EXPLOITING EXAMPLE STRUCTURE IN MULTIPLE INSTANCE LEARNING

by

Rama Krishna Sandeep Pokkunuri

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science

School of Computing

The University of Utah

May 2011

The University of Utah Graduate School

STATEMENT OF THESIS APPROVAL

The thesis of **Rama Krishna Sandeep Pokkunuri**

has been approved by the following supervisory committee members:

Hal Daumé III	, Chair	**3/14/2011** Date Approved
Ellen Riloff	, Member	**3/14/2011** Date Approved
Tom Fletcher	, Member	**3/14/2011** Date Approved

and by **Al Davis**, Chair of the Department of **School of Computing**

and by Charles A. Wight, Dean of The Graduate School.

ABSTRACT

Multiple Instance Learning (MIL) is a type of supervised learning with missing data. Here, each example (*a.k.a.* bag) has one or more instances. In the training set, we have only labels at bag level. The task is to label both bags and instances from the test set. In most practical MIL problems, there is a relationship between the instances of a bag. Capturing this relationship may help learn the underlying concept better. We present an algorithm that uses the structure of bags along with the features of instances. The key idea is to allow a structured support vector machine (SVM) to "guess" at the true underlying structure, so long as it is consistent with the bag labels. This idea is formalized and a new cutting plane algorithm is proposed for optimization.

To verify this idea, we implemented our algorithm for a particular kind of structure – hidden markov models. We performed experiments on three datasets and found this algorithm to work better than the existing algorithms in MIL. We present the details of these experiments and the effects of varying different hyperparameters in detail. The key contribution from our work is a very simple loss function with only one hyperparameter that needs to be tuned using a small portion of the training set.

The thesis of this work is that it is possible and desirable to exploit the structural relationship between instances in a bag, even though that structure is not observed at training time (*i.e.,* correct labels for all the instances are unknown). Our work opens a new direction to solving the MIL problem. We suggest a few ideas to further our work in this direction.

To my father

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank my adviser, Prof. Hal Daumé III, who proposed the initial idea and patiently guided me through this project despite being away from the university. I am grateful to Prof. Ellen Riloff and Prof. Tom Fletcher, who gave valuable input at every stage.

Thanks are due to my fellow students Amrish Kapoor and Siddharth Patwardhan for providing their valuable input for my experiments.

It would have been impossible to finish this thesis without my friends and fellow students Avantika Vardhan, Srikanth Kamesh Iyer, Prasanna Venkatesh, Srikanth Chikkulapelly, Shreyas Payal, Himabindu Nunna and Gopalkrishna Veni. Many thanks to them.

CHAPTER 1

INTRODUCTION

Machine Learning is a scientific discipline that is concerned with the design and development of algorithms that allow computers to evolve behaviors based on empirical data, such as from sensor data or databases. There are three primary branches of this discipline – supervised learning, unsupervised learning and semisupervised learning.

Supervised learning infers a function from the given labeled data (training data). For example, given a set of documents, each of which is labeled relevant or irrelevant, a supervised learning algorithm outputs a function that would take an unseen document and label it relevant or irrelevant. This particular task is also called classification.

Unsupervised learning tries to determine how the input data are organized. The primary difference between unsupervised learning and supervised learning is that the former takes unlabeled data as input. For example, consider the problem of mining different topics in a given set of documents. Here, the topics are not known ahead of time. The unsupervised learning algorithm is supposed to mine them.

Semisupervised learning is a combination of the above two scenarios. Here, both labeled and unlabeled data are given. Unlabeled data, when used in conjunction with a small amount of labeled data, can produce considerable improvement in learning accuracy. Since manual labeling of data could be expensive, labeling a small part of the data and applying semisupervised algorithms could be of great practical value.

Structured prediction is a variant of supervised learning. In conventional supervised learning, each label is a scalar *i.e.,* the classifier's output domain is $\mathbb{R}$. On the other hand, in structured prediction, each label is a vector. For example, consider the task of part-of-speech (POS) tagging in which input is a sequence of words and the output is a sequence of POS tags. The domain of output is the set of all possible POS tag vectors.

1.1 Multiple Instance Learning

Multiple Instance learning (MIL) (10) is a variation of supervised learning. In supervised learning, each example along with its label is a single unit of information. In Multiple

Instance learning, there are *bags* of information. Each bag[1] is a composition of one or more instances. Each instance has an unknown label. One or more of the instances cause the (known) label for the bag. The goal of MIL is to classify unseen bags and instances.

MIL is typically done in the context of binary classification, where there are only two mutually exclusive labels possible per instance (and bag) – positive or negative. A bag is said to be positive if there is *at least* one positive instance in it. A bag is said to be negative if *none* of its instances is positive.

MIL finds applications in many areas like drug activity prediction (10), content-based image retrieval (41), text categorization (1) *etc.*

We formalize our notation for the binary MIL problem in Table 1.1.

Let us assume that the instances of each bag X in χ_p and χ_n are arranged in a sequence. Now, each bag has a sequence of labels, with a one-to-one correspondence between instances and labels in it. We assume that the correct output vector (*a.k.a.* label sequence) for each bag X to be Y. Let us denote the set of possible output vectors for X be Υ. Now, the feature function for each bag depends both on the input X and the full output vector Y. Let us denote this by $\mathbf{\Phi}(X, Y)$. Our goal is to build a classifier that takes χ_p, χ_n and $\mathbf{\Phi}$ to output a model that can classify bags and instances.

1.2 Structure in MIL Bags

The key idea in our work is to use the structure of input bags in MIL. Consider the following example. Suppose you are given a set of images, each of which is labeled as to whether it contains a car or not. The task is to come up with a model which, when input a new image, tells whether it contains a car or not. If it says that the image does contain a car, it must also list down the pixels of the image which belong to the car. This problem

[1]A bag is also referred to as an example. Throughout this document, we do not refer to bags as examples.

Table 1.1: Notations

Legend	Description
χ	Set of bags used for training
$\chi_p \subset \chi$	Set of positive bags
$\chi_n \subset \chi$	Set of negative bags
$\tilde{\chi}$	Set of instances from all bags *i.e.,* $\tilde{\chi} = \{x \vert x \in X \in \chi\}$
$\tilde{\chi_p}$	Set of instances from positive bags *i.e.,* $\tilde{\chi_p} = \{x \vert x \in X \in \chi_p\}$
$\tilde{\chi_n}$	Set of instances from negative bags *i.e.,* $\tilde{\chi_n} = \{x \vert x \in X \in \chi_n\}$
$\phi(x)$	Feature vector of any instance x
$\phi(X)$	Sum of all feature vectors of all instances in a bag X *i.e.,* $\phi(X) = \sum_{x \in X} \phi(x)$

is an instance of MIL. Here, each image is a bag and its pixels are the instances. We know labels for the bags but not for the instances.

Let us call any pixel depicting a car a "car pixel." Each image with a car has *at least* one car pixel whereas an image without a car has *no* car pixels at all. Intuitively, car pixels do not appear in isolation *i.e.,* a car pixel is likely to be surrounded by other car pixels. Put in other words, there is a structure among the instances of each pixel. A typical classifier takes features of a pixel (RGB values, location in the image *etc.*). One could also try to use the information from neighboring pixels by including them as additional features, depending on the structure. Our idea is to come up with a general technique to solve the MIL problems in which bags are expected to have a structure that can be modeled.

In this thesis, we formulate a classifier that considers the structural information in the bags along with the instance features. In Chapter 2, we describe the literature from MIL and structured prediction in detail. This is necessary to understand the formulation of our classifier in Chapter 3. In Chapter 4, we describe the datasets used and the results of our experiments with a thorough discussion of our results. We conclude our work and describe our ideas for future work in Chapter 5.

CHAPTER 2

BACKGROUND

Our work is built on concepts from two areas - MIL and structured prediction. In this section, we look at the existing work in both these areas.

2.1 Multiple Instance Learning

MIL was originally proposed under this name by Dietterich *et al.* (10) although earlier examples (18) of such similar research exist. Many classical classification techniques have been adapted to work in the context of MIL. Dietterich *et al.* (10) proposed three learning algorithms, each of which illustrates an approach to constructing *Axis Parallel Rectangles* (APR) in the MIL context. The best of these algorithms starts with a point in the feature space. It "grows" a rectangle in such a way that it covers at least one instance from each positive bag and no instance from any negative bag.

Maron and Lozano-Pérez (23) proposed a *diverse density* (DD) based algorithm which tries to find out the region of high positive bag density and low negative bag density. Zhang and Goldman (40) built an EM algorithm upon this. In the E-step, the current hypothesis h is used to pick one instance from each bag that is most likely to be the one responsible for the label given to the bag. In the M-step, the standard DD algorithm is used to find a new hypothesis h' that maximizes $DD\{h\}$. These steps are repeated until convergence.

Several methods have been proposed to adapt support vector machines (SVM) to MIL settings. In the following sections, we discuss these ideas in detail.

2.1.1 Single Instance Learning

The most obvious way to use an SVM to solve the MIL problem is to apply the bag's label to all its instances and train a regular SVM on the resulting data. This is referred to as Single Instance Learning(*SIL*). This is illustrated in Figure 2.1.

The objective (Eq (2.1)) has two components – the regularizer ($\frac{1}{2}||w||^2$) which tries to keep the feature weights small and the slack component which tries to keep the "adjustments" on the scores of training instances (training error) small. There is a trade-off between these two which is controlled by the cost-factor C. A high value for C means that we want the training error to be low even if the model is complex whereas a low value for C says that

minimize:

$$F(\mathbf{w}, b, \xi) = \frac{1}{2}||\mathbf{w}||^2 + \frac{C}{L}\sum_{x \in \tilde{\chi}} \xi_x \tag{2.1}$$

subject to:

$$\mathbf{w}\phi(x) + b \leq -1 + \xi_x \qquad \forall x \in \tilde{\chi}_n \tag{2.2}$$

$$\mathbf{w}\phi(x) + b \geq +1 - \xi_x \qquad \forall x \in \tilde{\chi}_p \tag{2.3}$$

$$\xi_x \geq 0$$

Figure 2.1: SIL-SVM optimization problem

we require the model to be simple even if a few training instances are misclassified. This trade-off is related to the *bias-variance trade-off* that is well-documented. Equations (2.2) and (2.3) constrain that the instances from negative bags have a score of *at most* -1 (modulo slack) and the instances from positive bags have a score of *at least* $+1$ (modulo slack).

Depending on the nature of the dataset, SIL can give very decent results. Suppose that a dataset has a very high density of positive instances in positive bags, then the number of instances that SIL wrongly assumes to be positive is very low in comparison to the total number of instances. SIL's performance under such a condition will be very close to learning from the actual labels. However, when the positive bags are sparse (*i.e.,* the fraction of instances in a positive bag that are positive), SIL would not work well.

2.1.2 Statistic Kernel and Normalized Set Kernel

Gärtner *et al.* (12) proved some key results about the applicability of kernels to the MIL problem. They introduced the Statistic Kernel (STK) and the Normalized Set Kernel (NSK).

In STK, every bag is transformed into a feature vector, where each instance-level feature contributes two bag-level features: the minimum and the maximum value it takes across all the instances in the bag. Thus, each bag is transformed into two instances having the same label as the bag. Using these derived instances, an SVM is trained.

In NSK, each positive bag is represented by the sum of feature vectors of its instances divided by its ℓ_1 or ℓ_2-norm. The negative bags are used as they are. The resulting feature vectors are used for training a regular SVM. This formulation is shown in Figure 2.2.

These two approaches, just like SIL, tend to work well when most instances in positive bags are actually positive. They are less effective when only a few of the positive-bag instances deserve a positive label.

minimize:

$$F(\mathbf{w}, b, \xi) = \frac{1}{2}||\mathbf{w}||^2 + \frac{C}{|\tilde{\chi}_n|}\sum_{x\in\tilde{\chi}_n}\xi_x + \frac{C}{|\chi_p|}\sum_{x\in\chi_p}\xi_X \tag{2.4}$$

subject to:

$$\mathbf{w}\phi(x) + b \quad \leq -1 + \xi_x \qquad \forall x \in \tilde{\chi}_n \tag{2.5}$$

$$\mathbf{w}\frac{\phi(X)}{|X|} + b \geq +1 - \xi_X \qquad \forall X \in \chi_p \tag{2.6}$$

$$\xi_x \geq 0, \xi_X \geq 0$$

Figure 2.2: NSK optimization problem

2.1.3 MI-SVM and mi-SVM

Andrews *et al.* (1) suggested two iterative SVM approaches for MIL. The maximum pattern margin formulation (*mi-SVM*) is essentially a self-training algorithm. It begins by training a model using the SIL formulation. This model is used to relabel the instances in positive bags. If some positive bag contains no instances labeled as positive, then the instance in that bag that gives the maximum value of the decision function is relabeled as positive. The SVM is then retrained with this new dataset. This process of relabeling and retraining is repeated until no labels are changed.

In the maximum bag margin formulation (*MI-SVM*), a model is trained on the given dataset using NSK. For every positive bag, the learned decision function is used to select the bag instance that gives the maximum value. This instance becomes the new bag representation. The SVM is then retrained with this new dataset, and the process is repeated until no bag representation is changed.

2.1.4 Sparse MIL, Sparse Transductive MIL, Sparse Balanced MIL

Bunescu and Mooney (4) proposed three more SVM-based methods that are particularly effective when the positive bags are sparse. Sparse MIL (*sMIL*) constrains that all the instances of the negative bags be labeled negative and *at least* one instance in each positive bag be labeled positive. This formulation is shown in the Figure 2.3.

In this formulation, the objective (Eq (2.7)) states that we want a simple (low norm) weight vector that achieves low error on negative bags (first sum) and positive bags (second sum), where ξs measure the losses. The negative bag constraint (Eq (2.8)) is identical to the usual SVM constraint for negative data points. It states that the prediction on any instance, x from a negative bag should be less than -1, modulo slack of ξ_x. The positive bag constraint (Eq (2.9)) applies to a bag X of size $|X|$. It requires that the average per-instance

minimize:

$$F(\mathbf{w},\xi) = \frac{1}{2}||\mathbf{w}||^2 + \frac{C}{|\tilde{\chi}_n|}\sum_{x\in\tilde{\chi}_n}\xi_x + \frac{C}{|\chi_p|}\sum_{X\in\chi_p}\xi_X \tag{2.7}$$

subject to:

$$\mathbf{w}^T\phi(x) + b \leq -1 + \xi_x \qquad \forall x \in \tilde{\chi}_n \tag{2.8}$$

$$\mathbf{w}\frac{\phi(X)}{|X|} + b \geq \frac{2-|X|}{|X|} - \xi_X \qquad \forall X \in \chi_p \tag{2.9}$$

$$\xi_x \geq 0, \xi_X \geq 0$$

Figure 2.3: sMIL optimization problem

prediction (the l.h.s. term) be at least $(2-|X|)/|X|$ (modulo slack). This term appears because if exactly one of the instances gets a score of 1, and the rest get scores of -1, then the sum will be $1-(|X|-1) = 2-|X|$.

Sparse transductive MIL (*stMIL*) extends sMIL formulation with an additional constraint that each instance in the positive bag be classified positive or negative with a margin (essentially akin to a transductive constraint). This is shown in Figure 2.4.

The stMIL formulation (see Figure 2.4) is identical to that of sMIL, except for an additional term added to the objective, and an additional set of constraints. The new constraints (Eq (2.12)) are from the idea of semisupervised SVMs, where we say that we do not know the labels for unlabeled points, but we should be confident about them one way or the other. The new term in the objective, $\frac{C}{|\tilde{\chi}_p|}\sum_{x\in\tilde{\chi}_p}\xi_x$, tries to minimize the value of slack terms in the new constraint set.

The third algorithm that they proposed is Sparse balanced MIL (*sbMIL*). It is an iterative

minimize:

$$F(\mathbf{w},\xi) = \frac{1}{2}||\mathbf{w}||^2 + \frac{C}{|\tilde{\chi}_n|}\sum_{x\in\tilde{\chi}_n}\xi_x + \frac{C}{|\chi_p|}\sum_{X\in\chi_p}\xi_X + \frac{C}{|\tilde{\chi}_p|}\sum_{x\in\tilde{\chi}_p}\xi_x \tag{2.10}$$

subject to:

$$\mathbf{w}^T\phi(x) + b \leq -1 + \xi_x \qquad \forall x \in \tilde{\chi}_n \tag{2.11}$$

$$|\mathbf{w}\phi(x) + b| \geq +1 - \xi_x \qquad \forall x \in \tilde{\chi}_p \tag{2.12}$$

$$\mathbf{w}\frac{\phi(X)}{|X|} + b \geq \frac{2-|X|}{|X|} - \xi_X \qquad \forall X \in \chi_p \tag{2.13}$$

$$\xi_x \geq 0, \xi_X \geq 0$$

Figure 2.4: stMIL optimization problem

algorithm based on sMIL. An SVM is trained on a small labeled dataset, and then used to score all instances from the larger unlabeled dataset. The top $\eta\%$ of them as ranked by the decision function are labeled positive and the rest are labeled negative. Then, the algorithm proceeds in iterative fashion: at each iteration, it finds a pair of instances with different labels that were classified on the wrong side of the hyperplane, switches their labels and retrains the SVM. This is illustrated in Figure 2.5.

2.2 Structured Prediction

In structured prediction, both the examples and output are expected to be vectors which have a structure among their components. The goal is to model that structure. It is possible that each component of the vector has a "feature list." Consider the task of POS tagging where the goal is to determine the part of speech of each word in a sentence. Obviously, a sentence in English has to follow certain grammar – which we refer to as *structure*. For instance, a noun is likely to follow a determiner (the/an/a). So our input is a sequence of words and the output is a corresponding sequence of part of speech tags. For each component (words) of the input vector (sentence), there could be features like "Is the word capitalized?," "Is the word in our dictionary?" *etc.*.

Typically structured prediction algorithms concentrate on sequence labeling (*i.e.,* the input and output are sequences like in POS tagging). While some of the following algorithms are applicable to only sequences (HMM, MEMM *etc.*), there are some which are applicable to more general structures like graphs (*e.g.,* CRF). In this section, we describe the most popular methods to model structured data.

2.2.1 Hidden Markov Models

Hidden markov models (HMM) are one of the earliest methods to deal with *sequence labeling*. Rabiner (32) wrote an excellent tutorial about HMMs. Briefly, we describe them in this section.

