Transductive Learning for Document Classification and Handwritten Character Recognition

Leonidas Lefakis
Utrech University

Supervisors : Marco Wiering and Arno Siebes

Contents

1 Introduction

In the traditional supervised approach to learning, the supervised algorithm is presented with a set of inputs $X^{train} = (x_1, x_2...x_n)$ and corresponding outputs $Y^{train} = (y_1, y_2...y_n)$, this set $D^{train} = (X, Y)$ is known as the training set and the examples x_i belonging to this set are known as labeled examples. In general any data point whose corresponding output (label) is known is a labeled data point.

Using this training set D^{train}, supervised learning aims to create a learner L which having learned the relevant (and presumably sufficient) information from the training set D^{train} will be able to make predictions on the output of a set of inputs X^{test} whose corresponding output Y^{test} is presumably unknown and which we are interested in discovering. Examples whose corresponding output is unknown are known as unlabeled examples.

It is evident from the above that the supervised learning setting takes only advantage of labeled data during the training phase. In order for the training to be successful, i.e. the resulting learner L performs well on the unlabeled and previously unseen data, the training set must provide the learner with the necessary information. This often means that the training set must be of considerable size, as this is linked to its information content. The number of necessary training examples rises with the complexity of the learning algorithm as well as with the dimensionality of the data; the latter is a well known problem in machine learning known as the curse of dimensionality.

If the number of labeled examples is not sufficient then the danger of overfitting increases. Overfitted learners have adapted to irrelevant attributes of the training data which are not representative of the problem to be solved in general and thus the learner is not able to generalize well which translates to poor performance on examples outside the training set.

The above problem of overfitting can be remedied with a sufficiently large training data set. Unfortunately the acquisition of training examples is a costly endeavor. Training examples usually have to be manually labeled which is a time-consuming process. Peeking for a moment at the text classification problem to be presented below, we can see exactly how costly this process can be; each example to be labeled must be read by a human researcher (who could obviously be spending his time more productively) and then labeled. The process of reading the text itself can ,in certain cases, be extremely time-consuming as a text could easily consist of dozens of pages. Furthermore this labeling of examples, inserts a certain bias into the training procedure; though in most cases the subject (class/label) of a text is obvious, in other cases it is not as evident what the text's subject is (if it is about hardware or software for example); labeling such an example is ultimately based on the subjective opinion of the human labeler and this subjectivity invariably enters the training of the learner.

For these reasons it is agreeable to be able to conduct learning with as few labeled examples as possible. In contrast to labeled examples, there is a plethora of unlabeled examples readily available, for example the world wide web is full of texts that can be easily obtained with little to no cost. It is only the labeling process which is costly. Thus if we were able to use this unlabeled data to supplement the labeled data in the learning process then the problem of insufficient data could be overcome.

In cases where little labeled data is present while large amounts of unlabeled data are available, instead of supervised learning, semi-supervised learning [23] algorithms are preferably used. In semi-supervised learning, we have a training set $X^{train} = (x_1, x_2...x_n, x_{n+1}...x_{n+m})$ for which only a small number n of the corresponding outputs (labels) are known, i.e. $Y^{train} = (y_1, y_2...y_n)$ while for the rest of the examples $(x_{n+1}...x_{n+m})$ the corresponding outputs are unknown. Using these training examples the aim of semi-supervised learning algorithms is to construct a learner L that can generalize well and predict the output of unseen examples X_{test} whose labels are unknown but which are not the same examples as the subset $X^u = (x_{n+1}...x_{n+m}) \subseteq X_{train}$ whose labels are also unknown. Research in the field of semi-supervised learning, has shown that the addition of unlabeled examples to the training data set can help improve the performance of the learner when only small quantities of labeled examples exist.

As can be seen, just as in the case of supervised learning, semi-supervised learning aims to create a learner L which can generalize well on unseen data. The assumption is thus usually made that the training set is representative of the general problem (for example the assumption is often made that the training examples have been generated independently and identically according to an unknown but fixed distribution which is also assumed to reflect the problem in hand in general). This form of learning is based on inductive reasoning. Inductive reasoning uses observations (which are by nature limited in number) to reach more general conclusions. An example of such reasoning can be seen below :

All crows observed (by say the scientific community) are black.
Thus all crows (in existence) are black.

Having reached a more general rule (All crows are black) it is then possible to use deductive reasoning to reach conclusions about specific instances :

John's pet is a crow.
John's pet is black.

As can be seen from the above example the conclusions of deductive reasoning are logically sound. If all crows are black, and John's pet is a crow then it follows that John's pet must be black. In the case of inductive reasoning however the premise cannot ensure the (logical) validity of the conclusion. This becomes evident if we slightly alter our example :

All crows observed (by me) are black.
Thus all crows (in existence) are black.

It is obvious from the above that the assumption that the second clause holds true based on the first, cannot be one of high confidence.

This follows closely (and is in fact inspired by) the argument set forth by Bertrand Russell in his book "Problems of Philosophy" [16].

... the newness of the knowledge is much less certain if we take the stock instance of deduction that is always given in books on logic, namely, 'All men are mortal; Socrates is a man, therefore Socrates is mortal.' In this case, what we really know beyond reasonable doubt is that certain men, A, B, C, were mortal, since, in fact, they have died. If Socrates is one of these men, it is foolish to go the roundabout way through 'all men are mortal' to arrive at the conclusion that (probably) Socrates is mortal. If Socrates is not one of the men on whom our induction is based, we shall still do better to argue straight from our A, B, C, to Socrates, than to go

round by the general proposition, 'all men are mortal'. For the probability that Socrates is mortal is greater, on our data, than the probability that all men are mortal. (This is obvious, because if all men are mortal, so is Socrates; but if Socrates is mortal, it does not follow that all men are mortal.)

In inductive learning a function $f(x)$ is learned which is capable of predicting the correct output for the training data (observed examples) and it is assumed to be able to correctly predict the corresponding labels of unseen data. Both semi-supervised and supervised learning use inductive inference based on which they create learners L which hopefully will perform well on unseen data. This however entails making assumptions that are not logically safe.

The problem of unnecessarily solving a general problem in a machine learning setting was pointed out by Vladimir Vapnik [20] who also introduced the notion of transductive learning. In transductive learning, instead of aiming to compute a function $f(x)$ that will perform well on unseen data, the aim is to calculate directly the output on seen, albeit unlabeled data. Thus as in the case of semi-supervised learning, a transductive learner takes as input a training set $X^{train} = (x_1, x_2 ... x_n, x_{n+1} ... x_{n+m})$ for which only a small number n of the corresponding outputs (labels) are known, $(Y^{train} = (y_1, y_2 ... y_n))$ while for the rest of the examples $(x_{n+1} ... x_{n+m})$ the corresponding outputs are unknown; however unlike semi-supervised learners, transductive learners aim to find the labeling of the seen unlabeled examples $(x_{n+1} ... x_{n+m}) \in X^{train}$.

As can be seen transductive learning not only overcomes the problem of few labeled data examples but also eliminates the need for making assumptions which may or may not prove to be valid. For all these reasons we have chosen to focus the present project on transductive learning.

In particular we focus on transductive learning when applied to two well known and important problems : handwritten character recognition and document classification. Both these problems are well suited for experimenting with transductive learning as they both concern areas where labeled data can prove to be scarce. Furthermore both these problems are intensely researched and of considerable scientific interest and are both an integral part of a great variety of real-world applications. It is perhaps characteristic that the performance of semi-supervised algorithms (and by extension transductive algorithms) are almost invariably tested on a document classification problem so that in fact their performance on document classification serves as a kind of benchmark for the semi-supervised (and transductive) learning algorithms.

As a large number of transductive algorithms are presented via their application to document classification, we initially focused the project on this problem and investigate the performance of a variety algorithms on it. This allows for the experimentation with a number of algorithms in a setting where their application is more or less straightforward as they have been more or less tailored (where necessary) to the characteristics of the specific problem. We use the Newsgroup dataset to create a binary classification problem and experiment with a number of algorithms in order to compare between them but also to investigate whether, in this case, transductive learning can be useful, this usefulness translating to whether transductive learning algorithms do in fact surpass in performance supervised algorithms when the number of labeled data is small.

Having overviewed via document classification a number of well-known transductive algorithms and their performance on a binary classification problem, the project then focuses to applying these same algorithms to the handwritten character recognition problem. This allows us not only to observe the performance of these algorithms on an important application but also allows us to observe their performance on a multilabel problem. In order to experiment with transductive learning and handwritten character recognition we must also investigate how exactly these algorithms can be applied as this is not always straightforward.

Finally besides experimenting with applicable alternatives of the different transductive learning algorithms, we also propose and experiment with a novel approach to transductive learning for handwritten character recognition inspired by a well known supervised algorithm : convolutional networks which are known to perform extremely well on object recognition problems.

Through the research and the experimentation we hoped to ultimately answer the questions that served as the original impetus of this project.

Research Questions Does transductive learning in fact lead to higher performance when labeled data is scarce, as this can be investigated on document classification and which of these algorithms proves ultimately to be best suited for this particular problem? Furthermore can these algorithms be adapted to apply to handwritten character recognition, which proves best suited in this case and finally does the convolutional network analogous proposed here in fact increase performance?

Overview In the following, we experiment with document classification in chapter 2, while handwritten character recognition is investigated in chapter 3. Finally in chapter 4 the conclusions of the project are presented.

2 Document Classification

2.1 Introduction

An application for which transductive learning seems to be well suited and in which it proves to perform particularly well is that of document classification. Document classification aims to create classifiers that given a document d_i will be able to successfully predict its class y_i. This can be seen as predicting the subject of a document given a number of potential subjects. For example a document classifier can be used to discern whether a specific document talks about sports, religion or politics.

These classifiers prove to be very useful in many real-world applications. With the growth of the world wide web, a need has risen for search engines that can intelligently search the web for pages with specific content, a document classifier could look at the content of these pages (the text to be exact) and decide whether the subject of the specific pages is one that interests the user, if the specific user is known to systematically read articles on basketball then it would be useful to be able to find articles whose subject is basketball.

Another application which is closely related is that of recommending systems where the classes the documents are classified to are no longer subjects but rather the binary labels like/dislike. Thus for example a document classifier could learn to recommend movies to a user based on the summaries of previously seen and enjoyed movies.

Obviously a real-world system would be quite complicated and have a number of components beyond the document classifier but nonetheless the document classifier would be an integral part of the overall system.

All these intelligent systems however useful are in need of data to be trained. As stated in the introduction and is evident, this data is not always easy to come by. In the case of web pages the tedious work of reading through the content and deciding on the subject must be executed manually in order to obtain labeled data. In the case of recommending systems the acquisition of labeled data is even more difficult, as the labeling is no longer performed by a researcher who will willy-nilly complete the task but rather by the user himself who will be unwilling to perform significantly time-consuming tasks before he obtains the desired results.

This paucity of labeled data, makes the field of document classification especially suited for transductive learners. As we shall see by the results of the experiments conducted in the following sections, transductive learners are able to overcome the lack of labeled data and construct document classifiers that attain surprisingly high performance despite this lack.

2.2 Data

In our experimentations with transductive learning and document classification, we used the Newsgroup dataset. This dataset is comprised of approximately twenty thousand messages taken from twenty different Usenet newsgroups. These twenty different newsgroups fall into roughly six different categories, with the newsgroups belonging to the same category being closely related amongst themselves; these categories are : computers, science, recreation, politics, religion and miscellaneous.

In order to obtain a numerical representation of the data, which is originally in text form, the documents must be mapped against a dictionary. We thus obtain an initial representation of each document consisting of a vector, each coordinate of which corresponds to a specific word in the dictionary, with the value of that coordinate being the number of occurrences of the corresponding word in the document.

To this effect, the documents are parsed and the words that are encountered are used as a basis for the creation of the dictionary. While parsing the documents and creating the dictionary, a number of steps are taken in order to improve on the dictionary and the quality of the resulting representation (using the TMG Matlab toolbox [21]).

First, a number of words are encountered which consist of non-alphabetic symbols, though some of these may well represent words written with deviant spelling (as for example when using leet, 1337, language) it is very difficult to discern between these and nonsense and thus all words containing non-alphabetic symbols are stricken from the dictionary.

Second, a number of words which are considered excessively common are also removed as they are considered not to contain any important information concerning the classification of the texts they appear in as they are very likely to appear in almost all texts. Examples of such words are ”the” and ”and” whose appearance obviously has little to do with the topic at hand. This is accomplished by the use of a list of such common words.

Finally a technique named stemming is used to further refine the dictionary. Simply parsing the text and recording the appearing words, would lead to a dictionary (and a subsequent representation) which marked as different words that in fact differed only in their appearance but which were in fact identical. For example the words ”talk” and ”talks” would be considered distinct when in fact there are in fact the one and the same. Stemming uses a number of rules in order to replace words by their common roots thus rendering a much more meaningful representation of the data.

Having parsed the text using the rules explained above, we obtain the experimental dataset consisting of 18828 examples each with 80662 attributes. Because of the computational cost involved, the complexity of most transductive learning algorithms and the size of the original dataset, a number of steps were taken to reduce the size of this dataset in order to make it more practical for experimentation.

First, the dataset was reduced by only using a subset of the dataset. That is to say not all newsgroups were ultimately used. More specifically we used posts from four of the newsgroups (using however all the posts in each of these newsgroups); in particular, we used two newsgroups with subjects regarding computers (comp.graphics and comp.os.windows.misc) and two newsgroups with subjects regarding science (sci.crypt and sci.electronics). Furthermore in order to aid in the reduction of the complexity of the learning algorithms these four newsgroups were grouped into two groups of two newsgroups each (one computer related, the other science related) and thus the entire problem was reduced from a multiclass one to a binary classification problem. Thus the resulting experimental dataset consists of 3930 examples divided into two classes, one with 1958 examples and the other 1972 examples (i.e. a ratio of 50.18 % - 49.82 %).

Besides reducing the number of examples used, the dimensionality of the data was also reduced. The number of attributes (80662) of the original dataset is excessive and can lead to a number of problems. On one hand an excessively high number of dimensions results in a similar increase in computational demands. On the other hand there is also the problem of the so called "curse of dimensionality". As the dimensions of the data grow, the risk of overfitting also increases; overfitting leads to a learner that gives weights to features that do not in fact yield significant information about the class of the data. In order to avoid this phenomenon we must have examples proportional to the number of dimensions. Dimensionality reduction could thus not only decrease computational costs but hopefully lead to an increase in performance.

In order to reduce the dimensionality of the data, term selection techniques are used which attempt to find a subset T' of the terms T which will hopefully lead to an efficient resulting representation of smaller size. In particular the method applied here is the so called filtering approach which ranks terms according to a certain measure and keeps those terms that score the highest. In general these measures try to capture the intuition that the terms that must be preserved are those that are distributed most differently across classes.

In order to experiment with term selection, we used a variety of measures to compute the relative importance of each term. The measures used can be seen below, together with there mathematical forms [17].

Information gain $IG(t_k, c_i) = \sum_{c \in c_i, \bar{c_i}} \sum_{t \in t_k, \bar{t_k}} P(t, c) \cdot log \frac{P(t,c)}{P(t)P(c)}$

Mutual Information $MI(t_k, c_i) = log \frac{P(t_k,c_i)}{P(t_k)P(c_i)}$

Chi-square $\chi^2(t_k, c_i) = \frac{|T| \cdot [P(t_k,c_i) \cdot P(\bar{t_k},\bar{c_i}) - P(t_k,\bar{c_i}) \cdot P(\bar{t_k},c_i)]^2}{P(t_k) \cdot P(\bar{t_k}) \cdot P(c_i) \cdot P(\bar{c_i})}$

Odds ratio $OR(t_k, c_i) = \frac{P(t_k|c_i) \cdot (1 - P(t_k|\bar{c_i}))}{(1 - P(t_k|c_i)) \cdot P(t_k|\bar{c_i})}$

GSS coefficient $GSS(t_k, c_i) = P(t_k\,|c_i) \cdot P(\bar{t_k}\,|\bar{c_i}) - P(t_k\,|\bar{c_i}) \cdot P(\bar{t_k}\,|c_i)$

NGL coefficient $NGL(t_k, c_i) = \frac{\sqrt{|T|} \cdot [P(t_k,c_i) \cdot P(\bar{t_k},\bar{c_i}) - P(t_k,\bar{c_i}) \cdot P(\bar{t_k},c_i)]}{\sqrt{P(t_k) \cdot P(\bar{t_k}) \cdot P(c_i) \cdot P(\bar{c_i})}}$

Each of these measures returns for each term t_k and for each class c_i a corresponding value. However in order to decide which terms t_k to keep, we must have a value for each term t_k irrespective of the classes. In order to calculate these values we experimented with the use of following criteria [17]:

Max The maximum value observed for each term t_k is kept :

$$f_{max}(t_k) = max_{i=1}^{|C|} f(t_k, c_i)$$

Sum The sum of values over all classes are kept for each term t_k :

$$f_{sum}(t_k) = \sum_{i=1}^{|C|} f(t_k, c_i)$$

Weighted Sum The sum of values is weighted by the prior probabilities of each class :

$$f_{wsum}(t_k) = \sum_{i=1}^{|C|} P(c_i) \cdot f(t_k, c_i)$$

The results of these experimentations using the different term selection methods can be seen in figures 1-6. In the experiments conducted a fraction of the dataset was set aside as a test set while the rest of the dataset was used to train a Naive Bayes Classifier. The resulting classifier was then used to classify the examples in the test set. In figures 1-6, the y axis denotes the accuracy attained by the Naive Bayes Classifier while the x axis denotes the number of terms kept.

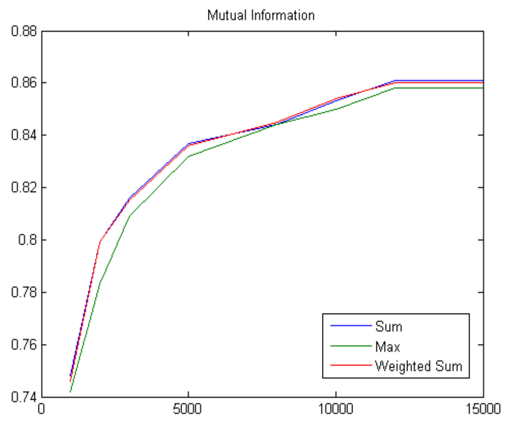


Figure 1

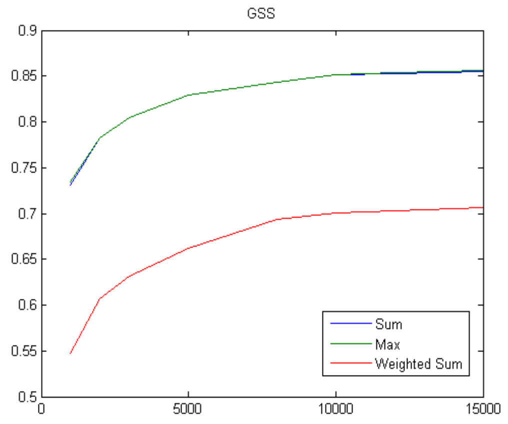


Figure 2

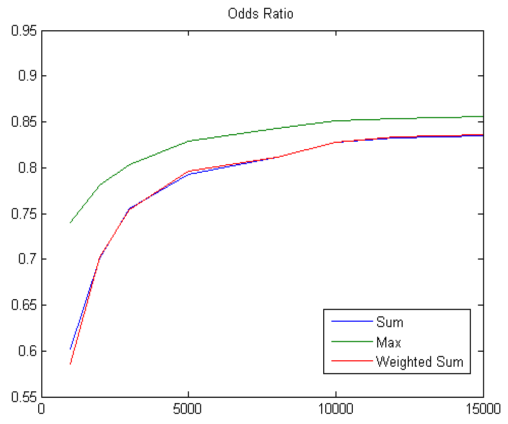


Figure 3

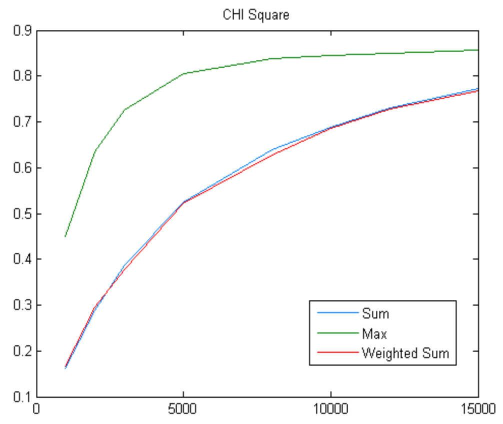


Figure 4

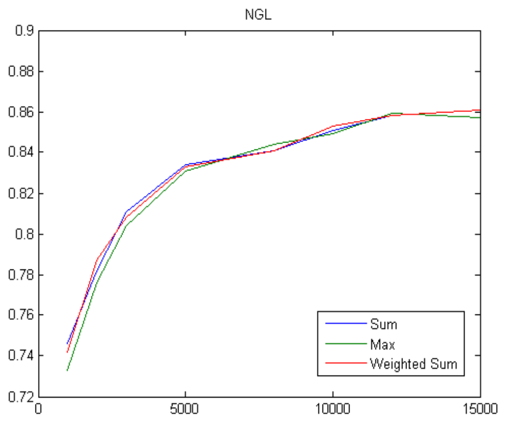


Figure 5

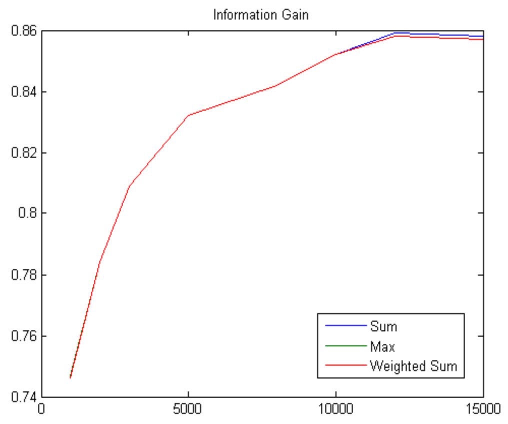


Figure 6

For each measure, we choose that criterium of value selection that yields the best results and then compare these results between them (Figure 7) in order to find that technique that will allow us to reduce the dimensionality of the data as much as possible without leading to a parallel decrease in the performance of the learning process.

