., y_n\}$, and each $y_i \in \gamma$, the set of labels. This information is gathered in the matrix Y of size $n \times c$. Each different row has the intuitive meaning of class predictions for a certain data point for all labels, each of which corresponds to

a column. The columns are confidence in class prediction for a certain class label for all data points, each of which corresponds to a row. In the initial matrix Y the entries of each row are all 0's for test data points, because we have no class information. The rows for the labelled training data points have values of 1 for the column of the true label, and 0 for all other labels. We also need an affinity matrix W of size $n \times n$ between all the data points we have. Intuitively, we want to spread label information between the data points proportional to their similarities.

To obtain a better classification matrix F^* , Zhou et al. [22] wanted to find an $n \times c$ matrix F, which minimizes the loss function

$$Q(F) = \frac{1}{2}\left(\sum_{i,j=1}^{n} W_{ij} \parallel \frac{1}{\sqrt{D_{ii}}}F_i - \frac{1}{\sqrt{D_{jj}}}F_j \parallel^2 + \mu \sum_{i=1}^{n} \parallel F_i - Y_i \parallel^2\right) \quad (2)$$

The first part of the formula is called "separation cost". The term represents the amount of label variation on each edge between data points , and is large when data points with different labels are connected in the affinity matrix. It penalizes large local variations and it makes the consistency learning method "smooth". The second term is "assignment cost", it penalizes deviating too much from the original classification. F consists of several columns $F_{.i}$, each of which is a vector of size $1 \times c$ and it is corresponding to all class predictions for a single data point. The value of $Q(F)$ is linear in each column corresponding to different class labels, and $Q(F) = \sum_i Q(F_{.i})$. Whenever we find it convenient we may minimize $\mathrm{Q}(F_{.i})$ for each class i and then the argument of the function Q will be a vector instead of a matrix. This will make finding F^* an easy quadratic programming problem, as will be shown in the next section.

4.2 Optimization Method

The next step in the algorithm is to compute the optimal F for equation (2) either iteratively or directly. For convenience, assume that we are calculating F for a single class and $F_{.i}$ used in the formula is a vector of size $1 \times n$. First, write (2) in pure matrix notation

$$Q(F) = \frac{1}{2}\left(FD^{\frac{-1}{2}}(D-W)D^{\frac{-1}{2}}F^t + \mu(FF^t - 2FY^t + YY^t)\right) \tag{3}$$

The first term was rewritten as the quadratic form of F with the normalized graph Laplacian, for details see Appendix A. The second term was obtained by rewriting the sum as $(F-Y)^t(F-Y)$

Since the quadratic term in equation 3 is positive semidefinite as a sum of two positive semidefinite matrices $D^{\frac{-1}{2}}(D-W)D^{\frac{-1}{2}}$ and the identity matrix, the equation will have a global minimum for the value of F, where the gradient is equal to 0. See Appendix B for proof.

Differentiating with respect to F leads to the following equation:

$$\frac{dQ}{dF} = F - SF + \mu(F-Y) \tag{4}$$

Setting this to 0 leads to the following optimal solution F^* of the minimization problem:

$$F^* = \frac{\mu}{1+\mu}\left(I - \frac{1}{1+\mu}S\right)^{-1}Y \tag{5}$$

or when setting α to equal $\frac{1}{1+\mu}$ and dropping the constant in front:

$$F^* = (I - \alpha S)^{-1}Y \tag{6}$$

We may solve this directly, and the complexity class will be proportional to the complexity of matrix inversion, which is $O(n^3)$ for a matrix of size $n \times n$. We may also rewrite this in a form allowing an iterative solution:

$$F(t+1) = \alpha SF(t) + (1-\alpha)Y \tag{7}$$

This iterative equation will converge to F^* and the exact proof of the equivalence of (2) and (7) is in Appendix C.

The complexity in this case will be $O(n^2)$ per iteration, and in practice the algorithm converges quite quickly, less than 10 iterations are enough usually, so this is the faster method. Note that Zhou et al. [22] started with the iterative equation (7) as an intuitive activation spreading network, where the similarity matrix used in the calculation represents a full graph. Then he showed that this formulation is equivalent to the regularization equation (2). Consistency learning may be illustrated with figure 8. We start with a graph, determined by a similarity matrix of the data points, some of which have labels and some of which have not. Activation is spread along the graph and at the end all unlabeled data points have received labels. Note that an activation spreading network is quite similar in spirit to a random walk process that jumps randomly from the different data points.

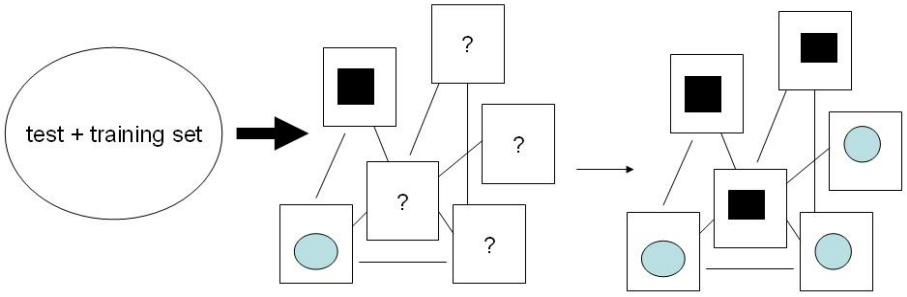

Figure 8: Consistency learning like graph spreading activation

4.3 Possible Consistency Learning Setups

Two different consistency learning setups were tested, one involving both test and training points, called the transductive setup. The other has only the

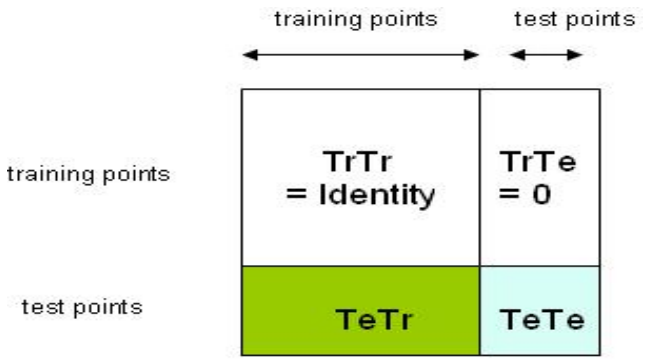

Figure 9: Similarity matrix for original transductive setup

test points, and is called smaller modified setup.

4.3.1 Original Setup with Combined Training and Test Data

We made some experiments with a setup similar to the one in figure 8. This setup is called sometimes transductive or semi-supervised, because we use the data points with known labels to infer the unknown labels, using the similarity of the test points with the training points. In it, the n objects to classify come from both the training and test data. This is the setup in both the works of Xu et al. [21] and Zhou et al. [22]. The initial label values in Y are the real, hard class values for the training data points, and the plain text classifier predictions for the test points. The similarity matrix to be used is modified in the following way: the edges test-to-test point and training-to-test point are calculated with the appropriate similarity measure. The $TeTe$ block is the matrix from the smaller modified setup explained in section 4.3.2, and when using 4-fold crossvalidation the additional $TeTr$ is three times its size. The edges training-to-training points should have no influence, so the corresponding matrix $TrTr$ is set to the identity matrix with 1 on the diagonal and 0 in the rest of the entries. $TrTe$ should be set to 0.

The purpose is to avoid changes to the labels of the data points with known labels. The matrices $TrTr$ and $TrTe$ are spreading activation to the training data points, so this setup will ensure that the values of F standing for rows of labels of training points don't change in the iterations in (7). The rest of the matrix has to be calculated with our similarity measure. The matrix and its submatrices are illustrated in figure 9. An advantage is that there are more edges to spread activation around, but this may also be a disadvantage, since some of those edges may be noisy false edges. The computational and memory cost is also much higher, since we work with larger matrices. In the case of 4-fold crossvalidation as in our experimental setup, there will be similarity matrices between 4 times as many data points, so the matrices will have 16 times more entries, which is quite costly in many practical situations. This may be optimized, because we don't need matrix rows for activations of the training points with known label, so we may set the matrix entries for them to 0, or use a more efficient sparse matrix implementation. With the $TeTr$ and $TeTe$ matrix blocks, the memory use will be then 4 times that of the other setup. The plain text learner is still necessary for initial class estimations on test data points, required by some of the similarity matrices introduced in the next section.

4.3.2 Modification with Smaller Matrix

In our work a novel modification in the initial setup is also presented and used in the experiments. The n data points we have here are all from the test data set from some fold of crossvalidation, i.e. none of them have hard true labels. Since we have no true labels in this set, but the consistency algorithm needs some initial information to function well, the values in the matrix Y are taken from the class distributions of some basic text classifier trained only on

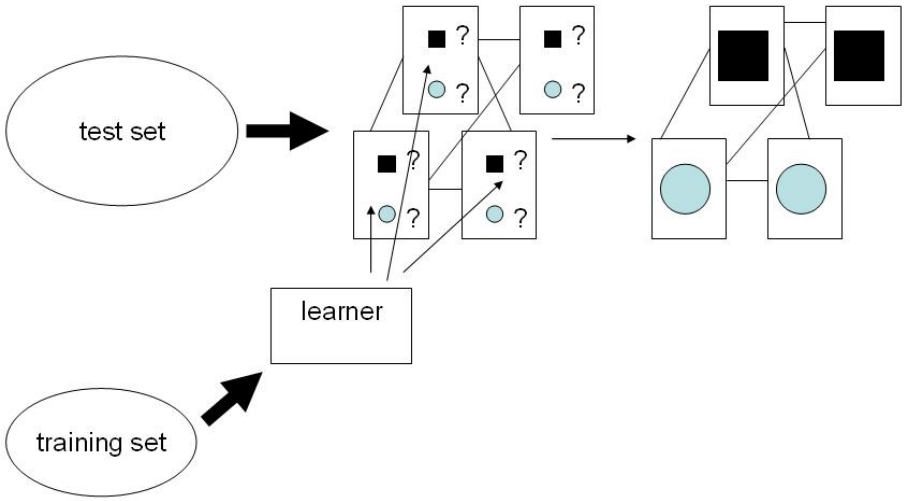

Figure 10: Modified consistency learning setup

the training data points' text features. So in the scenario in this thesis, there is partial information on all data points, and no training points are used directly, in contrast to the semi-supervised or transductive setting, where the training data points have a correct label, and the rest don't have any label information. The similarity matrix used is the small $TeTe$ block from figure 9. When classifying web pages or other objects, this is often a realistic scenario if we want to automate web page classification. In practice there are some sets of web pages with known classification, and some entirely new set is added to classify. For example, the entire domains of some universities may be crawled and organized manually into a directory structure. When we want to crawl the web domain of some other organization, the links between its pages will be more telling for the structure than the analogy to the already known university web pages. Is it is easier to have label information from some approximately correct learner, trained on the previously known data sets, than to have costly human assisted labelling of the newly added test

data sets. Another advantage is that we work on a smaller subset of all data points, and we have fewer entries in the affinity matrix, which has n^2 entries when using the consistency algorithm on n data points. This is particularly useful for databases with large training sets compared to the test set The information from the excluded training points with known labels is still present in the system because it was used to train the base text learners, and is represented in equation 2 in the assignment costs. The whole process is illustrated visually in figure 10. This setup will be called smaller modified setup.