SBMIL$(\chi_n, \chi_p, \phi, C, \eta)$

$(\mathbf{w}, b) \leftarrow$ solve_sMIL $(\chi_n, \chi_p, \phi, C)$
order instances $x \in \chi_p$ using $f(x)$
label instances in $\tilde{\chi}_p$:
    the top $\eta|\tilde{\chi}_p|$ as positive
    the rest $(1-\eta)|\tilde{\chi}_p|$ as negative
$(\mathbf{w}, b) \leftarrow$ solve_SIL$(\tilde{\chi}_n, \tilde{\chi}_p, \phi, C)$
return $(\mathbf{w}, b)$

Figure 2.5: Sparse balanced MIL

A stochastic process is a nondeterministic process in which there is uncertainty over the future state of the system. This uncertainty is described by a probability distribution. In its simplest form, a stochastic process amounts to a sequence of states known as *time series* *i.e.,* the state of the process at time $t = 1, \ldots, n$ is $s_1, \ldots, s_n$. Suppose we are noting down the weather of a city on a daily basis. For Sunday through Saturday, the observations are <sunny, cloudy, rainy, cloudy, sunny, sunny, cloudy>. This is a stochastic process because it is impossible to deterministically say what the weather on the following day would be, given the observations about the past few days. However, weather does follow a particular pattern. For instance, a cloudy day is more likely to be followed by a rainy day than a sunny day.

A stochastic process is said to have the *markov property*[1] if the conditional probability distribution of the future state of the process (conditional on both past and present values) depends only upon the present state. A process with this property is called a *markov process.* Let x_t be the random variable corresponding to the state of the process at time unit t. Markov property essentially means:

$$p(x_t = i|x_1, x_2, \ldots, x_{t-1}) = p(x_t = i|x_{t-1}) \qquad \forall i = 1 \ldots n$$

In the example mentioned above, if we are guaranteed that the weather on a particular day is influenced only by the previous day's weather, it can be modeled as a markov process.

Let us consider a markov process P which has n states $(s_1 \ldots s_n)$. When we try to model it from observed data, the parameters we are trying to estimate are the *initial* probabilities $(p(x = i|t = 1)$ for $i = 1 \ldots n)$ and the *transition* probabilities
$(p(x_t = i|x_{t-1} = j) \forall i, j \in [1, n])$.

In general, markov processes are of order 1 *i.e.,* the future state of the process depends on *only* the current state. If the future state depends on *both* the current state and the previous state, it is said to be of order 2. Markov processes of higher order are also used for modeling some data.

A hidden markov model is a markov process in which the states are *hidden.* Here, we cannot know the state of the system at any time, but we can make an observation that has a direct relationship with it. To modify our example to this situation, let us suppose that we are not allowed to visit the city directly. Instead, we are shown pictures of some trees in the city at the sunset. We know that the leaves of trees look different based on the

[1]Markov processes are named after the Russian mathematician Andrew Markovi.

weather during the day. Using the pictures and the markov property, one has to estimate the weather on the following day.

Let us denote the observation made at time t by o_t. To simplify the discussion, let us assume that any observation can take only a value between [2]$1 \ldots m$. In addition to the parameters of a regular markov process, we now have *emission* probabilities to estimate *i.e.,* the probability with which the observation i can be made while the system is in state $s_j : p(o_t = i | x_t = s_j)$.

HMMs make the three key assumptions which are as follows:

1. **Markov assumption:** the next state depends only on the current state.

2. **Stationary assumption:** state transition probabilities are independent of the actual time at which the transitions happen.

3. **Output independence assumption:** the current observation is independent of the previous outputs given the current state.

Typically, the observation made at time t is assumed to be dependent only on the state of the system at t. However, one could imagine a process which emits observations based on the previous states of the system. The number of states before the current state that have a say in the observation to be emitted now is called the emission order of the HMM.

Once we know the transition order and emission order of the model, there are efficient algorithms that answer relevant questions about the process. *Viterbi* algorithm takes the initial, transition and emission probabilities and outputs the most likely sequence of states that the process went through. *Forward* algorithm gives the probability of a given observation sequence. *Forward-backward* algorithm, using the same inputs, finds the most likely state of the system at any point of time. Given a few sequences of observations produced by the same process, one can estimate the parameters of an HMM using an expectation maximization algorithm (2).

HMMs have applications in a wide range of machine learning problems like speech recognition (32), handwriting recognition (31), POS tagging (14) and musical score following (28).

2.2.2 Maximum Entropy Markov Models

McCallum *et al.* (24) introduced Maximum Entropy Markov Models (MEMM) to overcome the following problems with the traditional HMMs:

1. In HMM, one has to list down all the observations (for Viterbi and Forward-backward algorithms) to utilize or estimate the emission probabilities. In some applications, it may

[2]The real observations need not be numeric. One could think of these numbers as identifiers to the distinct observations.

not be possible to enumerate all the possible observations. Consider parsing a document containing *frequently asked questions* (24). Each line in the document has to be labeled as one of { head, question, answer}. Obviously, there is a structure in the document. An answer is more likely to follow a question rather than a head. So, we could model this using an HMM. Here, the hidden states are { head, question, answer} and the lines of the document are observations. As one could see, there can be a very large number of possible lines. An HMM would try to build a multinomial distribution over all these lines – which is a very unreasonable goal. Typically, the distribution of lines is very sparse *i.e.,* we rarely see lines repeating within a document. This is again a problem for modeling them using a multinomial distribution.

2. Many applications benefit from a richer representation of observations. Consider the task of POS tagging. It is modeled as an HMM traditionally. Here, POS tags are states and the words are the observations. If one could use more features per observation like "Is the first letter capitalized?," "Does the word end in *tion*," "Does the word exist in our dictionary?" *etc.* As one can notice, these features need not be independent of each other *i.e.,* they are overlapping. But together, they increase the confidence of our prediction. With more features per observation, listing down the set of possible observations (combinations of features) would be hard.

3. In most text applications, the task is to predict the state sequence *given* the observation sequence. HMM algorithms maximize the likelihood of the observation sequence (*i.e.,* $p(s_t, o_t)$) rather than estimate the probability of a state given the observation (*i.e.,* $p(s_t|o_t)$) *i.e.,* they use a generative *joint* model to solve a *conditional* problem.

MEMM replaces transition and observation functions with a single function that gives the probability of the current state given the previous state s' and the current observation o. Formally, $p(s|s', o)$ (also denoted by $p_{s'}(s|o)$) is the probability of current state given the previous state and *current* observation. One could imagine this to be a probabilistic finite-state acceptor that goes from s' to s on an input o. It is important to note that unlike in common HMMs, MEMM *does not* require that the observation be dependent only on the current state.

Suppose that each feature of an observation can be obtained by a function $f_{<b,s>}(o_t, s_t)$ which is defined as follows:

$$f_{<b,s>}(o_t, s_t) = \begin{cases} 1 & \text{if } b(o_t) \text{ is true and } s = s_t \\ 0 & \text{otherwise} \end{cases}$$

$f_{<b,s>}$ is shortly written as f_a where $a =< b, s >$. Now, constraints are added so that the expected value of each feature in the learned distribution will be the same as its average on the training observation sequence $o_1, o_2, \ldots, o_m$. Put formally, this is:

$$\frac{1}{m^{s'}} \sum_{k=1}^{m^{s'}} f_{<b,s'>}(o_t, s_t) = \frac{1}{m^{s'}} \sum_{k=1}^{m^{s'}} \sum_{s \in S} P_{s'}(s|o_{t_k}) f_{<b,s'>}(o_{t_k}, s)$$
$$\forall s' \in \{s_0, s_1, s_2, \ldots, s_{m-1}\}, \text{ all features } b$$

The *maximum entropy distribution* [3] that satisfies these constraints has exponential form as shown below. Here, $Z(o, s')$ is the normalizing factor that makes the distribution sum to one across all the next states s.

$$P_{s'}(s|o) = \frac{1}{Z(o, s')} exp\big(\sum_a \lambda_a f_a(o, s)\big)$$

(24) suggest variants of Forward, Viterbi and Forward-backward algorithms for this formulation. In order to estimate the distribution, they suggest a generalized iterative scaling algorithm.

2.2.3 Conditional Random Fields

MEMMs have several advantages compared to HMMs but they suffer from a weakness called the *label bias problem* due to the fact that the transitions leaving a particular state compete only against each other, rather than against all the other transitions in the model. In other words, there is *conservation of score mass* whereby all the scores that arrive at a state must be distributed among the possible successor states. This results in a bias towards states with fewer outgoing transitions. In the worst case, a transition with only one outgoing transition effectively ignores the observation at that stage.

Lafferty *et al.* (19) introduced conditional random fields (CRF) to solve the label bias problem while retaining all the advantages of MEMMs. In addition, CRFs can be applied to structures other than sequences (unlike HMM, MEMM). In general, it can be applied to label graph structures where nodes represent labels and edges represent the dependencies.

---

[3] Maximum entropy is a framework for estimating probability distributions from data. In contrast to traditional learning methods like Naive Bayes classifiers, it does not assume statistical independence of the variables (*i.e.,* features of observations). Its basic principle is that the best model for the data is the one that is consistent with certain constraints with the training data, but otherwise makes the fewest possible assumptions *i.e.,* with the highest entropy.

Sequence labeling becomes a special instance of graph labeling where the graph is just a path (*a.k.a.* chain). Below, we describe the basic idea behind CRFs.

Let $G = (V, E)$ be a graph such that $Y = (Y_v)_{v \in V}$, so that Y is indexed by the vertices of G. If the random variables Y_v obey the markov property with respect to the graph when conditioned on X *i.e.,* $p(Y_v|X, Y_w, w \neq v) = p(Y_v|X, Y_w, w \sim v)$ where $w \sim v$ represents neighborhood in the graph G, then (X, Y) is called a CRF. Note that a CRF is a random field globally conditioned on the observation X. If the graph is a tree, then the joint distribution over the label sequence Y given X has the form:

$$p_\theta(y|x) \propto \exp\Big(\sum_{e \in E,k} \lambda_k f_k(e, \mathbf{y}|_e, \mathbf{x}) + \sum_{v \in V,k} \mu_k g_k(v, \mathbf{y}|_v, \mathbf{x}) \Big) \tag{2.14}$$

In Eq (2.14), x is a data sequence, y is a label sequence and $y|_s$ is the set of components of y associated with the vertices in subgraph S. f_k, g_k are features of edges and vertices, respectively. They are assumed to be given and fixed. For example, f_k may be true if the edge is between a determiner and noun and the word X_i is not found in the dictionary.

The parameter estimation problem for this formulation would be to determine $\theta = (\lambda_1, \lambda_2, \dots; \mu_1, \mu_2, \dots)$ from the given data $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}$ with empirical distribution $\tilde{p}(x, y)$. They describe two improved iterative scaling algorithms that maximize the log-likelihood objective function in Eq (2.15).

$$\begin{aligned} O(\theta) &= \sum_{i=1}^{N} \log p_\theta(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}) \\ &\propto \sum_{x,y} \tilde{p}(\mathbf{x}, \mathbf{y}) \log p_\theta(\mathbf{y}|\mathbf{x}) \end{aligned} \tag{2.15}$$

The key to avoiding the label bias problem is that the probabilities are not normalized at a transition level. Instead, they are normalized at the graph level (using the proportionality relationship in Eq (2.14)).

2.2.4 Maximum Margin Markov Networks

Kernel-based methods like SVMs which maximize the margin of confidence of the classifier are regularly used under normal classification settings (*i.e.,* without structure) due to their ability to handle high-dimensional feature spaces and theoretical guarantees. Initial methods to adapt these techniques to structured prediction involved representing the neighboring nodes as features. However, there are many limitations to techniques like this. On the other hand, probabilistic graphical models, such as HMMs and CRFs, can represent the

structure well but cannot handle high-dimensional feature spaces. Furthermore, they lack theoretical guarantees. Taskar *et al.* (35) proposed Maximum Margin Markov Networks (M3N) that combine the advantages from kernel methods and graphical models.

The key idea is to select the weight vector $\mathbf{w}$ for which the score of the correct label Y is uniformly most different from the closest runner-up, for each of the inputs X. This is expressed as a set of linear constraints that require a margin between the score obtained by the correct label and the rest. If this margin is a constant, then the formulation implicitly assumes zero-one loss over the bag. Since this is inappropriate for structured prediction where Υ could be large, they propose to *re-scale the margin* according to the loss incurred in the linear constraints. This would increase the penalty for violating the margin constraint involving $\hat{Y} \neq Y$ with the loss $\Delta(Y, \hat{Y})$. The resulting formulation is called $SVM_1^{\Delta m}$ (see Figure 2.6). The feature function, denoted $\mathbf{\Phi}(X, Y)$ now depends both on the input X and the full output sequence Y.

Here, as in most SVM-based models, we are optimizing (Eq (2.16)) for a simple solution (low norm) with low training error (here, we have written $\mathbf{1}^T\xi$ to denote the sum of slacks). In the structured SVM, the constraint says that the *difference* in score between the true output Y and any incorrect output $\hat{Y}$ should exceed the *loss* associated with predicting $\hat{Y}$. (In other words, very bad outputs should be ranked very low.)

2.2.5 SVM-ISO

Tsochantaridis *et al.* (36) have proposed an SVM formulation similar to M3N except that they *rescale the slack* instead of margin. The formulation is described in Figure 2.7. They proposed a cutting plane algorithm to solve the QP optimization problem and it finds a solution that is close to the optimal.

Rescaling the slack is more well-behaved than rescaling the margin. The problem with margin rescaling is that it allows the contribution of instances which are already well-

minimize:

$$F(\mathbf{w}, \xi) = \frac{1}{2}||\mathbf{w}||^2 + C\mathbf{1}^T\xi \tag{2.16}$$

subject to:

$$\begin{aligned} &\mathbf{w}^T\left[\mathbf{\Phi}(X, Y) - \mathbf{\Phi}(X, \hat{Y})\right] \geq \Delta(Y, \hat{Y}) - \xi_X \\ &\quad \forall X \in \chi, \forall \hat{Y} \in \Upsilon \backslash \{Y\} \\ &\xi \geq 0 \end{aligned} \tag{2.17}$$

Figure 2.6: SVM for structured data - $SVM_1^{\Delta m}$

minimize:

$$F(\mathbf{w}, \xi) = \frac{1}{2}||\mathbf{w}||^2 + C\mathbf{1}^T\xi \tag{2.18}$$

subject to:

$$\begin{aligned} &\mathbf{w}^T\left[\mathbf{\Phi}(X, Y) - \mathbf{\Phi}(X, \hat{Y})\right] \geq 1 - \frac{\xi_X}{\Delta(Y, \hat{Y})} \\ &\qquad \forall X \in \chi, \forall \hat{Y} \in \Upsilon \backslash \{Y\} \\ &\xi \geq 0 \end{aligned} \tag{2.19}$$

Figure 2.7: SVM for structured data - $SVM_1^{\Delta s}$

separated from the margin to be high. Suppose that an instance is very distant from the decision plane yet has a high loss value. The optimizer will automatically increase the $\xi_{\mathbf{X}}$ for it to accommodate for the unreasonable margin requirement. Once this "adjusting" happens, there is no incentive to check for better labeling of this instance. On the other hand, the slack rescaling will ignore instances that are separated by a margin of 1, and $\xi_{\mathbf{x}}$ is determined by instances which are close to the margin. However, if the loss function Δ satisfies some conditions (33), then finding the maximally violating constraint is much easier when the margin is rescaled compared to slack. So, rescaling the margin is more popular than rescaling the slack.

2.2.6 Incremental Perceptron

Most structured prediction algorithms deal with the problem $\bar{Y} = \text{argmax}_{Y \in \Upsilon}\, w^T\phi(X, Y)$ where X, Y, Υ denote the bag, a possible output vector and the range of Y, respectively. For many structured predictions, this problem can be solved efficiently[4]. But, in some conditions, exhaustive enumeration of the set of candidates for each input sequence is hard – both in the training and decoding phases. One way to overcome this problem is to re-rank the top N parses from an existing generative parser (6; 7). But this model presumes the existence of a supplementary baseline parser. The performance of algorithms based on re-ranking is dependent on the baseline parser. Many of such parsers are heuristic-based, which means that the performance is dependent on effective search.

As an alternative, Collins and Roark (9) introduced a perceptron-based algorithm for incremental parsing. This discriminative algorithm can be applied to other structured prediction problems as well. The basic idea is to use a beam-search parser from Collins (8)

[4]For example, there are dynamic programming solutions like Viterbi and CYK algorithm for finding out the most possible state sequence for an HMM and best parse for context free grammars, respectively.

with two changes in the search strategy to accommodate the perceptron feature weights. The changes are described below:

1. **Repeated use of hypotheses:** First, the model is updated with the current example (by regular perceptron principle). Then, the example is added to a cache (of N examples). Now, update the weight vector iteratively so long as there is an example in the cache for which it does not produce the gold standard parse (correct output vector) as the best parse[5]. The base algorithm looks at the only constraint on the current example and ignores those implied by the older examples. Using this cache helps to avoid a long time to return to a known constraint.

2. **Early update during training:** In the incremental parsing, if it is found that the parse until the j^{th} component of the input vector (*e.g.,* j^{th} word of a sentence in POS tagging) is incorrect, then exit the parsing process and pass it to the parameter estimation. This leads to less noisy input to the parameter estimation besides improving the efficiency.

While these refinements to the base algorithm are definitely intuitive and helpful (as proved by their experiments), it must be noted that it supports only 0/1 loss. Its incremental beam search is only a heuristic with no guarantees of finding the highest scoring parse.

[5]Of course, we stop after a certain number of iterations to avoid trying to separate inseparable data.

CHAPTER 3

OUR APPROACH

The basis for our approach is that there must be a reason for which a set of instances are grouped. For example, consider the task of *Drug Activity Prediction* (10). The goal is to measure the degree to which the molecule binds to a target "binding site." A binding site is a cavity into which the input molecule binds. Each molecule has many alternative *conformations i.e.,* the alternative shapes that the molecule can adapt by rotating its bonds. Only some of these shapes actually bind to the binding site and produce the observed result. When mapped to MIL settings, a molecule is represented by the set of conformations it can have *i.e.,* each molecule is a bag and each conformation is an instance. As one can see, there is a relationship between different conformations of the same molecule. This relationship, when captured, may improve our understanding of which molecule is more likely to bind. We mathematically formulate this idea in the following section.

3.1 SVM Formulation

As we discussed in Section 2.2, structured prediction takes input examples which have multiple components and tries to model the relationship between them. In MIL, each bag is a union of instances. The difference is that structured prediction expects the components of its example to be ordered *i.e.,* there is a structure in the example. If we are assured that the instances of a bag have a structure in them, then MIL becomes a variant of structured prediction with partial information.

Suppose we have instance labels for an MIL dataset in which the bags have structure (sequence, tree, context free grammar *etc.*). Then, we could directly use the formulation in Figure 2.6 to train a model. Even without the instance labels, we know that the negative bags contain only negative instances *i.e.,* we know the right output vector. So, the constraint (2.17) is still as good for negative bags. The problem is with the positive bags where we do not know the right output vector. It is here that we use the same logic as sMIL (Eq (2.9)). Since we do not know the correct output vector for positive bags, the easiest way to modify this constraint to positive bags is to ensure that the best sequence has at least one positive instance in it.