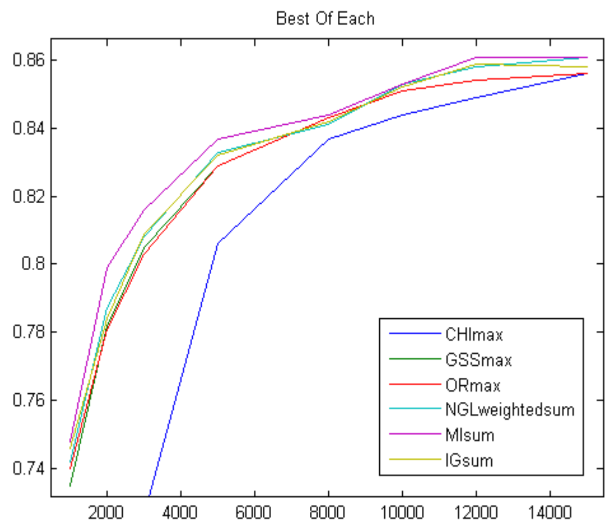


Figure 7

As can be seen for the approximately 6000 attributes which we would like to keep as a practical dimensionality of the dataset, mutual information proves to be the best measure for reducing the dimensionality of the data with only a relatively small (2%) decrease in performance.

Using the mutual information measure, we acquire the dimensionality reduced dataset which will be used for experimentations for those algorithms which make use at some stage of a Naive Bayes Classifier.

For the remaining algorithms (for example the support vector machines), the attributes of each example are weighted. When it comes to the occurrence of a specific word in a specific document, what interests us most is not the number of occurrences as this may easily be influenced by the documents length but rather what is more characteristic is the number of occurrences relative to the occurrences of the other words in the document. Thus instead of using a representation of the data using raw occurrences, an alternate representation is used employing term frequencies and inverse document frequencies which is known as tf-idf.

Tf-idf takes into account not only the frequency of the term in the specific document but also its frequency in the corpus. Thus a word which appears frequently throughout the corpus is not considered to yield considerable information about the class of the documents it appears in, while an infrequent term is considered more likely to be particular to the specific document and thus more indicative of its class. To this effect tf-idf takes into account the inverse document frequency of each term defined as :

$$idf_{ik} = \log \frac{|D|}{|t_k \ni d|}$$

where $|D|$ is the number of documents in the corpus and $|t_k \ni d|$ is the number of those documents in which t_k appears. The frequency of term k in document d_i is given by :

$$tf_{ik} = \frac{n_k}{\sum\limits_j n_j}$$

where n_k is the number of occurrences of term k in document d_i and the sum $\sum\limits_j n_j$ is over all terms j appearing in d_i.

Finally the tf-idf weight of the term k in document d_i is given as $tfidf_{ik} = tf_{ik} \cdot idf_{ik}$.

2.3 Supervised Learning Algorithms

In order to experiment whether in fact supervised algorithms perform poorly on the specific experimental dataset and for comparison with the transductive learners we use two different supervised learners, a Naive Bayes Classifier [11] and a Support Vector Machine [3].

2.3.1 NBC

Naive Bayes Classifiers make the assumption that the data has been created via a parametric model and estimate the parameters of the model based on a Bayesian theoretical framework. Having calculated the parameters, using the training data, the classifier then classifies any unlabeled examples (test set) using Bayes theorem i.e. by calculating which component of the model is most likely to have generated the example (posterior probability).

More analytically and in the case of text classification in particular, the data is considered to be generated by a mixture model with parameters ϑ and consisting of a number of mixture components $c_j \in C = \left\{c_1, c_2...c_{|C|}\right\}$. Under this framework, the probability of a specific document is given as the sum of probabilities of the document over all mixture components [11]:

$$P(d_i|\vartheta) = \sum_{j=1}^{|C|} P(c_j|\vartheta)P(d_i|c_j;\vartheta)$$

We can furthermore assume that each mixture component corresponds to exactly one class (and vice versa) and thus by c_j we indicate not only the component j but also the jth class. Thus the probability of each document is the sum of the probabilities of it being generated by the component that corresponds to each class.

In the experiments conducted here, a multinomial model is assumed which considers each document an ordered series of word events drawn from the same vocabulary V. Besides the assumption concerning the nature of the model, we make two further assumptions. First that the length of the document is independent of class and second that the probability of each word event is independent of the word's context and position in the document. Based on the above, the model calculates the probability of a document given its class using the formula[11] :

$$P(d_i\,|c_j;\vartheta) = P(|d_i|)\,|d_i|!\prod_{t=1}^{|V|}\frac{P(w_t|c_j;\vartheta)^{N_{it}}}{N_{it}!}$$

The parameters of the generative component for each class are the probabilities for each word, $\vartheta_{w_t|c_j} = P(w_t|c_j;\vartheta)$ and are given by[11] :

$$P(w_t\,|c_j;\vartheta) = \frac{1+\sum_{i=1}^{|D|} N_{it}P(c_j|d_i)}{|V|+\sum_{s=1}^{|V|}\sum_{i=1}^{|D|} N_{is}P(c_j|d_i)}$$

The class prior parameters are $\hat{\vartheta}_{c_j} = P(c_j|\vartheta)$ and are given by [11]:

$$P(c_j|\hat{\vartheta}) = \frac{\sum_{i=1}^{|D|} P(c_j|d_i)}{|D|}$$

Having calculated the Bayes-optimum estimates for these parameters using the training set, we can then proceed to classifying the test examples using the acquired classifier. This is accomplished by applying Bayes' rule [13]:

$$P(c_j|d_i;\hat{\vartheta}) = \frac{P(c_j|\hat{\vartheta})P(d_i|c_j;\hat{\vartheta})}{P(d_i|\hat{\vartheta})}$$

The document is then classified to that class j which returns the highest probability.

In figure 8 we can see the performance of the Naive Bayes classifier. As can be seen, though the algorithm performs reasonably well when the number of labeled examples per class is close to 100, when the number of labeled examples is low, the algorithm fails to reach high performance. In each case the classifier is tested on the remaining examples. Thus when 100 labeled examples are used per class, then the resulting classifier is tested on the remaining 3930-200=3730 examples. In order to obtain representative results, for each number of labeled examples per class, the experiments are repeated 10 times, each time the labeled examples being chosen by drawing with a uniform distribution without replacement from the set of 3930 examples.

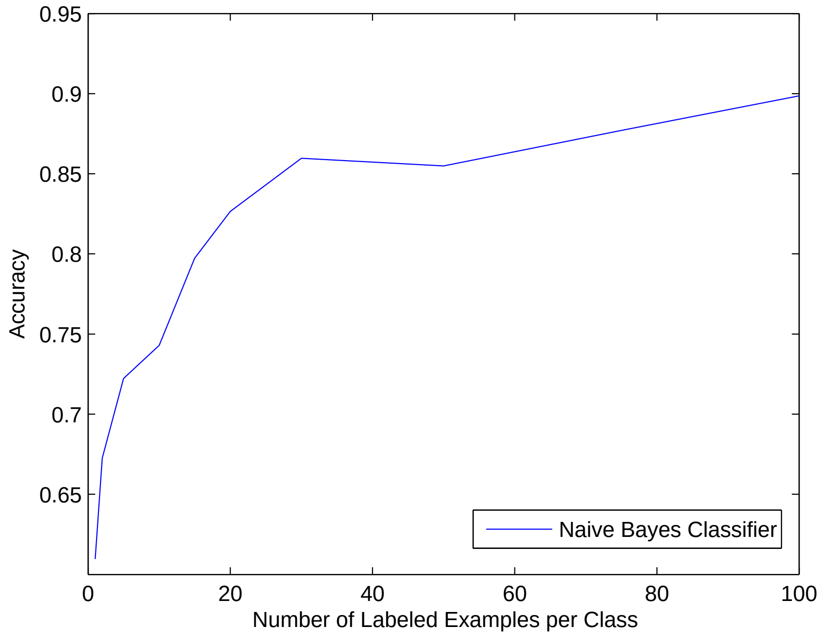


Figure 8

As can be seen by the results obtained the Naive Bayes classifier performs quite well when the number of examples is sufficient, reaching an accuracy of 90 percent when the number of labeled examples per class are 100, when the number of examples per class are few the accuracy of the classifier falls dramatically (as can be expected). This seems to be a first indication that document classification may very well benefit from transductive learning as at least this supervised algorithm performs unsatisfactorily for small amounts of labeled data.

2.3.2 SVM

Support Vector Machines have been used in a wide variety of applications and have been shown to attain high performance in most cases. They belong to a class of classifiers known as linear classifiers which aim to learn the solution to classification problems using a linear function. Specifically the aim of linear classification machines is given a set of examples (input) $\mathbf{x} = (x_1, x_2 ... x_n)$ and a corresponding set of labels (output) $\mathbf{y} = (y_1, y_2 ... y_n)$ where in the binary classification case $y_i \in \{-1, 1\}$, to learn a function $f(x) =< w \cdot x > +b$ such that $f(x_i) \geq 0$ if $y_i = 1$ and $f(x_i) < 0$ if $y_i = -1$.

The problem however of finding a separating hyperplane such that instances belong to one class lay on ones side of the hyperplane and instances belonging to the other class lay on the opposite side, does not have a single solution. Thus if the data is linearly separable, then there are an infinite number of functions $f(x) =< w \cdot x > +b$ that given a set of examples and corresponding labels, satisfy the restrictions $f(x_i) \geq 0$ if $y_i = 1$ and $f(x_i) < 0$ if $y_i = -1$ [4].

The simplest model of a support vector machine, works only for linearly separable data, is the so called maximal margin classifier. The margin of an example (x_i, y_i) with respect to a hyperplane (w, b) is defined to be the quantity:

$$\gamma_i = y_i(< w \cdot x_i > +b)$$

If $\gamma_i > 0$ then from the above equation it is evident that the example x_i is classified correctly by the function $f(x)$. By extension the margin distribution of a hyperplane (w, b) with respect to a training set $S = ((x_i, y_i), ..., (x_l, y_l)) \subseteq (X \times Y)$ is the distribution of the examples in S. The minimum of the margin distribution (i.e. the minimum margin over all the examples of the set S) is the margin of the hyperplane (w, b) with respect to the training set S.

By normalizing the linear function we obtain the separating hyperplane $(\frac{1}{\|w\|}w, \frac{1}{\|w\|}b)$ and the corresponding geometric margin which is in fact the Euclidean distance of the points from the decision boundary in the input space [4]. The margin of a training set S is the maximum geometric margin over all hyperplanes and the corresponding hyperplane is the maximal margin hyperplane.

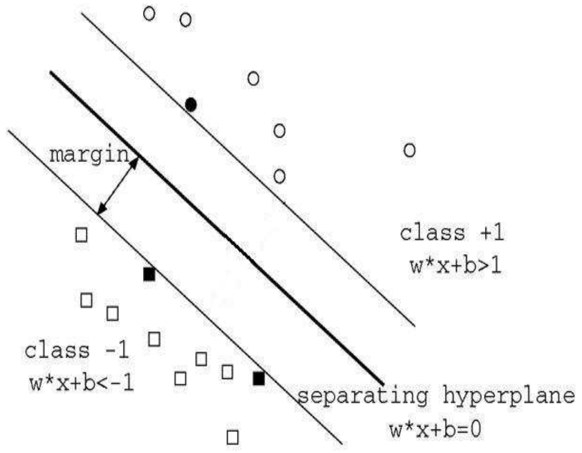


Figure 9

Thus a simple support vector machine solves the linear separation problem by finding a separating hyperplane such that the minimum geometric margin is maximal. This translates to the hyperplane whose distance from the closest points (to it) in the input space is as large as possible.

By fixing the functional margin to one we can optimize the corresponding the geometric margin by minimizing the norm of the weight vector. This is evident if we consider the following. Let us suppose that the closest positive point to the hyperplane is x^+ and the closest negative x^-, this implies [4] :

$$< w \cdot x^+ > +b = +1$$
and
$$< w \cdot x^- > +b = -1$$

and the geometric margin is given by [4]:

$$\begin{aligned}\gamma = \tfrac{1}{2}(< \tfrac{w}{\|w\|_2} \cdot x^+ > - < \tfrac{w}{\|w\|_2} \cdot x^- >) = \\ = \tfrac{1}{2\|w\|_2}(< w \cdot x^+ > - < w \cdot x^- >) = \\ = \tfrac{1}{\|w\|_2}\end{aligned}$$

Thus the problem can be transformed to the equivalent optimization problem:

$$\begin{gathered}\text{minimize} < w \cdot w > \\ \text{subject to } y_i(< w \cdot x_i > +b) \geq 1\end{gathered}$$

Unfortunately it is seldom the case that the examples of set S are linearly separable. In most cases it is not possible to find a hyperplane (w, b) such that the examples belonging to each class lay on opposite sides of the hyperplane. In order to overcome this problem and create a support vector machine that can solve the maximal margin classification problem even in the cases where the data is not linearly separable, we first introduce the notion of the margin slack variable of an example (x_i, y_i) [4].

For a specific fixed value $\gamma > 0$ the margin slack variable ξ_i of an example (x_i, y_i) with respect to a hyperplane (w, b) and the target margin γ is defined as :

$$\xi_i = max(0, \gamma - y_i(< w, x_i > +b))$$

This quantity can be seen as a measure of how much the specific point (x_i) fails to have a margin γ from the hyperplane (w, b). If $\xi > \gamma$ then obviously x_i is misclassified by the hyperplane (w, b).

The norm $\|\xi\|$ measures the amount by which the training set fails to have a margin of γ, taking into account any misclassifications that may appear in the training data.

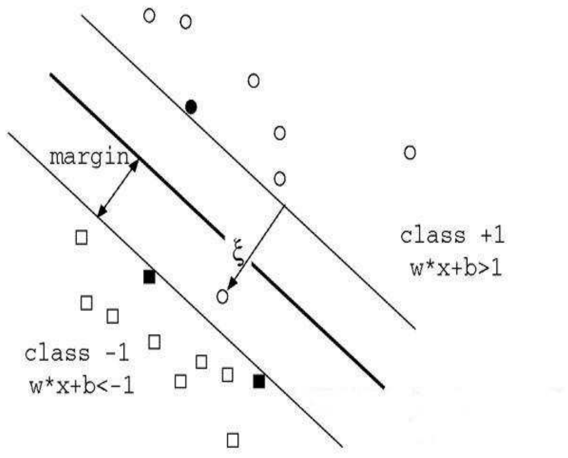


Figure 10

With the help of the slack margin variable we can restate the optimization problem to allow for the margin constraints to be violated. This gives the so called soft margin optimization problem which is [4]:

$$\text{minimize} < w \cdot w > +C \sum_{i=1}^{l} \xi_i$$
$$\text{subject to } y_i(< w \cdot x_i > +b) \geq 1 - \xi_i$$

This is the 1-norm soft margin case which will be used throughout the experiments conducted here. In the 2-norm case the objective function to be minimized is : $< w \cdot w > +C \sum_{i=1}^{l} {\xi_i}^2$.

Programming SVM As stated above, a support vector machine (in the 1-norm soft margin case) ultimately solves the optimization problem.

$$\text{minimize} < w \cdot w > +C \sum_{i=1}^{l} \xi_i$$
$$\text{subject to } y_i(< w \cdot x_i > +b) \geq 1 - \xi_i$$

The problem stated above is presented in the so called primal form. Before we continue to the actual programming of the support vector machine we first transform the optimization problem to its dual form using Lagrange multipliers [4]. Thus the optimization problem becomes :

$$\text{maximize } W(a) = \sum_{i=1}^{l} a_i - \tfrac{1}{2} \sum_{i,j=1}^{l} y_i y_j a_i a_j K(x_i, x_j)$$
$$\text{subject } \sum_{i=1}^{l} y_i a_i = 0$$
$$\text{and } 0 \leq a_i \leq C$$

where $K(x, z)$ is a kernel function which implicitly maps the input to a feature space. In the case where no kernel is used we have : $K(x, z) =< x \cdot z >$. The bias b no longer appears in the dual form but is given by the equation [4]:

$$b^* = -\frac{max_{y_i=-1}(w^* \cdot x_i) + min_{y_i=1}(w^* \cdot x_i)}{2}$$

where $w^* = \sum y_i a_i^* x_i$ is the solution to the optimization problem.

Having converted the optimization problem to its dual form, we can proceed to programming the support vector machine. The simplest approach is to use gradient descent, meaning that the vector a is sequentially updated following the steepest ascent in the direction of the gradient of $W(a)$.

Using this method, the examples of the training set are sequentially presented to the support vector machine and the component a_i is updated by the rule :

$$a_i = a_i + \eta_i (1 - y_i \sum_{j=i}^{l} a_j y_j K(x_i, x_j))$$

where η_i is the learning rate and can be proven to provide maximal gain for $\eta_i = \frac{1}{K(x_i, x_i)}$.

In order to satisfy the restrictions of the objective function, after each update we perform a check to make sure that the update a_i is not below zero or above C, if this is the case then we set $a_i = 0$ or $a_i = C$ accordingly.

This approach however proves to be impractical as its complexity makes it excessively time consuming. Instead we program a support vector machine using the sequential minimal optimization (SMO) algorithm [15]. This algorithm is based on a technique known as decomposition in which only a subset of the Lagrange multipliers a_i

are considered at each moment with the rest considered constant. In the case of the SMO algorithm this idea is taken to the extreme and only two multipliers are considered at a time. One of the main ideas is that in the case of only updating two multipliers at a time, there exists an analytical solution to the optimization problem.

More analytically, considering a_1 and a_2 to be the multipliers to be updated, then the new values for these two parameters must lie on a line in order not to violate the linear conditions $\sum_{i=1}^{l} a_i y_i = 0$. Thus we have :

$$a_1 y_1 + a_2 y_2 = constant = a_1^{old} y_1 + a_2^{old} y_2$$

where of course the constraints $0 \leq a_1, a_2 \leq C$ must also be satisfied.

This translates to the following constraints for the value a_2^{new} :

$$U \leq a_2^{new} \leq V$$

where

$$U = max(0, a_2^{old} - a_1^{old}),\ V = min(C, C - a_1^{old} + a_2^{old})$$

if $y_1 \neq y_2$, and

$$U = max(0, a_1^{old} + a_2^{old} - C),\ V = min(C, a_1^{old} + a_2^{old})$$

if $y_1 = y_2$.

It can be proven [4] that the maximum of the objective function can be achieved by first computing $a_2^{new,unc} = a_2^{old} + \frac{y_2(E_1 - E_2)}{\kappa}$ and clipping it to enforce the constraints $U \leq a_2^{new} \leq V$. Where $-\kappa = -K(x_1, x_1) - K(x_2, x_2) + 2K(x_1, x_2)$ is the second derivative of the objective function along the diagonal line and

$$E_i = f(x_i) - y_i = (\sum_{j=1}^{l} a_j y_j K(x_j, x_i) + b) - y_i$$

is the difference between function output and target classification of the point x_i.

The value a_1^{new} can then be obtained from a_2^{new} as follows :

$$a_1^{new} = a_1^{old} + y_1 y_2 (a_2^{old} - a_2^{new}).$$

SVM Results In figure 11 we can see the performance of the support vector machine. As before the experiments were repeated 10 times each time drawing the labeled examples from the dataset by using a uniform distribution without replacement. Once the SVM was trained using the examples, its performance was observed on the remaining examples in the data set. From the figure we can see that once again in the case of a supervised learning algorithm the problem cannot be solved to a satisfactory level when very few labeled examples exist. This reinforces the suspicion that when it comes to document classification, the problem cannot be solved satisfactorily with supervised algorithms if there is not adequate labeled data.