4.3.3 Comparison of the Setups

The assumption behind the modified setup is that different folds of the crossvalidation have different inner structure. This assumption holds for heterogeneous databases like WebKB, where each university has its own subdomain web page organization. We thus apply consistency learning assumption only within this subset of all data points, the test data fold of the crossvalidation. This is the case with the WebKB dataset, and the classical 4-fold crossvalidation performed with it. The training folds are three university domains, and the test fold is the fourth university. Each different university has different class and link structure, the web domain structure varies by computer departments in the real world. Trying to learn with manifolds may be easier on a single university, than on all four universities and their various structures. One of the goals of the experiments here is to see whether using both the test and training points for consistency learning may bring classification improvements, even at the cost of more computational time. The clear structure of 4 distinct folds is a peculiarity of WebKB which has four subsets with different hyperlink structure - the Wisconsin, Washington, Cornell and Texas

university domains, which is a rather unusual database property. Other web page classification databases like ODP have random samples of a single set as different folds, and mixing test and training point graphs there may bring more, but it will still be at a higher computational cost.

4.4 Separation and Assignment Cost Tradeoffs

A tuneable parameter for the consistency learning is the parameter α controlling the separation and assignment cost trade-off. Intuitively, if the original classification is very accurate, we should not change it much and should not influence it too much with the affinity matrix, just a bit of activation will be enough to improve classification. Conversely, if the initial learner is quite inaccurate, we may gain much by heavily modifying its initial classification with information from the graph structure. Usually a value of 0.8 was used in the experiments if no value for α is specified explicitly, since such a value had good performance on holdout data. More results illustrating this are found in Appendix D.

5 Enhancement of the Consistency Framework Through Similarity Matrix Selection

One may ask oneself the question how important for classification accuracy are the two pieces of information involved in the consistency learning equations, the affinity matrix and the original text data classification. In this work the initial text classification is performed by any simple and fast text classification algorithm, like Naive Bayes from the WEKA library. In this section we discuss and compare various options for defining the affinity matrix to improve the accuracy.

5.1 Activation Spreading by Affinity Matrices

What is the exact influence of the similarity matrix on the categorization that comes from the consistency learning? A useful analogy comes from looking at the iterative equation (7) as an activation spreading network. To obtain a better label assignment, we want that each data point activates data points just from its label class, and has minimal activation on other classes. In this way, wrongly classified data points for each class will gain values in the correct label from the correctly classified data points. Additionally, correctly classified data points should not gain activation in false classes. The activation is actually the matrix multiplication: the new class values for data point x_i and class l are in matrix entry F_{il} equal to $S_i^t F_{.l}$ and so each data point is activated proportionally to its similarity in the affinity matrix to every other data point and its label value. This is illustrated by figure 11, where the activation spreading for the second object out of a total of three different objects and two labels is shown. Note that the self similarity of object two to itself is set to 0, so only the labels of the other objects will

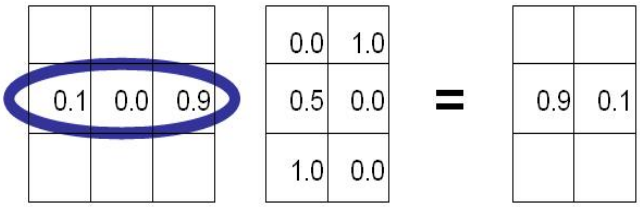

Figure 11: Matrix multiplication as spreading activation

influence it via the graph activation spreading. Of course, the $(1-\alpha)Y$ term in equation 7 still has adds label information on each iteration, but it is a constant term whose influence we control through setting α. The first object has large value in the fist label, and the third object has large value in the second label. The second object has larger similarity with the third one, so matrix multiplication in a single iteration will change its label distribution in favour of the first label.

5.2 Synthetic Similarity Matrix

5.2.1 Defining Synthetic Similarity Matrix

What is the upper limit of gains from the consistency learning method, if we have the best possible similarity matrix ? An ideal affinity matrix will have all 0's in entries corresponding to edges between data points in different classes, and all 1's in entries corresponding to same classes. When those web pages with same class label are ordered one after another, one may visualize their similarity matrix and see a nice block diagonal structure, as illustrated visually in figure 12. There the synthetic matrices have a very clear pattern for all labels, and the real cosine similarity matrix has good structure for some and bad for others. The ideal matrix should be the best possible affinity

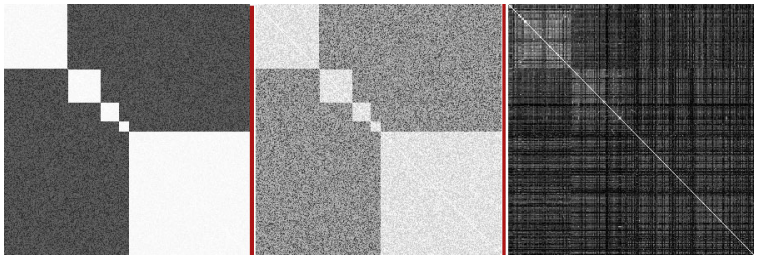

Figure 12: Image of synthetic matrix R19, R1.22, Text cosine similarity matrix

matrix, because all activations will be between data points with the same label. The correspondence to this pattern can be numerically illustrated by the ratio of the average similarity within the same class label, divided by the average similarity between web pages with different labels. The higher the ratio, the more activation flows between data points with same labels and leads to a better accuracy and smaller error rate. This is illustrated schematically in figure 13, where edges within same class are thicker and have larger weight than edges between different classes. To test the connection of edge ratio and classification performance, synthetic similarity matrices were generated with distribution of the edges with different ratio.

5.2.2 Experiments with Synthetic Matrices

The results are shown in tables 3 and 4, where 5 different matrices were created, and named Rx, where x is the ratio of the two types of edges in them, e.g. $R19$ has a ratio of 19 to 1 between the average weight of edges connecting same class data points and edges connecting different class data points. The different edge distributions were created by a uniform distribution. In $R19$ the expected mean value of edges in same class was 0.95, and of the other edges 0.5. For $R5.67$ it was 0.85 and 0.15, for $R3$ 0.75 and 0.25, for $R1.86$ 0.65 and 0.35, for $R1.22$ 0.45 and 0.55. PT is the baseline performance of

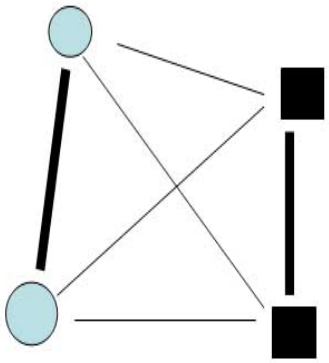

Figure 13: Activations in consistency learning

a plain text classifier. α was set to 0.8 for the consistency learning for all 4 tables. The results for both databases confirm our expectation. A better similarity matrix with larger within class to between class edge ratio leads to better performance. As can be seen in the table, the ideal matrix really improves performance, and adding more weight to the between class edges or less weight to within class edges worsens accuracy. The local neighbourhood parameter kNN can both improve and worsen performance, so it has to be tuned finely. It is also worth observing that even with a very large edge ratio, we can't get optimal web page classification, because the initial class information was not completely accurate. Another interesting observation from tables 3 and 4 is that even with an optimal matrix, no perfect classification is achieved. The reason is in the initial classification: if there are many false positives in some class, they will spread influence to other points in their real class and may degrade performance even on true positives.

If we used the second setup from Section 4.3.1, a much better classification will be possible, because the training data labels will be known. The results

		PT	$R19$	$R5.67$	$R3$	$R1.86$	$R1.22$
$kNN10$	error	0.254	0.140	0.155	0.181	0.222	0.279
	precision	0.422	0.517	0.508	0.484	0.441	0.406
	recall	0.415	0.463	0.448	0.426	0.390	0.342
$kNN30$	error	0.254	0.125	0.156	0.182	0.262	0.332
	precision	0.422	0.549	0.516	0.500	0.379	0.322
	recall	0.415	0.476	0.442	0.422	0.329	0.266
$kNN50$	error	0.254	0.123	0.158	0.187	0.282	0.338
	precision	0.422	0.544	0.517	0.477	0.342	0.281
	recall	0.415	0.477	0.441	0.413	0.306	0.258
$kNN70$	error	0.254	0.122	0.163	0.195	0.303	0.338
	precision	0.422	0.550	0.518	0.441	0.353	0.247
	recall	0.415	0.475	0.434	0.405	0.284	0.255
$kNN90$	error	0.254	0.127	0.165	0.203	0.304	0.343
	precision	0.422	0.549	0.488	0.440	0.354	0.287
	recall	0.415	0.470	0.433	0.396	0.282	0.252

Table 3: Performance of synthetic similarity matrices for WebKB, smaller modified setup.

are in tables 5 and 6. Since the transductive setup uses many more edges to spread activation. larger kNN values are appropriate. One may notice a small trend that for matrices with good ratios, the transductive setup outperforms the smaller modified setup, and for worse matrix edge ratios the transductive setup is worse. A possible explanation is that when we have correct edges the many true labels of the transductive setup all contribute to a good classification. When the edges are worse, the fact that there are more edges in the transductive setup means that more of them may bring bad activations from false labels due to noise effects. Also, in the transductive setup when we have more data points overall, the between class and within class edges counts change, and the average ratio of these has a bit different

		PT	$R19$	$R5.67$	$R3$	$R1.86$	$R1.22$
$kNN10$	error	0.246	0.091	0.135	0.149	0.207	0.240
	precision	0.647	0.766	0.736	0.724	0.687	0.663
	recall	0.627	0.755	0.718	0.707	0.658	0.631
$kNN30$	error	0.246	0.068	0.128	0.170	0.222	0.246
	precision	0.647	0.782	0.744	0.715	0.682	0.667
	recall	0.627	0.774	0.724	0.689	0.646	0.626
$kNN50$	error	0.246	0.070	0.157	0.195	0.230	0.257
	precision	0.647	0.780	0.721	0.696	0.677	0.662
	recall	0.627	0.772	0.699	0.669	0.640	0.617
$kNN70$	error	0.246	0.089	0.166	0.207	0.242	0.259
	precision	0.647	0.767	0.713	0.688	0.672	0.660
	recall	0.627	0.757	0.693	0.659	0.630	0.616
$kNN90$	error	0.246	0.089	0.170	0.215	0.242	0.257
	precision	0.647	0.767	0.711	0.684	0.670	0.662
	recall	0.627	0.757	0.690	0.652	0.630	0.617

Table 4: Performance of synthetic similarity matrices for ODP, smaller modified setup

significance from the smaller modified setup.