More formally, on negative bags, where one should always predict the all-negative sequence (which we denote as $\bar{Y}$), the loss is simply the number of positive instances predicted. This is just hamming loss measured against $\bar{Y}$. However, for positive bags, the loss is *zero* if anything *other than* $\bar{Y}$ is predicted (that is, if the hypothesized output has at least one positive instance) and is 1 otherwise. The optimization problem for this formulation is written in Figure 3.1.

The idea here is straightforward, if notationally abstruse. The objective (Eq (3.1)) states that we want a simple weight vector (the norm part) that makes few errors, where C is the trade-off between simplicity of the model and training error rate. As before, ξ is a vector of slacks, and $\mathbf{1}$ is a vector of ones. The constraints are as follows: Eq (3.2) states that for **any** negative bag, we must rank the completely-negative output vector highest (modulo slack). This is equivalent to the usual $SVM^{\Delta m}_1$ constraint when the full known output is equal to $\bar{Y}$. Eq (3.3) states that for **any** positive bag, we must rank the completely-negative output vector *lower* than **some** other sequences (modulo slack). In other words, the model must not think that $\bar{Y}$ is the best possible output. Here, the loss for making an error is 1.

If we know that each positive bag has to have at least k positive instances in it (say due to domain knowledge), the second constraint in the above equation becomes:

$$\mathbf{w}^T\left[\mathbf{\Phi}(X,\hat{Y}) - \mathbf{\Phi}(X,\bar{Y})\right] \geq k - \xi_X$$
$$\forall X \in \chi_p, \exists \hat{Y} \in \Upsilon \backslash \{\bar{Y}\}$$

Basically, this constraint is saying that there has to be an output vector ($\hat{Y}$) that has *at least* a difference of k labels compared to $\bar{Y}$ *i.e.,* it must have at least k positive labels in it.

minimize:

$$F(\mathbf{w},\xi) = \frac{1}{2}||\mathbf{w}||^2 + C\mathbf{1}^T\xi \tag{3.1}$$

subject to:

$$\mathbf{w}^T\left[\mathbf{\Phi}(X,\bar{Y}) - \mathbf{\Phi}(X,\hat{Y})\right] \geq \ell(\bar{Y},\hat{Y}) - \xi_X$$
$$\forall X \in \chi_n, \forall \hat{Y} \in \Upsilon \backslash \{\bar{Y}\} \tag{3.2}$$
$$\mathbf{w}^T\left[\mathbf{\Phi}(X,\hat{Y}) - \mathbf{\Phi}(X,\bar{Y})\right] \geq 1 - \xi_X$$
$$\forall X \in \chi_p, \exists \hat{Y} \in \Upsilon \backslash \{\bar{Y}\} \tag{3.3}$$
$$\xi \geq \mathbf{0}$$

Figure 3.1: SVM for structured MIL

3.2 Cutting Plane Optimization

The optimization problem Eq (3.1) is hard to solve for two reasons. First, the number of constraints is exponential in the number of possible output vectors. However, this is the case for most structured prediction problems, and can be addressed by clever optimization techniques, for instance by using cutting planes (17). The second, more serious difficulty, is the **existential** positive-bag constraint in Eq (3.3). We propose a method to overcome this problem.

We first briefly describe the cutting plane optimization. The idea is relatively straightforward – we initialize the algorithm with *no* "active" constraints, and solve the resulting optimization problem. In the first iteration, the solution will be a zero weight vector. We then *search* for the most violated constraint. We add this most violated constraint to the active constraint set, and re-optimize. We repeat this process until convergence within some tolerance. One can show (17) that this will converge in a polynomial number of iterations, meaning that only a polynomial number of constraints need to be active to obtain a good solution.

In order to cope with the existential constraint, we employ a parameter augmentation approach. For each positive bag X, we add a new variable to the model, Y_X. This can be interpreted as our model's current best "guess" as to what the best output vector for X is. In other words, Y_X is a representative that demonstrates that the positive bag constraint holds. The resulting formulation, which we refer to as Multiple Instance Structure Prediction (MISP), can be seen in Figure 3.2.

The resulting problem is now a mixed optimization problem over continuous variables ($\mathbf{w}$, ξ) and discrete variables (Y_X). For negative bags, we use the regular cutting plane

minimize:

$$F(\mathbf{w}, \xi, Y_x) = \frac{1}{2}||\mathbf{w}||^2 + C\mathbf{1}^T\xi \tag{3.4}$$

subject to:

$$\mathbf{w}^T\left[\mathbf{\Phi}(X, \bar{Y}) - \mathbf{\Phi}(X, \hat{Y})\right] \geq \ell(\bar{Y}, \hat{Y}) - \xi_X$$
$$\forall X \in \chi_n, \forall \hat{Y} \in \Upsilon \backslash \{\bar{Y}\} \tag{3.5}$$

$$\mathbf{w}^T\left[\mathbf{\Phi}(X, Y_X) - \mathbf{\Phi}(X, \bar{Y})\right] \geq k - \xi_X \tag{3.6}$$
$$\forall X \in \chi_p \text{ where}$$
$$Y_X = \arg \max_{\hat{Y} \in \Upsilon \backslash \{\bar{Y}\}} \mathbf{w}^T \mathbf{\Phi}(X, \hat{Y})$$
$$\xi \geq \mathbf{0}$$

Figure 3.2: SVM for structured MIL (updated) – MISP

technique to solve this. In order to apply the technique, we must be able to find maximally violated constraints. This can be solved using loss-augmented decoding with the MAP inference algorithm corresponding to the structure. For positive bags, we use the MAP inference algorithm to determine the most likely output vector (Y_X) for each input X and enforce the inequality in Eq (3.6).

At this stage, we propose more constraints on positive bags to tackle the uncertainty due to missing data. This is inspired from a standard paradigm in structured prediction – to require a margin (p) between the scores obtained by the best (correct) and the second best possible output vectors. Essentially this requires the classifier to be confident about its prediction apart from being accurate. This formulation is described in Figure 3.3.

One could imagine adding constraints similar to these on negative bags as well. In fact, in our experiments, we did that and found no difference in performance (see Section 4.3.4).

Since we are unsure about the original output vector for a positive bag, asking for a margin between the best two output vectors may be slightly aggressive. This may make the classifier adjust on negative bags, for which we know the right output vector. A slightly milder version would be to require the margin against $\bar{Y}$ rather than Y_2. Then, the constraints will appear like in Figure 3.4.

These additional constraints require us to find the second best output sequence for each bag. Typically, the same inference algorithm used to find the best output vector can be modified to deduce the second best output vector. For graphical models such as hidden markov models and trees, the k-best paths problem is well-studied in both pure algorithms community (27; 11; 3) and NLP/speech community (26; 25).

Coming back to the cutting plane algorithm, a point to be noted about all the constraints

if Y_X has at least k positive instances:

$$\mathbf{w}^T\left[\mathbf{\Phi}(X,Y_X) - \mathbf{\Phi}(X,Y_2)\right] \geq p - \xi_X$$

otherwise:

$$\mathbf{w}^T\left[\mathbf{\Phi}(X,Y_X) - \mathbf{\Phi}(X,\bar{Y})\right] \geq k - \xi_X$$

$\forall X \in \chi_p$ where:

$$Y_X = \arg \max_{\hat{Y} \in \Upsilon \setminus \{\bar{Y}\}} \mathbf{w}^T \mathbf{\Phi}(X,\hat{Y})$$

$$Y_2 = \arg \max_{\hat{Y} \in \Upsilon \setminus \{\bar{Y}, Y_2\}} \mathbf{w}^T \mathbf{\Phi}(X,\hat{Y})$$

$\xi \geq \mathbf{0}$

Figure 3.3: MISP with margin constraint between the top two most likely sequences - $MISP^{Y_2}$

if Y_X has at least k positive instances:

$$\mathbf{w}^T\left[\mathbf{\Phi}(X, Y_X) - \mathbf{\Phi}(X, \bar{Y})\right] \geq p - \xi_X$$

otherwise:

$$\mathbf{w}^T\left[\mathbf{\Phi}(X, Y_X) - \mathbf{\Phi}(X, \bar{Y})\right] \geq k - \xi_X$$

$\forall X \in \chi_p$ where:

$$Y_X = \arg \max_{\hat{Y} \in \Upsilon \setminus \{\bar{Y}\}} \mathbf{w}^T \mathbf{\Phi}(X, \hat{Y})$$

$\xi \geq \mathbf{0}$

Figure 3.4: MISP with margin constraint between the most likely sequence and $\bar{Y}$ - $MISP^{\bar{Y}}$

on positive bags is that in each iteration of the cutting plane algorithm Y_X may change. A constraint which is formed with Y_X found in the first iteration may not be the most violating constraint with another Y_X found in the second iteration. This is against the assumptions of the cutting plane algorithm. So, we store only one constraint per bag in our working set at any point. This further raises questions about convergence because, in successive iterations, we may be swinging between two different output vectors for the same bag. In order to tackle this, we decide to stop adding constraints after α iterations.

3.3 Loss Function

Since we use loss-augmented decoding to check whether the constraints in our formulation are being satisfied, most of our algorithm's logic is embedded into a loss function. We present our loss function below. This function takes three hyperparameters – k (expected number of positive instances per positive bag), p (margin on positive bags as explained in Figure 3.3), n (margin on negative bags just like p for positive bags). It takes a bag and one of its possible output vectors as the input. If the bag is positive, we check if $\hat{Y}$ has at least k positive instances. If it does not, then we output the difference between k and the number of instances in $\hat{Y}$. If the bag has the desired number of positive instances, then we enforce the constraints in Figure 3.3 by requiring the margin between the best two output vectors (as per the current model) for the bag. If the bag is negative and $\hat{Y}$ has positive instances, the output is the number of positive instances in it. Otherwise, we require a margin between the best two output vectors for the bag. Here, $[z]_+ = \max\{0, z\}$ is the hinge function.

In Figure 3.5, we have shown the implementation corresponding to the constraints in $MISP^{Y_2}$(Figure 3.3). One could set Y_2 to be $\bar{Y}$ (only for positive bags) in the above

implementation to replace those constraints by those in $MISP^{\bar{Y}}$(Figure 3.4). One could consider this as an additional hyperparameter for MISP.

MISP-LOSS($X, \hat{Y}$)

if this is a positive bag **then**
  $\eta \leftarrow min(\#$ instances in $\hat{Y}, k)$
  if $\eta > \#$ positive instances in $\hat{Y}$ **then**
    return $\eta - \#$ positive instances in $\hat{Y}$
  end if
  $\gamma \leftarrow p$
else
  if $\hat{Y}$ has positive instances **then**
    return $\#$ positive instances in $\hat{\mathbf{y}}$
  end if
  $\gamma \leftarrow n$
end if
$Y_1 \leftarrow \arg\max_{Y \in \Upsilon} \mathbf{w}^T \Phi(X, Y)$
$Y_2 \leftarrow \arg\max_{Y \in \Upsilon \setminus \{Y_1\}} \mathbf{w}^T \Phi(X, Y)$
return $\left[\gamma - \left[\mathbf{w}^T \mathbf{\Phi}(X, Y_1) - \mathbf{w}^T \mathbf{\Phi}(X, Y_2)\right]\right)]_+$

Figure 3.5: MISP loss function

CHAPTER 4

EXPERIMENTS

We conducted several experiments on five datasets. The purpose of these experiments was two-fold – to analyze the effect of varying the hyperparameters that we discussed in Section 3.3 and to compare the performance of MISP with the existing techniques for solving the MIL problem. To analyze the effect of varying the hyperparameters, we used two manually labeled small datasets (see Section 4.3). The best hyperparameters found in these experiments were used for the rest of the experiments. In all our experiments, we considered two tasks – bag labeling and instance labeling.

For the datasets on which we have done n-fold cross-validation (MUC4-Hum, ProMed-Hum, CR), we report the precision (P), recall (R) and F1-score obtained by using the cumulative true positives, true negatives, false positives and false negatives corresponding to the best F1-score in each fold. This is called micro-averaging. We decided to do away with accuracy since it does not show the right picture when the number of positive documents in the fold is significantly different. For the datasets in which we already have the train set and test set separated (MUC4-Ans, ProMed-Ans), we will just report the performance of algorithms on the test set with the parameters obtained after tuning on the tune set.

In our discussion through the Chapter 3, we have assumed a *structure* and a corresponding inference algorithm (given the parameters of the model). To validate our idea and evaluate its performance, we had to choose one particular structure. Due to the ease of obtaining them, we decided to use datasets which can be modeled as HMMs (see Section 2.2.1). Many such text datasets are publicly available.

4.1 Software

We compare the results of MISP against the following state of the art algorithms that we discussed in Section 2.1 – Single Instance Learning (SIL), Statistic Kernel (STK), Normalized Set Kernel (NSK), Sparse MIL (sMIL), Sparse Transductive MIL (stMIL), Sparse Balanced MIL (sbMIL).

In the following sections, we describe our implementation of MISP for HMMs (based on SVM^{hmm}) and UniverSVM – a software that implements all the aforementioned algorithms.

4.1.1 SVM$^{\text{hmm}}$

SVM$^{\text{struct}}$ (36) is an SVM algorithm for predicting multivariate or structured outputs[1]. It performs supervised learning by approximating a mapping $h : X \rightarrow Y$ using labeled training examples $(x_1, y_1), \ldots, (x_n, y_n)$. SVM$^{\text{struct}}$ can predict complex structured objects like trees, sequences and sets unlike regular SVMs which do univariate predictions.

SVM$^{\text{struct}}$ is a general idea that can be easily adapted to complex prediction algorithms. One of the original authors, Thorsten Joachims, has implemented it for many kinds of tasks. For instance, SVM^{cfg} learns weighted context free grammar, SVM^{rank} learns a rule for predicting rankings of instances in a bag and SVM$^{\text{hmm}}$ learns hidden markov models (HMM).

Since our goal was to model text datasets that are likely to exhibit HMM properties, we downloaded SVM$^{\text{hmm}}$ and modified its loss function as described in Section 3.3. For finding the second best likely sequence, we used the algorithm described in Nilsson and Goldberger (27). We modified the cutting plane optimization module in SVM$^{\text{hmm}}$ as described in Section 3.2. More details about these modifications and the input format are described in Appendix B.

SVM$^{\text{hmm}}$ has hamming loss (*a.k.a.* 0/1 loss) implemented by default (`-l 0`). We used it as it is to get numbers for the rows corresponding to SVMH-H in Section 4.4.

4.1.2 UniverSVM

UniverSVM (34) is an SVM implementation written in C/C++. SIL, STK, NSK, sMIL, stMIL, sbMIL have been implemented by Bunescu and Mooney (4) in UniverSVM. It can be obtained by emailing the authors. Details about how to format the input files are given in Appendix B. Basically, UniverSVM expects a set of instances with a common *qid* (query-id) indicating that they all belong to the same bag. It outputs a single label for the bag during the test phase. This brings up an important question – "How do we measure the performance of the above algorithms on instance labeling?" Our approach was as follows: For each of the tune and test sets, we created two files. In the first file, we maintained the qids as the instances are in the original bag. These were used to evaluate the performance on bag labeling. In the second file, we gave each instance a different qid *i.e.,* the fact that two instances are from the same bag was hidden from the classifier. This is reasonable because *none* of the above algorithms use the structure of the document while predicting the label. In other words, the model output by the above algorithms classifies each instance independent of the other instances in its bag.

---

[1]It can be downloaded from `http://www.cs.cornell.edu/People/tj/svm_light/svm_struct.html`

We describe the optimization parameters of UniverSVM in Section 4.1.3, along with those of SVM$^{\text{hmm}}$.

4.1.3 Parameters of the Classifiers

There are several functionalities in both SVM$^{\text{hmm}}$ and UniverSVM. In particular, UniverSVM is a general software that can be used for tasks like large-scale transduction, data-dependent regularization. But for our tasks, we used a very limited number of options from both of them. Here, we describe some of the settings we used for the software.

In SVM$^{\text{hmm}}$, one could set the transition order (between $0, \ldots, 3$) and emission order (0 or 1) (see Section 2.2.1). From our preliminary experiments[2], we concluded that a transition order of 1 along with an emission order of 0 gives the best performance. So, throughout our experiments, we used the same settings. We used the default values for all the remaining parameters corresponding to the SVM-formulation.

In UniverSVM, there are not many options related to the tasks at hand. Nevertheless, both SVM$^{\text{hmm}}$ and UniverSVM have a few options related to optimization in common that are of high importance. We used the default values for all of them except two. We set the tolerance for termination criterion (option `-e` in both of them) to 0.1. The cost-factor (option `-c` in both of them) has a very strong effect on the performance of the classifier. We felt that leaving it to their default values is not right. We decided to tune this parameter based on a small manually labeled dataset (to be cut away from the available train set).

UniverSVM provides scores for the test bags instead of labels. This lets us choose an appropriate threshold (*a.k.a. bias*) using a tune set. However, SVM$^{\text{hmm}}$ provides us the labels directly. In some sense, SVM$^{\text{hmm}}$ has one degree of freedom less than UniverSVM. So, we decided to tune the bias for SVM$^{\text{hmm}}$. There are two components of an SVM$^{\text{hmm}}$ model. One component tries to learn to associate the features of instances with a class while the other tries to model the parameters of HMM (*i.e.,* the initial and transition probabilities). We tried introducing a bias at both these places.

To introduce bias in the features component, we added a feature with unit weight to every instance in the train, tune and test sets. We identified the corresponding feature in the model and modified it. We noticed that the performance of the model only worsens with any change (positive/negative) to that feature. It means that SVM$^{\text{hmm}}$ is learning the right value for the bias. Then, we modified the weights of features corresponding to the transition probabilities. This showed a great improvement in performance. So, in all our

[2]We split the MUC4-Hum (and ProMed-Hum) dataset into train and test sets. We measured the performance of SVM$^{\text{hmm}}$ on it with different combinations of transition, emission orders and chose the best.

experiments, we varied the bias on transition probability scores[3] between 0 and 5 in steps of 0.1[4].

For all the datasets, we used bag-of-words features *i.e.,* unigrams. We stemmed the words and eliminated the stop words. The feature vector is normalized *i.e.,* the weight of a feature is the ratio of the number of times it appeared in the document divided by the total number of words in the document (after stemming and eliminating stop words).

4.2 Datasets

We used 5 datasets in our experiments. Below, we describe each of the datasets. For each dataset, we clearly mention the entities – bag, instance and labels (positive vs negative).

4.2.1 MUC-4 Dataset

The Message Understanding Conferences were initiated by NRAD, the RDT&E division of the Naval Command, Control and Ocean Surveilance Center (formerly NOSC, the Naval Ocean Systems Center) with the support of DARPA, the Defense Advanced Research Projects Agency (13). The purpose of these conferences is to assess and improve research on the automated analysis of military message containing textual information. Organized by Beth Sundheim, the goal of each MUC is a qualitative evaluation of the state of the art in message understanding.

For each MUC, the participating groups are given sample messages and instructions on the type of information that needs to be extracted. These participants develop systems that can process such messages. Shortly before the conference, participants are given a set of test messages. They are expected to run these through their system and the output of each participant's system is evaluated against a manually prepared answer key. Thus, the MUCs are different from regular conferences (13).