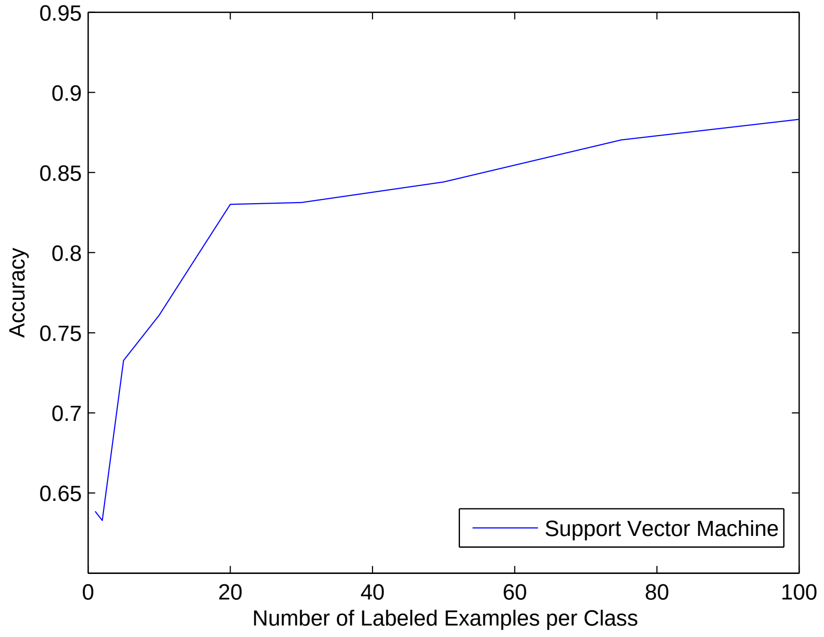


Figure 11

2.4 Transductive Learning Algorithms

2.4.1 Self-Training

One of the simplest transductive learning algorithms, is the self-training algorithm [12]. In self-training, a learner L is trained using the labeled examples of the training set. Once the learner has been trained, it is used to classify the unlabeled examples. The most confidently classified examples are then chosen and are labeled accordingly. These labeled examples are then added to the training set and the learner is retrained. The process continues iteratively, each time the learner being trained with a slightly augmented training set. This can continue until every example in the original test set is labeled or (as this can prove to be especially time consuming) after a fixed number of steps.

In the experiments conducted here, the learner used was a Naive Bayes Classifier. Thus after the training set has been used to obtain the parameters $\vartheta_{c_j} = P(c_j|\vartheta)$ and $\vartheta_{w_t|c_j} = P(w_t|c_j;\vartheta)$ of the model, the self-training algorithm calculates the probabilities of each document d_i for each of the classes c_j:

$$P(c_j|d_i;\hat{\vartheta}) = \frac{P(c_j|\hat{\vartheta})P(d_i|c_j;\hat{\vartheta})}{P(d_i|\hat{\vartheta})}$$

and for each of the classes c_j find those documents d_i which have the highest probability $P(c_j|d_i;\hat{\vartheta})$ and labels them accordingly.

In the experiments conducted here, at each iteration of the algorithm, only the most confident example for each class is labeled, the process continues for 10 iterations, i.e. 10 examples are labeled for each class. Of course the process could continue until all the unlabeled examples are labeled (which translates to $3730/2 = 1865$ iterations for the case of 100 labeled examples per class) but that proves impractical as the complexity of the Naive Bayes classifier makes it prohibitive. As in the cases before, the experiments are repeated 10 times and the resulting classifier tested on the unlabeled examples. The results of these experiments can be seen in figure 12. As can be seen the resulting classifier continues to perform badly for minimal labeled data (1 or 2 per class) but nonetheless performs quite well when the number of labeled examples are few (but not minimal).

The sudden jump observed in the accuracy of the algorithm can be easily explained if one takes into consideration the nature of the algorithm. As at each iteration the algorithm augments its training set and retrains the classifier using the augmented training set, in order for its performance to increase the examples labeled must in fact belong to the class they are labeled to, otherwise they will hinder the training process. When the number of original labeled data is very small then the examples are labeled more or less randomly and as the probability of the training set incorporating falsely labeled data increases, the accuracy of the algorithm instead of increasing with every iteration in fact deteriorates. When however the number of originally labeled examples surpasses a certain threshold then the examples chosen for labeling are highly likely to be classified correctly (as the original classifier is of a sufficient accuracy) and the algorithm no longer deteriorates but instead improves with each iteration.

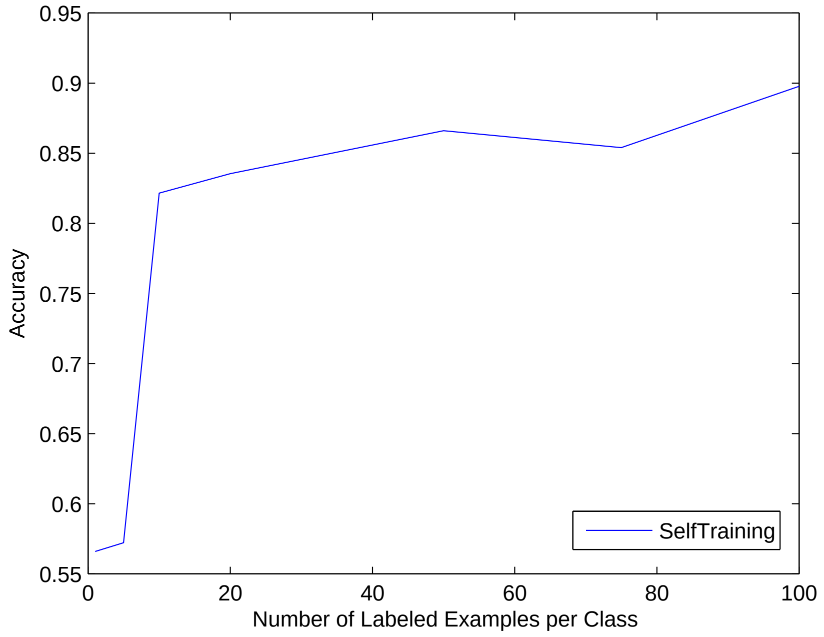


Figure 12

2.4.2 Transductive Expectation Maximization

A well known algorithm in statistical learning theory is the expectation maximization algorithm [14]. This algorithm uses iterative optimization in order to estimate some unknown parameters ϑ of a probabilistic model. In a supervised learning setting, expectation maximization is usually used when there is incomplete data present, i.e. the values of the different attributes are not all known for each example. Thus in these cases, expectation maximization is used to make a maximum likelihood estimation of the missing values.

In the case of transductive learning, the missing values are considered to be the labels of the unlabeled data. In particular, we consider a Bayesian probabilistic framework and originally train a Naive Bayes classifier using the labeled data. Having built the classifier using the labeled data, it is then used to estimate the component membership of each document. This translates to calculating the probability that each mixture component generated each document and based on these probabilistically weighted class labels are assigned to the unlabeled data.

The algorithm then continues to train a new Naive Bayes classifier (i.e. new parameters $\hat{\vartheta}$) this time using the entire data set. The process of building classifiers and estimating labels for the unlabeled data is repeated iteratively until there is no change to the values of the estimated parameters $\hat{\vartheta}$ (in practice the algorithm halts when the change in the log-likelihood of the data falls beneath some threshhold, in this case 10^{-4}). Thus we effectively execute a hill-climbing search which ultimately calculates those labels for the unlabeled data that maximizes the value $l_c(\vartheta\,|D) = log(P(\vartheta)P(D\,|\vartheta))$ which is equivalently[14]:

$$l_c(\vartheta\,|D; z) = log(P(\vartheta)) + \sum_{d_i \in D} \sum_{j=1}^{|C|} z_{ij} log(P(c_j\,|\vartheta)\, P(d_i\,|c_j; \vartheta))$$

where z_{ij} are the indicator values of document d_i and where $z_{ij} = 1$ iff $y_i = c_j$, otherwise $z_{ij} = 0$. Consequently, the hill-climbing procedure effectively computes the expected value of z at each E-step and uses these values to estimate the maximum a posteriori estimates for the parameters of the mixture model $\hat{\vartheta}$. As the labels of the unlabeled data are not in fact binary, but rather, as stated, probabilistically weighted estimations of the true labels, when estimating the maximum a posteriori values for the parameters $\hat{\vartheta}$ we use the following equations[14]:

$$\hat{\vartheta}_{w_t|c_j} = P(w_t|c_j; \hat{\vartheta}) = \frac{1+\sum_{i=1}^{|D|} N(w_t,d_i)P(y_i=c_j|d_i)}{|V|+\sum_{s=1}^{|V|}\sum_{i=1}^{|D|} N(w_s,d_i)P(y_i=c_j|d_i)}$$

where $N(w_t, d_i)$ is the count of the number of times word w_t appears in document d_i. The value $P(y_i = c_j\,|d_i) \in \{0, 1\}$ for labeled data, while in the case of labeled data we have $P(y_i = c_j\,|d_i) \in [0, 1]$.

The class prior probabilities are given by [14]:

$$\hat{\vartheta}_{c_j} = P(c_j|\hat{\vartheta}) = \frac{1+\sum_{i=1}^{|D|} P(y_i=c_j|d_i)}{|C|+|D|}$$

.

The above equations are derived from the maximum a posteriori parameter estimation, if we consider the prior distribution over the parameters $(P(\vartheta))$ to be a Dirichlet distribution[14]:

$$P(\vartheta) \propto \prod_{c_j \in C}((\vartheta_{c_j})^{a-1} \prod_{w_t \in V}(\vartheta_{w_t|c_j})^{a-1})$$

The parameter a affects the strength of the prior, if we consider $a = 2$ which is equivalent to Laplace smoothing, then we obtain the formulas shown above[14]. In figure 13 we can see the performance of the Expectation Maximization algorithm. As in the case of Self-training, the experiments were repeated ten times each time drawing the labeled examples by a uniform distribution without replacement and the resulting classifier each time being tested on the remaining (originally) unlabeled examples.

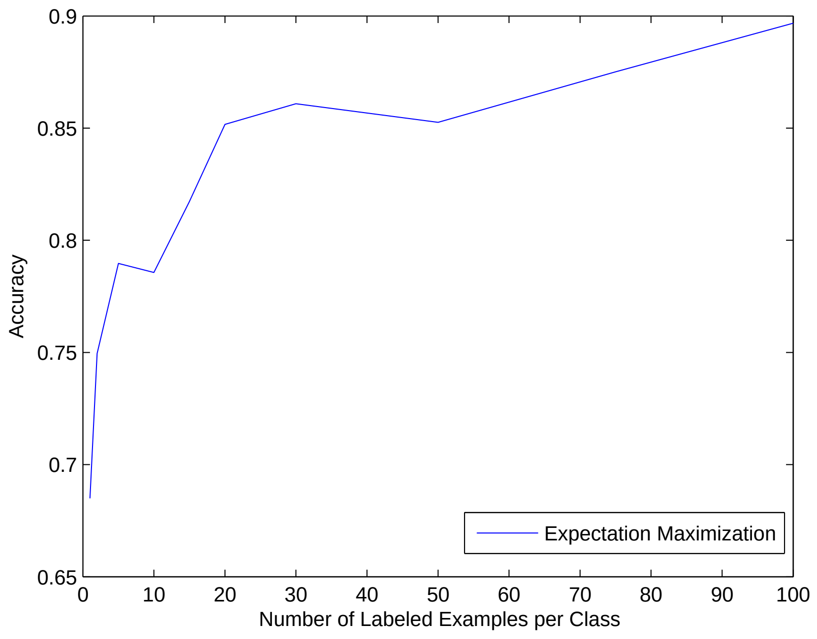


Figure 13

As can be seen from these results, the algorithm quickly reaches a high performance as the number of labeled examples rises. Unlike the self-training algorithm however, expectation maximization does not deteriorate for minimal amounts of labeled data. On the contrary it reaches a relatively high performance ($\approx 68\%$) even when originally there is only one labeled example per class. This performance is higher than both the supervised algorithms and the self-training transductive algorithm.

The expectation maximization transductive learning algorithm, after having trained the first learner, in all subsequent iterations and training of a new classifier (at the M-step), uses both the labeled and the (originally) unlabeled data to estimate the parameters $\hat{\vartheta}$. These two sets of data are taken into consideration with equal weight and influence any estimations made equally. However it is obvious that there should be more confidence in the label of the labeled data than the labeling of the unlabeled data. Thus it would be preferable if during the M-step of the algorithm, i.e. the training of the Naive Bayes Classifier, the labeled data was taken more into account than the unlabeled data, especially seeing as the unlabeled data is far larger in quantity than the labeled data and subsequently it is highly probable that the information provided by the labeled data be lost under the weight of the information provided by the unlabeled data.

In order to overcome these problems, an extra parameter λ_i is introduced which weights the contribution of each example d_i to the learning process. By introducing this variable, the quantity which is maximized by the expectation maximization algorithm is[14]:

$$l_c(\vartheta \,|D; z) = log(P(\vartheta)) + \sum_{d_i \in D^l} \sum_{j=1}^{|C|} z_{ij} log(P(c_j \,|\vartheta) \, P(d_i \,|c_j; \vartheta)) +$$
$$\lambda(\sum_{d_i \in D^u} \sum_{j=1}^{|C|} z_{ij} log(P(c_j \,|\vartheta) \, P(d_i \,|c_j; \vartheta)))$$

where D^u and D^l are the sets of labeled and unlabeled documents. As λ approaches zero, the influence of the unlabeled data on the learning diminishes, the algorithm effectively reverts to supervised learning for $\lambda = 0$, while for $\lambda = 1$ the algorithm takes both labeled and unlabeled data into equal consideration. For the training process realised in the M-step, we define $\Lambda(i)$ to be[14]:

$$\Lambda(i) = \lambda \text{ if } d_i \in D^u$$
$$\text{and}$$
$$\Lambda(i) = 1 \text{ if } d_i \in D^l$$

Using these values, the formulas for computing the parameters $\hat{\vartheta}$ become[14]:

$$\hat{\vartheta}_{w_t|c_j} = P(w_t|c_j;\hat{\vartheta}) = \frac{1+\sum_{i=1}^{|D|} \Lambda(i) N(w_t,d_i) P(y_i=c_j|d_i)}{|V|+\sum_{s=1}^{|V|}\sum_{i=1}^{|D|} \Lambda(i) N(w_s,d_i) P(y_i=c_j|d_i)}$$

and the class prior probabilities are given by :

$$\hat{\vartheta}_{c_j} = P(c_j|\hat{\vartheta}) = \frac{1+\sum_{i=1}^{|D|} \Lambda(i) P(y_i=c_j|d_i)}{|C|+\left|D^l\right|+\lambda|D^u|}$$

In figure 14 we see the performance of the algorithm for different values of lambda. The settings of the experiments remain the same as before

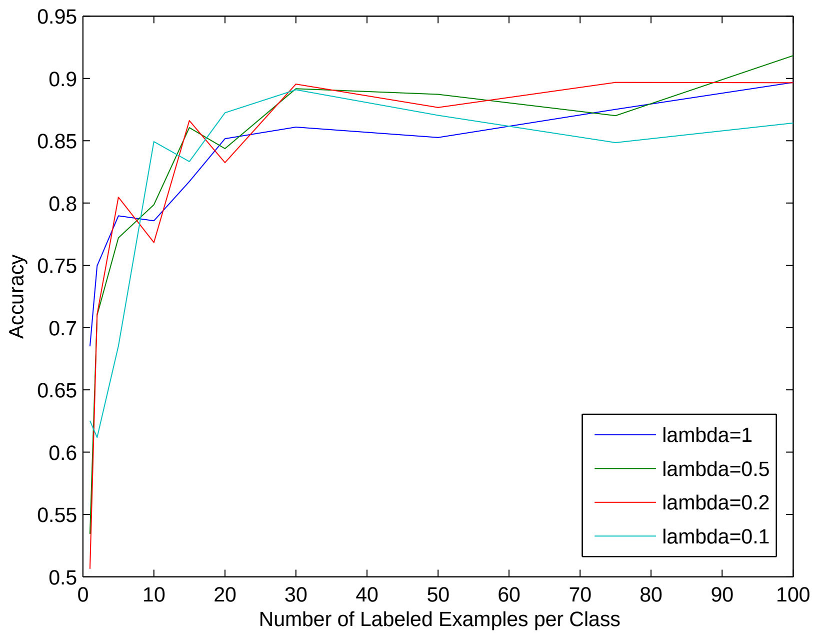


Figure 14

From the results it can be seen that the introduction of the parameter λ does in fact improve the performance of the algorithm. However this improvement exhibits itself mostly when the number of labeled examples is not minimal. Thus for minimal amounts of labeled data, the algorithm performs best for $\lambda = 1$ which is equivalent to the original algorithm. The introduction of the parameter λ only leads to improvement as the number of labeled examples augments leading to a clear advantage when the number of labeled examples is above 20 per class. This makes sense if ones considers that for minimal amounts of labeled data, the information received by these examples is also minimal and thus the algorithm must rely on the unlabeled data. As however the number of labeled examples increases, the information for the labeled data also increases and so not only decreases the need for information from the unlabeled examples, but also so increases the need to protect this information from being buried under the unlabeled data's information.

2.4.3 Transductive Learning Using Graph Mincuts

Another approach to transductive learning is that of using graph mincuts [1]. One of the main advantages of the graph mincut algorithm is that unlike the expectation maximization algorithm presented in the previous chapter, the graph mincut finds the global maximum of the objective function. Furthermore this global maximum can be found in polynomial time. However, while in the case of the expectation maximization algorithm, the hill-climbing procedure can be applied to a wide number of objective functions, in the case of the graph mincuts algorithm the objective functions are limited to those functions which depend only on pairwise relations amongst examples.

In particular, the graph mincuts algorithm can perform the following optimization which is of great interest[1]:

Given a set of labeled examples (in this case documents) D^l and a set of unlabeled examples D^u, label the unlabeled examples in such a way that the leave one out cross validation error of the k-nearest neighbor algorithm is minimized over the entire data set $D = D^l \cup D^u$.

Given these two sets of data D^u and D^l, the algorithm first constructs a weighted graph $G = (V, E)$, where the vertices V consist of one vertice for each example in the labeled and unlabeled data set and also a source vertix u_+ and sink vertix u_-; thus we have $V = D^l \cup D^u \cup \{u_+, u_-\}$. The vertices u_+ and u_- are also called classification vertices, while the other vertices are called example vertices.

Each edge $e \in E$ between the vertices in the graph is assigned a weight $w(e)$. This weight can be assigned by any number of functions (for example analogously to the distance between the two examples that the edge connects), in the experiments conducted here the value of $w(e)$ for edges e between two example vertices u_i and u_j is set to be $w(e) = \frac{cos(\vec{d_i},\vec{d_j})}{\sum\limits_{m \in kNN(d_i)} cos(\vec{d_i},\vec{d_m})}$ if example d_j is amongst the k nearest neighbors of example d_i (or vice versa), where $cos(\vec{d_i}, \vec{d_j})$ is the cosine similarity function between documents d_i and d_j and which is equal to :

$$\cos(\vec{d_i}, \vec{d_j}) = \frac{\vec{d_i} \cdot \vec{d_j}}{\|\vec{d_i}\|_2 * \|\vec{d_j}\|_2} = \frac{\sum\limits_{k=0}^{n} w_{ik} w_{jk}}{\sqrt{\sum\limits_{k=0}^{n} w_{ik}^2}\sqrt{\sum\limits_{k=0}^{n} w_{jk}^2}}$$

Otherwise if d_j is not amongst the nearest neighbors of d_i then the weight of the edge between the two vertices is set to 0, i.e. the vertices are not considered to be connected.

In the case of the classification vertices u_+ and u_-, these are connected to those vertices which correspond to labeled data with the same label as the corresponding classification vertex. Thus vertices corresponding to positively labeled examples (i.e. examples with a label of +1) are all connected to the vertex u_+, while vertices corresponding to negatively labeled examples (i.e. examples with a label of -1) are all connected to the vertex u_-. These edges between classification vertices and example vertices are all assigned an infinite weight, thus we have $w(u, u_-) = \infty$ xor $w(u, u_+) = \infty$ $\forall u \in D^l$.

The algorithm then proceeds to calculate a minimum cut for the graph, considering u_+ as the source and u_- as the sink which translates to finding the minimum total weight set of edges that disconnect u_+ and u_-. This can be achieved with the help of a maximum flow algorithm like for example the Edmonds-Karp algorithm [2] which is used in the following experiments.

Before analyzing in detail the Edmonds-Karp algorithm, we must first introduce the notion of the residual network. Given a graph

$G = (V, E)$ with capacity $c(v, u)$ and flow $f(v, u)$ between vertices v and u, the residual network $G_f(V, E_f)$ is the network with capacity $c_f(v, u) = c(v, u) - f(v, u)$.