5.3 Affinity Matrices from Different Features

Which affinity matrix to choose to improve from? The first possibility is to use some meaningful data similarity measure. For text classification, cosine similarity between the text vectors of the documents, or document similarity derived by a semantic decomposition technique like PLSA, are meaningful and easy to calculate similarities. In the domain of web page classification, additional matrices may be defined that include the hyperlink graph information. For example, the similarity between two web pages may be the average label similarity between the class predictions of their link anchor texts. The

		PT	$R19$	$R5.67$	$R3$	$R1.86$	$R1.22$
$kNN10$	error	0.254	0.100	0.142	0.177	0.242	0.326
	precision	0.422	0.593	0.519	0.521	0.470	0.380
	recall	0.415	0.524	0.451	0.417	0.346	0.286
$kNN60$	error	0.254	0.040	0.078	0.140	0.278	0.367
	precision	0.422	0.692	0.606	0.516	0.398	0.366
	recall	0.415	0.622	0.530	0.451	0.295	0.225
$kNN110$	error	0.254	0.036	0.091	0.145	0.281	0.368
	precision	0.422	0.694	0.622	0.518	0.371	0.224
	recall	0.415	0.627	0.515	0.445	0.288	0.224
$kNN160$	error	0.254	0.037	0.088	0.144	0.288	0.369
	precision	0.422	0.695	0.611	0.525	0.305	0.261
	recall	0.415	0.631	0.521	0.446	0.282	0.226
$kNN210$	error	0.254	0.023	0.082	0.141	0.287	0.366
	precision	0.422	0.701	0.546	0.535	0.270	0.226
	recall	0.415	0.684	0.521	0.446	0.282	0.227

Table 5: Performance of synthetic similarity matrices for WebKB, transductive setup

domain of hyperlinked web pages allows mining for multiple different features, having both local and global information. Figure 14 shows three different information sources for web pages: the text of the web page itself, the classes of hyperlinked web page neighbours, and the textual information from the links themselves, like anchor text. Below follow concrete descriptions on the implementations of matrices arising from different similarity functions used here. Many of those similarity functions will involve measuring how close two vectors a and b are. Here we use the cosine similarity function $cos(a, b)$, which is defined as

$$cos(a, b) = \frac{ab}{|a||b|}$$

		PT	$R19$	$R5.67$	$R3$	$R1.86$	$R1.22$
$kNN10$	error	0.246	0.023	0.054	0.114	0.199	0.263
	precision	0.647	0.814	0.790	0.741	0.676	0.625
	recall	0.627	0.814	0.788	0.736	0.665	0.612
$kNN60$	error	0.246	0.013	0.013	0.033	0.118	0.203
	precision	0.647	0.833	0.833	0.808	0.740	0.674
	recall	0.627	0.833	0.833	0.804	0.733	0.662
$kNN110$	error	0.246	0.013	0.013	0.041	0.101	0.184
	precision	0.647	0.833	0.833	0.801	0.754	0.695
	recall	0.627	0.833	0.833	0.797	0.746	0.678
$kNN160$	error	0.246	0.013	0.008	0.043	0.104	0.201
	precision	0.647	0.833	0.827	0.800	0.753	0.678
	recall	0.627	0.833	0.826	0.794	0.744	0.664
$kNN210$	error	0.246	0.013	0.012	0.058	0.122	0.207
	precision	0.647	0.833	0.823	0.789	0.739	0.681
	recall	0.627	0.833	0.822	0.782	0.729	0.659

Table 6: Performance of synthetic similarity matrices for ODP, transductive setup

$$Euclidnorm(a) = |a| = \sqrt{\sum_i a_i^2}$$

$$Vectorproduct(a,b) = ab = \sum_i a_i b_i$$

The cosine similarity is useful, simple and returns a number between 0 (most different) to 1 (same) as a result, when all the vector entries are positive, as is the case with class and term distributions used here. Some other vector similarity functions are possible, like measuring vector correlation or a kernel, for example RBF kernel $exp(\frac{-|a-b|}{\sigma})$. Investigating the performance of these vector similarity functions may be the work of further experiments, or each of these may be included as an additional similarity matrix in the framework.

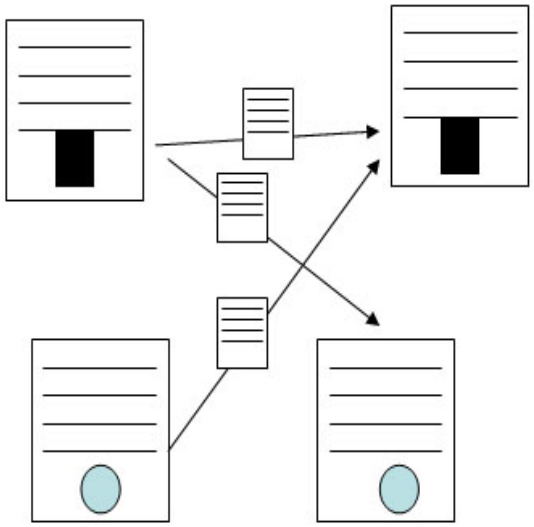

Figure 14: Different possible web page features. Text of webpage and its neighbours. Label predictions for neighbours. Anchor text and other textual features of hyperlinks themselves.

5.3.1 Similarity Functions based On Text Similarities

The first type of similarity matrix that the domain of web pages gives rise to is based only on the term document matrix of the webpage text. Each web page is parsed using `java.swing` classes and its text content is gathered in a document.

TFIDF Cosine Similarity The text of each web page is presented as a vector in high dimensional space, the term vector space model, where each different word is a dimension. Suppose we have a total of d different words in the dictionary. The value in each dimension is the number of times the word occurs in the document. This corresponds to the more simple TF weighting scheme, where for word i and document j TF_{ij} is defined as the

count of i in j. One may also use the slightly more complicated TF-IDF weighting scheme, which is also well known and outperforms empirically the TF scheme. DF_i is the number of web pages which have word i. If n was the size of the documents in the database, IDF_j is equal to $log(n/DFi)$. The $TFIDF(i)$ vector representation for a webpage i is defined as a vector of size $1 \times d$ where $TFIDF_{ij} = TF_{ij} \cdot IDF_j$ where j ranges from 1 to d over all different word terms in the dictionary. Intuitively, the second term gives a bonus for words occurring in fewer documents. The construction of the similarity matrix W^{cos} requires to calculate the $TFIDF(wp_i)$ vector for each web page wp_i and then calculate the pairwise similarity between the web page documents in TF-IDF space:

$$W_{ij}^{cos} = cos(TFIDF(wp_i), TFIDF(wp_j))$$

Whenever we use the cosine text similarity matrix, it will be shortened to CC in result tables.

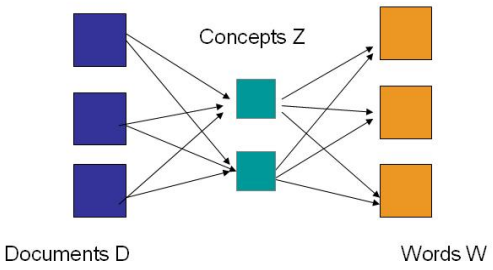

Figure 15: PLSA schema: document, concepts and words

PLSA Cosine Similarity A logical enhancement of the term vector space similarity measure is to use similarity in some abstract "concept" space. There are many ways to do this, like classical SVD semantic decomposition,

but in this work we use Probabilistic Latent Semantic Analysis , as described in the work of Hofmann [7]. PLSA expresses each text document d and word w in lower dimensional latent concepts Z, as shown in figure 15. Z was simply defined to have 100 concepts z_1,..,z_{100} for the experiments in this thesis. This is a sensible parameter choice for databases with less than 1000 documents, but further testing may be done to tune the number of concepts optimally. The probabilities $P(w|d)$ are decomposed as $\sum_z P(w|z)P(z|d)$. The advantage of PLSA over traditional LSI with Singular Value Decomposition is that PLSA is more efficient to compute, using the iterative EM algorithm. Details can be seen in Hofmann [7]. It is also useful that each document is represented as a non-negative combination of concepts, so this may be used in an intuitive similarity measure. Each web page wp is assigned its concept vector $z(wp)$ of dimension 1×100 for which each vector entry $z_i = \mathrm{p}(z_i|d)$ for $i = 1...100$. The construction of the similarity matrix W^{rplsa} requires computing the concept space representation of each web page $z(wp_i)$. Afterwards, the similarity matrix entry for web pages wp_i and wp_j is defined simply as the cosine between them:

$$W_{ij}^{plsa} = cos(z(wp_i), z(wp_j))$$

Whenever we use the PLSA similarity matrix, it will be shortened to $PLSA$ in result tables.

5.3.2 Similarity Functions based on Neighbour Labels

The work of Lu and Getoor [10] uses the classes of neighbouring web sites in the hyperlink graph to make inference about the class of web page *wp*. A "neighbour" may mean different things. Web sites having links pointing to *wp* are called *in-neighbours*, websites that *wp* points to are *out-neighbour* and

web sites that share *in-neighbours* with *wp* are *co-cited neighbours*. This is shown in figure 16, where we want to classify web page d. Web pages a and b are its *in-neighbours*, c is a *co-cited neighbour* and e and f are *out-neighbours*. We will use only the *in-neighbours* as example, the feature construction for out and co-cited neighbours is analogous.