We used data from the fourth MUC (called MUC-4) for our experiments. This dataset consists of news reports about terrorist activities in Latin America. It has 1700 texts divided into 5 sets. The DEV (development *a.k.a.* train) set contains 1300 documents. Four test sets (TST1, TST2, TST3, TST4) contain 100 documents each. The information extraction task was to identify relevant entities like: *perpetrator individuals, perpetrator organizations, physical targets, victims and weapons.* MUC-4 data comes with the answer keys for this

[3] SVM^{hmm} stores the probabilities as scores which need not be between 0 and 1.

[4] We modified the probability of a negative sentence following a positive one. It gave good results. Modifying the rest of the probabilities did not help much.

task[5].

For the purpose of our experiments, we considered each document a bag and each sentence an instance. Any document that contained a relevant event was considered relevant (positive). Obviously, any document that did not contain at least one relevant event was considered irrelevant (negative)[6]. Labeling the 1700 documents by this definition is trivial given the answer keys. So, there is a sufficient amount of data for the document labeling task. However, this dataset does not come with sentence annotated data. Patwardhan and Riloff (30) hav e employed two human judges who annotated 100 documents from the MUC-4 test set (TST3), at the sentence level. To verify that these two judges annotate consistently, each of them has been asked to annotate 30 common documents individually. They had a Cohen's κ (5) of 0.77 on these documents, which is satisfactory. This data was used for evaluation on both the bag labeling and sentence labeling tasks.

We performed two kinds of experiments with the MUC-4 dataset – training on data annotated using answer keys and training on data annotated by humans. Below, we describe each of these datasets.

4.2.1.1 MUC4-Hum dataset. The 100 manually annotated TST3 documents are hereby referred to as the MUC4-Hum dataset. Out of these 100 documents, 69 are relevant. There are 281 relevant and 800 irrelevant sentences in the relevant documents while there are 628 sentences in the irrelevant documents.

Our basic idea is that the structure in positive documents is notably different from that in negative documents. The initial and transition probabilities for negative documents are trivial. Since this dataset has annotations at the sentence level, we computed the parameters of the HMM from the positive documents, to verify that there is a notably different structure in them. They are summarized in Tables 4.1 and 4.2.

As one can see, the probability of a positive sentence following a positive sentence is much higher than that of it following a negative sentence. This is indicative of substantial structure. Our hope is that we would be able to model these numbers reasonably well.

With this dataset, we do 10-fold cross-validation. In each round, the number of documents used for train, tune and test is $80, 10$ and 10, respectively.

4.2.1.2 MUC4-Ans dataset. The 1600 MUC-4 documents (DEV, TST1, TST2, TST4) that we used for document level annotations and the 100 TST3 documents that we used for sentence level annotations are hereby referred to as the MUC4-Ans dataset. In

[5]MUC-3/4 data is available for free download at `http://www-nlpir.nist.gov/related_projects/muc/muc_data/muc_data_index.html`

[6]Unlike some other IE datasets, many of these documents do *not* describe a relevant event.

Table 4.1: Initial probabilities estimated from MUC4-Hum

+	-
0.5072	0.4928

Table 4.2: Transition probabilities estimated from MUC4-Hum

From (↓) To (→)	+	-
+	0.5506	0.4594
-	0.1329	0.8671

this, we used the 1600 documents for training, 10 randomly chosen documents from TST3 to tune the parameters of the classifiers and the other 90 documents as the test set.

4.2.2 ProMed Dataset

The second dataset that we use is from Patwardhan and Riloff (29). 4958 documents were collected from ProMed-mail[7], an open-source global electronic reporting system for outbreaks of infectious diseases. Most of the ProMed documents contain email headers, footers, citations, and other snippets of nonnarrative text. So, they wrote a "zoner" program to automatically strip off some of this extraneous information.

Similar to the MUC4-HUM dataset, 120 of these documents have been manually annotated. Again, 30 documents have been annotated by both the annotators, individually. They had a Cohen's *kappa* of 0.72 on these documents, which is slightly lower than MUC-4 but still reasonable. Out of the 120 annotated documents, 99 contain relevant events. Going by this estimate, the majority of the documents from ProMed data are relevant.

For irrelevant documents, they have collected 10191 biomedical abstracts from PubMed8, a free archive of biomedical literature. To ensure that the PubMed articles are truly irrelevant (*i.e.,* did not contain any disease outbreak reports) they used specific queries to exclude disease outbreak abstracts.

Again, similar to the MUC-4 dataset, we considered each document a bag, each sentence an instance and any sentence that contained a relevant event a positive sentence. We have performed two kinds of experiments with the ProMed dataset as well – training on approximated data and training on manually annotated data. Below, we describe both the datasets.

4.2.2.1 ProMed-Hum dataset. The 120 manually annotated documents are hereby referred to as the ProMed-Hum dataset. Out of the 99 relevant documents, there are 708 relevant and 1793 irrelevant sentences, whereas the irrelevant documents contained 568

[7]http://www.promedmail.org/

sentences. Just like in Section 4.2.1.1, we computed the parameters of the HMM from the positive documents in this dataset. They are summarized in Tables 4.3 and 4.4.

The initial probabilities in the ProMed-Hum dataset are more skewed than those in the MUC4-Hum dataset. The transition probabilities are not much different. The more skewed the initial and transition probabilities are, the better it is for modeling the data (because of the contrast with the negative documents). Therefore, we expect the results of the ProMed-Hum dataset to be better than those on the MUC4-Hum[8].

We have done 10-fold cross-validation on this dataset with the train, tune, test set sizes being $80\%, 10\%$ and 10%, respectively, just like with the MUC4-Hum dataset.

4.2.2.2 ProMed-Ans dataset. As mentioned in Section 4.2.2, there are a total of 4958 ProMed documents which are labeled positive and 10191 PubMed documents which are labeled negative *i.e.,* a total of 15149 documents. All of these could be used for training. The manually annotated 120 documents were used for tuning (10) and testing (110). There are two important points to be noted here.

1. On average, a PubMed document is *half* the size of a ProMed document. In order to ensure that a classifier sees the the same amount of text from both the positive and negative documents, we used *nearly* twice the number of PubMed documents as there are ProMed documents. Nevertheless, since all the documents in the test set are from ProMed and the classifier has never seen words from ProMed documents being associated with negative documents, it may be misled (by the bag-of-words component of the model) against classifying any document (from the test set) as negative.

2. A classifier may depend heavily on the stark difference between the sizes of ProMed and PubMed documents in its model. Again, this difference does not exist in the test set. Therefore, there will be a further tendency to mark everything positive.

[8]Of course, assuming that the bag of words models of both the datasets are equally informative.

Table 4.3: Initial probabilities estimated from ProMed-Hum

+	-
0.6970	0.3030

Table 4.4: Transition probabilities estimated from ProMed-Hum

From (↓) To (→)	+	-
+	0.5759	0.4241
-	0.1373	0.8627

Due to the above reasons, we expect that any classifier tends to mark documents (from the test set) positive aggressively *i.e.,* there will be more *recall* and less *precision* with this dataset.

Since 15149 documents is too large a dataset for training, we chose 1500 documents from this, at random[9]. Out of these, 985 are from PubMed (negative) and 515 are from ProMed (positive). This would be used for training. Note that there are roughly twice the number of PubMed documents as there are ProMed documents.

4.2.3 Customer Review Dataset

Minging and Bing (15) have manually annotated customer reviews for 5 different products from `http://www.amazon.com` and `http://www.cnet.com`. There are 45 reviews for Canon G3 digital camera, 34 for Nikon Coolpix 4300 digital camera, 41 for Nokia 6610 cellular phone[10], 95 for Creative Labs Nomad Jukebox Zen Xtra 40GB mp3 player and 99 for Apex AD2600 Progressive-scan DVD player – a total of 314 reviews. Each review has the following annotations:

A title: It is the headline given by the customer for his review.

Opinion strengths: A sentence marked with $[+n]$ is a positive opinion with strength n. Similarly, a sentence marked with $[-n]$ is a negative opinion with strength n. A sentence without any opinion strength is considered neutral.

Finer annotations: Other opinions like a suggestion about the product, comparisons with a product from the same brand or different brand are given.

For our experiments, we considered any sentence with a positive comment about a product to be positive[11]. Any negative or neutral comments are considered negative. There are 244 positive documents in the corpus. We refer to this dataset as the CR dataset.

There are 1093 positive and 2383 negative sentences in the positive documents while there are 468 sentences in the negative documents. We computed the parameters of HMM from the positive documents in the CR dataset. They are summarized in Tables 4.5 and 4.6.

Obviously, these numbers are surprising. That a positive review begins with a neutral or negative sentence is unexpected. The reason for this is that a typical review begins with

---

[9]Using the command `$sort -R file.txt | head -1500`

[10]On the 118^{th} line of the file `Nokia 6610.txt`, there were three asterisks before the title marker `[t]`. We removed them.

[11]There are about 28 sentences which have both a positive and negative comment *e.g.,* `this is a very nice phone , but there is no warranty on it.`

Table 4.5: Initial probabilities estimated from CR

+	-
0.4016	0.5984

Table 4.6: Transition probabilities estimated from CR

From (↓) To (→)	+	-
+	0.4727	0.5273
-	0.2333	0.7667

how the customer ended up buying the product. For instance, a review could begin with a sentence like "`after much research i decided on the nikon coolpix 4300`".

In the MUC4-Hum dataset and the ProMed-Hum dataset, we have noticed higher probability of a positive sentence following a positive sentence, and the probability of a positive sentence following a negative sentence was much lower. We note that the average length of a document in the CR dataset is much smaller (12.56 sentences per document) compared to the MUC4-Hum dataset (17.09) or ProMed-Hum dataset (25.58) *i.e.,* customer reviews tend to be shorter than news reports. Customers tend to review different features of the same product in consecutive lines due to space constraint. While one feature of a product is liked immensely, another could be considered suboptimal. As we have seen before, sometimes customers write both positive and negative comments about the product in the same sentence. This may have resulted in the surprising transition probabilities.

We have done 10-fold cross-validation on the CR dataset with train, tune and test set sizes of $80\%, 10\%$ and 10%, respectively. Note that $(314 \mod 10) = 4$ documents will be left out of cross-validation. But it is a very small set compared to 310 to influence the numbers significantly.

4.3 Effect of Varying Hyperparameters

In Section 3.3, we said that the following are the hyperparameters of the MISP loss function.

1. Number of iterations in the cutting plane algorithm (*alpha*)
2. Penalty for misclassifying a positive bag as negative (k and f)
3. Margin constraint set for positive bags ($MISP^{\bar{Y}}$ vs $MISP^{Y_2}$)
4. Margins for positive bags (p) and negative bags (n)

In this section, we describe the effect of all these factors on the performance of MISP on the MUC4-Hum and ProMed-Hum datasets. There are two reasons for choosing these two datasets for these experiments – they are small, making the experiments faster and they are manually annotated, having less chances of erroneous labels. The best performing

hyperparameters in the experiments described in this section were used for the rest of the experiments.

In general, we feel that the results on instance labeling are more important than those on bag labelings. This is because the current results on bag labeling are satisfactory whereas there is plenty of scope for improvement on instance labeling. So, our choice for certain hyperparameters may be slightly biased towards the performance on instance labeling.

4.3.1 Number of Iterations in Cutting Plane Optimization (α)

In Section 3.2, we described the parameter α which controls the number of iterations for which the cutting plane optimization module runs. Since we do not have guarantees on convergence, it is important to see the effect of varying this parameter on the performance of our classifier. We tried five different values for α – $\{10, 20, 30, 40, 50\}$. We kept the other parameters constant throughout these experiments. They are – $k = 1, p = 0, n = 0$. Note that the constraint on positive bags does not matter since the margin is 0. The results of these experiments are summarized in Tables 4.7 and 4.8.

With the MUC4-Hum dataset, the best F1-scores obtained on bag labeling and instance labeling are for $\alpha = 40$ and $\alpha = 50$, respectively. With increasing *alpha*, F1-score more or less increases for instance labeling whereas there is no clear trend for bag labeling. This reiterates the fact that our cutting plane algorithm does not necessarily give better results as we increase *alpha*. Looking at the results of both the tasks, $\alpha = 40$ balanced them well.

Surprisingly, With the ProMed-Hum dataset, for $\alpha = 20$, we get the best F1-score for

Table 4.7: α vs performance on MUC4-Hum

	Bag labeling			Instance labeling		
α	P	R	F	P	R	F
10	68.04	95.65	79.52	19.84	34.52	25.19
20	70.45	89.86	78.98	28.82	40.93	33.82
30	72.09	89.86	80.00	25.62	47.69	33.33
40	73.26	91.30	81.29	25.09	50.53	33.53
50	70.13	78.26	73.97	26.51	51.60	35.02

Table 4.8: α vs performance on ProMed-Hum

	Bag labeling			Instance labeling		
α	P	R	F	P	R	F
10	83.19	94.95	88.68	37.57	56.78	45.22
20	86.11	93.94	89.86	40.12	64.27	49.40
30	85.85	91.92	88.78	38.80	75.14	51.18
40	85.71	90.91	88.24	39.55	74.58	51.69
50	85.71	90.91	88.24	38.74	72.18	50.42

bag labeling and it becomes worse later on. The best F1-score for instance labeling happens at $\alpha = 40$ and decreases slightly later on. Clearly, any value of $\alpha \geq 30$ does about the same job on both the tasks.

It is very important to note that the F1-score of instance labeling on the ProMed-Hum dataset were better than training a classifier with the instance labels (50.00). This asserts our proposition that ProMed has stronger structure in it.

From the above experiments, we feel that 40 is a reasonable value for the hyperparameter for these datasets. However, there is a clear necessity to come up with a better cutting plane algorithm which is guaranteed to converge, however slow it is. In practical scenarios, we may need to try out different values of α, like with most iterative algorithms without guarantees on convergence.

4.3.2 Penalty for Misclassifying a Positive Bag

By the definition of MIL, one is assured to have at least one positive label per bag. However, there could be more. If we had the instance labels, the ideal penalty for misclassifying a positive bag is hamming loss. But we do not have the instance labels. Suppose we know the expected number of positive instances per bag. This would be the ideal penalty for misclassifying a positive bag.

Let us define the *density* of a positive bag as the ratio of the number of positive instances in it. If we could somehow estimate this – either from domain knowledge or by manually labeling a small sample of the training set, then we could have a tighter loss function for positive bags.

One more important reason to study this parameter is because we have plenty of correctly labeled negative data but not positive data. That is, when it comes to misclassification, the larger the size of the bag, the more the opportunity to learn about positive instances that we are missing. So, this needs to be penalized much heavier than misclassifying a negative bag.

We have tried two ways of varying the penalty for positive bags. One is to expect a fixed fraction (f) of instances in the bag to be positive, and another is to expect an absolute number (k) of instances in the bag to be positive. For these experiments, we set $p = n = 0$ and $\alpha = 40$.

4.3.2.1 Effect of varying f. On the MUC4-Hum dataset (Table 4.9), clearly, there is no trend for both bag labeling and instance labeling. However, the best result is obtained for $f = 1.0$ on bag labeling. The same is the case with the ProMed-Hum dataset (Table 4.10). $f = 1.0$ is nothing but SVMH-H (*i.e.,* SVM^{hmm} with hamming loss).

Table 4.9: f vs performance on MUC4-Hum

	Bag labeling			Instance labeling		
f	P	R	F	P	R	F
0.1	70.59	86.96	77.92	26.85	53.02	35.65
0.2	71.76	88.41	79.22	27.67	63.70	38.58
0.3	72.37	79.71	75.86	26.35	59.07	36.44
0.4	73.33	79.71	76.39	23.83	52.67	32.82
0.5	69.88	84.06	76.32	24.39	60.50	34.76
0.6	70.45	89.86	78.98	20.29	69.40	31.40
0.7	69.41	85.51	76.62	22.17	71.17	33.81
0.8	70.11	88.41	78.21	22.53	74.73	34.62
0.9	69.89	94.20	80.25	22.01	73.31	33.85
1.0	69.89	94.20	80.25	22.07	79.72	34.57

Table 4.10: f vs performance on ProMed-Hum

	Bag labeling			Instance labeling		
f	P	R	F	P	R	F
0.1	83.49	91.92	87.50	37.13	74.15	49.48
0.2	86.67	91.92	89.22	32.83	82.91	47.04
0.3	86.36	95.96	90.91	34.92	77.54	48.16
0.4	85.32	93.94	89.42	29.35	90.11	44.27
0.5	85.32	93.94	89.42	29.69	90.54	44.72
0.6	85.59	95.96	90.48	27.36	95.48	42.53
0.7	83.33	95.96	89.20	26.64	95.90	41.69
0.8	83.93	94.95	89.10	26.52	91.10	41.08
0.9	83.76	98.99	90.74	27.55	87.29	41.88
1.0	83.19	100.0	90.83	25.52	90.40	39.80

Therefore, it appears that, for bag labeling, a heavy penalty for positive bag misclassification helps.

For the ProMed-Hum dataset, the performance on instance labeling keeps going down with increasing f. The average number of sentences in a ProMed document is 25 with the mean and mode of the positive instance density being 0.3380 and 0.5, respectively. With increasing f, we are asking the SVM to mark more and more sentences positive. Thus, we see an increase in recall but the precision goes down more rapidly (compare $f \leq 0.3$ vs $f > 0.3$).

The average number of sentences in the MUC4-Hum dataset is 17 (much smaller than ProMed-Hum). However, the mean and mode of positive instance density are 0.3628 and 1.0, respectively. This means that many (possibly small) documents contain *only* positive sentences. Thus, for low values of f, we are losing out on some of these sentences. For high values of f, we may be asking for too many sentences in positive documents to be labeled

positive. Just like the ProMed-Hum dataset, as f increases, we have better recall. But, the precision is not so badly effected.

4.3.2.2 Effect of varying k. We report the results of these experiments in the tables presented in Appendix A under the columns corresponding to $p = 0, n = 0$. For ease of analysis, we provide graphs here.

From the tables in Section 4.3.2.1 and Appendix A it is obvious that varying k gives much better results than varying f. We perform better on instance labeling with the MUC4-Hum dataset (43.51 vs 38.58 from Table 4.9) and on both bag labeling (92.38 vs 90.83 from Table 4.10) and instance labeling (51.69 vs 49.48 from Table 4.10). This trend is a little different than what one would think. The standard deviation of positive instance density in the MUC4-Hum dataset is 0.2935 which is very high given that the mean is just 0.3628. The corresponding numbers for the ProMed-Hum dataset are 0.2034 and 0.3380, respectively. This tells us that the ratio of positive instances per document varies highly even between the documents of the same dataset. Thus, using the same fraction for every document may set wrong expectations for the SVM.