The Edmonds-Karp algorithm is based on the Ford-Fulkerson algorithm [2] which uses the residual network to solve the maximum flow problem of a graph, as follows :

```
While there is a path $p$ in $G_f$ from $s$ (source) to $t$ (sink) such that $c_f(u, v) > 0 \; \forall(u, v) \in p$ then :
Set $c_f(p) = min(c_f(u, v)|(u, v) \in p)$.
$\forall(u, v) \in p$ do :
$f(u, v) = f(u, v) + c_f(p)$
$f(v, u) = f(v, u) - c_f(p)$
```

The Edmonds-Karp algorithm is a variation of the Ford-Fulkerson algorithm which specifically uses depth-first search to find the path p in the residual network G_f.

Returning to the transductive learning graph mincut algorithm, having found the mincut of the constructed graph, the graph is essentially partitioned into two sets V_+ and V_- each consisting of vertices situated on opposite sides of the mincut. Thus vertices in V_+ are those vertices which can be reached by a path p in the final residual graph and which form a connected component which includes the vertex u_+ while conversely vertices in the set V_- cannot be reached by a path p in the final residual graph. Finally the unlabeled examples corresponding to the vertices in each of the two subsets are labeled accordingly.

Figure 15 shows the performance of the graph mincut algorithm for various quantities of labeled examples. As before, the experiments are repeated 10 times, each time the labeled examples being drawn from the extended data set using a uniform distribution without replacement. The graph was constructed considering the 3-nearest neighbors of each example [1].

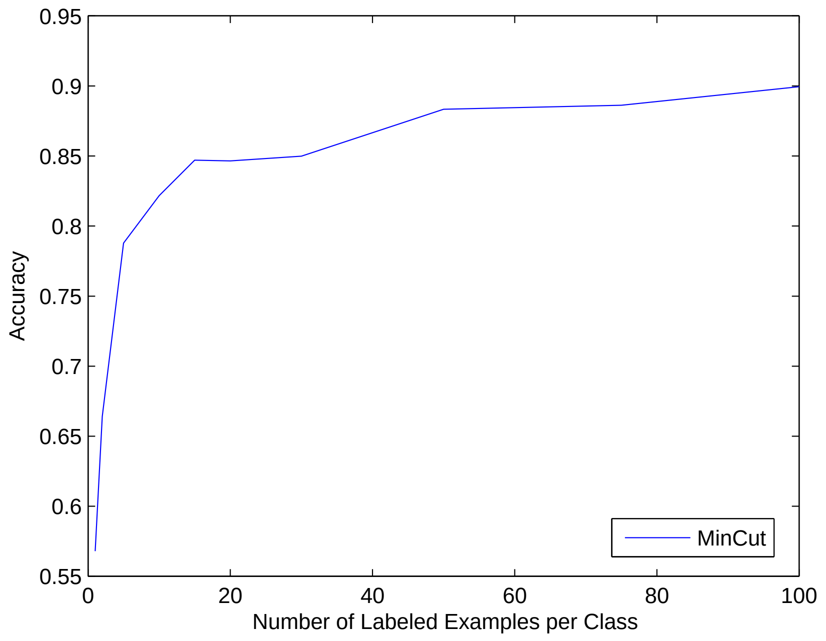


Figure 15

The results obtained using the graph mincut transductive learner show that once again the transductive learning algorithm is able to reach relatively high levels of performance with a few labeled examples per class. However unlike the expectation maximization algorithm in the previous section, the algorithm is unable to attain a high performance when the number of labeled examples is minimal. In fact in the case of one labeled example per class the algorithm performs worse than the supervised algorithms, its performance bordering on that of a random classifier. A possible explanation for this phenomenon is presented in the next section along with an alternative graph partitioning algorithm which aims to remedy this problem.

2.4.4 Tranductive Spectral Graph Partitioning

The graph mincut algorithm presented in the previous chapter is a intuitively appealing approach to the transductive learning problem, however it does not in all cases lead to similarly appealing solutions. Consider the case where there are only two labeled examples, one positive x_+ and one negative x_-. If we connect each point to its three closest neighbors then it is very likely that the minimum cut of the graph consists of partitioning the graph into two sets S and $\bar{S}$ such that $S = u_+ \cup, x_+$ and $\bar{S} = u_- \cup x_- \cup D^u$. This partition obviously will result in every example in the unlabeled data set D^u being labeled with a negative label. In this case it is obvious that the resulting classifier performs substantially poorer than if we had not taken the unlabeled data into consideration at all.

The problem of degenerated cuts can be overcome if we consider the graph Laplacian and instead of the minimum cut problem, we solve the ratiocut problem instead [8]. The ratio of a cut $S, \bar{S}$ is defined as :

$$R(S, \bar{S}) = \frac{C(S,\bar{S})}{\sum\limits_{i \in S} d_i \cdot \sum\limits_{i \in (\bar{S})} d_i}$$

where d_i is the size of vertex i and $C(S, \bar{S})$ is the sum of the weights of the edges between the vertices of S and $\bar{S}$. The ratio cut problem consists of finding the non-empty cut $S, \bar{S}$ for which the value $R(S, \bar{S})$ is minimum. Thus instead of minimizing the sum of the weights of the edges of the cut as in the case of the mincut problem, the ratiocut problem finds the cut $S, \bar{S}$ which minimizes the average weight of the cut.

The Laplacian L_G of a graph $G = (V, E)$ is a matrix whose entries l_{ij} are given by :

$l_{ij} = -1$ if $(i, j) \in E$
$l_{ij} = d_i$ if $i = j$ where d_i is the degree of vertex i
$l_{ij} = 0$ otherwise.

The Laplacian of a graph provides information about various characteristics of the graph through its eigenvalues λ_i and corresponding eigenvectors v_i. For example the multiplicity of 0 as an eigenvalue of L_G equals the number of connected components of G. In the case of the ratiocut problem, the eigenvalue λ_2 (i.e. the smallest non-negative eigenvalue) is important as it yields information concerning the quality of the best cut of the graph.

If we redefine the ratio of a cut $(S, \bar{S})$ to be :

$$\phi(S) = \frac{\left|C(S,\bar{S})\right|}{min(|S|, \left|\bar{S})\right|}$$

then the isoperimetric number $\phi(G)$ of a graph can be defined as :

$$\phi(G) = min_S \phi(S)$$

It is obvious from the above that the ratiocut problem consists of finding a cut in the graph whose ratio is equal to the isoperimetric number. The isoperimetric number is related to the eigenvalue λ_2 via Cheeger's inequality :

$$\phi(G) \geq \lambda_2 \geq \frac{\phi^2(G)}{2d}$$

where d is an upper bound of the degree of every vertex in the graph. Thus the eigenvalue λ_2 of the Laplacian L_G bounds the isoperimetric number of the graph, giving both a lower bound ($\phi(G) \geq \lambda_2$) and a higher bound ($\sqrt{2d \cdot \lambda_2} \geq \phi(G)$).

Before presenting the spectral graph transducer [8] that solves the spectral graph partitioning problem in a transductive setting, we first formally state the problem to be solved. Given a graph $G = (V, E)$ the problem consists of minimizing the value $\frac{C(S,\bar{S})}{|i:y_i=1||i:y_i=-1|}$ with the constraints a) $y_i = 1$ if $i \in D^l$ and positive, b) $y_i = -1$ if $i \in D^l$ and negative and c) $\vec{y} \in \{+1, -1\}^n$.

Ignoring these constraints which are related to the labels of the labeled data set, we obtain the unsupervised ratiocut problem which is equivalent to the following[8]:

$$min_{\vec{z}} \frac{\vec{z}^T L \vec{z}}{\vec{z}^T \vec{z}}$$

where $\vec{z_i} \in \{\gamma_+, \gamma_-\}$ and $\gamma_+ = \sqrt{\frac{|i:z_i<0|}{|i:z_i>0|}}, \gamma_- = -\sqrt{\frac{|i:z_i>0|}{|i:z_i<0|}}$

The above problem is NP-hard to solve[8]. However the relaxation of the above problem:

$$\begin{gathered} min_{\vec{z}} \vec{z}^T L \vec{z} \\ \text{subject to } \vec{z}^T \vec{z} = n \text{ and } \vec{z}^T 1 = 0 \end{gathered}$$

where n is the number of vertices in the graph, has an analytical solution which is the eigenvector v_2 which corresponds to the eigenvalue λ_2 of the Laplacian[8]. The solution to the relaxed problem can then be used as a good approximation of the solution of the original unsupervised ratiocut problem.

For the supervised ratiocut problem, we add a quadratic penalty to the objective function and obtain the following supervised ratiocut problem :

$$min_{\vec{z}} \vec{z}^T L \vec{z} + c(\vec{z} - \vec{\gamma})^T C (\vec{z} - \vec{\gamma})$$
$$\text{subject to } \vec{z}^T \vec{z} = n \text{ and } \vec{z}^T 1 = 0$$

where the value of element i of the vector $\vec{\gamma}$ is γ_+ or γ_- if the corresponding example d_i is positively labeled or negatively, and it is 0 if $d_i \in D^u$. The parameter c allows for a trade-off between training error and cut-value, while C is a diagonal cost matrix that allows for misclassification costs (not necessarily the same) for each example.

Using eigen-decomposition, we can decompose the Laplacian : $L = U \Sigma U^T$, and subsequently by introducing a new parameter vector $\vec{w}$ such that $\vec{z} = U\vec{w}$, we obtain the equivalent problem :

$$min_{\vec{w}} \vec{w}^T D \vec{w} + c(V\vec{w} - \vec{\gamma})^T C (V\vec{w} - \vec{\gamma})$$
$$\text{subject to } \vec{w}^T \vec{w} = n$$

Based on the above formulation of the ratiocut problem, the spectral graph transducer labels the examples of the unlabeled data set D^u by solving the ratiocut problem as follows.

First the similarity-weighted k-nearest neighbor graph is constructed. As in the case of the graph mincut algorithm presented in the previous chapter, the vertices of the graph, each represent an example of the extended data set $D = D^l \cup D^u$. Two vertices i, j of the graph are connected by an edge e_{ij} in the graph if the example d_j is amongst the k nearest neighbors of the example d_i. The weight of the edge $w(e_{ij})$ is assigned the value :

$$w(e_{ij}) = \frac{\cos(\vec{d_i}, \vec{d_j})}{\sum\limits_{\vec{d_k} \in kNN\vec{d_i}} \cos(\vec{d_i}, \vec{d_k})} \text{ if } \vec{d_j} \in knn\vec{d_i}$$
$$w(e_{ij}) = 0 \text{ otherwise.}$$

Having constructed the similarity-weighted k-nearest neighbor graph A, the algorithm computes the diagonal degree matrix B, the elements of which are :

$$B_{ij} = 0 \text{ if } i \neq j$$
$$B_{ij} = \sum_k A_{ik} \text{ if } i = j$$

The normalized Laplacian $L = B^{-1}(B - A)$ is then computed as well as its smallest 2 to $d+1$ eigenvalues $(\lambda_2 ... \lambda_{d+1})$ and eigenvectors $(v_2 ... v_{d+1})$, where the parameter d is defined by the user. The eigenvectors are then stored in the matrix V, whose columns correspond to the eigenvectors (i.e. $V(:, i) = v_{i-1}$). In order to normalize the spectrum of the graph, instead of using the diagonal matrix D whose diagonal elements are eigenvalues of L (i.e. $D_{ii} = \lambda_{i-1}$), some monotonically increasing function is used to assign values to the diagonal elements of D. More specifically, in the experiments conducted here, we use $D_{ii} = i^2$.

Next the values $\gamma_+ = \sqrt{\frac{l_-}{l_+}}$ and $\gamma_- = \sqrt{\frac{l_+}{l_-}}$ are estimated, where l_+ is the number of positively labeled examples and l_- the number of negatively labeled examples. The vector $\vec{\gamma}$ is then computed where:

$$\begin{gathered} \gamma_i = \gamma_+ \text{ if } d_i \in D^l \text{ and } y_i = +1 \\ \gamma_i = \gamma_- \text{ if } d_i \in D^l \text{ and } y_i = -1 \\ \gamma_i = 0 \text{ if } d_i \in D^u \end{gathered}$$

The diagonal cost matrix is then computed by setting :

$$\begin{gathered} C_{ii} = \tfrac{l}{2l_+} \text{ if } d_i \in D^l \text{ and } y_i = +1 \\ C_{ii} = \tfrac{l}{2l_-} \text{ if } d_i \in D^l \text{ and } y_i = -1 \\ C_{ij} = 0 \text{ otherwise.} \end{gathered}$$

The solution $\vec{w}^*$ to the optimization problem :

$$\begin{gathered} min_{\vec{w}} \vec{w}^T D \vec{w} + c(V\vec{w} - \vec{\gamma})^T C (V\vec{w} - \vec{\gamma}) \\ \text{subject to } \vec{w}^T \vec{w} = n \end{gathered}$$

is given by $\vec{w}^* = (G - \lambda^* I)^{-1} \vec{b}$, where $G = (D + cV^T CV)$ and $\vec{b} = cV^T C\vec{\gamma}$ and λ^* is the smallest eigenvalue of the matrix :

$$\begin{pmatrix} G & -I \\ \frac{-1}{n} \cdot \vec{b}\vec{b}^T & G \end{pmatrix}$$

where I is the identity matrix.

Based on the above the algorithm computes a prediction :

$$\vec{z}^* = V(G - \lambda^* I)^{-1} \vec{b}$$

Using this prediction, the spectral graph transducer, makes a hard class assignment to the unlabeled examples by setting $y_i = sign(z_i - \Theta)$ where

$$\Theta = \tfrac{1}{2}(\gamma_+ + \gamma_-).$$

The performance of the algorithm can be seen in figure 16. The experiments, as before, were all repeated ten times and the labeled examples drawn each time using a uniform distribution without replacement. The parameter c was set to $c = 3200$ while the number of nearest neighbours used to calculate the adjacency matrix was set to 50.

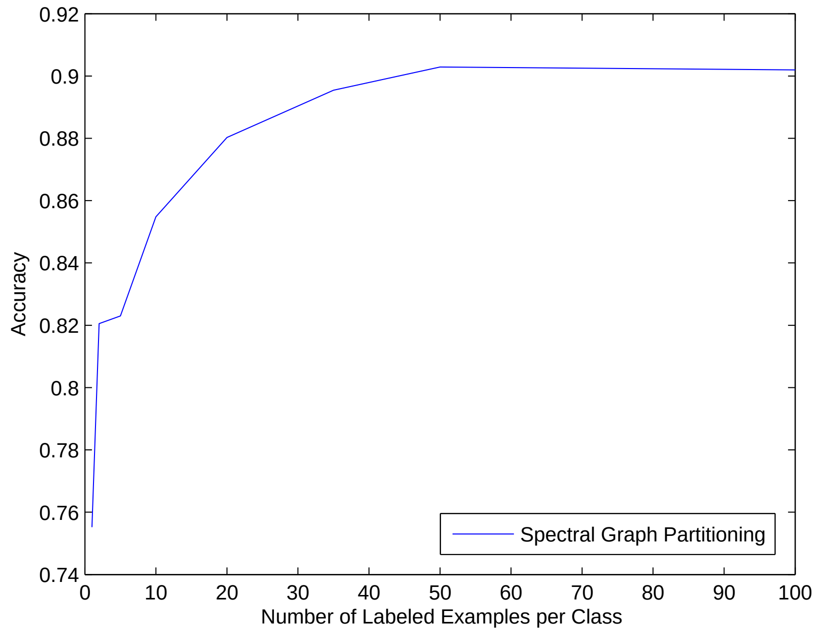


Figure 16

From the results it is evident that the spectral graph transducer is able to overcome the shortcomings of the graph mincut algorithm of the previous section. Even for very small, minimal, amounts of labeled data, the spectral graph transducer reaches a very high level of performance, the greatest observed up till now, attaining an accuracy of more than 80% with two labeled examples per class.

2.4.5 Learning with Local and Global Consistency

Another approach to transductive learning is that of learning with local and global consistency [22]. This algorithm is based on two assumptions concerning the consistency of the data set. The first assumption concerns local consistency and is that nearby points are likely to have the same label. The second assumption concerns global consistency and is that points on the same structure (for example a cluster) are likely to have the same label, this assumption is also known as the cluster assumption.

In order to formalize the above assumptions, the algorithm uses a classifying function which is considered to be sufficiently "smooth" as regards the intrinsic structure of the data. The algorithm classifies the unlabeled examples with the help of a matrix F. This matrix is a $n \times c$ matrix (where n is the number of examples in the extended data set $D^{all} = D^u \cup D^l$ and c the number of classes) which corresponds to a classification of the examples in D^{all}, it effectively assigns a vector F_i (i.e. the row i of the matrix F) to each example d_i, this vector can then be used to obtain a label for the example d_i by setting $y_i = argmax_{j \leq c} F_{ij}$.

We define the matrix Y to be a $n \times c$ matrix for which :

$$Y_{ij} = 1 \text{ if } y_i = j \text{ i.e. if example } d_i \text{ is labeled and belongs to class } j$$
$$Y_{ij} = 0 \text{ otherwise.}$$

Based on the above the learning with local and global consistency algorithm consists of the following steps [22]:

1. The matrix W, called the affinity matrix, is constructed. This matrix can be seen as the weighting function of the edges E of a graph $G = (V, E)$ where as in the previous learners presented heretofore, the vertices represent the examples of the data set D^{all} . More analytically in the case of text classification the elements of the affinity matrix W are given by :

$$W_{ij} = \cos(\vec{d_i}, \vec{d_j}) \text{ if } i \neq j$$
$$W_{ii} = 0$$

2. The matrix $S = D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$ is constructed. The matrix D is a diagonal matrix whose diagonal elements are equal to the sum of the elements of the corresponding row of the affinity matrix W, i.e. $D_{ii} = \sum_j W_{ij}$.

3. The matrix F is calculated by iterating $F(t+1) = aSF(t) + (1-a)Y$ until convergence, where $a \in (0, 1)$ is a user defined parameter.

4. Having calculated F^* (the limit of the sequence $F(t)$), the unlabeled examples $d_i \in D^u$ are labeled by assigning $y_i = argmax_{j \leq c} F^*_{ij}$

The above iterative process is closely related to spreading activation networks (SANs). As stated above, the affinity matrix W can be seen as the weighting function of a graph $G = (V, E)$ where each node $v_i \in V$ represents a specific example d_i of the extended data set D^{all}. Equivalently in a spreading activation network each node represents a specific concept (or on this case example) and two nodes i, j are connected between them by a connection with which a weight w_{ij} is associated. The iteration process in step 3, propagates information from the nodes of the network to its neighbors based on the weights of the connections between them. Just as in the case of spreading activation networks, initially only the input nodes are activated (equivalently only the vertices pertaining to labeled examples are labeled i.e. $Y_{ij} = 1$ if $y_i = j$) and the rest of the nodes have an output of 0 ($Y_{ij} = 0$ otherwise). Thus at each point every node receives information regarding its label from its neighbors and ultimately is set to the class of which it has received the most information. It should be noted that during the iteration procedure self-reinforcement is avoided as the diagonal elements of W are all zero ($W_{ii} = 0$).

The execution of the iteration procedure in step 3 can be avoided if we consider the limit :

$$\lim_{t \to \infty} F(t) = \lim_{t \to \infty} aSF(t) + (1-a)Y$$

Since $F(t+1) = aSF(t) + (1-a)Y$ [22], we have :

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

As $a \in (0, 1)$ and the eigenvalues of $S \in [-1, 1]$, it follows that :

$$\lim_{t \to \infty}(aS)^{t-1} = 0$$

and

$$\lim_{t \to \infty}\sum_{i=0}^{t-1}(aS)^i Y = (I - aS)^{-1} Y$$

Based on the above we obtain a closed form for the solution F^* :

$$F^* = (1-a)(I-aS)^{-1}Y$$

The above algorithm can also be analyzed in a regularization framework where the algorithm can be seen as minimizing a cost function $Q(F)$. In particular we consider the cost function :

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

where the parameter μ is a regularization parameter which is set by the user.

The first term of the cost function $\sum_{i,j=1}^{n} W_{ij}\left\|\frac{1}{\sqrt{D_{ii}}}F_i - \frac{1}{\sqrt{D_{jj}}}F_j\right\|^2$ is the smoothness constraint and ensures that the resulting classifying function will not have greatly different values for similar input, that is two points that are nearby will be less likely to be ultimately classified to different classes. This is connected to the local consistency assumption.

The second term $\sum_{i=1}^{n}\|F_i - Y_i\|^2$ is the fitting constraint which ensures that the classifying function will not return results which differ significantly from the original label assignment i.e. the resulting classifier will have a low error-rate on the labeled data set D^l. The parameter μ thus gives us a trade-off between the two terms.

In order to minimize the cost function $Q(F)$ the solution F^* must satisfy :

$$\begin{gathered}\frac{\partial Q}{\partial F}|_{F=F^*} = 0\\ \Rightarrow F^* - SF^* + \mu(F^* - Y) = 0\\ \Rightarrow F^* = \beta(I-aS)^{-1}Y\end{gathered}$$

where $a = \frac{1}{1+\mu}$ and $\beta = 1-a$. The two closed forms obtained through considering the limit of the sequence $F(t)$ and considering the minimization of the cost function $Q(F)$ are obviously equivalent.