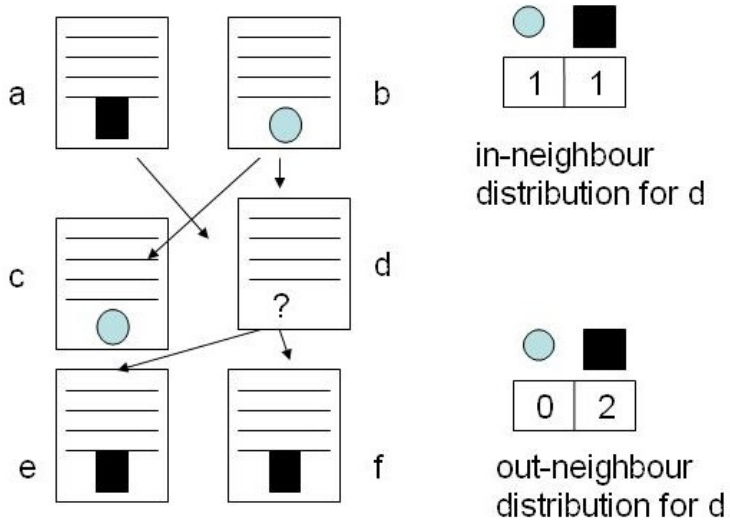

Figure 16: Web page features from Getoor, W_d^{in} and W_d^{out} vectors shown.

Each web page *wp* is assigned some vector of class counts of its *in-neighbours* $C^{in}(wp)$ of size $1 \times c$ where c is the number of different classes. The value of each entry of the vector C_i^{in} for $i = 1 \ldots c$ is equal to the number of *in-neighbours* of class i. Note that if the web pages are from the test set, these are not the actual classes, but the class distribution obtained from the initial text learner. This idea is modified slightly in this thesis and the classes of the neighbouring features are used to define a similarity measure between each web page. The affinity matrix W^{in} for in-neighbours is constructed in the following steps. First, for $i = 1...n$ and each webpage wp_i the class count

vector $C^{in}(wp_i)$ is calculated. Then for all i and j the corresponding matrix entry is defined as the cosine of the class counts of the two web pages:

$$W_{ij}^{in} = cos(C^{in}(wp_i), C^{in}(wp_j))$$

The construction of W^{out} and W^{co} is analogous. Note that this similarity matrix uses the initial label predictions from the Naive Bayes classifier for each web page. In the course of the iterations of the consistency learning algorithm, the labels will change and we may recalculate the neighbour label similarity matrices. However, this was not implemented in this thesis and such dynamic matrix recalculation remains an idea for a future enhancement. Whenever we use these similarity matrices, they will be shortened to GLI, GLO and $GLCO$ in the result tables.

5.3.3 Similarity Functions based on Link Features

The work of Fürnkranz [5] uses features from the in-hyperlinks themselves. For example, the anchor text of the link, or the text of the webpage before and after the link. Those features may be considered as text vectors and used in some text classifier together with the text of the original web page. The similarity matrix defined here is slightly different, because it does not use the individual words of the links as features, but uses the class of the links. To do so, first all hyperlinks and their textual information is gathered (from the training set) and a text classifier is trained on them. Afterwards, each link e is assigned its class distribution assignment $DL(e)$ of size $1 \times c$ from the output of a plain text classifier trained on the anchor texts of edge from the training set web pages. Each page wp is assigned some vector of the class counts of all its incoming links $L^{in}(wp)$ of size $1 \times c$. The construction of the similarity matrix W^l requires the calculation of $L^{in}(wp_i) = \sum_{e \in \delta(wp_i)} DL(e)$

for each web page wp_i and its incoming hyperlinks $\delta(wp_i)$. Afterwards, each matrix entry corresponding to pairwise web page similarity is defined as the cosine between the link feature class counts:

$$W_{ij}^{l} = cos(L^{in}(wp_i), L^{in}(wp_j))$$

For this thesis, we limit ourselves to the anchor text feature only, since it is the most crucial one for performance according to the original work of Fürnkranz [5]. The anchor text of a web link is the text seen by a user when looking at the position of the link in his web browser. For example, the html code for the hyperlink < a href= `"http://minimalnik.blogspot.com/">` Nik´s Blog < / a > has the anchor text "Nik's Blog". Since we have text vectors, one may consider using text cosine similarity between the anchor text vectors, but this has the drawback that anchor texts are too sparse and have few words, so many of the similarities will be 0 and won't lead to a similarity matrix with desirable properties. Note that to have enough anchor links in order for the methods of Fürnkranz to work, we used the links from pages with the "other" label pointing to pages in the set of interest. Whenever we use the cosine text similarity matrix, it will be shortened to JFLA in result tables.

5.4 Experiments with Different Similarity Matrices

5.4.1 Experimental Results

Tables 7 and 8 show the results of consistency learning with several similarity matrices with the best of several tested kNN parameters. The consistency setup with web pages only from the test fold was used. The abbreviation PT means the baseline Naive Bayes Text Classification. The other abbreviations are short for different similarity matrices, as indicated in the above

sections. $JFLA$ is link anchor similarity, GLI, GLO and $GLCO$ are different neighbour class distribution similarities, CC is TFIDF text cosine similarity, $PLSA$ is semantic concept similarity. The two different databases have different optimal kNN parameters for their manifolds for the modified setup.

The results show that different matrices are best for different setups. For WebKB the $JFLA$ matrix was best for smaller modified setup, but all the gains there are not too impressive, just 5 percent. The $GLCO$ matrix was best in the transductive setup for WebKB, with 24 percent less misclassified instances.

For the ODP database the cosine text kernel CC was second best for both the smaller modified setup and the transductive setup. The $JFLA$ matrix was best in both setups, with 10 percent less misclassified instances in the web setup and 30 percent less misclassified instances in the transductive setup.

		PT	CC	GLI	GLO	$GLCO$	$JFLA$	$PLSA$
WebKB	error	0.254	0.250	0.256	0.255	0.257	**0.242**	0.256
$\alpha = 0.8$	precision	0.422	0.426	0.422	0.420	0.421	**0.454**	0.423
kNN =10	recall	0.415	0.411	0.412	0.399	**0.413**	0.400	**0.413**
smaller modified	edge ratio	*	1.586	0.694	1.189	1.087	1.472	1.800
WebKB	error	0.254	0.343	0.345	0.354	**0.194**	0.418	0.329
$\alpha = 0.7$	precision	0.422	0.290	0.349	0.367	**0.462**	0.349	0.399
no kNN	recall	0.415	0.245	0.323	0.269	**0.461**	0.205	0.270
transductive	edge ratio	*	0.709	0.741	0.655	0.623	0.529	0.728

Table 7: Performance of several base matrices for WebKB

		PT	*CC*	*GLI*	*GLO*	*GLCO*	*JFLA*	*PLSA*
ODP	error	0.246	0.236	0.244	0.244	0.244	0.248	0.240
$\alpha = 0.8$	precision	0.647	0.656	0.648	0.648	0.648	0.659	0.651
$kNN = 30$	recall	0.627	0.635	0.628	0.628	0.628	0.624	0.632
smaller modified	edge ratio	*	2.512	2.313	2.202	2.202	1.005	1.917
ODP	error	0.246	0.238	0.246	0.246	0.246	**0.222**	0.253
$\alpha = 0.7$	precision	0.647	0.654	0.647	0.647	0.647	**0.673**	0.641
$kNN = 50$	recall	0.627	0.633	0.627	0.627	0.627	**0.647**	0.621
smaller modified	edge ratio	*	2.355	1.793	1.461	1.822	1.926	1.672
ODP	error	0.246	0.215	0.236	0.236	0.236	**0.174**	0.485
$\alpha = 0.9$	precision	0.647	0.670	0.655	0.656	0.655	**0.693**	0.432
no kNN	recall	0.627	0.653	0.635	0.635	0.635	**0.687**	0.427
transductive	edge ratio	*	1.196	1.069	1.543	1.107	1.104	1.073

Table 8: Performance of several base matrices for ODP

5.4.2 Results Discussion

One observation is that the link anchor texts are a really good information source, as indicated first in the work of Fürnkranz [5]. It is difficult to select a single best similarity measure for all databases, so that is why combining all of them may bring more, as will be seen in the next section.

As noted in 3.4.1, the results show the performance with the best kNN parameter only, which was tested from 10 to 90 for the smaller modified setup and 10 to 260, as well as no nearest neighbour, for the transductive setup. The optimal neighbourhood parameter for each database may be determined by testing on a small subset set aside for this purpose, holdout data, and taking the value which was overall best for most of the different matrices. For the transductive setup, the nearest neighbour methods did not

work well, so no neighbour edges trimmed was the best kNN setting. The edge ratio is a bad predictor for matrix performance. A possible explanation is that class distributions play a role as well and just the average values of edges are not enough to predict good matrix performance, even though this seems like a logical rule. The recall and precision values follow similar trends like the error rate and don't require a special separate result interpretation.

6 Combining All Similarity Edge Information into a Single Matrix

In this section a novel method to combine the available similarity matrices and construct a flexible local similarity measure is presented. Each edge between each pair of data points is defined as an object for a second classification problem with the goal of inferring a similarity matrix better representing class structure. Available information from different similarity measures and web page features is used to find a similarity matrix better representing data label structure. Each pair-wise data similarity value is represented as an edge between data points, with a feature vector combining information from the similarity measures and the features of the edge endpoints. A linear discriminant is trained to give more weight to edges connecting web pages with the same label than those edges connecting web pages with different labels, and thus obtain a similarity matrix that will improve classification performance on the original web page categorization task.