Using an absolute expectation per positive bag does not seem wise at the first look. But as one can see, it performs better. One general observation about the datasets is that the smaller relevant documents tend to have higher density. As described in Section 3.3, the penalty for misclassifying a positive bag is: $max(k, length)$. For small bags (*i.e.,* number of instances $\leq k$), this would evaluate to the bag length and for large bags it would be an absolute number. In some sense, using k adapts to the situation based on the length of the bag. This could be the reason for its better performance.

In Figures 4.1 and 4.2, we see the precision, recall and F1-score trends for bag labeling on the MUC4-Hum and ProMed-Hum datasets, respectively. Increasing the penalty does not show a clear trend when k is small (≤ 7). After that, it steadily increases the recall. Essentially, an increase in penalty is causing the model to learn a more "general" positive bag structure that is fine with a higher number of misclassifications on negative bags. However, this backfires after a certain value of k on each of the datasets due to overfitting. The precision values are unsteady throughout.

In Figures 4.3 and 4.4, we see the precision, recall and F1-score graphs for instance labeling on the MUC4-Hum and ProMed-Hum datasets, respectively. For instance labeling, increasing the penalty increases the recall and reduces the precision in general. This can be seen very markedly on the ProMed-Hum dataset (Figure 4.4. It is important to note that the optimal values of k for bag labeling and instance labeling for a particular dataset do not have any relation. For bag labeling, higher values of k seem to be working better whereas lower values of k do well for instance labeling.

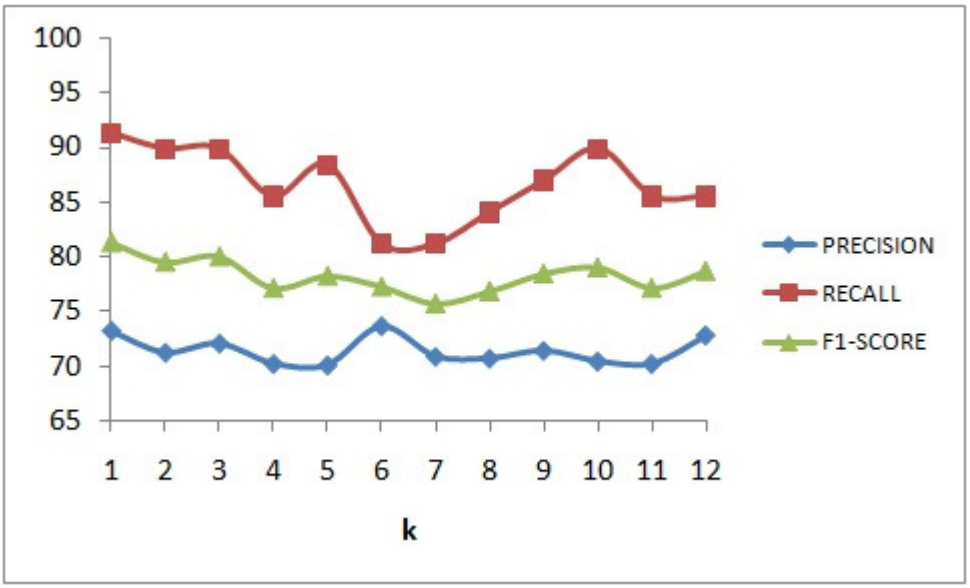

Figure 4.1: k vs bag labeling on MUC4-Hum

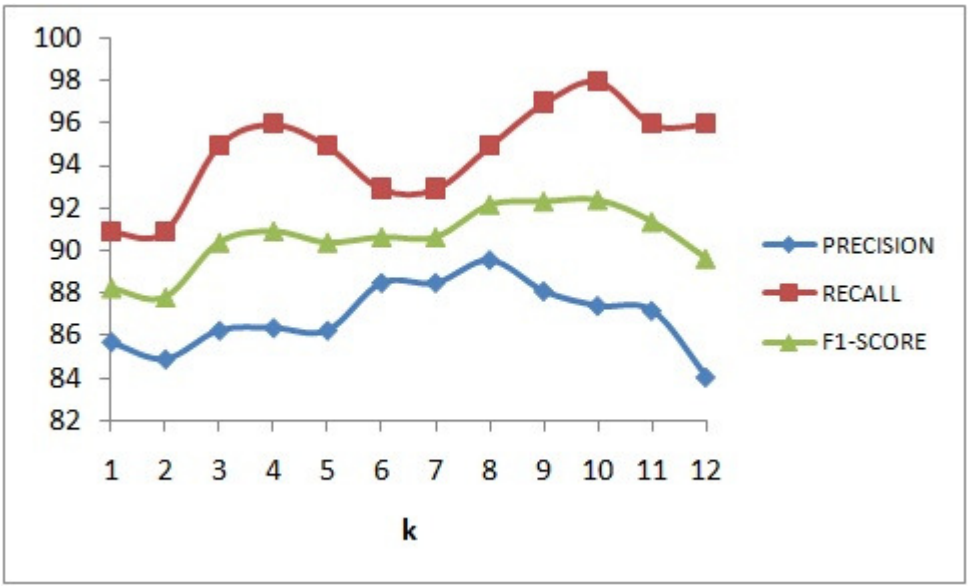

Figure 4.2: k vs bag labeling on ProMed-Hum

4.3.3 Margin Constraint Set for Positive Bags ($MISP^{\bar{Y}}$ vs $MISP^{Y_2}$)

In Section 3.1, we referred to two possible sets of constraints for positive bags. $MISP^{Y_2}$ employs the set of constraints which require that the best possible labeling for a bag have a margin of at least 1 from the second best labeling. $MISP^{\bar{Y}}$ describes the set of constraints which require that the best possible labeling for a bag have a margin of at least 1 from the all-negative sequence. For the experiments described in this section, we set $\alpha = 40$ and $k = 1$. We do experiments for three different values of $< p, n > - < 0, 1 >, < 1, 0 >,$

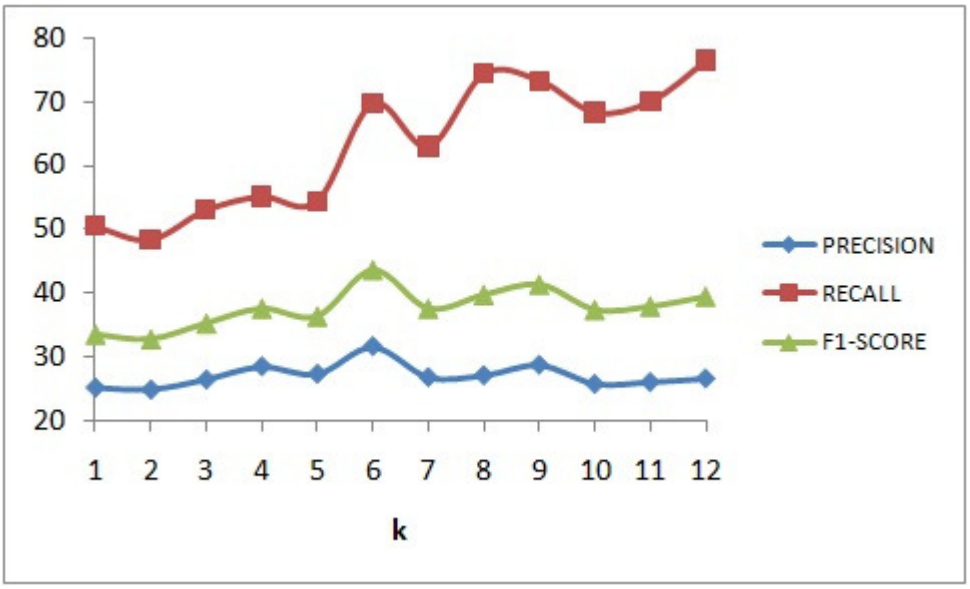

Figure 4.3: k vs instance labeling on MUC4-Hum

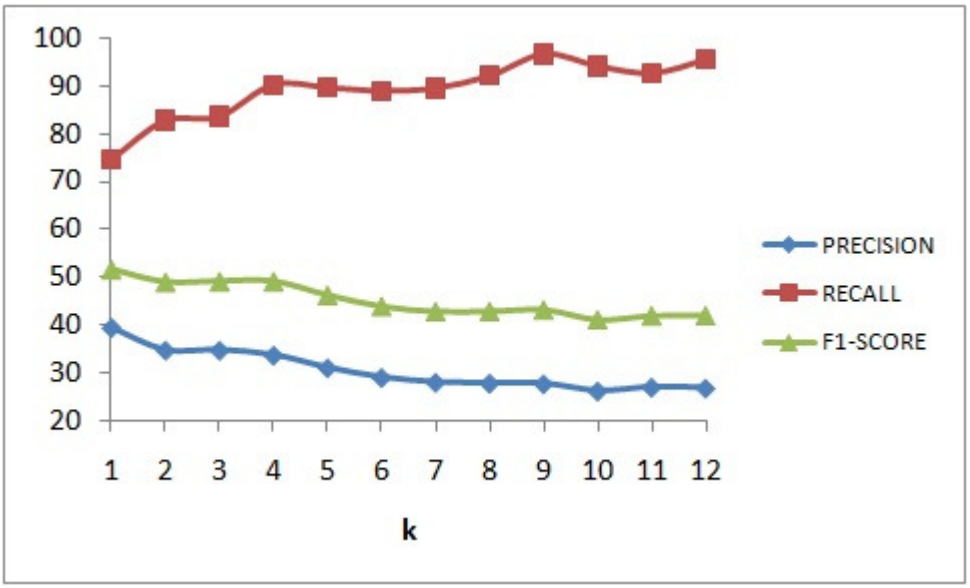

Figure 4.4: k vs instance labeling on ProMed-Hum

$< 1, 1 >$.

Tables 4.11 and 4.12 summarize the results of these experiments. From these tables, we see that $MISP^{Y_2}$works better than $MISP^{\bar{Y}}$for all values of p, n. As we mentioned in Section 3.1, the constraints in Figure 3.3 subsume those in Figure 3.4. Hence, this behavior is not very surprising. However, if the classifier tries hard to achieve the margin and tries to overfit to the data, it may not perform well on the test set. This is what seems to have happened with instance labeling, although the results are pretty close with either choice.

As we mentioned in Section 4.1.3, there are two components to the SVM$^{\text{hmm}}$ model –

Table 4.11: Positive bags constraint vs performance on MUC4-Hum

		Bag labeling					
		$MISP^{\bar{Y}}$			$MISP^{Y_2}$		
p	n	P	R	F	P	R	F
0	1	73.26	91.30	81.29	73.26	91.30	81.29
1	0	70.24	85.51	77.12	70.79	91.30	79.75
1	1	70.24	85.51	77.12	70.79	91.30	79.75
		Instance labeling					
		$MISP^{\bar{Y}}$			$MISP^{Y_2}$		
p	n	P	R	F	P	R	F
0	1	25.09	50.53	33.53	25.09	50.53	33.53
1	0	28.63	49.82	36.36	25.47	53.38	34.48
1	1	28.63	49.82	36.36	25.47	53.38	34.48

Table 4.12: Positive bags constraint vs performance on ProMed-Hum dataset

		Bag labeling					
		$MISP^{\bar{Y}}$			$MISP^{Y_2}$		
p	n	P	R	F	P	R	F
0	1	85.71	90.91	88.24	85.71	90.91	88.24
1	0	84.91	90.91	87.80	85.71	90.91	88.24
1	1	84.91	90.91	87.80	85.71	90.91	88.24
		Instance labeling					
		$MISP^{\bar{Y}}$			$MISP^{Y_2}$		
p	n	P	R	F	P	R	F
0	1	39.55	74.58	51.69	39.55	74.58	51.69
1	0	37.49	76.84	50.39	37.95	72.32	49.78
1	1	37.49	76.84	50.39	37.95	72.32	49.78

the structure model (initial probabilities and transition probabilities) and the bag-of-words model for individual sentences (emit probabilities). Depending on the data, the classifier may have to compromise between these two. If the structure model has more emphasis than the bag of words model, then the classifier will rely on the bag structure to predict the bag label. but this works negatively on instance labeling.

4.3.4 Margins for Positive Bags (p) and Negative Bags (n)

To evaluate the performance of adding margins to positive and negative bags, we used the three manually annotated datasets (MUC4-Hum, ProMed-Hum and CR) as the results on the MUC4-Hum and ProMed-Hum datasets were not strong enough to ascertain certain observations. We used $MISP^{Y_2}$constraints with $\alpha = 40$ for these experiments. k is varied in $\{1, 2, \ldots, 12\}$. There are four combinations of p, n possible since each of p, n can be $0, 1$. Thus, we performed $12 * 2 * 2 * 3 = 144$ experiments. Since it is superfluous to report the

results of bag labeling and instance labeling for all these experiments, we tabulate and plot them in Appendix A. In the following discussion, we compare precision, recall and F1 scores of one combination of p, n values against another. In each comparison, only one of p, n is varied. In most of our comparisons, we see that neither of the contestants wins every time. At times, one betters the other marginally. In order to do a fair comparison, we consider only a difference of *at least* 2% against the base *significant*.

4.3.4.1 Margin on positive bags. Figures 4.5, 4.6, 4.7 show the bag labeling performance of $p = 1, n = 0$ (with margin on positive bags) against $p = 0, n = 0$ (without margin on positive bags) *i.e.,* the effect of requiring a margin on positive bags in the absence of a margin requirement on negative bags. On the MUC4-Hum (Figure 4.5), there is no clear trend of how any one of precision, recall and F1-score vary with increasing k. On both ProMed-Hum (Figure 4.6) and CR (Figure 4.7), we see that recall drops drastically for $p = 1$ after $k = 8$. So, when the penalty for misclassifying a positive bag is already high, margin requirement is hurting the performance. It is very likely that this is due to overfitting of the positive bag structure.

Figures 4.8, 4.9, 4.10 show the instance labeling performance of $p = 1, n = 0$ against $p = 0, n = 0$. Clearly, margin requirement on positive bags does not result in much difference in any of them. In general, for instance labeling, the tuned bias values (for transition probabilities) are much higher than in the case of bag labeling. This could be in order to reduce the emphasis on the learned structure and put the bag of words model to better use while classifying instances.

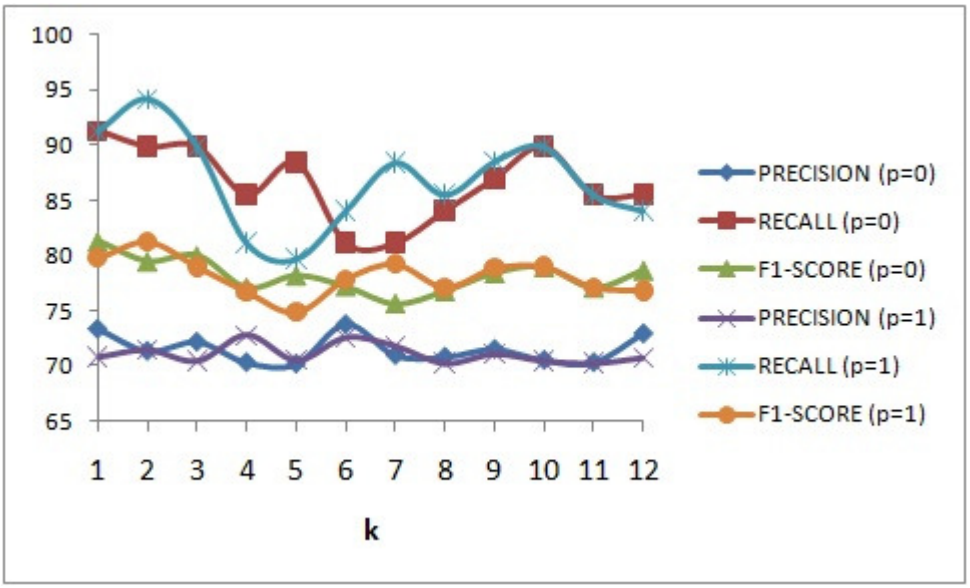

Figure 4.5: p vs bag labeling on MUC4-Hum

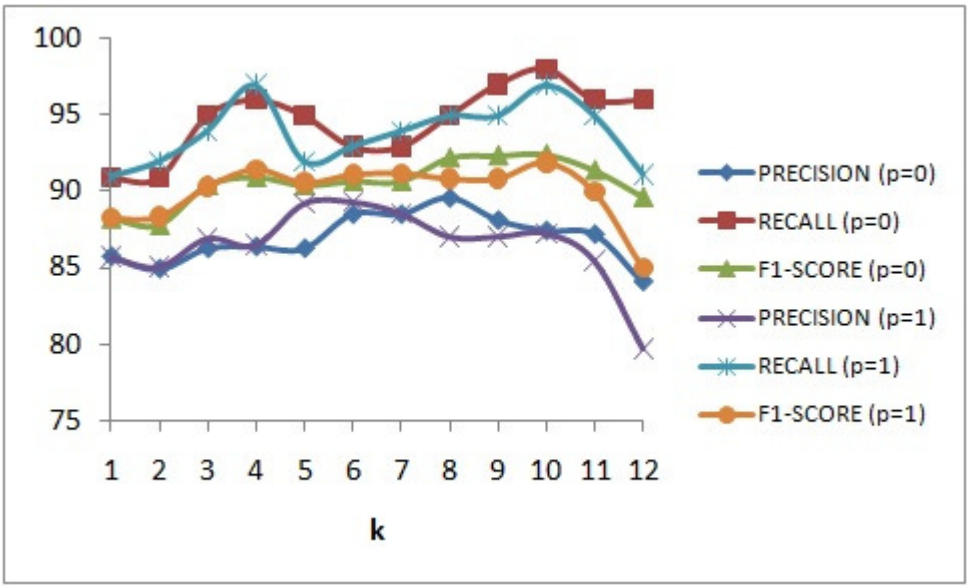

Figure 4.6: p vs bag labeling on ProMed-Hum

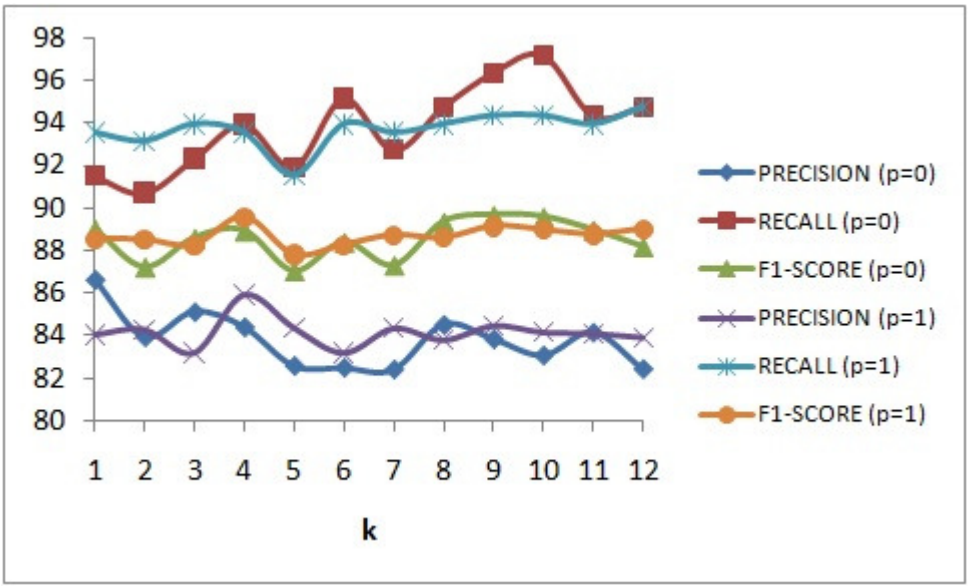

Figure 4.7: p vs bag labeling on CR

4.3.4.2 Margin on negative bags. Figures 4.11, 4.12, 4.13 show the bag labeling performance of $p = 0, n = 1$ (with margin on negative bags) against $p = 0, n = 0$ (without margin on negative bags) for the three manually annotated datasets. The effect on bag labeling for the MUC4-Hum dataset (see 4.11) is not very clear. Although from $k = 6 \ldots 9$, margin requirement helps in increasing recall, its behavior overall is very erratic with a huge decrease in performance at $k = 3, 5$. On the ProMed-Hum (Figure 4.12) and CR (Figure 4.13) datasets, recall drops after $k = 8$ (just as with margin requirement on positive bags).