In figure 17 we can see the accuracy attained by the algorithm for different amounts of labeled examples where the setting are the same as always, ten repetitions, the labeled examples drawn with a uniform distribution without replacement, and the resulting classifier tested on the remaining unlabeled examples. Regarding the parameter μ it is set to 0.99.

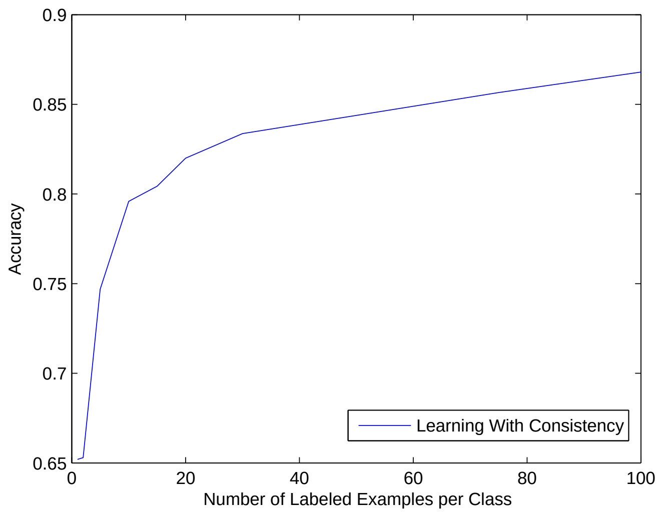


Figure 17

The results obtained seem to indicate that the learning with local and global consistency algorithm is not particularly appropriate for the document classification problem, the problem of low performance for minimal labeled examples has returned and furthermore unlike the other transductive classifiers the performance of the learning with local and global consistency algorithm remains relatively low even for larger amounts of labeled examples.

2.4.6 Results

In the figure 18 we can see the performance of all the transductive learning algorithms presented here compared to the two supervised learning algorithms used as a base for comparison.

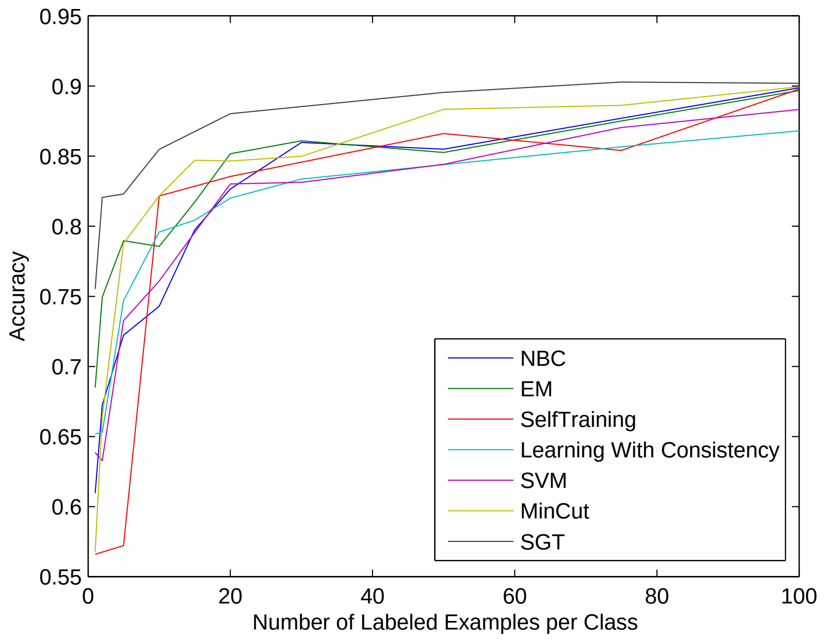


Figure 18

As can be seen the supervised learning algorithms perform significantly poorer than the transductive learners when labeled data is sparse. This can be better seen in the next three figures (19-21) which zoom in on different parts of the previous figure.

When the amount of labeled data is below twenty examples per class, both supervised learning examples perform below the transductive learners as can be seen in figure 19.

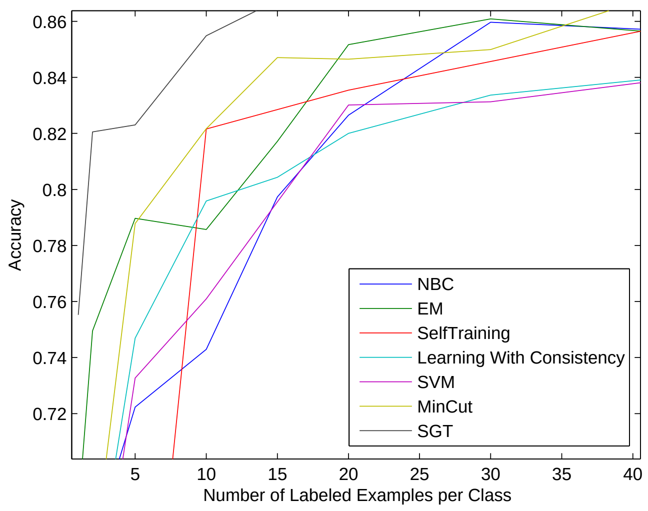
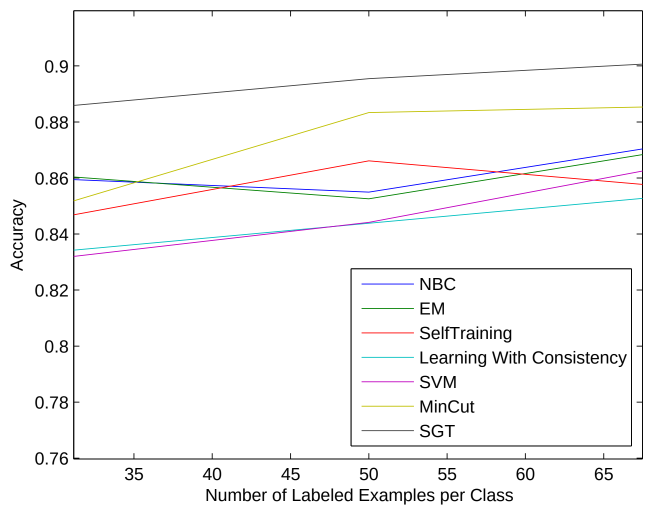


Figure 19

As the number of labeled examples increases, approaching seventy examples per class, the supervised learners start surpassing certain of the transductive learners. The Naive Bayes Classifier outperforms the Expectation Maximization, Learning with Consistency and SelfTraining algorithms while the Support Vector Machine outperforms the two latter algorithms. Both supervised algorithms however continue to perform worse than the Spectral Graph Transducer and the Mincut learner.



Figue 20

The Naive Bayes Classifier finally reaches the performance of the two transductive learners as the number of labeled examples reaches about 100 examples per class. At this level there seems to be enough labeled data that there is no need for additional information from the unlabeled data in order to solve the classification problem. Nonetheless the Support Vector Machine continues to perform below the transductive learners.

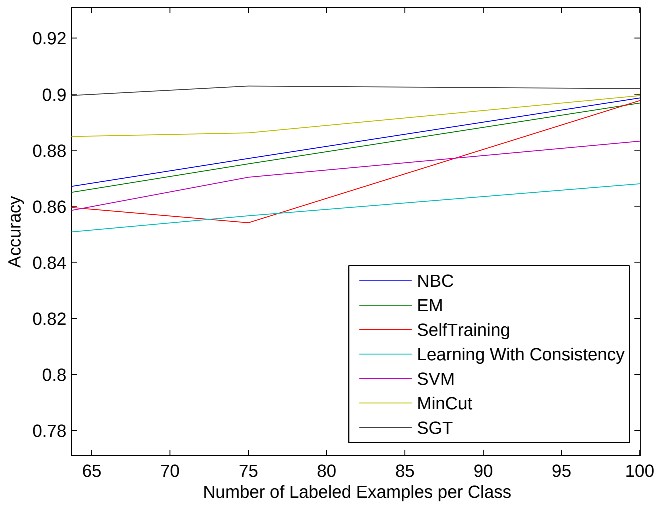


Figure 21

Finally figure 22 shows the performance of only the best two transductive learners, the Spectral Graph Transducer and the graph Mincut learner, and that of the best supervised learner,the Naive Bayes Classifier, for better comparison. From the graph the dominance of the transductive learners is evident.

From the experiments conducted it is easy to see that in the case of document classification, classifiers can benefit from transuctive learning to increase performance by making use of the abundance of unlabeled data in order to overcome the problem of scarce labeled data which is common is such applications.

The graph Mincut and Spectral Graph Transducer prove to perform especially well, at least as concerns the problem at hand, and would seem to be appropriate for applications of this sort. The graph Mincut algorithm does suffer from complexity (Solving the mincut problem on a graph $G(V, E)$ is $O(VE^2)$) which makes it perhaps not so attractive but the Spectral Graph Transducer is especially speedy making it an ideal approach to the document classification problem.

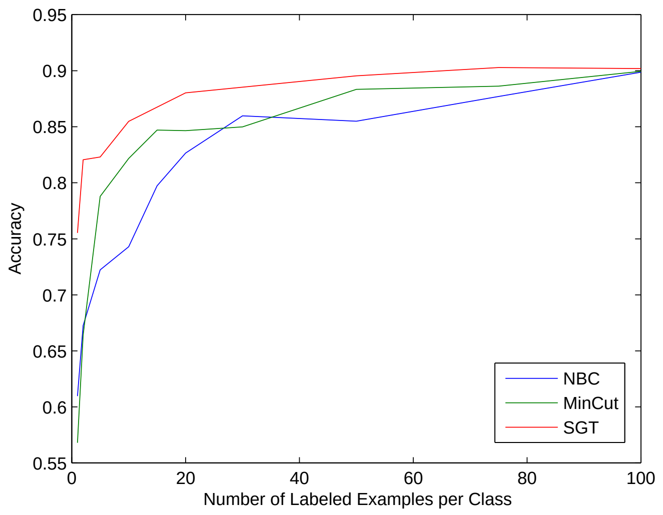

0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
Accuracy
0
20
40
60
80
100
Number of Labeled Examples per Class
NBC
MinCut
SGT

Figure 22

3 Handwritten digit recognition

3.1 Introduction

Another application in which transductive learning can prove to be particularly useful is that of handwritten text recognition. In handwritten text recognition, the aim is to create learners which will ultimately be capable of recognizing handwritten text whether by recognizing isolated characters or entire words. Thus effectively handwritten text recognition can be seen as an approach to learning machines to read.

In the case of handwritten character recognition the system learns to recognize one character at a time, thus in order to read a specific word it must parse each of its letters and then based on the letters recognized, ultimately recognize the word they make up. As however the system will not be 100% accurate in the recognitions of letters it is quite likely that the system will not correctly recognize all the letters of a word. This problem is even more prominent in the case of cursive text where the limits of each letter in a word are not clear cut. Thus for example while parsing the word "dog", the system may mistakenly recognize the "o" as an "a" and thus ultimately read the word "dag". A human being would be unlikely to make such a mistake as he is well aware that the word "dag" does not exist. In order to overcome this problem, handwritten text recognizers which are based on character recognition are usually equipped with a dictionary; this however is not always possible as certain texts may be written in a language for which a concise dictionary is not readily available (in Old Dutch for example).

In cases like these it is perhaps preferable to create a system that attempts to recognize handwritten words directly without resorting to recognizing the individual components (letters) of a word. However in this case another problem often arises. In the case where the system is to be trained to recognize characters, obviously the input of the learning algorithm will also be characters; labeled examples of such characters are usually easy to obtain as the number of different characters are limited and there exist large data sets of different characters (as for example the MNIST data set for digits). On the other hand however in the case where whole words are to be recognized, the presence of large amounts of labeled data cannot be guaranteed (one is highly unlikely to find a data set consisting of handwritten Old Dutch words) and as the number of different words is high, the acquisition of such a large data set by manual labeling is not a realistic endeavor.

Thus it is obvious that although in the specific problem of handwritten digit recognition transductive learning may not be particularly helpful on a practical level as large amounts of labeled data already exist, nonetheless transductive learning can prove to be particularly useful in the wider area of handwritten text recognition. As the two approaches to handwritten text recognition (character and word) are based on many common principles, the experiments presented below which address handwritten digit recognition can be seen representative of the value of transductive learning in the case of handwritten text recognition in general.

3.2 Data

The data set of handwritten digits that will be used in the following experiments is the MNIST dataset [9]. It consists of two sets of data, one dataset containing sixty thousand hand written digits and another containing ten thousand samples of hand written digits. These samples were taken from a mixture of US census workers and high school students and is thus considered to represent two different distributions. In the original NIST dataset these distributions were used separately, the first as a training set and the second as a test set. In the MNIST dataset however these two distributions have been mixed together yielding two datasets (training and test set) which contain samples from both distributions. In figure 23 we see examples from each distributions. The first image shows hand written digits obtained from the (paid) US census workers. The second image shows the samples obtained from the high school students.

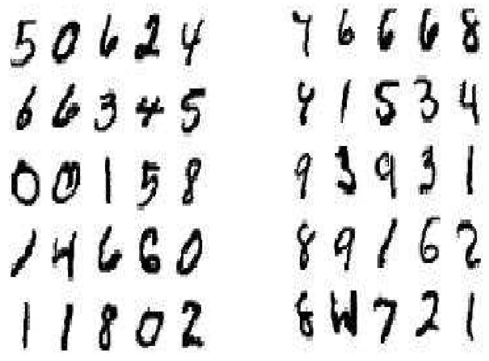


Figure 23

These obtained digits are of a size at most 20x20 pixels. These digits are then set centered on a blank background of 28x28 pixels. Each pixel of the obtained digit is originally represented by a value in the interval [0,255] signifying the intensity of the pixel. Before using the digits in the following experiments however, the values of their pixels are first normalized. Thus the value of each pixel is divided by the maximum value (255) and as a result the value of each pixel lays in the interval [0,1].

As stated the entire MNIST dataset consists of seventy thousand examples of handwritten digits. In the following we will be experimenting with datasets consisting of very few labeled examples (one to twenty per class) and a significantly larger quantity of unlabeled examples. As the maximum number of labeled examples that will be used is 150 (15 examples per class / 10 classes) that leaves us with just under seventy thousand unlabeled examples to complement the labeled data set. Though transductive learning allows us to take advantage of large repositories of unlabeled data to enhance performance, the number of available unlabeled examples in this case in excessive. With such a huge amount of unlabeled data, any information provided by the labeled data set is very likely to be buried due to the sheer size of the unlabeled data set. Furthermore a data set of that size greatly increases the computational cost of the various transductive learning algorithms making them impractical and the execution of multiple experiments unfeasible.

Due to the above, instead of using the entire MNIST data set, a smaller data set is created consisting of four hundred examples of each digit, these examples are drawn from the original data set using a uniform distribution (without replacement). Thus the data set used for the experiments presented below consists of four thousand handwritten digits.

3.3 Supervised Learning Algorithms

As in the case of text classification, the transductive learning algorithms are compared against a number of supervised learning algorithms in order to discern if in fact these supervised learning are insufficient when there are only small amounts of labeled data available and if transductive learning can in fact increase performance.

In particular two algorithms ,Networks pretrained with Restricted Boltzmann Machines and Convolutional Networks are tested and compared to the transductive learners; these algorithms are known to perform exceptionally well on the handwritten digit recognition problem and the second of the two (Convolutional Networks) is in fact specifically tailored to deal with exactly this kind of problems (optical recognition in general). Thus if these two algorithms are seen not to perform adequately with little labeled data then it is highly probable that supervised learning algorithms in general are not capable of overcoming the scarcity of labeled data.

3.3.1 Convolutional Networks

As stated convolutional neural networks [10] are specially tailored for machine vision problems. Convolutional networks use a feature mapping and sub sampling techniques, applying these consecutively onto the data through the multiple layers of the network in order to create a classifier that is ultimately invariant to distortions in the original data, making the classifier effectively invariant to shifting or rotation of the original image. This resembles the way human beings recognize images, or in this case digits; a human would not look to see whether a digit has a line at the exact same place as a prototype digit, for example ”4”, rather a human would simply look to see if certain lines (features) that constitute a digit ”4” are approximately where they should be (invariance to shift and rotation). In the experiments conducted here, we use a specific convolutional network called LeNet-5 which has a total of seven layers (the input excluded) using a series of feature mappings and sub-sampling layers. An overview of the architecture of LeNet-5 [10] can be seen in figure 24.

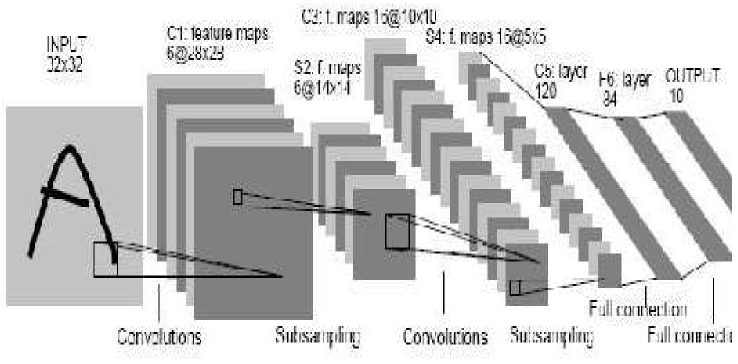


Figure 24

Convolutional networks use feature detectors to create feature mappings of the data, these detectors consist of a set of weights that are applied to a small neighborhood in the data (a bias is then added and the result passed through a sigmoid function), this operation is the equivalent of a convolution and thus lends its name to the entire network. The application of a feature detector over a series of such small neighborhoods (effectively scanning the entire imag) creates a feature mapping corresponding to the specific feature detector. More precisely each unit on a feature mapping has as input a specific region of the data (called the receptive field of the unit), the value of each unit depends on the input of the unit and the value of the weights of the feature detector, these weights are identical for each unit, it is this fact that allows for the creation of feature maps.

This architectural idea is called weight replication (or shared weights). Thus all unit will have the exact same output value for identical input regardless of the location of their respective receptive fields in the data as the shared weights perform the exact same operation over each receptive field. Each feature map is created by passing over the data using the feature detector creating a mapping where consecutive units have consecutive receptive fields. Each consecutive receptive field is slightly shifted on the data from the first field, thus neighboring units on the feature maps correspond to receptive fields that overlap on the data.

As can be seen from the above a slight shift in the data will lead to an equivalent shift in the feature map. This property of feature mappings accounts for the ability of convolutional networks to be invariant to shift (and distortion). Typically a convolutional network will have a number of layers, with the feature detectors of each layer mapping the features of the previous layer. A layer will typically consist of a number of feature detectors in order to create mappings of a variety of different features. Feature maps in subsequent layers can have receptive fields in several of the feature maps of the previous layer, allowing thus for the detection of higher-order features.

In the case of the LeNet-5 convolutional network we have three layers of feature detecting, the first, third and fifth layers labeled C1,C3,C5 respectively. LeNet-5 uses 5×5 receptive fields in all its feature detecting layers. Below we can see how the output of a node i, j on the kth feature map of layer $C1$ when the input is the image I.

$$C1_{k,i,j} = b_k + \sum_{n=1}^{n=5} \sum_{m=1}^{m=5} w_{C1,k,n,m} * I_{i+n-1,j+m-1}$$
and
$$C1_{k,i,j} = A * tanh(S * C1_{k,i,j})$$

Where the bias b_k is the shared bias of all the units belonging to map k and the weights $w_{C1,k,n,m}$ are the shared weights of the connections of these units. The parameters A, S are constants set by the user.

When training, convolutional networks use backpropagation to propagate the error in the output layer back to the nodes in the previous layers, due to the weight replication, a slight alteration of the classical backpropagation is used. Specifically, for each set of weights, the error corresponding to each of the nodes they are applied to is calculated as well as the corresponding updates to the weights. This overall sum of updates is then averaged out and this average used to update the set of weights. Below we see these calculations in the case of the first feature detecting layer C_1 :

$$a_{C1,k,i,j} = a_{S2,k,ceil(i/2),ceil(j/2)} * A * S * \frac{(1-C1^2_{k,i,j})}{A^2} * \frac{w_{S2,k}}{4}$$

and

$$w_{C1,k,i,j} = w_{C1,k,i,j} + \frac{1}{N} * lr * \sum_{n=1}^{\sqrt(N)} \sum_{m=1}^{\sqrt(N)} a_{C1,k,n,m} * I_{n+i-1,m+j-1}$$

Where N is the number of nodes in the feature map $C1_k$ and $1 \leq i, j \leq 5$.