6.1 Constructing an Edge Feature Vector

6.1.1 New Feature Representation

When one considers how many different similarity measures are useful for improving performance, it seems a promising idea to create a combined affinity matrix incorporating all the different information sources we have - the initial class labels and the different affinity matrices. This is a secondary classification problem with the goal of learning the best affinity matrix for optimal classification accuracy in the primary problem of web page classification. We define a new feature ϕ for each edge between pairs of data points. It is a

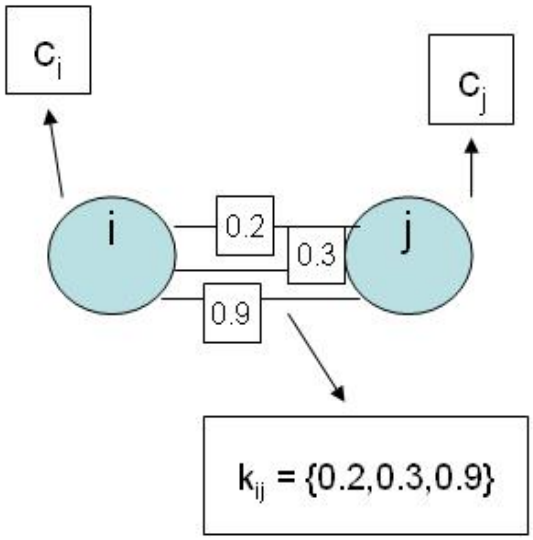

Figure 17: The new edge feature vector

Cartesian product between all the values for this edge in the affinity matrices and the labels for the classes in both edge endpoints. The exact motivation for such a formulation is discussed in Section 6.1.2. The Cartesian product operation on two vectors u and v produces a vector w with dimension the product of the dimensions of u and v, and the entries of w are all ordered pairs of entries of u and v. For example $(1,2) \times (1,2,3) = (1,2,3,2,4,6)$. If we have data points or web pages i and j, we may gather their different similarity measures in a vector k_{ij}. The class label information for each of the two edge endpoints may be gathered in vectors c_i and c_j. The new feature representation for the edge between i and j will be then:

$$\phi_{i,j} = k_{ij} \times k_{ij} \times c_i \times c_j \tag{8}$$

This is a novel representation combining features from both vertices and edges in the graph, and treating the label predictions of classifiers as features.

These different types of information involved in the feature are shown in figure 17, where an edge representation from three different similarity measures is shown, and the class distribution of the endpoints c_i and c_j is also taken into account. The dimension of the new feature is k^2c^2, where k is the number of different affinity matrices, or in other words different similarity measures, between the web page data points. The dimension of the data point feature vectors is c, which is exactly the number of different labels in the simple case implemented here.

6.1.2 Motivation of the Different Feature Components

The intuitive reason behind this Cartesian-product representation with both data point and edge feature is to allow classification of the edges with rules like "if the first endpoint is of this class, and the second from that class, and the similarity value of this measure between them is so large, then this is for sure an in-class edge". The reason behind the $k_{ij} \times k_{ij}$ term, is to have features which capture the cooccurance of different similarity measures, i.e. allow rules like "if this similarity value is large, and the other is not too small, than the combined similarity value should be so large". Similarly, the idea of the whole equation 8 is to allow weights for similarity values to vary locally as a function of data features. This allows for quite flexible local similarity functions. In contrast, previous attempts to combine similarity functions as convex combinations, for example in the work of Argyriou et al. [2], are much less flexible to adapt to local data neighbourhoods and the convex combination of similarity matrices corresponds to just the first part of our Cartesian product edge feature, the vector k_{ij}. The vector c_i can be enhanced to contain more information, like class information obtained as output from different learners, or even more differently defined feature vectors. The basic

form uses just the label predictions from the plain text learner, and the other possible learners are different plain text learners, or even more fundamentally different classifiers, like the Hyperlink Ensemble from Fürnkranz [5] and the iterative label relaxation from Lu and Getoor [10]. As one may expect, using more features presents more information for edge classification, because each learner may classify better or worse on different data types, but having too many features may also lead to overfitting and worse results on the edge classification, besides being more costly computationally.

6.2 Defining Edge Classification as a Learning Problem

6.2.1 Linear Discriminant Classifier

We have constructed the new feature representation, now the point is to learn from it a model to distinguish class edges that connect data points with the same labels, from those edges that are between two different labels. A learner should take an edge as input and give a real number as output, as shown in figure 18. For the consistency framework to work, it is enough that on the average edges in same class have greater value than cut edges. This is a slight simplification of the problem, since we need to be correct only on the average about the two edge types, not for each edge individually.

We want a linear discriminant w to distinguish between edges in the cut and in the same class. This is a rather simple classification function, but it often works well if we have an appropriate input vector space, and we assume that the new feature representation forms such a space. The connection of input space and separating hyperplane is further discussed in Section 6.3.3. Traditionally linear discriminants are used in statistics to find the linear

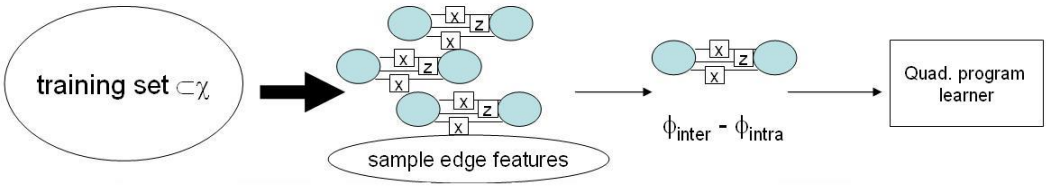

Figure 18: Learning edge values

combination of features which best separate two or more classes of objects. In our concrete scenario it should give large values to edges within same class, i.e. intra edges or edges outside of the cut, and small ones to edges between different labels, i.e., inter edges or edges inside of the cut. We need a vector w where the value of $w^t\phi$ is telling for the class of the edge ϕ - inter or intra class edges should receive more or less weight respectively. We define ϕ_{intra} and ϕ_{inter} as the average data point of all edges between training points with same class labels and between different labels respectively. The discrimination itself is modelled as a quadratic program:

$$min\,(\phi_{inter} - \phi_{intra})^t\, w + \beta w^t w \tag{9}$$

The first summand ensures that on the average the within class edges will have greater value than the between class ones. This is a slight relaxation of the 0-1 pattern criterion, but it is also necessary for a good affinity matrix, since activation is spread linearly from all data points. Another advantage of this scheme is that we have a quadratic problem with a complexity in the dimension of the constructed feature, which is constant. Very large data sets may be handled by sampling random edges, with no significant accuracy loss. For the WebKB and ODP datasets, sampling just 5 percent of the total edges was enough for calculation of ϕ_{inter} and ϕ_{intra} resulting in improved combined

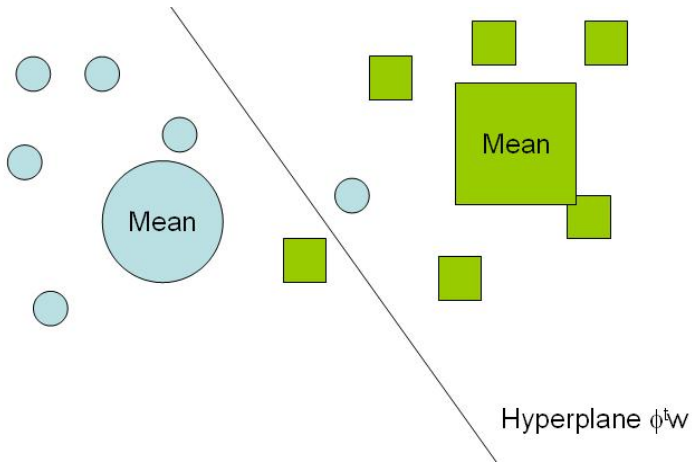

Figure 19: Hyperplane of linear discriminant

similarity matrix and the results in Section 6.5 are with such sampling.

The function of the second term $w^t w$ is to have a smooth hyperplane w which does not overfit. Intuitively, $w^t w$ corresponds to the squared Euclidean norm, where very large values are penalized. If we used the 1-norm $|w|$, which is just the sum of absolute values of the components of w, we may get an overfitting linear discriminant, which has a very large component corresponding to the smallest component of $\phi_{inter} - \phi_{intra}$ and a very small one to the largest value. The value of β is not too critical for the performance and has been set to 1 in the experiments.

The weight vector w represents a hyperplane in the vector space of the ϕ feature. Solving the optimization problem produces a hyperplane that separates optimally the mean vector of all edges connecting web pages with the same label from the mean vector of all edges connecting web page with a different label, as can be seen in figure 19. After w has been found, finding the test data affinity matrix W is straightforward - for each edge e with endpoints i and j and feature ϕ, multiply $w^t \phi$, and store the values in the affinity

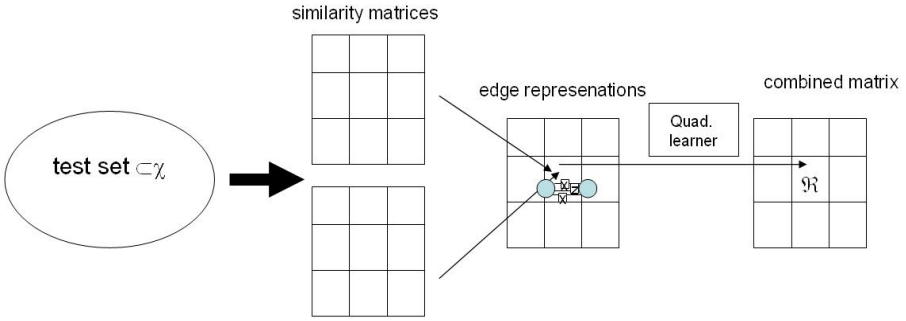

Figure 20: Inferring similarity matrices for test set, from input matrices to new edge representations to final matrix.

matrix entry W_{ij}. Afterward normalize W and rescale so that the minimum value is 0, to avoid negative entries, which would violate the definition of similarity measure. Note that the vector w, representing weights for the different features and defining the hyperplane of discrimination, is our learner function. The whole process of finding W is illustrated by figure 20. We examine the different similarity matrices of the test set data points, create the new edge representations, and then infer the final combined similarity matrix using our learner, in this case a simple linear discriminant.

6.2.2 Finding the linear discriminant as quadratic optimization problem

Optimizing (9) is made easier by quadratic optimization theory, which proves that quadratic programming problems are convex and have global minimums when the quadratic term is a positive semidefinite matrix. In the case of (9), this is the identity matrix. When there are no inequality constraints, differentiating with respect to w and setting the gradient to 0 delivers a straightforward analytical solution: $w = -\left(\phi_{inter} - \phi_{intra}\right)/(2\beta)$ It is an interesting variation to demand that all the entries of w are positive and

add this as inequality constraints to the quadratic program. This is some indirect knowledge encoding that all of the similarity matrices denote some similarity and we can't have components of ϕ that denote dissimilarity and are thus weighted negatively. When inequality constraints are present, we need a quadratic program solver, for example active set or interior point solver. There are some theoretical complexity results for such solvers, but in practice the solvers perform much better, so the theoretical bounds are not too informative. The interior point Matlab solver needs less than 10 seconds for an edge feature of dimension more than 3000, so it is not hurting overall performance time. Note also that the whole definition of the linear discriminant working on the mean cut and no cut edges scales extremely well with large databases. We only need to sample enough edges to get the mean vectors, and the edge feature dimension is constant regardless of the number of web pages to be classified.