Again, for instance labeling (see Figures 4.14, 4.15, 4.16), margin requirement does not

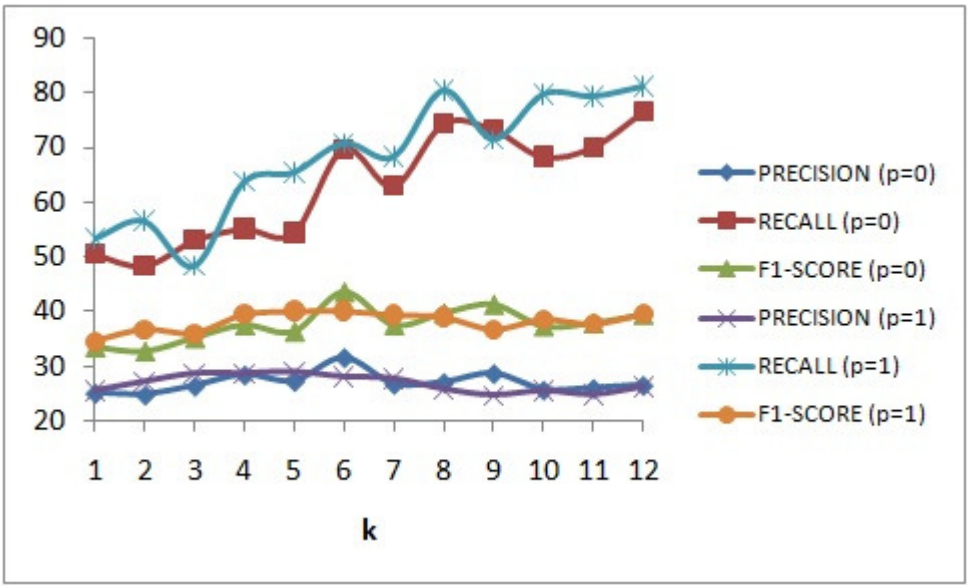

Figure 4.8: p vs instance labeling on MUC4-Hum

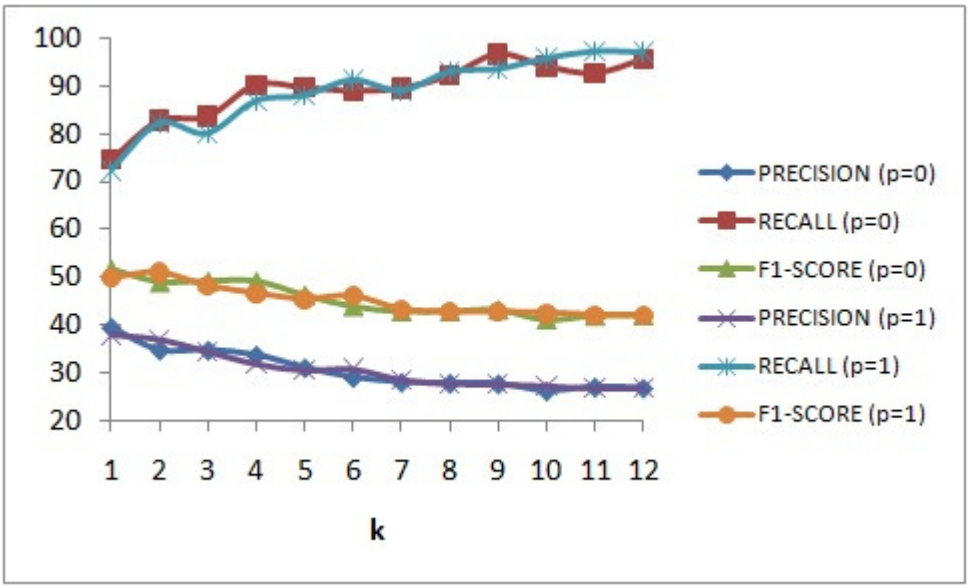

Figure 4.9: p vs instance labeling on ProMed-Hum

result in much of a difference, just like in the case of $p = 1$.

4.3.4.3 Margins on both types of bags. We observed that the precision, recall and F1-score values corresponding to those with $p = 1, n = 0$ and $p = 1, n = 1$ are identical almost all the time (see Appendix A). Out of the 6 tables corresponding to the manually annotated datasets, with 12 rows in each, these columns differ only 4 times. Only 2 times, this difference is significant. From this observation, we can safely conclude that requiring a margin on negative bags does not help much when there is a margin requirement on positive bags.

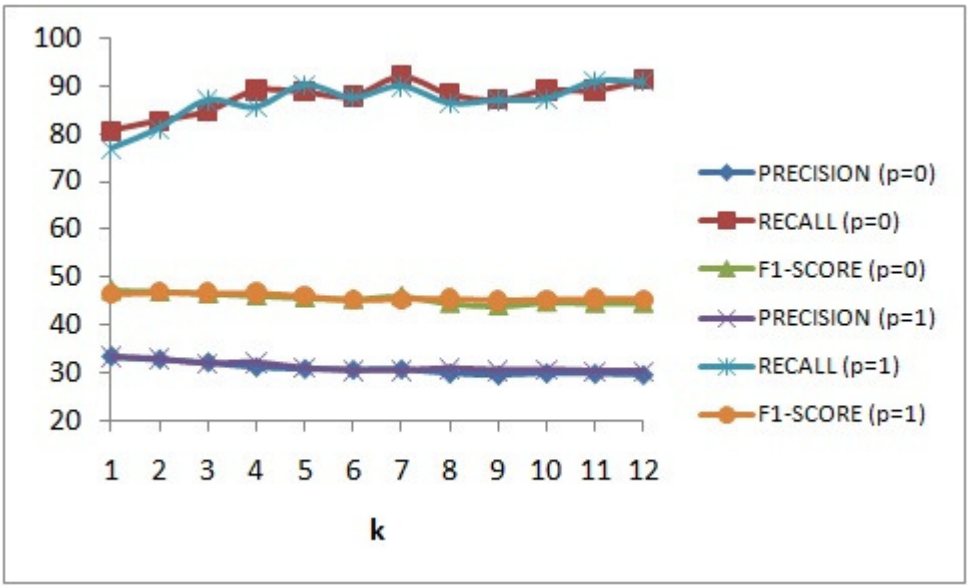

Figure 4.10: p vs instance labeling on CR

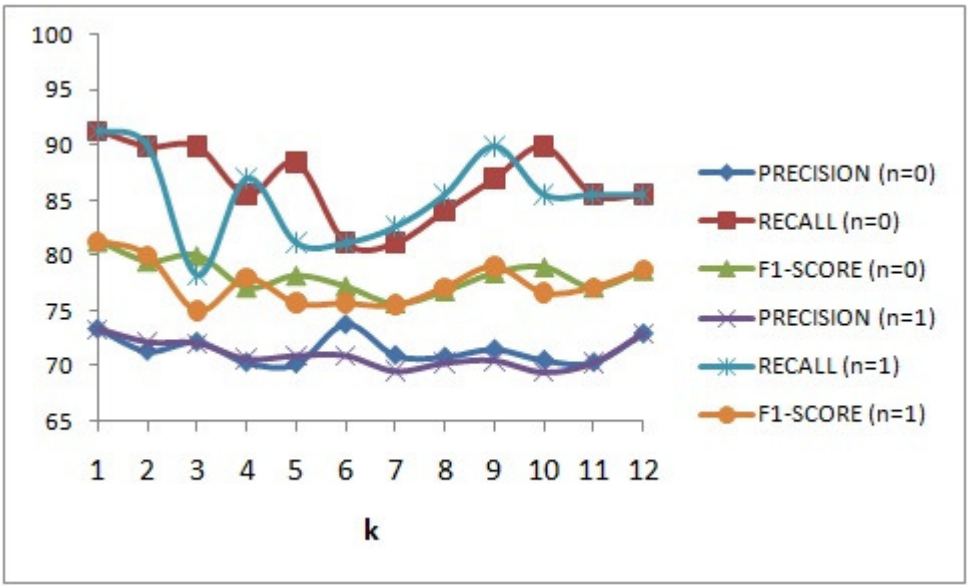

Figure 4.11: n vs bag labeling on MUC4-Hum

4.3.4.4 Conclusion. In general, any margin requirement does not help instance labeling much. Margin requirement on positive bags may help bag labeling when the value of k is low (in our experiments $k < 7$) but hurts for higher values. The best F1-scores across the four columns of each table in Appendix A do not differ significantly. So, instead of tuning k, p and n, one could just vary k and choose the best value. This is good news because this essentially means that we need not compute the second best output vector (which is time consuming).

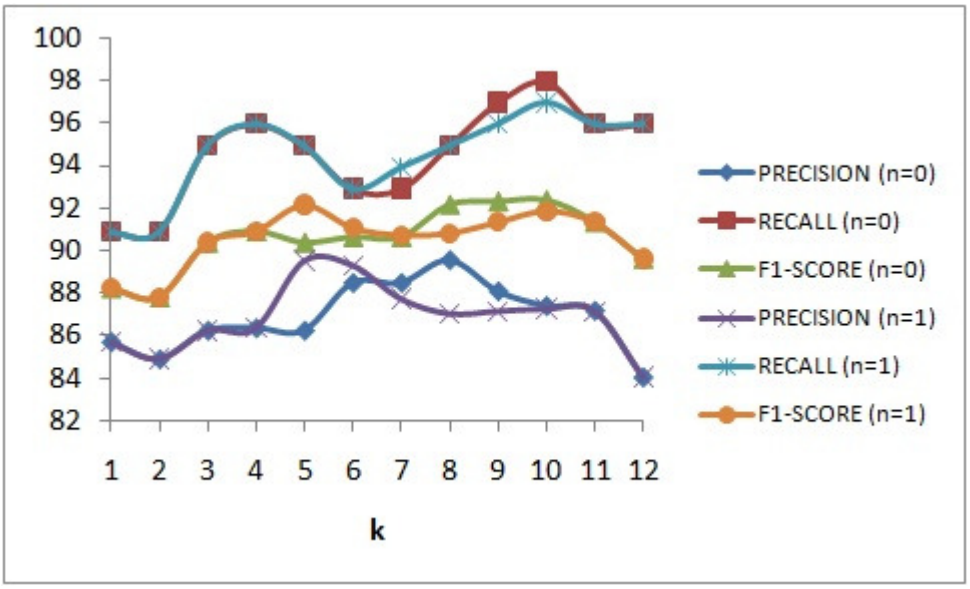

Figure 4.12: n vs bag labeling on ProMed-Hum

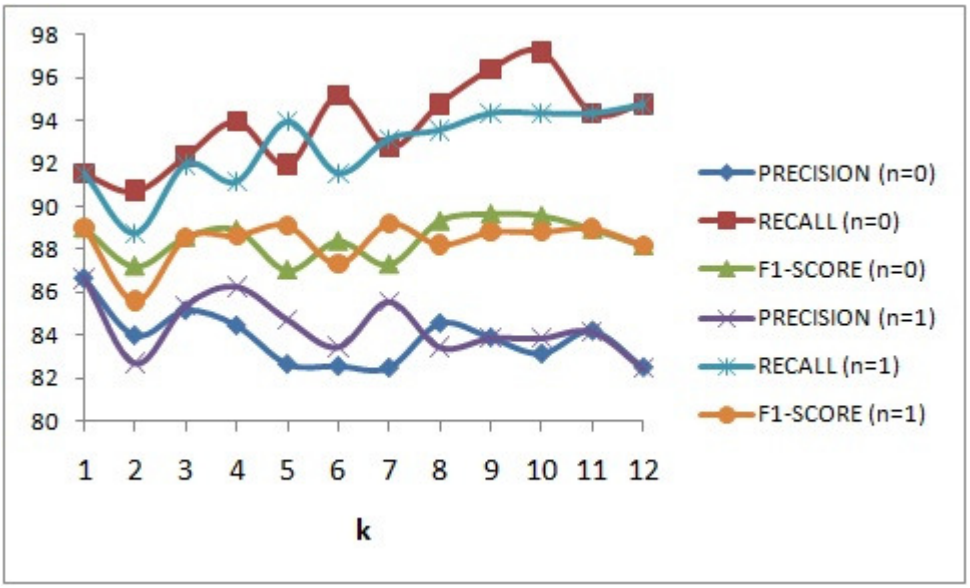

Figure 4.13: n vs bag labeling on CR

4.4 Results

In this section, we compare the results of MISP on all the datasets with the state of the art algorithms for MIL. From our previous experiments (see Section 4.3.4), it is evident that the margins on positive and negative bags do not help as much as increasing the penalty for positive bags. Therefore, we vary k between $\{1, 2, \ldots 12\}$ for MISP. The margins p, n are set to 0 which means that there is no question of whether to use $MISP^{Y_2}$or $MISP^{\bar{Y}}$.

In each of the tables in this section, we give the results of 8 algorithms for both bag labeling and instance labeling. Similar to the previous sections, we report the precision (P),

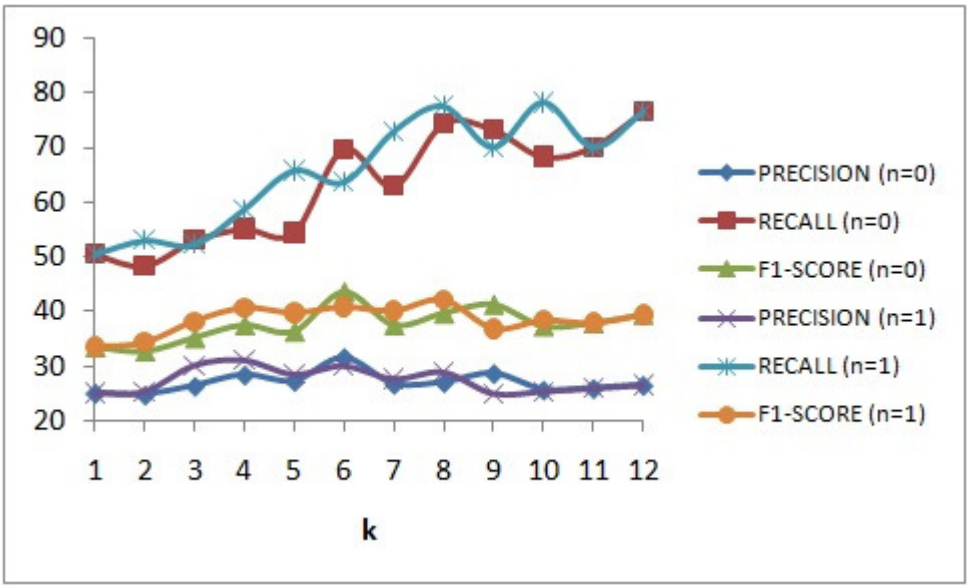

Figure 4.14: n vs instance labeling on MUC4-Hum

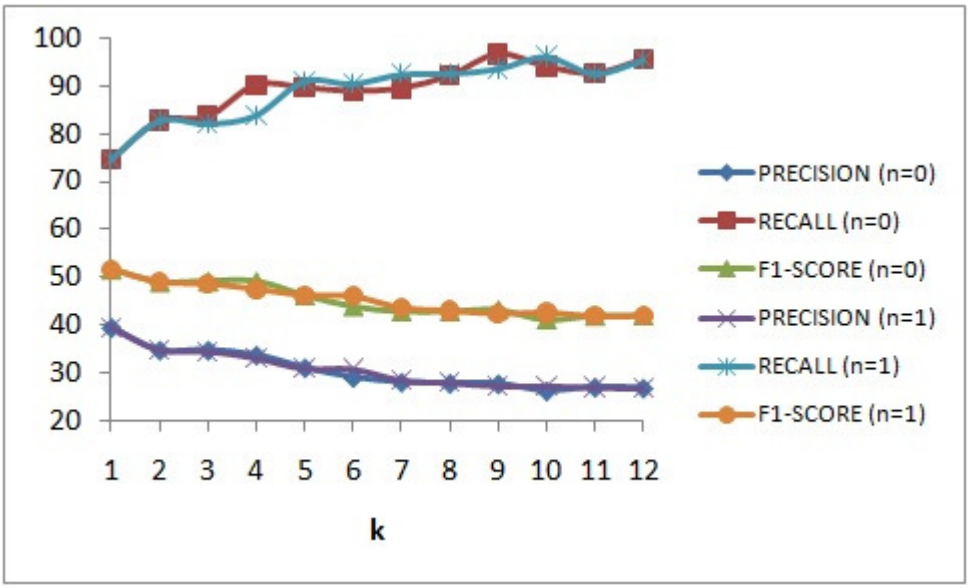

Figure 4.15: n vs instance labeling on ProMed-Hum

recall (R) and F1-score (F). While we do an analysis of precision and recall values, we stick to F1-score for comparing the performance. The row following the results of MISP shows the hyperparameters for which the result has been obtained.

4.4.1 Results on the MUC4-Hum Dataset

Table 4.13 summarizes the performance of different algorithms on the MUC4-Hum dataset.