Feature maps detect the presence of different features in the data and record their location. However in certain cases (as in digit recognition) the exact location of the occurrence of different features is not important. What is more important is the approximate location. In fact in cases such as digit recognition the strict recording of locations of features is not only not helpful but indeed can lead to a lower performance. This is easy to see if we imagine a classifier based on such a strict representation. A classifier like this would expect to find specific features characteristic of each digit (say the upper circle in an "8") in specific locations. This, as can be readily seen, is not a realistic expectation, as a bunch of handwritten eights could not possibly be expected to have this feature (the upper circle) in the exact same location. On the other hand the classifier could only be interested in the approximate location of the digit (say that it was in the upper half of the image), this obviously is a much more realistic expectation. In order to achieve the above effect, convolutional neural networks use sub-sampling layers.

Sub-sampling consists of averaging over the data of the feature maps belonging to the previous layer. This leads to a reduction in the resolution of the feature maps. The network is thus less sensitive to shifts and distortions. More precisely each unit in a sub-sampling layer is the result of the averaging of the data in its receptive field with consequent multiplication by a trainable coefficient, the addition of a bias and the application of a sigmoid function. As is the case with feature mappings, the trainable coefficient used by each unit is identical for all units in the sub-sampling layer. Thus sub-sampling also uses weight replication for the trainable coefficient and the bias used. However unlike feature mappings, in the case of sub-sampling layers consecutive units correspond to consecutive but not overlapping receptive fields in the previous layer. Depending on the value of the shared trainable coefficient, the output of the sub-sampling can be considered to be a "noisy OR" or a "noisy AND" if the coefficient is sufficient.

In the case of the LeNet-5 convolutional network we have two layers of sub-sampling, the second and fourth layers labeled S2,S4 respectively. LeNet-5 uses 2×2 receptive fields in all its feature detecting layers. Below we can see how the output of a node i, j on the kth feature map of layer $S2$.

$$S2_{k,i,j} = b_k + \tfrac{w_{S2,k}}{4} \sum_{n=0}^{n=1} \sum_{m=0}^{m=1} C1_{k,2i-1+n,2j-1+m}$$
and
$$S2_{k,i,j} = A * tanh(S * S2_{k,i,j})$$

As in the case of the feature detecting layers, backpropagation of the network error is executed using a slightly altered procedure :

$$a_{S2,k,i,j} = \sum_{a=1}^{K} \sum_{C3_{a,k,o,p} \in AS2_{k,m,n}} a_{C3,a,k,m,n} * A * S * \frac{(1 - S2^2_{k,i,j})}{A^2} * w_{C3,a,k,o,p}$$
and
$$w_{S2,k} = w_{S2,k} + \tfrac{1}{N} * lr * \sum_{i=1}^{\sqrt{N}} \sum_{j=1}^{\sqrt{N}} (a_{S2,k,i,j} * \sum_{n=0}^{n=1} \sum_{m=0}^{m=1} \tfrac{C1_{k,2i-1+n,2j-1+m}}{4})$$

where N is the number of nodes in the feature map $S2_k$ and K the number of feature maps in layer $S2$.

Finally regarding the final two layers of the LeNet-5 convolutional network (the layer F6 and the output layer O, the layer F6 is a regular layer of a neural network fully connected to the previous layer C5 without weight replication, feature detecting or subsampling; the output layer O consists of ten softmaxed output units fully connected to the neurons of the previous layer C5. The calculation of the output of the neurons of these layers and of the backpropagation of the network error is straightforward.

Despite the suitability of convolutional networks for the handwritten digit recognition problem and the high accuracy they attain when labeled data is abundant, experiments conducted with small labeled data sets (up to 15 examples per class) showed that in these cases, convolutional networks are not able to learn at all. Having been trained using these data sets, the convolutional network achieved an accuracy of a little over 10% in every case which is approximately the accuracy that a random classifier would attain. Thus in this case convolutional networks perform very poor and are not appropriate for the task in hand.

3.3.2 Pretrained Networks

Neural networks typically suffer from the problem of weight initialization. As they are not guaranteed to find the global minima of the objective function but rather a local one, their performance can differ significantly according to the initial values of their weights. Different initial weights mean different starting points for minima search and ultimately different solutions to the objective function minimization problem. As an objective function can be expected to have a number of local minima, neural networks typically arrive at a solution that is suboptimal. The difference of this solution from the optimal one can in some cases be substantial.

In order to overcome this problem of initial weight selection, instead of randomly assigning initial values to the weights, the network can be pretrained in order to begin its search from a more appropriate position. One method of pretraining that has been proposed in the case of handwritten digit recognition is that of pretraining the network with the help of restricted Boltzmann machines (RBM) [6].

In this approach in order to pretrain the network, initially each pair of sequential layers in the network are treated as a restricted Boltzmann machine. The input layer of the restricted Boltzmann machine [19] is considered to consist of linear units with Gaussian noise and the hidden layer is considered to consist of units with binary states.

The restricted Boltzmann machine then proceeds to use the available data in order to pretrain the respective weights. More specifically, the RBM minimizes the following objective function which is called the energy of of the RBM:

$$E(v,h) = -\sum_{i \in vis} b_i v_i - \sum_{j \in hid} b_j h_j - \sum_{i,j} v_i h_j w_{ij}$$

where the *vis* units are the units of the input layer and *hid* refers to the units of the output layer. Accordingly, h_i (v_i) are the values of the unit i of the input (output) layer. The value of the weight between two units i,j in the input and output layer respectively is denoted by w_{ij} and the biases of these units by b_i and b_j respectively.

During the training of the RBM, as each example is presented to its input layer, the restricted Boltzmann machine assigns a probability $P(h_j = 1)$ to each of the units of the output layer according to [6]:

$$P(h_j = 1) = \frac{1}{1 + e^{-b_j - \sum_i v_i w_{ij}}}$$

Based on this value, a confabulation is produced for each unit in the input layer :

$$v_i = \frac{1}{1 + e^{-b_j - \sum_j h_j w_{ij}}}$$

Having calculated the above confabulation, the RBM then once again calculates probabilities for the units of the hidden layer using the same equation as before.

After the Boltzmann machine has been presented with all the data, it updates the weights between the two layers according to[6]:

$$\Delta w_{ij_k} = m * \Delta w_{ij_{k-1}} + lr * (< v_i, h_j >_d - < v_i, h_j >_c + cost * w_{ij})$$

The value of $< v_i, h_j >_d$ is the fraction of times the units i and j, in the input and output layer respectively, are both "on" when the input of the machine is an example of the data set; the value of $< v_i, h_j >_c$ is accordingly the fraction of times the units i and j, in the input and output layer respectively, are both "on" when the input of the machine is a confabulation. At each update k, the value of the previous update $k - 1$ is also taken into account via the term $m * \Delta w_{ij_{k-1}}$. Furthermore the value of the weight itself is also taken into consideration via the term $cost * w_{ij}$. As can be seen by the update rule given, the training process aims to minimize the energy over the data while raising the energy of the confabulated data.

Once the restricted Boltzmann machine is trained, the process of pretraining weights continues to the next pair of network layers where obviously the new input layer is the previous output layer. Having pretrained the network and obtained weights that hopefully are more suitable, the network can then be trained. In the case of the pretrained network presented here, the network is subsequently trained using a conjugate gradient method [18].

In the conjugate gradient method the search for the (local) minima of the function is conducted in directions that are A-orthogonal.

Two vectors are A-orthogonal (or conjugate) if the following equation holds :

$$d_i{}^T * A * d_j = 0$$

Choosing the search directions to be A-orthogonal guarantees the method will find the (local) minima after at most n steps, where n is the dimensionality of the search space. As is known the minimization of a quadratic function $f(x)$ requires the determination of a set of values for which $f'(x) = 0$. These values are determined by the solution of an equation of the form:

$$Ax = b$$

In each of the A-orthogonal directions d_{i-1} a search is conducted to find the value x_i for which the error vector $e_i = Ax_i - b$ is A-orthogonal to d_{i-1}. The directions chosen for search are given by the following iterative equations:

$$\begin{gathered} d_{i+1} = r_{i+1} + \beta_{i+1} d_i \\ \beta_i = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}} \\ \text{and} \\ r_i = -Ae_i \end{gathered}$$

Where the vectors r_i are the residuals. For these residuals we have also that $r_i = b - Ax_i$. In the case of nonlinear problems these vectors are in fact the directions of steepest descent.

In the experiments conducted for the conjugate gradient fine-tuning, Carl Rasmussen's minimize code was used .Batch training was implemented in the fine-tuning stage and for each epoch and for each batch presented for training, three line searches were conducted. The restricted Boltzmann machines were each trained for 50 epochs with a learning rate of 0.1, a cost weight of 0.0002, the parameter m was set to 0.5 for the first five epochs and 0.9 for all subsequent. The experiments were repeated ten times, each time drawing the labeled examples from the data set using a uniform distribution without replacement and the trained network tested on the remaining unlabeled examples. The results of these experiments can be seen in figure 25.

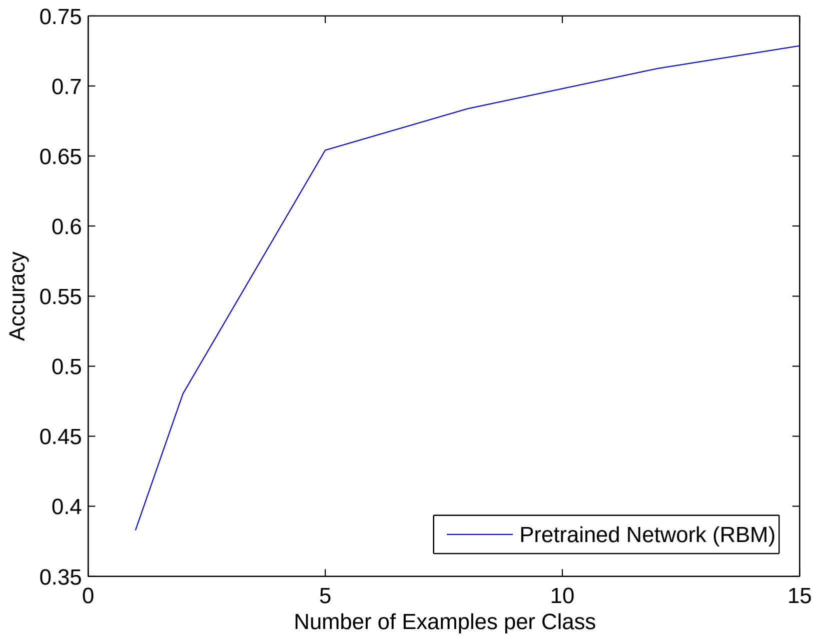


Figure 25

The results show that the pretrained network ,unlike the convolutional networks, are in fact able to learn from such small amounts of data. However the results are far from satisfactory, it is characteristic that for one or two labeled examples per class the network has an accuracy of below 50% and though this is much better than random (which would be 10%) it still means that the resulting classifier is wrong more times that it is right.

3.4 Transductive Learning

3.4.1 Expectation Maximization

In the case of transductive learning using expectation maximization, when applied to handwritten digit recognition, we can no longer use a naive Bayes classifier as a base learner. A naive Bayes classifier (based on a multinomial model) considers each example to be an ordered series of events. This approach does not seem appropriate when it comes to images, as in this case the classifier would consider a pixel of intensity 255 (for example) to be the result of a specific event (corresponding to specific pixel) occurring 255 times, which would seem to make little sense. Thus rather then use a naive Bayes classifier as a base learner, we must turn towards other solutions. An alternate framework that would seem to be well suited for the transductive expectation maximization is that of considering the data as generated by a mixture of normal (Gaussian) densities[5]:

$$p(x) = \sum_{i=1}^{c} P(\omega_i) \frac{1}{\sqrt{2\pi}|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)}$$

where A^T is the transpose of A. The probabilities $P(\omega_i)$ are the prior probabilities of each class ω_i, while the values μ_i and Σ_i are the mean and covariance matrix respectively that correspond to class ω_i.

Based on the above assumption given a specific example x_k, the probability $P(\omega_i|x_k)$ is given by[5]:

$$P(\omega_i|x_k) = \frac{p(x_k|\omega_i)P(\omega_i)}{\sum\limits_j p(x_k|\omega_j)P(\omega_j)}$$

and the most probable class for the example x_k is given by[5]:

$$\omega(x_k) = argmax_i P(\omega_i|x_k) = argmax_i \frac{p(x_k|\omega_i)P(\omega_i)}{\sum\limits_j p(x_k|\omega_j)P(\omega_j)}$$

As the denominator $\sum\limits_j p(x_k|\omega_j)P(\omega_j)$ is common for all $P(\omega_i|x_k)$ it can be ignored and thus the above equation becomes:

$$\omega(x_k) = argmax_i p(x_k|\omega_i)P(\omega_i)$$

and substituting $p(x_k|\omega_i) = \frac{1}{\sqrt{2\pi}|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)}$ we obtain the final form :

$$P(\omega_i|x_k) = P(\omega_i) \frac{1}{\sqrt{2\pi}|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)}$$

In the above final equation, the mean of each class ω_i is the expected value of an example generated by the mixture component corresponding to that class $\mu_i = E[x]$. Given a data set D, if the maximum-likelihood estimates are used then the mean of each class ω_i is given by[5]:

$$\mu_i = \frac{1}{n} \sum_{k=1}^{n} x_k$$

where obviously, x_k is considered to belong to class ω_i. Equivalently the maximum-likelihood estimate of the covariance matrix Σ_i of class ω_i is given by[5]:

$$\Sigma_i = \frac{1}{n}\sum_{k=1}^{n}(x_k - \mu_i)^T(x_k - \mu_i)$$

where as above x_k is considered to belong to class ω_i. Finally the probabilities $P(\omega_i)$ are given by [5]:

$$P(\omega_i) = \frac{n}{N}$$

where n is the number of examples in the data set belonging to class ω_i and N is the size of the data set.

Based on the above we can use this framework for transductive learning with the help of the expectation maximization algorithm. At each E-step of the algorithm the parameters μ_i,$P(\omega_i)$ and Σ_i are calculated for each class ω_i using the data. In the first E-step only the labeled data is used while in subsequent steps the extended data set is used as after the first M-step all the examples are assigned a label. Subsequently in the M-step, the probabilities $p(x_k|\omega_i)$ are calculated $\forall k, i$ and each example x_k is assigned a label accordingly.

Unfortunately even though the presented framework ties in naturally with the expectation maximization algorithm, in the case of the experiments presented here it cannot be applied due to a number of problems that arise and make the algorithm in its present form inapplicable.

In particular, the experiments conducted here focus on the case where there are very small amounts of labeled data. These settings are natural when conducting experiments on transductive learning as when large amounts of labeled data are present, supervised methods are more appropriate.

This scarcity of labeled data however means that though it is possible to calculate the maximum-likelihood estimates of the covariance matrices Σ_i, these estimates will not be inversible. This is due to the fact that in order for the covariance matrix $\Sigma_i = \frac{1}{n}\sum_{k=1}^{n}(x_k - \mu_i)^T(x_k - \mu_i)$ to be inversible the number of examples n must be at least as large as the dimensionality of the data d. As the data has 784 dimensions and the labeled data sets used here are of a size of maximum 15 examples per class, it is evident that the resulting matrices will not be inversible.

One way to overcome this problem is to assume that every class has the same covariance matrix Σ. The entire data set (both labeled and unlabeled data) can then be used to calculate this covariance matrix and since the size of the data set N is 4000 examples it is obvious that $N \geq D$. Thus we have :

$$\Sigma = \frac{1}{n} \sum_{k=1}^{N} (x_k - \mu_i)^T (x_k - \mu_i)$$

Unfortunately, though this approach overcomes the initial approach of insufficient labeled data, it suffers from a further setback. As stated the data consists of images of digits which have been centered on a 28×28 background. This means that the pixels near the border of the image tend to have a value of zero, as the pixels of the digit tend to be more towards the center. In fact there are lines of pixels whose value is constantly zero over the entire data set. This unfortunately means that the respective rows in the covariance matrix Σ are also zero which in turn means that $rank(\Sigma) < d$. Thus once again the covariance matrix is not inversible.

In order to overcome this obstacle, we employ two different approaches that ultimately prove successful. The first approach is to use a technique known as shrinkage which "'shrinks"' the covariance matrix Σ towards the identity matrix I[5]:

$$\Sigma_\beta = (1 - \beta)\Sigma + \beta I$$

The results of experiments conducted using this approach can be seen in figure 26 where the accuracies for different numbers of labeled examples per class in relation to the number of steps of the expectation maximization algorithm are shown. The settings of the experimentation (creation of training set, test set and repetitions) remain the same as in the supervised experiments. The parameter β is set experimentally to 0.8.

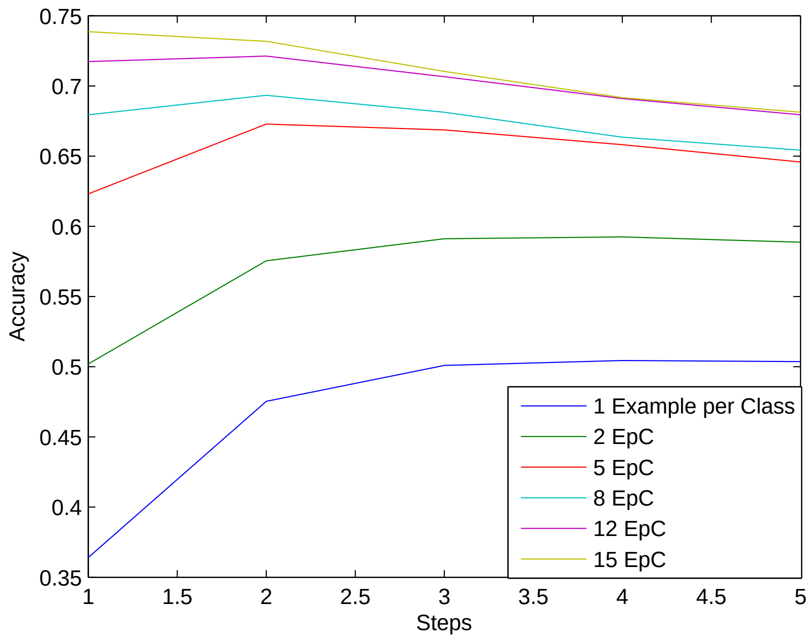


Figure 26

As can be seen in almost every case, after one or at the most two iterations of the algorithm, the performance of algorithm ceases to improve and from then on any further iterations are counterproductive. If we chose to run only two iterations (which seems to be the optimal) then the performance of the algorithm can be seen in figure 27.

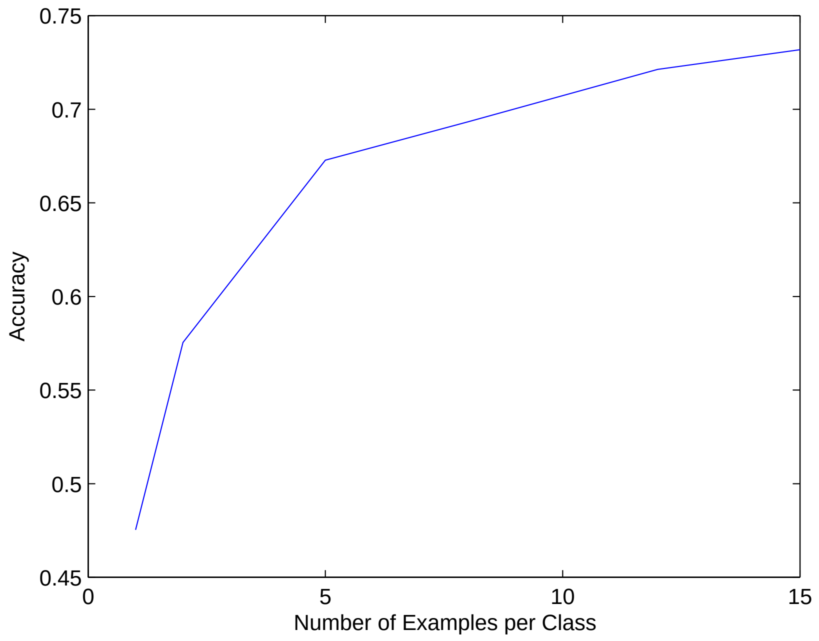


Figure 27

It is evident from the results obtained that though the self-training algorithm may perform better than the supervised pre-trained networks, they nonetheless do not seem to completely overcome the problem of poor performance for small amounts of labeled data. Thus in this case the presence of unlabeled data may lead to an improved performance, however this performance is still relatively low.

The second approach to calculating an inversible covariance matrix is to assume that the features are statistically independent and that each of the features has the same variance σ^2. This leads to a common covariance matrix of the form[5]:

$$\Sigma = \sigma^2 I$$

which is obviously inversible. In order to calculate the variance σ^2, an average of the variance over the data set of each feature is calculated. The results obtained using this approach can be seen in figure 28 for various numbers of iterations of the algorithm.