6.3 Discussion of Alternatives to the Linear Discriminant

6.3.1 Classical Instance Based Learning Methods

A naive approach will be to train on each of the edges from the training data a model to predict the class of each edge as 0 or 1. However, this is not practical. The number of edges is quadratic to the number of points and even for the relatively small WebKB and ODP databases we will have hundreds of thousands of edges. The traditional learner functions, for example classifiers from WEKA like Naive Bayes or SVM or Decision Trees, are both too slow on so many training edges and can't generalize well on the test data edges. We did experiments with this and the performance was too bad to be of

interest as method for learning combined similarity measure. For the success of learning with similarity matrix methods to work, it is not so important that each edge is perfectly classified as being in the cut or outside of the cut. Only the relative difference of all the edges matters, and a strict binary 0/1 classification of a more traditional approach focusing on single instances is not the best formulation of the task to get a better activation spreading matrix for consistency learning.

6.3.2 Comparison with a SVM edge classifier

Support Vector Machines suffer from the same drawbacks of binary classifiers described in the previous section. The classical SVM formulation for our problem of edge classification would be:

$$\begin{aligned} \min\ & w^t w + C \sum_e \xi_e \\ \text{for edge e outside cut}\ & w^t \phi_e \geq 1 - \xi_e \\ \text{for edge e inside cut}\ & w^t \phi_e \leq 0 + \xi_e \end{aligned}$$

Each of the inequalities has the function to control the algorithm to give a correct classification for this edge, and gives a penalty ξ when the edge is on the wrong side of the hyperplane. The drawbacks of SVM for our secondary classification problem, the edge classification, is that we have too many edges and they can't all be included in the SVM formulation due to the enormous computational cost for solving a quadratic problem with so many inequalities. We need to sample the edges to still have a computationally viable SVM formulation, and sampling from so many data points, around one million for the WebKB dataset, is a difficult task in itself. SVM formulations are also better suited for binary classifications, and it is not trivial to interpret

their classification results as smooth probabilities for the class of the edge. If desired one may try to mix the linear discriminant formulation with the SVM one and add support vectors to (9) in the form of inequalities for the edges, but this slows down the algorithm considerably and leads to no significant improvement. Thus, it was not tested for the final experiments. It may be investigated further which edges are appropriate to include as support vectors.

6.3.3 Discussion of Consistency Learning vs. SVM with Kernels on the First Classification Problem

One of the reasons for the popularity of SVM's in the machine learning community is that they may employ flexible pairwise data similarities for classification. Through the use of kernel functions, the original data space is transformed to a higher dimensional one. The Cartesian product feature we constructed for edges corresponding to pairs of data points, bears some resemblance to kernel learning in a SVM formulation of the first web page classification problem. More specifically the polynomial kernel $(x_i^t x_j + 1)^p$ for some parameter p has some similarities. It can be expanded as a polynomial with terms the vector entries of x_i and x_j . The simple Cartesian product $x_i \times x_j$ is equivalent to a polynomial kernel with $p = 2$. Both our Cartesian product feature and kernels used in SVM learning increase the dimension of the feature space and allow classification with a hyperplane, which could not be possible in lower dimensions. This is illustrated in figure 21, where the circle optimal decision boundary in the initial 2-dimensional space is a hyperplane in 3-dimensional space. A kernel corresponding to similarity in the 3-dimensional space will allow a hyperplane to separate well the two classes, which was not possible in 2-dimensional space, where no single line

(hyperplane in two dimensions) can separate the classes optimally.

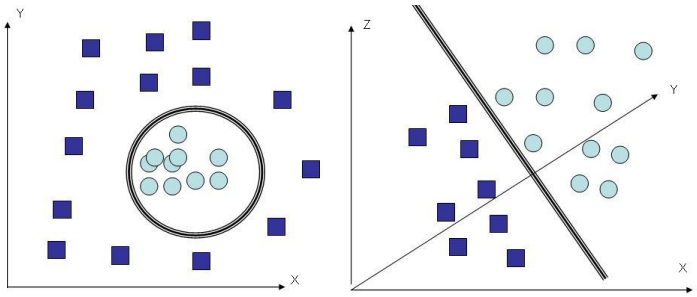

Figure 21: High dimensional feature spaces and classifying hyperplane

However, empirically SVM performed just as good as or even worse than the simple Naive Bayes text classifier for flat text classification on our databases ODP and WebKB, and had much higher training time. For this reason SVM is not the best choice for initial flat text learner for the particular web page collections used here, but may be excellent for other databases. Another possible problem is that not all similarity measures may be easily expressed as a kernel function between data feature vectors. We start with data similarity features that have some meaning by themselves, like a certain property of the database neighbourhood. It will be an unnecessary complication of the model to have to go from pairwise data similarities back to feature vectors of the individual data points. It is also not clear how to incorporate nearest neighbour graph matrix modifications. Another issue is SVM uses the kernel similarity between all training data points and each test data point to make a classification, and in the domain of web page classification edges between test data points are crucial for good performance. When we already have good similarity matrices, the multiple text similarity measures and neighbourhood similarity functions from the previous section, it is better to try to find some manifold structure and use them via con-

sistency learning. All those considerations did not allow for an easy SVM formulation incorporating all our features and similarity matrices, but this may be an interesting topic for further research.

6.4 From Multiclass to Binary Class Classification

We may also consider one-against-all binary classification as an alternative to multiclass classification. This means that we will try to solve several subproblems, each label against all other labels combined, and choose as the final classification the label with greatest predictive confidence. The loss function of the consistency framework $Q(F)$ is linear in the dimension of the different class labels c. In other words, it is independent in the matrix dimension corresponding to different class labels. We can reformulate the task of minimizing $Q(F)$ as the task of minimizing all $Q(F_{.i})$ separately. Each of these problems may be solved iteratively for each class i:

$$F_{.i}(t+1) = \alpha S F_{.i}(t) + (1-\alpha) Y_{.i} \tag{10}$$

We may split the problem of finding the optimal F^* into several subproblems of finding optimal column vector $F_{.i}$ for each class label i. In this way, we split the original multiclass problem into several one vs. all binary classification problems. The final class is the one where the binary classifier has the largest value. If we model the problem like this, we may train discriminants for each class label by taking intra and inter edges involving data points with this label. The assumption is that edges separated like this in subproblems have a better structure and are easier to discriminate correctly. The problem of an overrepresented class spreading too much activation will be better dealt with in this way. The intuition is that the binary discriminants are better trained on edges involving at least one endpoint from class i, as shown in

figure 22. Edges with both endpoints in other classes may be learned worse, but this is not so crucial, since data points from other classes will have low starting activation in class i, assuming the initial classifier is good enough. It is important that data points from class i are activated with edges weighted much from other data points in class i, and data points from other classes connected to class i are activated with edges with low values. This we avoid false positives and false negatives. How points from other classes are activated is not immediately clear, the discriminant did not even train on them, but since those have low activation in class i anyway, it is unlikely they will receive activation in class i larger than in all other classes.

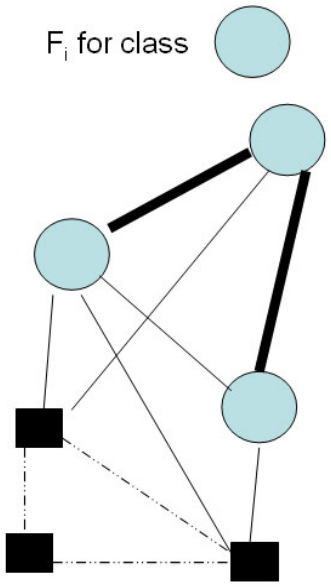

Figure 22: Edge weights for Binary Consistency Learning

6.5 Experiments with Combined Features

6.5.1 Experimental Results

The tables 9 and 10 show how well learned combined similarity matrices performed. The vector k in equation 8 has information from all six different similarity matrices defined in the previous section. The abbreviation $MLMA$ means multiclass problem, i.e. a single learned similarity matrix was used for activation spreading for all different classes. $BCLMA$ means that for each class a different matrix was learned. Both $MLMA$ and $BCLMA$ use the analytic solution of the edge discriminant problem and allow negative weights. $MLMP$ and $BCLMP$ are analogous, but with the extra condition in the quadratic program that $w \geq 0$. To use such extra conditions, a quadratic program solver was used, which brings a slightly bigger computational cost, but also improves performance. The consistency learning parameter α was set to 0.8 for all experiments.

Looking at the result tables, one may observe that for WebKB the edge combination methods work well and achieve the best performance in the smaller modified setup for this database with 11 percent less misclassified instances, better than all single matrices. The transductive setup is much worse for WebKB than the smaller modified setup, so there is no use of learning a combined matrix for the setup with both training and test data points.

For ODP, the results are shown in table 10. The combination method $BCLMP$ in smaller modified setup manages to improve the performance of Naive Bayes with 12 percent less misclassified instances, slightly better than the best single matrix, $JFLA$, which had 10 percent improvement. For the transductive setup for ODP $MLMP$ has 24 percent less misclassified

instances than plain text classification, which is worse than the transductive result of $JFLA$ alone, 30 percent less misclassified instances than plain text classification. This still shows the possible benefit of matrix combination methods when having multiple matrices. Tuning of α, β and kNN might improve the combination method performance even better.