On bag labeling, we see that MISP performed the best with SVMH-H coming very close

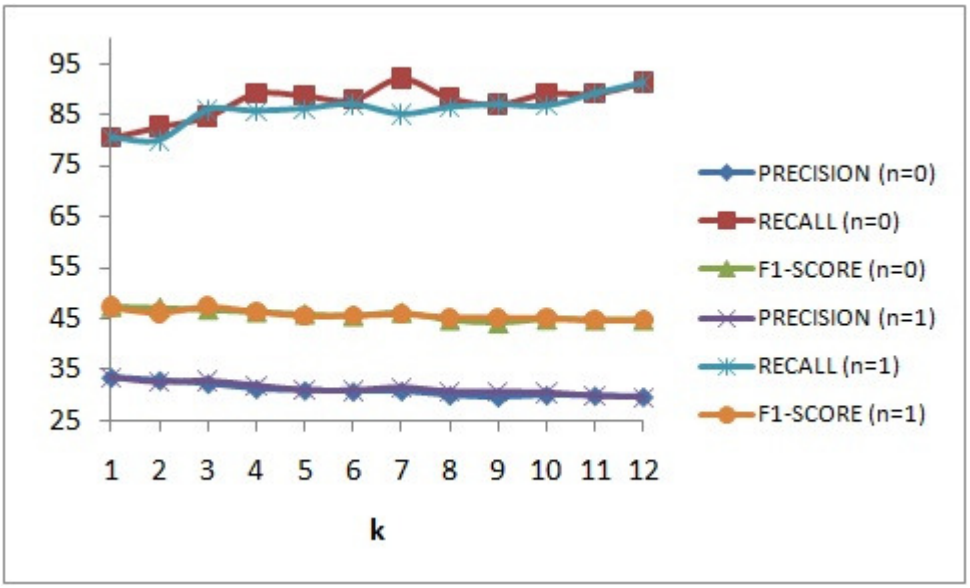

Figure 4.16: n vs instance labeling on CR

Table 4.13: Results on MUC4-Hum

	Bag labeling results			Instance labeling results		
	P	R	F	P	R	F
SIL	68.49	72.46	70.42	25.43	52.31	34.23
NSK	68.57	69.57	69.06	36.87	49.47	42.25
STK	73.17	86.96	79.47	30.66	53.02	38.85
sMIL	68.82	92.75	79.01	26.45	56.94	36.12
stMIL	66.67	89.86	76.54	23.36	59.43	33.53
sbMIL	68.48	91.30	78.26	21.57	57.65	31.40
SVMH-H	69.89	94.20	80.25	21.83	79.00	34.21
MISP	73.26	91.30	81.29	31.61	69.75	43.51
		$k = 1$			$k = 6$	

to it. Among the other algorithms, only STK got a reasonable balance between precision and recall. sMIL got much higher precision than recall. Note that SVMH-H obtained the highest recall among all the algorithms.

On instance labeling, NSK and SVM$^{\text{hmm}}$ gave the best precision and recall, respectively. MISP obtains the best F1-score, marginally better than NSK. Note that the difference in precision between these two algorithms is $\sim 5\%$ points whereas the difference in recall is $\sim 20\%$. Clearly, the recall gain obtained is much more significant.

4.4.2 Results on the ProMed-Hum Dataset

Table 4.14 summarizes the performance of different algorithms on the ProMed-Hum dataset.

Table 4.14: Results on ProMed-Hum

	Bag labeling results			Instance labeling results		
	P	R	F	P	R	F
SIL	82.73	91.92	87.08	24.61	70.62	36.50
NSK	83.04	93.94	88.15	29.81	62.85	40.44
STK	82.30	93.94	87.74	32.94	58.62	42.17
sMIL	84.40	92.93	88.46	29.70	68.50	41.44
stMIL	81.65	89.90	85.58	25.44	79.94	38.60
sbMIL	81.65	89.90	85.58	26.03	84.89	39.84
SVMH-H	83.19	100.0	90.83	25.65	86.44	39.56
MISP	87.39	97.98	92.38	39.55	74.58	51.69
		$k = 10$			$k = 1$	

On bag labeling, MISP and SVM$^{\text{hmm}}$ obtained the best precision and recall values, respectively. MISP got the best F1-score by trading off ~ 2 points of precision with ~ 4 points of recall.

On instance labeling, MISP and SVM$^{\text{hmm}}$ got the best precision and recall values, respectively. MISP dominated the rest of the algorithms in precision by a large margin. Even though stMIL and sbMIL obtained good recall scores, they performed very poor on precision. SVM$^{\text{hmm}}$ obtained a very dominating recall with a reasonably low standard deviation but was very poor on precision. MISP balances precision and recall, therefore it gets the highest F1-score by a huge margin.

4.4.3 Results on the CR Dataset

Table 4.15 summarizes the performance of different algorithms on the CR dataset.

On bag labeling, MISP performed better than every other algorithm with the highest precision, recall and F1-score. It must be noted that MISP increased precision and recall

Table 4.15: Results on CR

	Bag labeling			Instance labeling		
	P	R	F	P	R	F
SIL	80.22	92.92	86.10	30.34	80.32	44.04
NSK	81.32	92.50	86.55	38.88	66.58	49.09
STK	82.16	92.08	86.84	36.06	69.72	47.53
sMIL	80.45	89.17	84.58	32.28	68.55	43.89
stMIL	82.12	93.75	87.55	33.73	66.58	44.77
sbMIL	79.55	87.50	83.33	29.18	91.55	44.26
SVMH-H	82.35	95.97	88.64	28.94	86.34	43.35
MISP	83.86	96.37	89.68	33.50	80.50	47.31
		$k = 9$			$k = 1$	

in nearly equal proportions when compared to the other algorithms.

On instance labeling, NSK and sbMIL obtained the best precision and recall, respectively. Although NSK shows a higher F1-score than sbMIL, it must be noted that it trades off ~ 25 points of recall for ~ 10 points of precision. Since F1-score looks for a balance of precision and recall, it ranked NSK above sbMIL.

4.4.4 Results on the MUC4-Ans Dataset

Table 4.16 summarizes the performance of different algorithms on the MUC4-Ans dataset.

We see the same story with the MUC4-Ans dataset on both bag labeling and instance labeling. Interestingly, SIL performed much better than most other algorithms (F1-score). This is mainly due to its high precision compared to the rest. Most of the other algorithms obtained high recall but low precision. MISP got better F1-scores for both bag and instance labeling by balancing them.

4.4.5 Results on the ProMed-Ans Dataset

Table 4.17 summarizes the performance of different algorithms on the ProMed-Ans dataset.

Most of the algorithms perform well on bag labeling. On instance labeling, although MISP obtained the best F1-score, it must be noted that it gained ~ 4 points on precision for ~ 27 points on recall (compared to NSK, STK, sMIL, sbMIL, stMIL). So, it cannot be declared the winner. All the algorithms in UniverSVM but for SIL performed much better on precision compared to MISP and SVM^{hmm}.

Table 4.16: Results on MUC4-Ans

	Bag labeling			Instance labeling		
	P	R	F	P	R	F
SIL	85.25	85.25	85.25	56.74	49.19	52.70
NSK	67.78	100.0	80.79	23.48	71.77	35.39
STK	67.78	100.0	80.79	45.28	56.05	50.09
sMIL	67.78	100.0	80.79	30.81	77.02	44.01
stMIL	67.78	100.0	80.79	21.48	89.92	34.68
sbMIL	67.78	100.0	80.79	16.06	88.31	27.17
SVMH-H	70.24	96.72	81.38	38.95	86.69	53.75
MISP	77.33	95.08	85.29	50.76	66.94	57.74
		$k = 7$			$k = 8$	

Table 4.17: Results on ProMed-Ans

	Bag labeling			Instance labeling		
	P	R	F	P	R	F
SIL	91.46	80.65	85.71	29.98	55.64	38.97
NSK	84.40	98.92	91.09	24.09	99.25	38.77
STK	83.96	95.70	89.45	24.11	98.50	38.73
sMIL	87.13	94.62	90.72	24.09	99.25	38.77
stMIL	84.38	87.10	85.71	24.11	99.25	38.79
sbMIL	81.40	75.27	78.21	24.09	99.25	38.77
SVMH-H	84.55	100.0	91.63	24.81	54.14	34.03
MISP	90.43	91.40	90.91	27.97	71.88	40.27
		$k = 2$			$k = 4$	

4.4.6 General Observations

The value of structure in these datasets is evident from the fact that SVM^{hmm} with hamming loss performed better than every algorithm in UniverSVM on all the datasets for bag labeling. It performed marginally better than MISP on the ProMed-Ans dataset. One possible reasoning for this is based on the very nature of MIL – with negative bags, you already have all the information needed; with positive bags, even if you get one positive instance right, you are correct on the whole bag. Since we enforce some of the negative sentences in the positive bags to be labeled positive (0/1 loss function), SVM^{hmm} probably decided to concentrate less on the word features in the individual sentences (which it found confusing due to contradictory evidences) and tried to learn the general flow of positive documents. This also explains the fact that it did not match the performance of MISP for instance labeling.

In Section 4.2.2.1, we mentioned that the ProMed-Hum dataset has better structure than the MUC4-Hum dataset. This structure was particularly exploited by MISP. It performed $\sim 10\%$ better than any other algorithm on instance labeling (see Table 4.14).

From the Tables 4.13 and 4.16, we can see the effect of increasing the training data on MUC-4 based datasets. Clearly, there is an improvement on both bag labeling ($\sim$ 5 points) and instance labeling ($\sim$ 13 points). It means that with more training data, MISP is able to learn the structure better. One would expect a similar result from Tables 4.14 and 4.17. However, there was no improvement. In fact, the results were only worse on both bag labeling (by $\sim$ 1 point) and instance labeling ($\sim$ 11 points). This behavior is easy to explain from our discussion in Section 4.2.2.2. The problem with the ProMed-Ans dataset is that the negative documents in the train set are from a different distribution compared to the distribution of negative documents in the test set. To aggravate the problem further, the negative documents in the test set come from the same distribution as the positive

documents in the train set. This caused severe dip in the performance on both the tasks.

One can clearly notice from these tables that MISP obtains higher precision scores than every other algorithm, on both the tasks. In some cases, it has better recall as well. Also, the major difference between SVMH-H and MISP is that the latter has lower recall values. While the former aggressively marks instances from positive bags as positive (due to hamming loss on bags), the latter is careful with it due to an MIL-adapted loss function. So, it gets precision and recall values which are more balanced.

4.4.7 On the Hyperparameter k

It would be fair to ask about the hyperparameter k which may turn out to be an advantage for MISP compared to the other algorithms. If the dataset is small enough, it may be worth trying a few different values for k and pick the best (like in Section 4.4). Otherwise, one could use the method which Bunescu and Mooney (4) use to estimate the positive instance density for sbMIL *i.e.,* get human annotations for a small number of bags and tune k on them[12].

In our case, we already have small annotated samples for both MUC4-Ans (MUC4-Hum) and ProMed-Ans (ProMed-Hum). Table 4.18 shows the performance of MISP on the larger datasets using the value of k which gave the best result on the manually annotated datasets.

From the Table 4.18, we note that on the ProMed-Hum dataset, the performance of MISP with the tuned value of k was better than any other algorithm (see Table 4.17). On bag labeling for the MUC4-Hum dataset, it performed worse than the two basic algorithms (SIL and SVM^{hmm}) although it matched the performance of every other algorithm. On instance labeling, it performed better only compared to NSK, stMIL and sbMIL. In fact, its performance was nowhere close to the performance of SVM^{hmm}. Our intuition as to why there was such a mismatch between the performances on the smaller and larger datasets is that the MUC4-Hum dataset does not have a uniform structure. We had seen such erratic behavior with this dataset while varying k, p and n as well (see Figures 4.1, 4.5, 4.11, 4.14).

[12]It may be useful to try one or two neighboring values for the tuned k since the results on small datasets may not be stable.

Table 4.18: Results on the "Ans" datasets with tuned k

	Bag labeling			Instance labeling		
	P	R	F	P	R	F
MUC4-Hum	67.78	100.0	80.79	32.19	49.19	38.92
		$k = 1$			$k = 6$	
ProMed-Hum	85.44	94.62	89.80	35.23	44.66	39.39
		$k = 10$			$k = 1$	

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

In our work, we have seen a novel approach to solving the MIL problem based on the structure of the bags. We proposed a variation to the cutting plane technique to handle existential constraints on positive bags. We implemented an instance of our idea on SVM$^{\text{hmm}}$with several hyperparameters. We performed experiments on several datasets and all of them show that this idea indeed works very well. We presented a detailed analysis of how different hyperparameters affect our algorithm.

It turned out that the margin constraints are not always useful. The key hyperparameter to be tuned is the penalty on misclassifying a positive bag, for both bag labeling and instance labeling. We could not find a genuine estimator for finding this hyperparameter given a sample of the dataset and thus, we had to choose it using trial-and-error method. Nevertheless, given that the loss function becomes trivial without the margin and that a very small number of iterations is sufficient to get a decent solution, MISP is of great practical value.

We intend to work on the following ideas hereupon:

• If a dataset has known distribution (of number of instances in a positive bag), can we estimate the value of k (penalty for misclassifying a positive bag) for which the performance is maximized? If the answer is yes, then one could manually label a small sample of the training set and avoid the trial-and-error method to find the right value of k.

• Can we modify our variant of the cutting plane technique to ensure convergence (possibly at the cost of ignoring a few opportunities to reach a better solution)? If yes, how fast can this algorithm converge, in terms of the parameters of the dataset? One recent approach to a very related problem is that of Chang *et al.* (22). They consider a structured prediction problem (for instance, phonetic alignments of transliterated words) with a natural associated binary classification problem ("Are these two words transliterations of each other?"). They formulate an SVM-style model that uses both labeled structured data as well as labeled binary data (which is presumably cheaper to obtain) and achieve impressive results. The precise formulation of their approach differs from ours (as the task is somewhat different), but theirs also leads to an existential constraint. Their approach to dealing with

the existential constraint is different than ours, though it is unclear whether one is better than the other or not.

• MIL falls under the category of machine learning algorithms which learn from partial data (*a.k.a.* learning from missing data). In MIL, there is labeled negative data ($\tilde{\chi}_n$). The missing piece of information is as to which of the instances in each $X \in \chi_p$ is negative. Here, MIL is close to semisupervised learning. Adding transductive constraints in binary classification settings helps SVMs significantly (37). Bunescu and Mooney (4) report that moving from *sMIL* to *stMIL* (see Section 2.1.4). One could imagine formulating similar transductive constraints for our formulation as well. The idea is to require a margin for the classification on instances rather than examples.

• Recently, there has been plenty of work on multiclass multiinstance learning for image categorization (38; 16; 42). One could imagine extending MISP to multiple classes as well. This could be done using methods similar to those employed for multiclass MIL (16).

• PU Learning is the task of learning from only positive examples and unlabeled examples. This is *roughly* the opposite of MIL except there is no information at the bag level. Techniques for PU learning (21; 39; 20) involve identifying other examples that are close to the known positive examples. One could imagine a similar idea for MIL – to find instances from positive bags that are close to those in negative bags (say, $\tilde{\chi}_q$). Then, we can train an SVM or a Naive Bayes' classifier with $\tilde{\chi}_n \cup \tilde{\chi}_q$ as negative examples and $\tilde{\chi}_p - \tilde{\chi}_q$ as positive examples.

APPENDIX A

RESULTS OF ALL EXPERIMENTS

In this appendix, we report the results of all the experiments with MISP that we described in Sections 4.3.2, 4.3.4 and 4.4. We do not report the standard deviation values for the three datasets, MUC4-Hum, ProMed-Hum and CR because of space constraints. Like in previous sections, P, R, F represent precision, recall and F1-score, respectively. For the manually labeled datasets, we use $\alpha = 40$. For the larger datasets, we use $\alpha = 25$. We used $MISP^{Y_2}$constraints for all the experiments. We varied k in $\{1, 2, \ldots, 12\}$.

After the tables, just for completeness, we present graphs showing the effect of p, n on the MUC4-Ans and ProMed-Ans datasets (similar to the ones shown in Section 4.3.4).

Table A.1: Bag labeling results on the MUC4-Hum dataset

	$p=0, n=0$			$p=1, n=0$			$p=0, n=1$			$p=1, n=1$		
k	P	R	F	P	R	F	P	R	F	P	R	F
1	73.26	91.30	81.29	70.79	91.30	79.75	73.26	91.30	81.29	70.79	91.30	79.75
2	71.26	89.86	79.49	71.43	94.20	81.25	72.09	89.86	80.00	71.43	94.20	81.25
3	72.09	89.86	80.00	70.45	89.86	78.98	72.00	78.26	75.00	70.45	89.86	78.98
4	70.24	85.51	77.12	72.73	81.16	76.71	70.59	86.96	77.92	72.73	81.16	76.71
5	70.11	88.41	78.21	70.51	79.71	74.83	70.89	81.16	75.68	70.51	79.71	74.83
6	73.68	81.16	77.24	72.50	84.06	77.85	70.89	81.16	75.68	72.50	84.06	77.85
7	70.89	81.16	75.68	71.76	88.41	79.22	69.51	82.61	75.50	71.76	88.41	79.22
8	70.73	84.06	76.82	70.24	85.51	77.12	70.24	85.51	77.12	70.24	85.51	77.12
9	71.43	86.96	78.43	71.05	88.52	78.83	70.45	89.86	78.98	73.17	86.96	79.47
10	70.45	89.86	78.98	70.45	89.86	78.98	69.41	85.51	76.62	70.45	89.86	78.98
11	70.24	85.51	77.12	70.24	85.51	77.12	70.24	85.51	77.12	70.24	85.51	77.12
12	72.84	85.51	78.67	70.73	84.06	76.82	72.84	85.51	78.67	70.73	84.06	76.82

Table A.2: Instance labeling results on the MUC4-Hum dataset

	$p=0, n=0$			$p=1, n=0$			$p=0, n=1$			$p=1, n=1$		
k	P	R	F	P	R	F	P	R	F	P	R	F
1	25.09	50.53	33.53	25.47	53.38	34.48	25.09	50.53	33.53	25.47	53.38	34.48
2	24.77	48.40	32.77	27.13	56.58	36.68	25.38	53.02	34.33	27.13	56.58	36.68
3	26.37	53.02	35.22	28.63	48.40	35.98	30.12	52.31	38.23	28.63	48.40	35.98
4	28.44	55.16	37.53	28.64	63.70	39.51	31.07	58.72	40.64	28.64	63.70	39.51
5	27.27	54.45	36.34	28.93	65.48	40.13	28.51	65.84	39.78	28.93	65.48	40.13
6	31.61	69.75	43.51	28.03	70.82	40.16	30.03	63.70	40.82	28.03	70.82	40.16
7	26.74	62.99	37.54	27.71	68.33	39.43	27.67	72.95	40.12	27.71	68.33	39.43
8	27.04	74.38	39.66	25.77	80.43	39.03	28.91	77.58	42.13	25.77	80.43	39.03
9	28.69	73.31	41.24	24.65	71.54	36.67	25.03	70.11	36.89	26.21	73.31	38.61
10	25.67	68.33	37.32	25.43	79.72	38.55	25.40	78.29	38.36	25.43	79.72	38.55
11	25.96	70.11	37.88	24.75	79.36	37.73	25.96	70.11	37.88	24.75	79.36	37.73
12	26.54	76.51	39.41	26.18	81.14	39.58	26.54	76.51	39.41	26.18	81.14	39.58