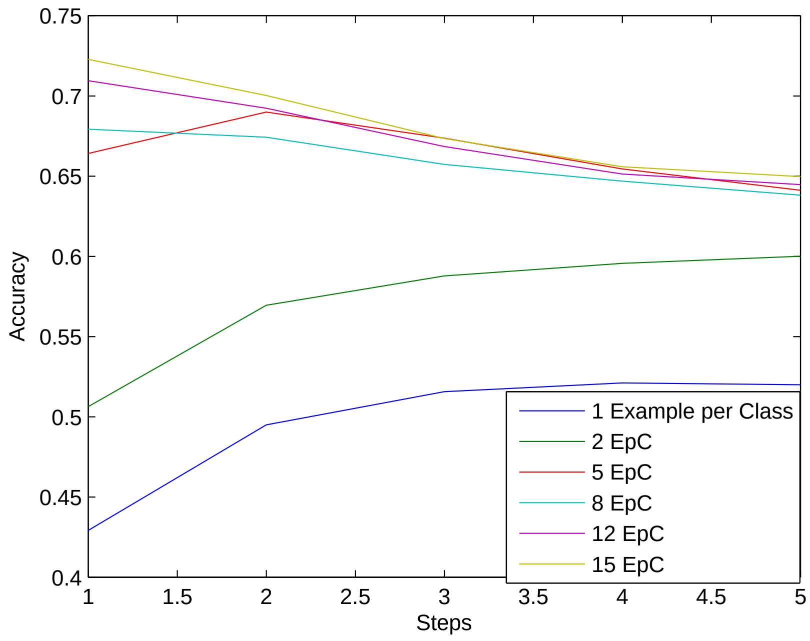


Figure 28

As can be seen with the exception of the cases where only one or two examples per class are labeled, the algorithm does not gain in performance from using expectation maximization. Thus in order to use the expectation maximization algorithm we must revert back to the previous approach.

3.4.2 Self-Learning

As was the case with the transductive learning algorithm using expectation maximization, in the case of self-learning once again a naive Bayes classifier is not a suitable base learner for the same reasons as above.

Instead of using a naive Bayes classifier, again as before we assume that the data has been generated by a mixture of Gaussian distributions. The self-training algorithm in this case, calculates the covariance matrix Σ (using the approaches described) the means of the classes μ_i and the prior probabilities $P(\omega_i)$, the latter two using the labeled data while the former using the entire data set.

The algorithm then calculates the probabilities $P(\omega_i|x_k)$ and for each class ω_i labels the example x_k for which the value $P(\omega_i|x_k)$ is maximal over the data set. The algorithm then continues to iteratively calculate parameters and label examples, extending the labeled data set each time by one example per class. Unfortunately as can be seen from figure 29, in the case of self-training the results obtained show that the performance of the learner does not improve. The results are obtained using the standard experimentation settings used in the previous sections.

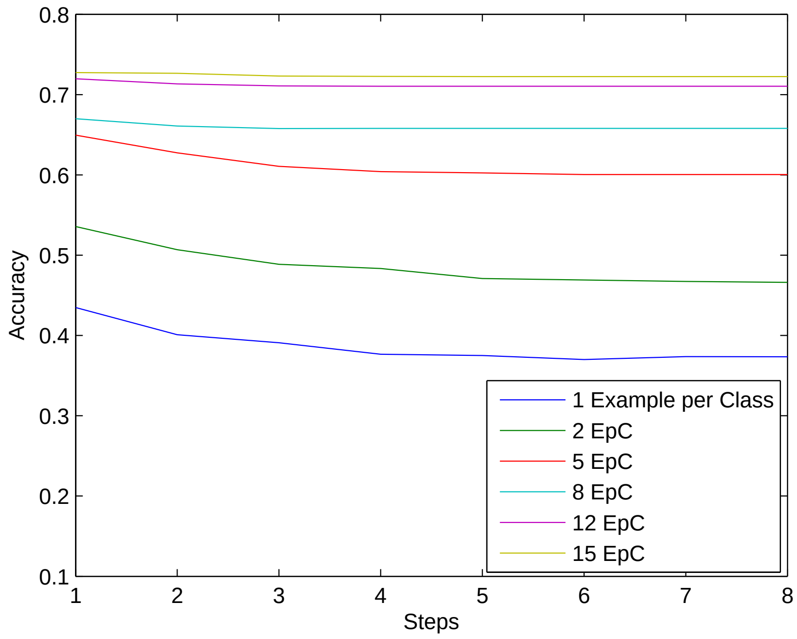


Figure 29

As the framework used with the expectation maximization algorithm does not seem to work in the case of self-learning, we must choose a different learner as a base, in order to obtain a self-learning transductive classifier with enhanced performance. For this purpose we experiment with using a class of neural networks known as autoencoders [7].

Autoencoders are typically used to reduce the dimensionality of the data. The main architectural idea behind these networks is a bottlenecked network which has as many output neurons as input and has a hidden layer with as many neurons as the desired dimensionality of the data (obviously less than the original dimensionality). By equating the target of the network with the input presented each time we force the bottlenecked hidden layer to learn a representation of the data. As the autoencoding network is trained to replicate its input in its output layer, this means that the values of the neurons of the bottlenecked hidden layer hold adequate information so as to allow for the (optimally perfect) reconstruction of the input by the succeeding part of the network. In figure 30 the architecture of such an autoencoder can be seen.

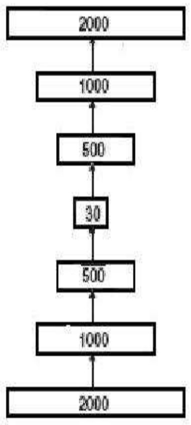


Figure 30

In the experiments conducted here we use a series of autoencoding networks in a slightly different fashion. Instead of aiming to reduce the dimensionality of the data, we use a number of autoencoders (as many as the different classes in the data) to create a classifier. Each of the autoencoders is trained using only that data which belongs to a specific class. This way, each autoencoder is trained to replicate only those examples that belong to its specific class. The unlabeled data is then passed through each of the autoencoding networks and each example is classified according to which autoencoder best replicates the example in its output.

As in the case of the supervised pretrained network, the weights of the autoencoding networks are pretrained before the actual training phase. The weights between layers up until the bottlenecked layer are as before considered to belong to a restricted Boltzmann machine whose input layer consists of linear units with Gaussian noise and whose hidden layer consists of units with binary states.

The training process of this RBM is very similar to the process in the supervised case. For each example presented to its input, the RBM calculates the probabilities of the output units h_j being "'on"' [6]:

$$P(h_j = 1) = \frac{1}{1+e^{-b_j - \sum_i v_i * w_{ij}}}$$

As before, based on these values, a confabulation is also calculated :

$$v_i = \frac{1}{1+e^{-b_j - \sum_j h_j w_{ij}}}$$

and subsequently probabilities $P(h_j = 1)$ are calculated anew.

After all the appropriate examples have been presented to the RBM, its weights are updated. The update rule used for the weights (for the kth update) is :

$$\Delta w_{ij_k} = m * \Delta w_{ij_{k-1}} + lr * (< v_i, h_j >_d - < v_i, h_j >_c + cost * w_{ij})$$

where at each update k, the value of the previous update $k - 1$ is also taken into account via the term $m * \Delta w_{ij_{k-1}}$. Furthermore the value of the weight itself is also taken into consideration via the term $cost * w_{ij}$).

The terms $< v_i, h_j >_d$ and $< v_i, h_j >_c$ are as before the fraction of times the units i and j, in the input and output layer respectively, are both "on" when the input of the machine is an example from the data set or a confabulation accordingly. The values of these terms are given by :

$$< v_i, h_j >= \frac{1}{N} \sum_t v_{it} * P_t(h_j = 1)$$

where v_{it} is the value of unit v_i when the RBM is presented with the example (or confabulation) t.

In the case of the bottlenecked layer, the RBM which has the units of this layer as an output is considered to have an output layer consisting of linear units. For each of these units, given a state of the input units $\mathbf{v}$, the mean μ_j of the output unit h_j is calculated

$$\mu_j = b_j + \sum_i w_{ij} * v_i$$

The variance for each output unit is considered to be the same and equal to : $\sigma^2 = 1$.

Having calculated the corresponding means, the machine then calculates the values of the output units by drawing from a normal distribution $N(\mu_j, \sigma^2)$. Based on these values, as before, a series of confabulations is calculated along with the corresponding values of the output units for those confabulations.

Finally the weights are updated using the same update rule as before, only that in this case, the values of the terms $< v_i, h_j >_d$ and $< v_i, h_j >_c$ are given by :

$$< v_i, h_j >_{recon} = \frac{1}{N} \sum_t v_{it} * h_j$$

The process of pretraining the weights of the autoencoder is conducted up until the weights prior to the bottlenecked layer. The weights of the subsequent layers are obtained by "unrolling" the network. Thus the weights between two layers situated after the bottlenecked layer are assigned the same values as the weights between the two corresponding layers prior to the bottleneck. This process of "unrolling" can be seen in figure 31.

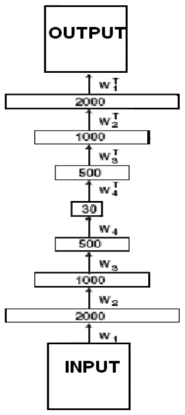


Figure 31

Subsequently the autoencoder is trained using the labeled examples as both input and target values. As before the method used during the training phase is the conjugate gradient method.

After the ten autoencoders have been trained, the unlabeled examples of the data set are presented to each of these autoencoders. For each example x_k and for each autoencoder i the reconstruction error $E_i(x_k)$ is calculated. If o_i is the output of autoencoder i, then the reconstruction error is given by:

$$E_i(x_k) = \|o_i - x_k\|_2$$

Having passed through the entire set of unlabeled data, for each class i the unlabeled example x_k for which $E_i(x_k)$ is minimal over all the unlabeled data set is found and appropriately labeled, thus extending the labeled data set by one example per class.

In figure 32 we can see the performance of the self-training algorithm based on autoencoders for various numbers of iterations.

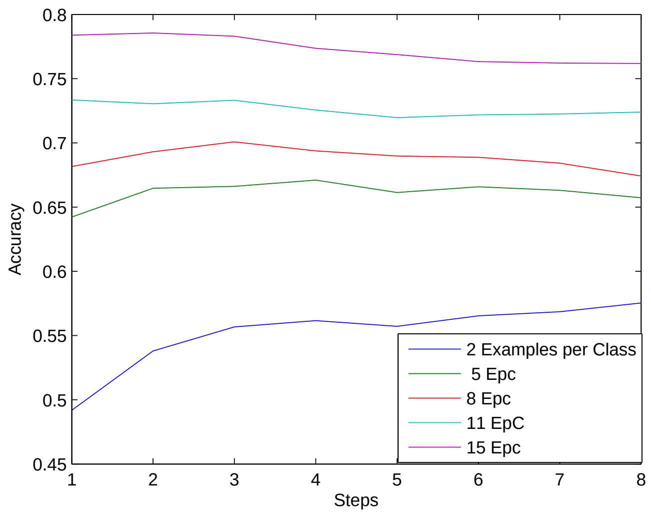


Figure 32

From figure 32 we can see that the performance of the algorithm seems to decline after a certain number of iterations, especially as the number of labeled examples per class rises. By fixing the number of iterations to three which seems according to the previous results to be the optimal, we obtain the results shown in figure 33, where the usual settings are used.

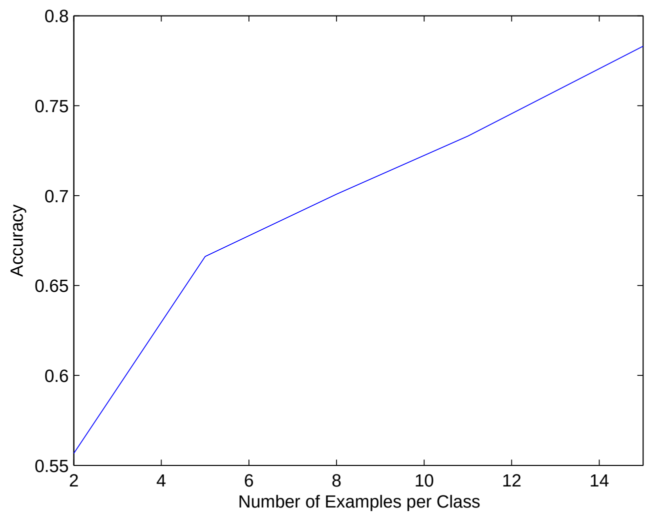


Figure 33

As can be seen by the results, the expectation maximization algorithm using autoencoders does not completely overcome the problem of low amounts of labeled data, as we shall see this transductive algorithm surpasses the supervised pretrained network algorithms in performance but as is evident from the results seen in figure 33, its performance is still not high.

3.4.3 A Convolutional Network Analogous

As can be seen by the results of the experiments conducted using convolutional networks, this supervised approach is not suitable when the number of labeled examples is very low. This inappropriateness can be readily explained if we consider the size of the convolutional network and the number of trainable parameters it has. Networks of this size typically require thousands of examples in order to be trained so it is not surprising that training fails when only a handful of labeled examples are available.

Despite the extremely poor performance of convolutional networks in these settings, we can use the ideas it is based on (feature mapping and sub-sampling) to come up with a transductive learner which employs these two techniques with the help of the large quantities of unlabeled data to obtain a new representation of the data which allows for higher performance. As such the algorithms presented in this chapter are not classifiers themselves but rather a transductive method of acquiring a new data representation that can subsequently be combined with a classifier which will attain a higher performance.

Feature Mapping In order to create a feature mapping of the data that can exploit the available unlabeled data, we create feature detectors that can manipulate this data. To this extent we utilize a self organizing map (SOM) that will be trained to recognize the presence of certain features and thus serve as a group of feature detectors. This self organizing map takes a small number of input units which are situated in a small neighborhood of the data point (i.e. the SOM's receptive field), and reveals in its output which feature is present in that neighborhood according to which of the SOM's units is activated by the input.

The SOM is originally trained using as input neighborhoods of units in the training data. Thus in the case where the SOM has a 5×5 input layer (as will be used here) the data used to train the SOM are 5×5 neighborhoods of input units taken from the original training data. By presenting the SOM with receptive fields taken from the data, it is trained to recognize a number of different features that are present in these 5×5 windows. Thus each neuron in the SOM's output layer becomes sensitive to a different feature.

Having trained the SOM, we subsequently scan each data point using this SOM in order to obtain a feature mapping of the data. For each data point the SOM is applied to consecutive receptive fields on the data point. Each of these receptive fields is shifted slightly in relation to the previous receptive field. Thus two consecutive receptive fields are overlapping as can be seen in Figure 34.

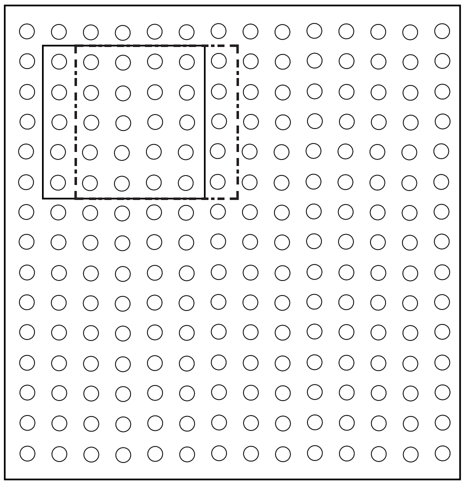


Figure 34

By scanning each data point with the SOM, a feature mapping of the data point is created by recording for each receptive field which neuron in the SOM's output field is activated. Thus the feature mapping of the data consists of a list of numbers as can be seen in Figure 35 where each of these numbers corresponds to a specific neuron in the SOM's output field and denotes the presence of a specific feature in the corresponding receptive field.

1	4	7	12	11	9	3	5	6	1
5	9	1	4	10	8	7	1	3	12
8	9	3	5	4	1	11	7	2	1
10	11	4	7	1	2	5	9	3	1
1	12	1	7	11	4	1	3	1	9
3	1	2	5	7	1	9	10	4	1
1	9	1	2	1	4	7	1	3	1
5	1	9	1	12	1	3	2	7	4
1	11	5	4	1	7	9	1	3	1
1	1	8	4	3	2	9	11	5	7

Figure 35

Sub-Sampling Once a feature map of the data has been created, it is useful to find a way to execute sub-sampling. By using sub-sampling the algorithm will become even less sensitive to small distortions in the data as the exact positions of the occurring features will no longer be as important as their approximate positions.

To this effect, after creating the feature maps of the data, these feature maps are clustered in K different clusters. For each one of these clusters k we follow the following procedure :

1. A specific region (neighborhood) of units is extracted from the data points belonging to that cluster (here 2×2 neighborhoods are chosen).
2. The center of the cluster formed by these subdimensions is calculated. This center is treated as an archetype for that subregion in the specific cluster k
3. The above steps 1-2 are repeated scanning all the subdimensions of the data points. The subregions are chosen in such a way that they do not overlap. By this repetition we obtain a series of archetypal subregions for the entire data vector.
4. Steps 1-3 are repeated for all the clusters k.
5. Finally for each example in the data set, we compare each of its 2×2 subregions with the respective archetypes obtained for each of the clusters. The subregions of the data are then replaced with the archetype with which it has the greatest similarity.

After this procedure has been executed the subdimensions of each data point consist of a series of archetypal subregions. Thus slight shifts or distortions in the original feature maps are lost as subregions exhibiting similar features will most likely be clustered together, and ultimately be replaced by a common archetype.

Distance between Examples Once a feature mapping of the data has been created and sub-sampling has been executed, the data acquires a new form of representation. However as the data is represented by a list of numbers denoting the presence of features and their locations, the Euclidean distance function is no longer an appropriate distance measure between two points. Instead the distance between two examples is calculated using a distance function based on the distances between neurons in the SOM's output layer.

More specifically the distance between two data points I_i and I_j is set to be the sum of the distances between the neurons in the SOM, for each pair of neurons activated by the same receptive fields in the SOM. More formally the distance between these two data points is calculated as

$$dist(I_i, I_j) = \sum_{k=1}^{M} d_{SOM}(I_i(k), I_j(k))$$

where $I_i(k), I_j(k)$ denote the neurons activated by the k^{th} receptive field of the two data points and where the distance function $d_{SOM}(x, y)$ returns the distance between neurons x and j in the self organizing map.

Results using these methods can be seen in the following sections where the convolutional network analogous is combined with other transductive methods to form improved transductive classifiers.

3.4.4 Spectral Graph Partitioning

As analyzed in the previous chapter, the spectral graph transducer calculates the cut of a graph such that the average weight of the cut is minimal. Having found this cut, the algorithm labels the examples corresponding to nodes on the one side of the cut as positive and the examples corresponding to the rest of the nodes as negative. As such the spectral graph transducer is appropriate for binary classification problems.

In the case however of the MNIST data set for handwritten digit recognition, the problem at hand is a multiclass one. Thus in order to be able to apply the spectral graph transducer to the handwritten digit recognition problem, it must first be altered in order to be able to handle multilabel problems.

This adaptation is achieved by reconstructing the problem from a single multiclass problem to a series of binary classification problems. Thus in the case of c classes, we construct as many spectral graph transducers and train them using a one against all approach, for example the first spectral graph transducer is trained to discern between the digit 1 and the rest; in this case the examples of class "-" are considered to be 1s while the examples in class "+" are considered to be one (any) of the other digits. In this case the adjacency matrix is constructed based on the Euclidean distance of the examples,].

The results when using the original representation of the data can be seen in figure 36 using the usual settings. We set $c = 3200$ and the adjacency matrix is constructed using the 5 nearest neighbours of each example.

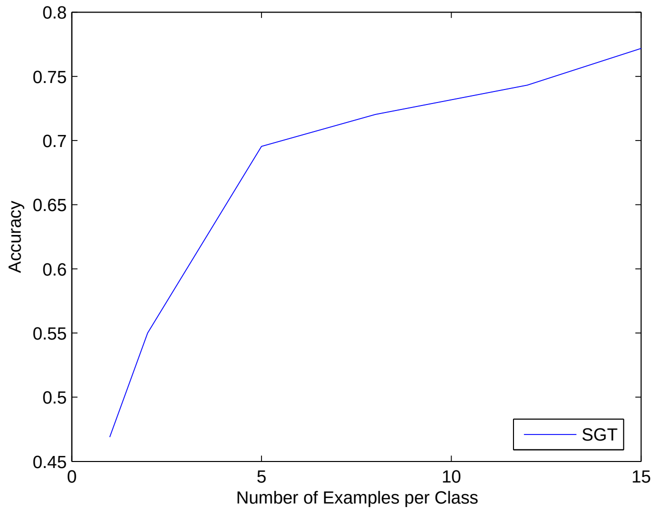


Figure 36

Besides however applying the spectral graph transducer on the data using the original representation, the convolutional analogous presented in the previous chapter can also be used to obtain a new representation of the data. The adjacency matrix of the data is then constructed using the distance function analyzed in the previous chapter. Figure 37 shows the result of applying the transductive learner on the data when only feature mapping has been applied and when both feature mapping and sub-sampling have been applied; the previous results on the original representation are also shown for comparison.

In the case where only feature mapping is applied, the experiments are repeated 25 times, 5 different self-organizing maps are trained using the data and for each of these SOMs we obtain a different representation of the data, for each of these representations we calculate an adjacency matrix for the data and repeat 5 experiments with the spectral graph transducer.