		PT	$MLMA$	$BCLMA$	$MLMP$	$BCLMP$
WebKB	error	0.254	0.239	0.240	**0.227**	0.232
$\beta = 1$	precision	0.422	0.431	0.433	**0.466**	0.461
$kNN10$	recall	0.415	0.410	0.409	0.410	0.407
smaller modified setup	edge ratio	*	1.997	1.977	1.859	1.973
WebKB	error	0.254	0.237	0.242	0.232	0.245
$\beta = 1$	precision	0.422	0.432	0.429	0.459	0.429
$kNN50$	recall	0.415	**0.412**	0.411	0.408	0.408
smaller modified setup	edge ratio	*	1.970	1.833	1.939	1.759
WebKB	error	0.254	0.248	*	0.244	*
$\beta = 1$	precision	0.422	0.429	*	0.430	*
$kNN10$	recall	0.415	0.408	*	0.408	*
transductive setup	edge ratio	*	1.061	*	1.578	*

Table 9: Performance of several different matrix combination methods for WebKB

6.5.2 Results Discussion

A possible explanation why combination methods work well for WebKB in the smaller modified setup is that this database comes from a structured domain of university computer departments. Probably there is university-specific logic for the placement of links between course and faculty web page, for example. The smaller modified setup exploits the graph structure only of the test points, which all are from the same university, so this contributes

		PT	$MLMA$	$BCLMA$	$MLMP$	$BCLMP$
ODP	error	0.246	0.228	0.230	0.224	**0.217**
$\beta = 4$	precision	0.647	0.661	0.659	0.666	**0.671**
$kNN50$	recall	0.627	0.642	0.640	0.645	**0.650**
smaller modified setup	edge ratio	*	2.242	2.023	2.228	2.036
ODP	error	0.246	0.195	*	**0.188**	*
$\beta = 4$	precision	0.647	0.682	*	**0.688**	*
$kNN110$	recall	0.627	0.669	*	**0.674**	*
transductive	edge ratio	*	1.163	*	1.164	*

Table 10: Performance of several different matrix combination methods for ODP

to good activation flow. ODP is a database where the different subsets are sampled randomly, so having a transductive setup where both training and test points are together helps with the extra true labels.

The methods using a strictly positive hyperplane vector w are slightly better than the others, so the assumption that all similarity measures should be combined with positive coefficients was correct.

The methods learning several binary matrices can be better than those with a single matrix for all classes, but this comes at a computational cost and further experiments may be needed to utilize the full potential of binary class methods. The binary classification methods were not tested in the transductive setup, because they use as many matrices as they are different class labels. Multiplying the memory usage by 5 in the ODP case or 6 for WebKB is too impractical for the slight performance gains, but is still a viable option for the smaller modified setup computationally and may lead to interesting edge learning algorithms in the future.

One may also notice that the edge ratios are improved over the ratios of

the constituent basic similarity matrices, shown in tables 7 and 8, so the linear discriminant learned successfully the task of improving this ratio. However, as noted already, these edge ratios are not a good guarantee for classification performance, so the combined matrices for ODP are worse than some of the basic matrices regardless of the edge ratio.

7 Final Words

7.1 Conclusion

This work examined the consistency algorithm for smooth web page classification. Different similarity measures applicable to web pages and their influence on the performance of a text based classifier were examined in two setups, a smaller one building a graph on the test data points only and a larger transductive setup on both the test and training points. On WebKB, the $JFLA$ similarity measure based on hyperlink anchors had 5 percent improvement in the smaller setup, and the $GLCO$ similarity based on co-cited web page neighbours had 24 percent improvement in the transductive setup. For ODP, $JFLA$ had 10 percent improvement over plain text classification in the smaller setup with test points only, and 30 percent in the transductive setup. With careful parameter selection, consistency learning can outperform plain text web page classification significantly, but requires careful parameter selection and more memory and processing time, especially for the transductive setup.

A novel algorithm for similarity matrix combination was also evaluated and tested. Different features, some of them predicted classes from basic text learners and others taken from the data, its link graph and various data similarity measures, are combined for a novel graph edge representation. Edges are learned via an efficient discriminant method and an affinity matrix better representing the combined data similarity was found. The learned matrix is computationally fast to learn, and can have competitive performance with any of the matrices used to compute it, even outperforming them in the smaller setup.

More experiments will be needed to get the best of the consistency learn-

ing methods, and to tune them finely, but they are clearly very useful for improving classification accuracy in the presence of additional information in the form of pairwise similarities.

7.2 Further Enhancements

7.2.1 Theoretical Framework Enhancement

The methods described in this thesis can be fine-tuned in many different ways. More similarity measures may be defined and tested, for example using a nonlinear kernel between vectors instead of cosine similarity as mentioned in Section 5.3 . Each of the six similarity measures used may be modified with such kernels or other modifications, for example different text vector spaces than TFIDF. The $PLSA$ kernel may be tuned more finely with different number of concepts, or replaced with a Singular Value Decomposition semantic analysis algorithm. Some more work may be done to improve the neighbourhood structure of the similarity matrices and reduce noisy edges. There are methods to inspect more carefully the local neighbours of a graph vertex and delete edges that are most probably connecting objects from different classes, as described in the work of Xu et al. [21]. The work of Gao et al. [6] also presents an interesting algorithm to find outliers in local neighbourhoods, defined as an optimization problem. Further work may also be done to find a measure of matrix quality that may predict the consistency learning gain better than the edge ratio. Different learning methods may be tested with the new edge features, like efficient SVM formulations, or other fast discriminant methods like Fisher Linear Discriminant.

7.2.2 Practical Applications in Different Domains

The framework described here for web page classification may be easily modified to solve interesting problems in other areas. For example, the domain of scientific papers and the papers quoted by them is again a graph with directed links between documents. A web crawler of an online digital library, like `http://citeseer.ist.psu.edu/`, may create a database of documents linked to other documents and then apply the classification methods from this thesis.

Another interesting application may be in Bioinformatics and classification of enzyme functions. The different similarity measures come from different biological origins: DNA microarrray gene expressions, phylogenetic gene profiles and localization of genes in different cell compartments, as presented in the work of Vert and Yamanishi [18]. All of those measurements are expressed as vectors of numbers and may give rise to various similarity measures.

A Appendix: Graph Laplacian

Let W be some similarity matrix of dimension $n \times n$ with positive entries. Define the degree matrix D as a matrix with $d_{ii} = \sum_{j=1}^{n} w_{ij}$ for $i = 1...n$. Define $L := D - W$ as the unnormalized Graph Laplacian. Then we have:

$$\sum_{i,j} w_{ij}(y_i - y_j)^2 = \frac{1}{2}\sum_{i,j} w_{ij}(2y_i^2 - wy_iy_j)) =$$

$$= \sum_i y_i^2 d_i - \sum_{i,j} w_{ij}y_iy_j = \sum_{i,j} y_iy_jd_{ij} - \sum_{i,j} y_iy_jw_{ij} =$$

$$= y^tDy - y^tWy = y^tLy$$

L satisfies the definition of positive semidefiniteness - $y^tLy \geq 0$ for all vectors y, because it can be rewritten as a sum of positive numbers, $(y_i - y_j)^2$.

If we want, we may introduce the normalized Graph Laplacian $D^{\frac{-1}{2}}LD^{\frac{-1}{2}}$ and

$$y^tD^{\frac{-1}{2}}LD^{\frac{-1}{2}}y = y^tD^{\frac{-1}{2}}(D - W)D^{\frac{-1}{2}}y =$$

$$= \sum_{i,j} w_{ij}(y_i - y_j)^2 = \sum_{i,j=1}^{n} W_{ij} \parallel \frac{1}{\sqrt{D_{ii}}}y_i - \frac{1}{\sqrt{D_{jj}}}y_j \parallel^2$$

This corresponds to weighting each vector entry y_i by its total edge sum and so overly connected data points won't have too large influence. In the equation (2) y is named F. Thus, the equivalence of equations (2) and (3) was shown.

B Appendix : Global Minima for Convex Quadratic Programs

Theorem: For a quadratic program of the form $minf(x) = xQx^t + cx + b$ with the Hessian Q positive semidefinite, $f : Z \longrightarrow R^n$ has a global minima at the point x^*, where $\frac{df}{dx^*} = 0$.

Proof: We are minimizing without side conditions, so we know that the region Z is convex. Then for all $x, y \in Z$ and $t \in$ [0,1] $(1-t)x + ty \in Z$. Then set $s = y - x$ and the Taylor expansion of $f(x)$ can be written for some $t \in [0, 1]$ as:

$$f(y) - f(x) = \nabla f(x)^t s + \frac{1}{2} s^t \nabla^2 f(x + ts)s \geq \nabla f(x)^t s = \nabla f(x)^t s(y - x)$$

The inequality follows because $\nabla^2 f$ is the matrix Q in the case of our quadratic function f and Q was positive semidefinite. This means that for a x^* that satisfies $\nabla f(x) = 0$ we have $f(y) \geq f(x^*)$ and this fits the definition of a global minimum for x^*. This finishes the proof.

The gradient $\nabla f(x) = Qx$ and $f(y) - f(x) \geq \nabla f(x)^t s(y - x)$ is a necessary and sufficient condition for f being a convex function, where every local minima is a global minima, but this is a simplified proof, with the notation of Ulbrich [17] aimed specifically at our learning problems without side conditions. To present a proof about quadratic programs with side conditions one needs to define the KKT conditions and prove that every local minima satisfying them is a global minima when the Hessian matrix Q of the quadratic program is positive semidefinite. For brevity such a proof is omitted here, but may be found for example in Nocedal and Wright [11].

C Appendix: Iterative Consistency Equation

We want to show that the iterative equation $F(t+1) = \alpha S F(t) + (1-\alpha)Y$ will converge to the matrix $(I - \alpha S)^{-1} Y$. The initial value in the first round of the iteration for $F(0)$ is Y. The iteration equation may be rewritten as

$$F(t) = (\alpha S^{t-1})Y + (1-\alpha)\sum_{i=0}^{t-1}(\alpha S)^i Y$$

$\lim_{t\to\infty}(\alpha S^{t-1}) = 0$ as shown in the Lemma below and $\lim_{t\to\infty}\sum_{i=0}^{t-1}(\alpha S)^i = (I - \alpha S)^{-1}$ Thus, $F^* = \lim_{t\to\infty} F(t) = (1-\alpha)(I-\alpha S)^{-1}Y$. We may drop the constant in front since it does not affect classification, and thus we proved the validity of the iterative method.