In the case of both applying both feature mapping and sub-sampling, for each of the five representations (from the five different

SOMs) we cluster the data four times each time receiving a new sub-sampled representation. Thus we effectively obtain 20 different representations of the data. For each of these representations we repeat the experiment five times, effectively running 100 experiments for each amount of labeled data.

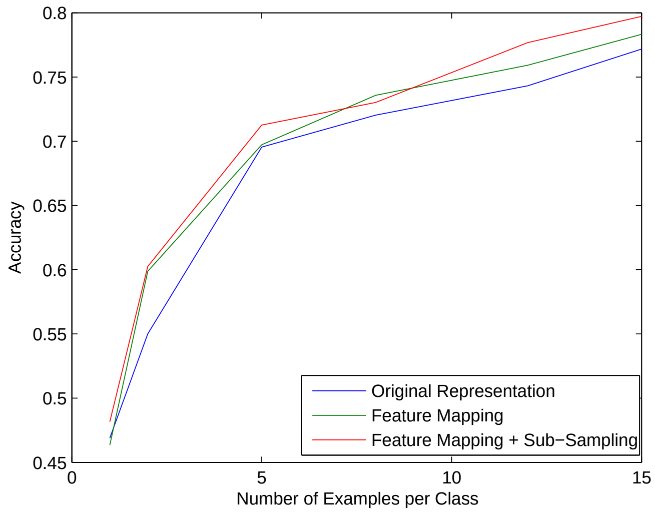


Figure 37

As can be seen from the results above, applying feature mapping to the data prior to executing the spectral graph partitioning leads to an increase of the performance of the resulting classifier. Furthermore if sub-sampling is applied subsequent to the feature mapping and then the algorithm executed, the performance improves even further. Based on these results we can see that the convolutional analogous proposed does in fact contribute to the learning process.

3.4.5 Learning with Local and Global Consistency

In the case of transductive learning with local and global consistency, the application of the algorithm to the handwritten digit recognition problem is straightforward. As the algorithm is already adapted to handle multiclass problems, no alterations to the algorithm is needed. The only difference with the algorithm applied to the document classification problem is that the construction of the affinity matrix is slightly changed to better represent the nature of the problem.

The values of the affinity matrix W are given by :

$$W_{ij} = e^{-\|x_i - x_j\|^2/2\sigma^2}$$

when $i \neq j$ and $W_{ij} = 0$ when $i = j$.

Using this affinity matrix, we obtain the results in figure 38 when applying the tranductive learner to the original representation of the data. The value of σ is set to 1.25. The experiments are repeated ten times, the labeled examples being drawn using a uniform distribution without replacement and the resulting classifier tested on the remaining examples.

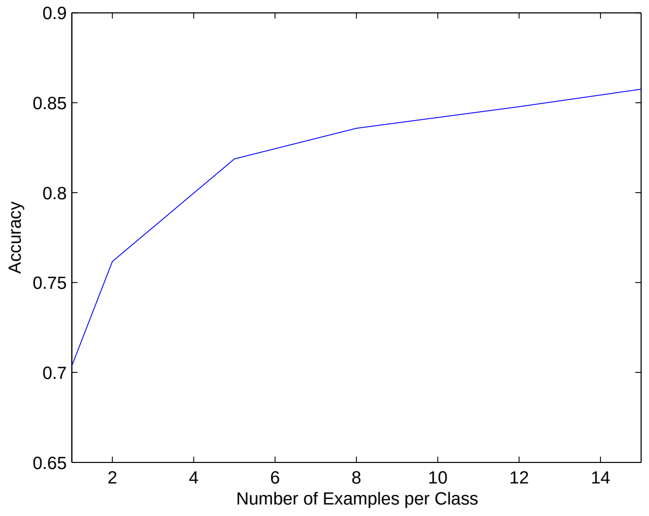


Figure 38

As can be seen the algorithm attains a very high performance on the handwritten digit recognition problem when applied to the original representation of the data. Unfortunately in this case applying the algorithm to the representations obtained by the convolutional network analogous, is not straightforward, this is due to the parameter σ.

The algorithm is extremely sensitive to the parameter σ, the performance varying greatly in relation to its value. Thus although the value 1.25 proves to be optimum when the data is in its original representation, in the case where feature mapping has been applied the performance is very low. This is not due to the expressive power of the new representation but rather is due to the poor choice of σ.

A search for a suitable value for σ in the case where feature mapping has been applied, yielded the results shown in figure 39 where the number of labeled examples per class is 15. The settings of the experimentation are the same as before.

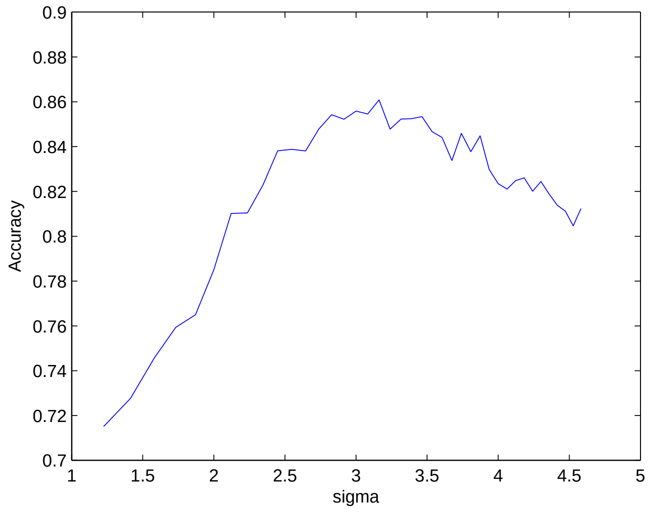


Figure 39

Based on the above the optimum value for σ is 3.16, for which we obtain the following results which are shown together with the previous results for comparison. As can be seen in the case of the learning with local and global consistency algorithm, using feature mapping only slightly increases the performance of the algorithm. Furthermore we have set the value σ with hindsight, though the value in the case of the original representation was also set to its optimum value. As in the case of feature mapping and spectral graph partitioning the experiments have been repeated 25 times.

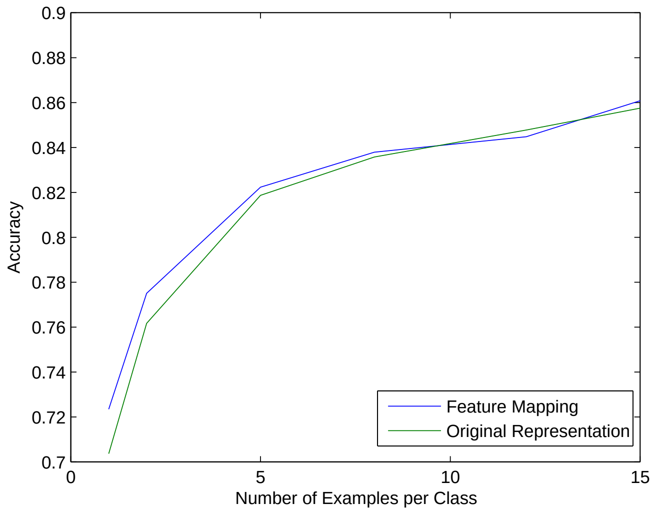


Figure 40

In the case of using sub-sampling, a similar search as above was conducted and an optimum value of $\sigma = \sqrt{\frac{1}{2}}$ was found. Unfortunately even for this value the algorithm performs poorly on this representation (compared to the other two representations), as can be seen in figure 41. As in the case of sub-sampling and spectral graph partitioning, the experiments have been repeated 100 times.

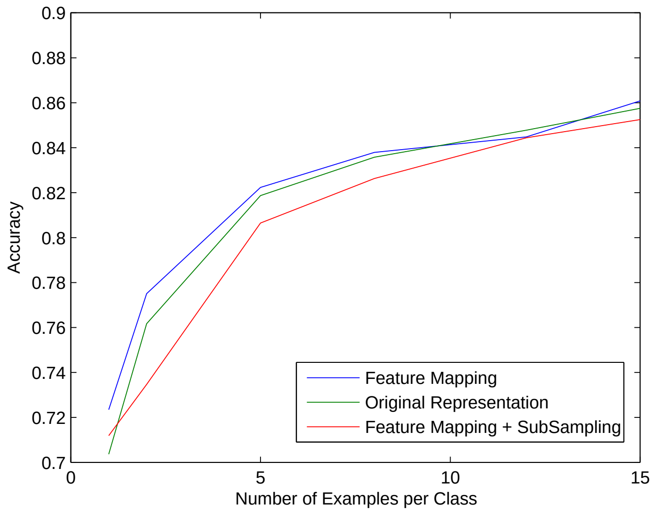


Figure 41

3.4.6 Transductive Clustering

In the case of handwritten digit classification, besides the algorithms already presented we can also apply another transductive algorithm which we could not practically apply in the case of document classification due to the dimensionality of the data. The algorithm in question is transductive clustering.

In the case of transductive clustering, the data is originally clustered using an unsupervised clustering algorithm like for example the k-means algorithm. In this step of the algorithm all the data is treated as unlabeled. The reason this algorithm could not be (practically) applied in the case of document classification is exactly due to this step; the complexity of a single iteration of the k-means algorithm is $O(knd)$ where n is the number of examples, k is the number of clusters and d is the dimensionality of the data. Thus as the dimensionality of the data rises, the k-means algorithm tends to become computationally intractable and thus impractical.

Returning to the transductive clustering, once the data has been clustered, the algorithm finds those labeled examples which are closest to each of the cluster centers. The examples belonging to each cluster are then labeled according to the label of the closest data point to the cluster's center.

Using the original representation of the data, we conduct experiments with transductive clustering and obtain the results seen in figure 42. The experiments here are repeated 25 times, the data is clustered 5 times and for each of these clusterings, experiments are repeated 10 times by choosing ten different sets of labeled examples by choosing these sets by a uniform distribution without replacement.

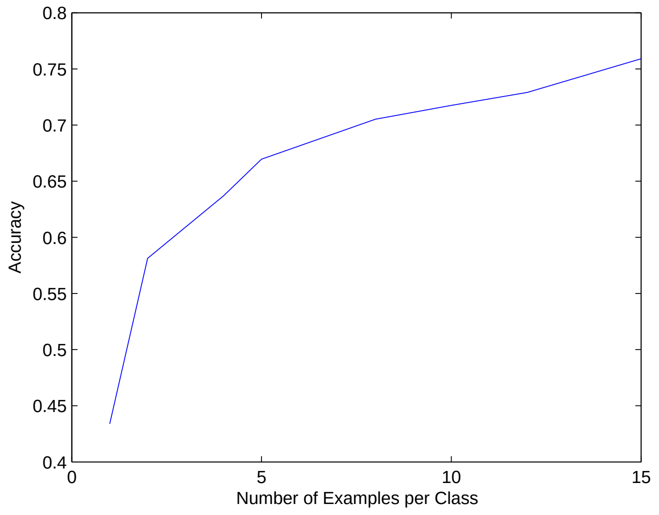


Figure 42

We furthermore conducted experiments using the representations obtained by feature mapping and subsampling and obtained the results seen in figure 43. As in the case of spectral graph partitioning the experiments are repeated 25 and 100 time respectively for feature mapping and sub-sampling. From the results it is evident that transductive clustering achieves a quite higher performance when it is applied to the new data representations using the convolutional network analogous.

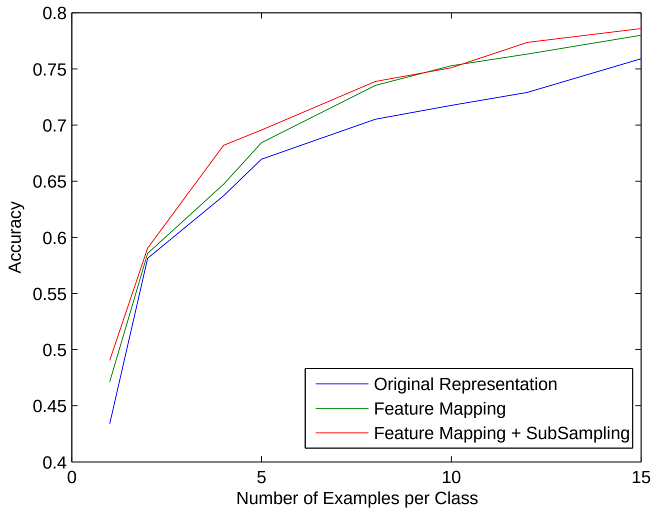


Figure 43

3.5 Results

In figure 44, we can see the performance of the different transductive learning algorithms as well as that of the supervised pretrained networks algorithm.

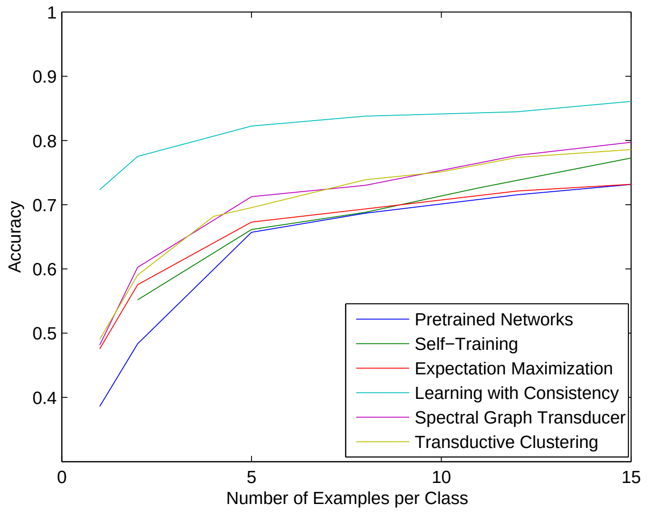


Figure 44

It is evident from these results that in the case of handwritten digit recognition transductive learning can prove to be very helpful when only small amounts of labeled data are present. As can be seen all of the transductive learning algorithms surpass the performance of the supervised methods (the convolutional networks effectively randomly classify the data and thus their results have been omited). Learning with local and global consistency performs especially well surpassing the rest of the algorithms by a large margin.

Of the rest of the algorithms, the spectral graph transducer and transductive clustering perform substantially better than the expectation maximization and the self-training algorithms, surpassing them by 2-5%.

Finally the last two transductive learners, expectation maximization and self-training perform only slightly better than the supervised algorithm when the labeled examples are about 10 to 15 examples per class. However when the labeled examples are truly few (1 to 5 per class) than even these algorithms greatly outperform the supervised pretrained networks as can be seen in the figure 45.

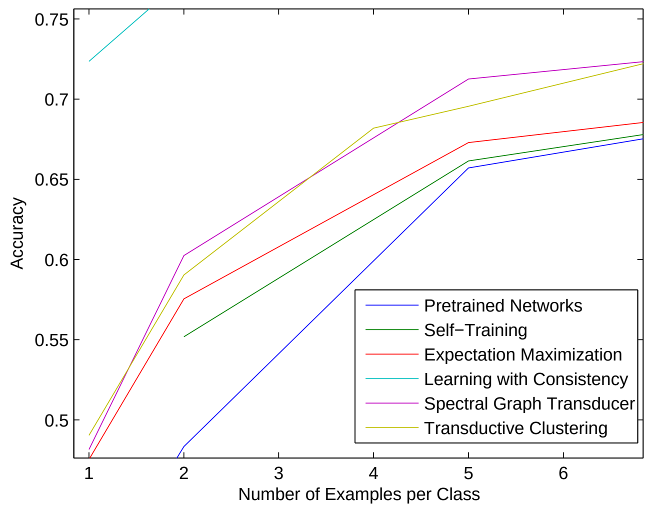


Figure 45

Finally regarding the convolutional network analogous proposed here for obtaining a new representation of the data with the help of large amounts of unlabeled data, as was seen in the previous chapters, wherever it was applicable, when applied it increased the performance of the respective transductive learner (though in the case of learning with local and global consistency only marginally). This holds true both for the feature mapping technique proposed but also for the sub-sampling technique proposed. This form of feature extraction is especially suited for a transductive learning setting as in those cases we have large amounts of unlabeled data which the convolutional network analogous can readily use.

4 Final Conclusions

In the present project we concentrated on transductive learning for both document classification and handwritten character recognition. Besides experimenting with the performance of well known transductive classifiers on document classification we further investigated how these could be adapted to apply to handwritten character recognition. This allowed us to experiment with the performance of transductive learning algorithms not only on to different applications but also on both a binary and multi label problem.

The aim of the these experimentations was to ultimately answer the simple question:

`Can transductive learning overcome the lack of labeled data with the help of the unlabeled data and return classifiers with increased performance?`

Based on the results obtained, the answer in both cases, document classification and handwritten character recognition, binary label and multi label, straightforward application and adapted application is yes; transductive learning was shown in both cases to lead to improved performance, in fact when the number of labeled examples was very low (one or two examples) the transductive learning algorithms almost invariably outperformed the supervised classifiers.

Furthermore in both cases, certain transductive classifiers consistently surpassed both the supervised classifiers and the remaining transductive classifiers even as the number of labeled examples per class increased. This dominance makes them seem ideal for the specific application. In the case of document classification the spectral graph partitioning algorithm outperformed all others, while for handwritten character recognition, learning with local and global consistency performed considerably better than the rest. Unfortunately, the learning with local and global consistency performed quite poorly in the document classification problem meaning that its success is application dependent. On the other hand the spectral graph transducer performed quite well on the handwritten character recognition problem which possibly indicates its general appropriateness when labeled data is scarce.

Finally in this project a novel method based on convolutional networks was presented which uses a transductive setting to obtain an alternate representation of the data and which can be combined with a number of transductive learners and which hopefully results in improved learners. This method proposes an alternate approach to both feature mapping and sub-sampling so that they may take advantage of the large amounts of unlabeled examples. This brings us to the second question this project was concerned with:

`Does the convolutional network analogous proposed lead to a representation of the data which ultimately leads to improved classification?`

Based on the results obtained the answer would again be yes. At least as far as the experiments conducted here are concerned, both the feature mapping and sub-sampling techniques proposed lead to improved performance in most cases. When combined with a spectral graph transducer or with transductive clustering the resulting classifiers are improved as opposed to the same algorithms executed on the original representation of the data. Only in the case of learning with local and global consistency does the sub-sampling technique result in an inferior classifier. These results make us confident that the method proposed can be applied successfully in a more broad field, either by combining it with other transductive learning algorithms or by applying it to other problems, object recognition problems seeming especially germane. This could perhaps become the focus of a future project.

References

[1] Avrim Blum and Shuchi Chawla. Learning from labeled and unlabeled data using graph mincuts. In Carla E. Brodley and Andrea Pohoreckyj Danyluk, editors, *ICML*, pages 19–26. Morgan Kaufmann, 2001.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, September 2001.

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

[4] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, March 2000.

[5] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience Publication, 2000.

[6] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

[7] Geoffrey E. Hinton and Richard S. Zemel. Autoencoders, minimum description length and helmholtz free energy. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *NIPS*, pages 3–10. Morgan Kaufmann, 1993.

[8] Thorsten Joachims. Transductive learning via spectral graph partitioning. In Tom Fawcett and Nina Mishra, editors, *ICML*, pages 290–297. AAAI Press, 2003.

[9] Y. LeCun, L. Botou, L. Jackel, H. Drucker, C. Cortes, J. Denker, I. Guyon, U. Muller, E. Sackinger, P. Simard, and V. Vapnik. Learning algorithms for classification: A comparison on handwritten digit recognition, 1995.

[10] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[11] A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification, 1998.

[12] Rada Mihalcea. Co-training and self-training for word sense disambiguation. In Hwee Tou Ng and Ellen Riloff, editors, *HLT-NAACL 2004 Workshop: Eighth Conference on Computational Natural Language Learning (CoNLL-2004)*, pages 33–40, Boston, Massachusetts, USA, May 6 - May 7 2004. Association for Computational Linguistics.

[13] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[14] Kamal Nigam, Andrew McCallum, Sebastian Thrun, and Tom M. Mitchell. Text classification from labeled and unlabeled documents using em. *Machine Learning*, 39(2/3):103–134, 2000.

[15] J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines, 1998.

[16] Bertrand Russell. *The problems of philosophy*. Oxford University Press UK, 1912.

[17] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, 2002.

[18] Jonathan R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Pittsburgh, PA, USA, 1994.

[19] P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart, J. L. McClelland, et al., editors, *Parallel Distributed Processing: Volume 1: Foundations*, pages 194–281. MIT Press, Cambridge, 1987.

[20] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory (Information Science and Statistics)*. Springer, November 1999.

[21] D. Zeimpekis and E. Gallopoulos. Tmg : A matlab toolbox for generating term-document matrices from text collections. In Charles Nicholas Jacob Kogan and Marc Teboull, editors, *Grouping Multidimensional Data Recent Advances in Clustering*, pages 187–210. Springer Berlin Heidelberg, 2006.

[22] D. Zhou, O. Bousquet, T. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency, 2003. In 18th Annual Conf. on Neural Information Processing Systems.

[23] Xiaojin Zhu. Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2005.