Lemma: $\lim_{t\to\infty}(\alpha S^{t-1}) = 0$

Proof: Suppose that ν is eigenvector to eigenvalue λ for the matrix $D^{\frac{-1}{2}}(W)D^{\frac{-1}{2}}$. This means that

$$D^{\frac{-1}{2}} W D^{\frac{-1}{2}} \nu = \lambda\nu$$

After multiplication from the left with $D^{\frac{-1}{2}}$ we get

$$D^{\frac{-1}{2}} D^{\frac{-1}{2}} W D^{\frac{-1}{2}} \nu = \lambda D^{\frac{-1}{2}} \nu$$

Thus $D^{\frac{-1}{2}}\nu$ is eigenvector to eigenvalue λ for the matrix $D^{-1}W$ This means that the matrix $S = D^{\frac{-1}{2}}(W)D^{\frac{-1}{2}}$ has the same eigenvalues as the matrix $D^{-1}W$, i.e. they are similar. $D^{-1}W$ is a stochastic matrix, i.e. the row sums are 1, so it is well known that its eigenvalues are smaller or equal to 1 in absolute value. Thus, taking in consideration that α is smaller than 1, $\lim_{t\to\infty}(\alpha S)^{t-1} = 0$. The reason is that $\lim_{t\to\infty}\alpha^{t-1}$ will converge to 0, and $\lim_{t\to\infty} S^{t-1}\nu = \lim_{t\to\infty}\lambda^{t-1}\nu$ can't have any entries tending to infinity due to the eigenvalues being smaller or equal than 1 in absolute value.

D Appendix: more Result Tables

The tables in this appendix illustrate that fine tuning of the α and kNN parameters influences the classification performance. $kNN30$ means that the nearest neighbour parameter was set to 30, analogously for other kNN values. The abbreviation PT means plain text classifier performance, which is the baseline. The results for the smaller modified setup are in Tables 11 for WebKB and 13 for ODP, using in both cosine text similarity. For the transductive setup larger kNN parameters are better. For ODP the transductive setup works really well and the text cosine similarity matrix brings much greater performance gain than in the other setup, as seen in table 14. A probable explanation is that the extra training data points with their labels bring correct activations. For WebKB. the extra data points in the graph worsen performance, as shown in table 12. A possible reason may be that the web pages from different universities have different features even if from same class labels, a course web page in different universities may have different link structure. Table 15 is an example where a slight activation influence by a small α may bring performance improvements.

		PT	$\alpha = 0.1$	$\alpha = 0.3$	$\alpha = 0.5$	$\alpha = 0.7$	$\alpha = 0.9$
$kNN10$	error	0.254	0.254	0.251	0.253	0.252	0.250
	precision	0.422	0.422	0.424	0.426	0.427	0.430
$kNN30$	error	0.254	0.253	0.251	0.251	0.249	**0.243**
	precision	0.422	0.424	0.426	0.427	0.425	**0.432**
$kNN50$	error	0.254	0.253	0.250	0.248	**0.243**	0.253
	precision	0.422	0.424	0.427	0.430	0.428	0.294
$kNN70$	error	0.254	0.253	0.251	0.247	0.246	0.257
	precision	0.422	0.424	0.425	0.432	0.428	0.403
$kNN90$	error	0.254	0.253	0.250	0.248	0.245	0.272
	precision	0.422	0.424	0.427	**0.432**	0.429	0.409

Table 11: Performance of text cosine similarity matrix for WebKB, smaller modified setup

		PT	$\alpha = 0.1$	$\alpha = 0.3$	$\alpha = 0.5$	$\alpha = 0.7$	$\alpha = 0.9$
$kNN10$	error	0.254	0.254	0.258	0.256	0.253	0.356
	precision	0.422	0.423	0.423	0.422	0.426	0.342
$kNN60$	error	0.254	0.253	0.250	0.271	0.271	0.321
	precision	0.422	0.424	0.426	0.408	0.384	0.285
$kNN110$	error	0.254	0.255	0.254	0.281	0.287	0.326
	precision	0.422	0.422	0.426	0.411	0.392	0.308
$kNN160$	error	0.254	0.254	**0.249**	0.271	0.306	0.321
	precision	0.422	0.424	**0.429**	0.411	0.368	0.271
$kNN210$	error	0.254	0.255	0.251	0.262	0.304	0.320
	precision	0.422	0.422	0.420	0.409	0.385	0.251

Table 12: Performance of text cosine similarity matrix for WebKB, transductive setup

		PT	$\alpha = 0.1$	$\alpha = 0.3$	$\alpha = 0.5$	$\alpha = 0.7$	$\alpha = 0.9$
$kNN10$	error	0.246	0.244	0.244	0.244	0.240	0.240
	precision	0.647	0.648	0.648	0.648	0.651	0.651
$kNN30$	error	0.246	0.244	0.242	0.240	0.242	0.238
	precision	0.647	0.648	0.651	0.653	0.652	0.656
$kNN50$	error	0.246	0.244	0.242	0.236	0.236	0.236
	precision	0.647	0.648	0.651	0.657	0.658	0.657
$kNN70$	error	0.246	0.244	0.244	**0.234**	**0.234**	**0.234**
	precision	0.647	0.648	0.648	0.659	**0.660**	0.659
$kNN90$	error	0.246	0.244	0.242	**0.234**	**0.234**	**0.234**
	precision	0.647	0.648	0.650	0.659	0.659	0.659

Table 13: Performance of text cosine similarity matrix for ODP, smaller modified setup

		PT	$\alpha = 0.1$	$\alpha = 0.3$	$\alpha = 0.5$	$\alpha = 0.7$	$\alpha = 0.9$
$kNN10$	error	0.246	0.246	0.242	0.236	0.244	0.375
	precision	0.647	0.647	0.647	0.652	0.641	0.523
$kNN60$	error	0.246	0.246	0.240	0.234	0.236	0.255
	precision	0.647	0.647	0.649	0.654	0.648	0.630
$kNN110$	error	0.246	0.244	0.236	0.224	0.230	0.226
	precision	0.647	0.648	0.651	0.657	0.658	0.657
$kNN160$	error	0.246	0.244	0.236	0.226	0.232	0.209
	precision	0.647	0.648	0.655	0.661	0.654	0.672
$kNN210$	error	0.246	0.244	0.232	0.228	0.228	**0.195**
	precision	0.647	0.648	0.659	0.660	0.659	**0.685**

Table 14: Performance of text cosine similarity matrix for ODP, transductive setup

		PT	$\alpha = 0.1$	$\alpha = 0.3$	$\alpha = 0.5$	$\alpha = 0.7$	$\alpha = 0.9$
$kNN10$	error	0.246	0.244	0.230	0.226	**0.219**	**0.219**
	precision	0.647	0.648	0.658	0.663	0.667	0.670
$kNN30$	error	0.246	0.244	0.230	0.224	0.224	0.240
	precision	0.647	0.649	0.659	0.663	0.672	0.667
$kNN50$	error	0.246	0.246	0.234	0.226	0.222	0.244
	precision	0.647	0.647	0.657	0.662	**0.673**	0.665
$kNN70$	error	0.246	0.246	0.238	0.230	0.232	0.263
	precision	0.647	0.647	0.653	0.660	0.665	0.659
$kNN90$	error	0.246	0.246	0.238	0.228	0.234	0.267
	precision	0.647	0.647	0.653	0.663	0.664	0.659

Table 15: Performance of $JFLA$ similarity matrix for ODP, smaller modified setup

References

[1] R. Angelova and G. Weikum. Graph-based text classification: Learn from your neighbours. In *Proceedings of SIGIR 2006*, pages 485–492.

[2] A. Argyriou, M. Herbster, and M. Pontil. Combining graph laplacians for semi-supervised learning. In *Advances in Neural Information Processing Systems 18*, pages 67–74, Cambridge, MA, 2006. MIT Press.

[3] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[4] S. Chakrabarti, B. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. In *SIGMOD Conference*, pages 307–318, 1998.

[5] J. Fürnkranz. Hyperlink ensembles: a case study in hypertext classification. *Information Fusion*, 3(4):299–312, 2002.

[6] J. Gao, H. Cheng, and P.-N. Tan. Semi-supervised outlier detection. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 635–636. ACM Press, 2006.

[7] T. Hofmann. Probabilistic Latent Semantic Indexing. In *Proceedings of the 22nd Annual ACM Conference on Research and Development in Information Retrieval*, pages 50–57, Berkeley, California, August 1999.

[8] G. Lanckriet, M. Deng, N. Cristianini, M. I. Jordan, and W. S. Noble. Kernel-based data fusion and its application to protein function prediction in yeast. In *Proceedings of the Pacific Symposium on Biocomputing, January 3-8, 2004. pp. 300-311.*, pages 300–311.

[9] D. P. Lewis, T. Jebara, and W. S. Noble. Nonstationary kernel combination. In *ICML '06: Proceedings of the 23rd International Conference*

on Machine Learning, pages 553–560. ACM Press, 2006. ISBN 1-59593-383-2.

[10] Q. Lu and L. Getoor. Link-based text classification. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*.

[11] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer-Verlag, 1999.

[12] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[13] S. Seiermann. Vergleich von methoden zur hypertext-klassifikation. Master's thesis, TU Darmstadt, 2007. URL `http://www.ke.informatik.tu-darmstadt.de/lehre/arbeiten/diplom/2007/Seiermann_Stefan`.

[14] P. Sen and L. Getoor. Empirical comparison of approximate inference algorithms for networked data. In *International Conference on Machine Learning Workshop on Statistical Relational Learning (SRL)*, 2006.

[15] A. Stahl and T. Gabel. Using evolution programs to learn local similarity measures. In *International Conference on Case-Based Reasoning*, pages 537–551, 2003.

[16] M. Szummer and T. Jaakkola. Partially labeled classification with markov random walks. In *Neural Information Processing Systems*, pages 945–952, 2001.

[17] S. Ulbrich. *Nonlinear Optimization*. TU Darmstadt, 2005.

[18] J.-P. Vert and Y. Yamanishi. Supervised graph inference. In *Advances in Neural Information Processing Systems*, pages 1433–1440, Cambridge, MA, 2005. MIT Press.

[19] U. von Luxburg. *Statistical Learning with Similarity and Dissimilarity Functions*. PhD thesis, TU Berlin, 2004.

[20] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. J. Russell. Distance metric learning with application to clustering with side-information. In *NIPS*, pages 505–512, 2002.

[21] Y. Xu, X. Yi, and C. Zhang. A random walks method for text classification. *SIAM Conference On Data Mining, Maryland, USA*, 2006.

[22] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Scholkopf. Learning with local and global consistency. *Advances in Neural Information Processing Systems*, 16:321–328, 2004.