Table A.3: Bag labeling results on the ProMed-Hum dataset

	$p=0, n=0$			$p=1, n=0$			$p=0, n=1$			$p=1, n=1$		
k	P	R	F	P	R	F	P	R	F	P	R	F
1	85.71	90.91	88.24	85.71	90.91	88.24	85.71	90.91	88.24	85.71	90.91	88.24
2	84.91	90.91	87.80	85.05	91.92	88.35	84.91	90.91	87.80	85.05	91.92	88.35
3	86.24	94.95	90.38	86.92	93.94	90.29	86.24	94.95	90.38	86.92	93.94	90.29
4	86.36	95.96	90.91	86.49	96.97	91.43	86.36	95.96	90.91	86.49	96.97	91.43
5	86.24	94.95	90.38	89.22	91.92	90.55	89.52	94.95	92.16	89.22	91.92	90.55
6	88.46	92.93	90.64	89.32	92.93	91.09	89.32	92.93	91.09	89.32	92.93	91.09
7	88.46	92.93	90.64	88.57	93.94	91.18	87.74	93.94	90.73	88.57	93.94	91.18
8	89.52	94.95	92.16	87.04	94.95	90.82	87.04	94.95	90.82	87.04	94.95	90.82
9	88.07	96.97	92.31	87.04	94.95	90.82	87.16	95.96	91.35	87.04	94.95	90.82
10	87.39	97.98	92.38	87.27	96.97	91.87	87.27	96.97	91.87	87.27	96.97	91.87
11	87.16	95.96	91.35	85.45	94.95	89.95	87.16	95.96	91.35	85.45	94.95	89.95
12	84.07	95.96	89.62	79.69	91.07	85.00	84.07	95.96	89.62	83.93	94.95	89.10

Table A.4: Instance labeling results on the ProMed-Hum dataset

	$p=0, n=0$			$p=1, n=0$			$p=0, n=1$			$p=1, n=1$		
k	P	R	F	P	R	F	P	R	F	P	R	F
1	39.55	74.58	51.69	37.95	72.32	49.78	39.55	74.58	51.69	37.95	72.32	49.78
2	34.80	82.77	49.00	36.98	82.20	51.01	34.80	82.77	49.00	36.98	82.20	51.01
3	34.80	83.62	49.15	34.47	80.08	48.19	34.48	82.20	48.58	34.47	80.08	48.19
4	33.76	90.25	49.13	31.88	86.86	46.64	33.13	83.90	47.50	31.88	86.86	46.64
5	31.11	89.69	46.20	30.57	87.99	45.38	30.94	91.10	46.19	30.57	87.99	45.38
6	29.07	88.98	43.83	30.81	91.24	46.06	30.85	90.54	46.02	30.81	91.24	46.06
7	28.07	89.55	42.74	28.60	88.98	43.28	28.53	92.51	43.61	28.60	88.98	43.28
8	27.86	92.09	42.78	27.74	92.94	42.73	28.01	92.66	43.02	27.74	92.94	42.73
9	27.74	96.61	43.10	27.65	93.50	42.68	27.36	93.64	42.35	27.65	93.50	42.68
10	26.22	94.07	41.01	27.31	95.76	42.49	27.36	96.19	42.60	27.31	95.76	42.49
11	27.03	92.66	41.85	26.83	97.18	42.05	27.03	92.66	41.85	26.83	97.18	42.05
12	26.83	95.48	41.88	26.79	96.98	41.99	26.83	95.48	41.88	26.82	97.88	42.10

Table A.5: Bag labeling results on the CR dataset

	$p=0, n=0$			$p=1, n=0$			$p=0, n=1$			$p=1, n=1$		
k	P	R	F	P	R	F	P	R	F	P	R	F
1	86.64	91.53	89.02	84.06	93.55	88.55	86.64	91.53	89.02	84.06	93.55	88.55
2	83.96	90.73	87.21	84.31	93.15	88.51	82.71	88.71	85.60	84.31	93.15	88.51
3	85.13	92.34	88.59	83.21	93.95	88.26	85.39	91.94	88.54	83.21	93.95	88.26
4	84.42	93.95	88.93	85.93	93.55	89.58	86.26	91.13	88.63	85.93	93.55	89.58
5	82.61	91.94	87.02	84.39	91.53	87.81	84.73	93.95	89.10	84.39	91.53	87.81
6	82.52	95.16	88.39	83.21	93.95	88.26	83.46	91.53	87.31	83.21	93.95	88.26
7	82.44	92.74	87.29	84.36	93.55	88.72	85.56	93.15	89.19	84.36	93.55	88.72
8	84.53	94.76	89.35	83.81	93.95	88.59	83.45	93.55	88.21	83.81	93.95	88.59
9	83.86	96.37	89.68	84.48	94.35	89.14	83.87	94.35	88.80	84.48	94.35	89.14
10	83.10	97.18	89.59	84.17	94.35	88.97	83.87	94.35	88.80	84.17	94.35	88.97
11	84.17	94.35	88.97	84.12	93.95	88.76	84.17	94.35	88.97	84.12	93.95	88.76
12	82.46	94.76	88.18	83.93	94.76	89.02	82.46	94.76	88.18	83.93	94.76	89.02

Table A.6: Instance labeling results on the CR dataset

	$p=0, n=0$			$p=1, n=0$			$p=0, n=1$			$p=1, n=1$		
k	P	R	F	P	R	F	P	R	F	P	R	F
1	33.50	80.50	47.31	33.36	76.73	46.50	33.50	80.50	47.31	33.36	76.73	46.50
2	32.90	82.66	47.07	32.96	81.04	46.86	32.52	79.96	46.23	32.96	81.04	46.86
3	32.24	84.64	46.69	31.96	86.97	46.74	32.77	85.89	47.44	31.96	86.97	46.74
4	31.22	89.13	46.25	32.15	85.53	46.74	31.87	85.71	46.47	32.15	85.53	46.74
5	30.87	88.77	45.80	30.97	90.21	46.11	30.97	86.16	45.56	30.97	90.21	46.11
6	30.67	87.87	45.47	30.44	87.51	45.17	30.90	87.06	45.61	30.44	87.51	45.17
7	30.75	92.09	46.11	30.40	89.94	45.44	31.46	85.09	45.94	30.40	89.94	45.44
8	29.90	88.32	44.67	30.90	86.34	45.51	30.71	86.43	45.31	30.90	86.34	45.51
9	29.50	87.06	44.07	30.48	86.97	45.14	30.58	86.97	45.25	30.48	86.97	45.14
10	29.94	89.13	44.83	30.59	87.24	45.30	30.50	86.70	45.13	30.59	87.24	45.30
11	29.82	89.13	44.68	30.31	90.93	45.46	29.82	89.13	44.68	30.31	90.93	45.46
12	29.58	91.28	44.68	30.27	90.84	45.41	29.58	91.28	44.68	30.27	90.84	45.41

Table A.7: Bag labeling results on the MUC4-Ans dataset

	$p=0, n=0$			$p=1, n=0$			$p=0, n=1$			$p=1, n=1$		
k	P	R	F	P	R	F	P	R	F	P	R	F
1	67.78	100.0	80.79	69.88	95.08	80.56	74.29	85.25	79.39	69.88	95.08	80.56
2	70.89	91.80	80.00	78.33	77.05	77.69	70.89	91.80	80.00	78.33	77.05	77.69
3	75.68	91.80	82.96	74.03	93.44	82.61	75.68	91.80	82.96	74.03	93.44	82.61
4	78.12	81.97	80.00	74.19	75.41	74.80	78.12	81.97	80.00	74.19	75.41	74.80
5	71.60	95.08	81.69	72.15	93.44	81.43	67.78	100.0	80.79	72.15	93.44	81.43
6	76.39	90.16	82.71	75.00	93.44	83.21	76.39	90.16	82.71	75.00	93.44	83.21
7	77.33	95.08	85.29	85.94	90.16	88.00	67.78	100.0	80.79	85.94	90.16	88.00
8	75.00	88.52	81.20	76.32	95.08	84.67	76.12	83.61	79.69	76.32	95.08	84.67
9	73.42	95.08	82.86	76.71	91.80	83.58	71.60	95.08	81.69	76.71	91.80	83.58
10	71.08	96.72	81.94	67.78	100.0	80.79	79.17	93.44	85.71	67.78	100.0	80.79
11	78.46	83.61	80.95	78.12	81.97	80.00	78.46	83.61	80.95	78.12	81.97	80.00
12	78.12	81.97	80.00	78.12	81.97	80.00	78.12	81.97	80.00	78.12	81.97	80.00

Table A.8: Instance labeling results on the MUC4-Ans dataset

	$p=0, n=0$			$p=1, n=0$			$p=0, n=1$			$p=1, n=1$		
k	P	R	F	P	R	F	P	R	F	P	R	F
1	16.35	49.19	24.55	52.94	43.55	47.79	56.98	39.52	46.67	52.94	43.55	47.79
2	54.61	33.47	41.50	56.55	33.06	41.73	54.61	33.47	41.50	56.55	33.06	41.73
3	46.39	62.10	53.10	34.51	35.48	34.99	46.39	62.10	53.10	34.51	35.48	34.99
4	61.48	33.47	43.34	52.85	52.42	52.63	61.48	33.47	43.34	52.85	52.42	52.63
5	34.71	64.52	45.13	34.20	52.82	41.52	55.36	50.00	52.54	34.20	52.82	41.52
6	32.19	49.19	38.92	32.51	37.10	34.65	41.88	66.53	51.40	32.51	37.10	34.65
7	51.60	52.02	51.81	51.18	70.16	59.18	49.83	58.87	53.97	51.18	70.16	59.18
8	50.76	66.94	57.74	58.08	53.63	55.77	56.72	54.44	55.56	58.08	53.63	55.77
9	40.70	70.56	51.62	34.61	58.47	43.48	36.90	73.79	49.19	34.61	58.47	43.48
10	41.96	67.34	51.70	55.50	48.79	51.93	40.85	81.85	54.50	55.50	48.79	51.93
11	39.83	75.00	52.03	45.45	68.55	54.66	39.83	75.00	52.03	45.45	68.55	54.66
12	40.04	72.18	51.51	44.14	65.32	52.68	40.04	72.18	51.51	44.14	65.32	52.68

Table A.9: Bag labeling results on the ProMed-Ans dataset

	$p=0, n=0$			$p=1, n=0$			$p=0, n=1$			$p=1, n=1$		
k	P	R	F	P	R	F	P	R	F	P	R	F
1	87.00	93.55	90.16	88.30	89.25	88.77	87.00	93.55	90.16	88.30	89.25	88.77
2	90.43	91.40	90.91	88.57	66.67	76.07	90.43	91.40	90.91	88.57	66.67	76.07
3	86.73	91.40	89.01	89.89	86.02	87.91	86.73	91.40	89.01	89.89	86.02	87.91
4	91.11	88.17	89.62	90.91	86.02	88.40	91.67	94.62	93.12	90.91	86.02	88.40
5	91.36	79.57	85.06	91.01	87.10	89.01	91.36	79.57	85.06	91.01	87.10	89.01
6	90.22	89.25	89.73	90.59	82.80	86.52	90.91	86.02	88.40	90.59	82.80	86.52
7	88.66	92.47	90.53	89.25	89.25	89.25	91.57	81.72	86.36	89.25	89.25	89.25
8	86.41	95.70	90.82	89.36	90.32	89.84	90.70	83.87	87.15	89.36	90.32	89.84
9	90.48	81.72	85.88	90.48	81.72	85.88	90.11	88.17	89.13	90.48	81.72	85.88
10	85.44	94.62	89.80	90.00	77.42	83.24	90.59	82.80	86.52	90.00	77.42	83.24
11	90.32	90.32	90.32	89.47	91.40	90.43	90.32	90.32	90.32	89.47	91.40	90.43
12	88.66	92.47	90.53	87.76	92.47	90.05	88.66	92.47	90.53	87.76	92.47	90.05

Table A.10: Instance labeling results on the ProMed-Ans dataset

	$p=0, n=0$			$p=1, n=0$			$p=0, n=1$			$p=1, n=1$		
k	P	R	F	P	R	F	P	R	F	P	R	F
1	35.23	44.66	39.39	35.14	43.01	38.67	35.23	44.66	39.39	35.14	43.01	38.67
2	33.73	46.47	39.09	34.34	42.71	38.07	33.73	46.47	39.09	34.34	42.71	38.07
3	28.06	49.62	35.85	29.34	55.64	38.42	28.06	49.62	35.85	29.34	55.64	38.42
4	28.77	63.16	39.53	31.83	53.98	40.04	28.38	65.26	39.56	31.83	53.98	40.04
5	29.75	53.23	38.17	28.47	61.35	38.89	29.46	49.92	37.05	28.47	61.35	38.89
6	26.31	69.47	38.17	27.81	64.36	38.84	28.72	64.21	39.68	27.81	64.36	38.84
7	26.13	77.14	39.04	25.61	71.43	37.70	28.39	63.61	39.26	25.61	71.43	37.70
8	24.40	82.11	37.62	25.70	74.74	38.25	27.81	63.31	38.64	25.70	74.74	38.25
9	27.86	63.31	38.69	27.94	59.70	38.06	27.35	73.08	39.80	27.94	59.70	38.06
10	25.42	76.84	38.21	29.43	63.46	40.21	28.57	59.85	38.68	29.43	63.46	40.21
11	27.97	71.88	40.27	25.46	79.70	38.59	27.97	71.88	40.27	25.46	79.70	38.59
12	24.87	79.55	37.89	23.54	80.90	36.47	24.87	79.55	37.89	23.54	80.90	36.47

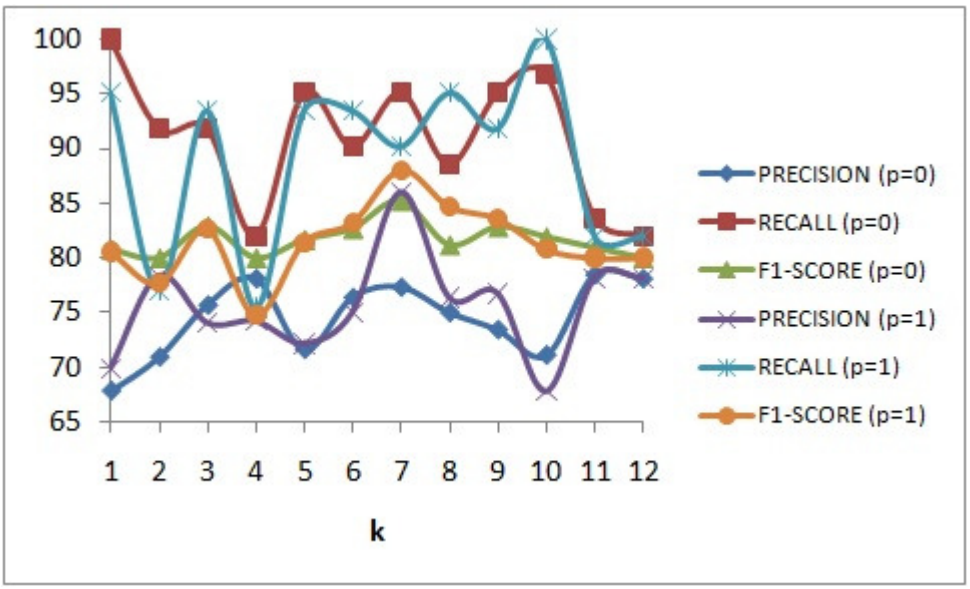

Figure A.1: p vs bag labeling on MUC4-Ans

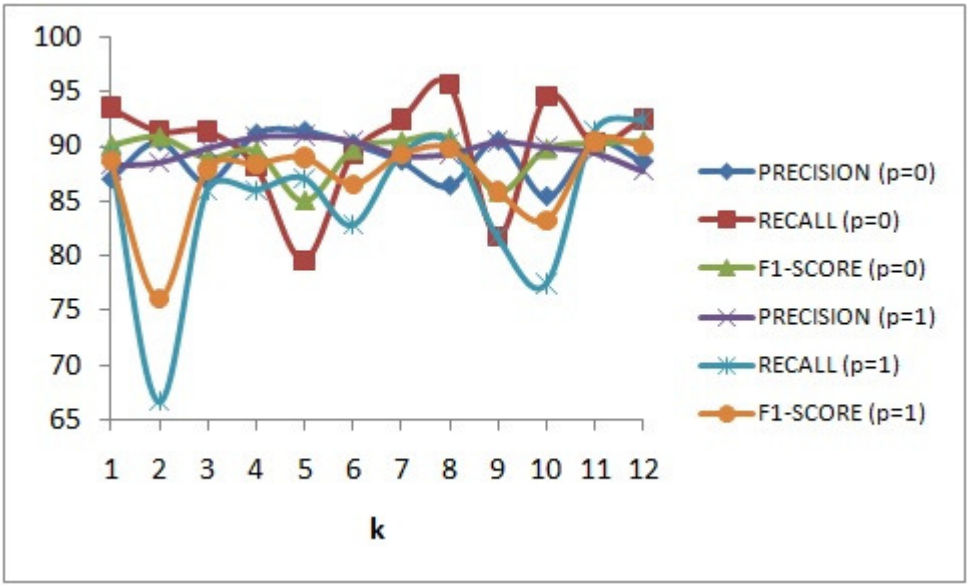

Figure A.2: p vs bag labeling on ProMed-Ans

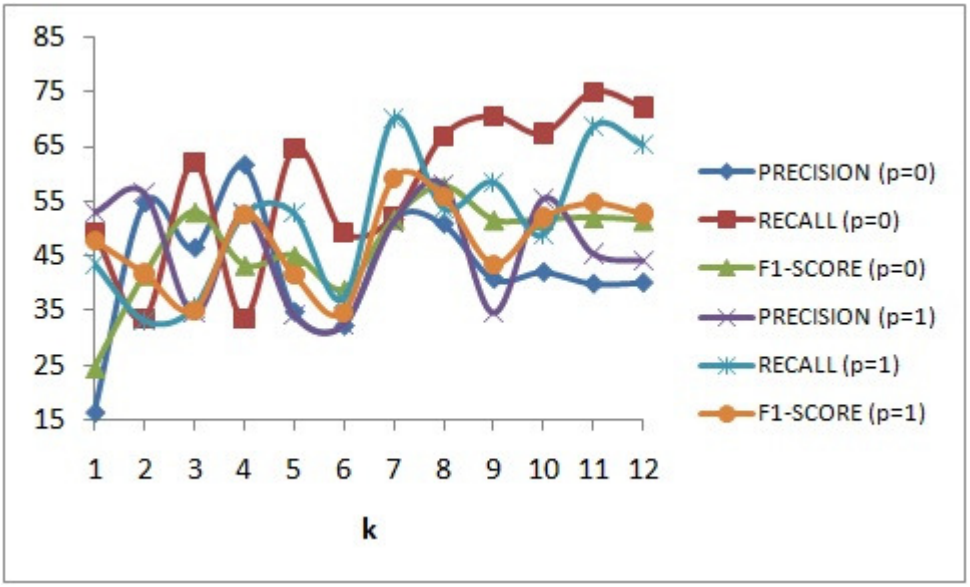

Figure A.3: p vs instance labeling on MUC4-Ans

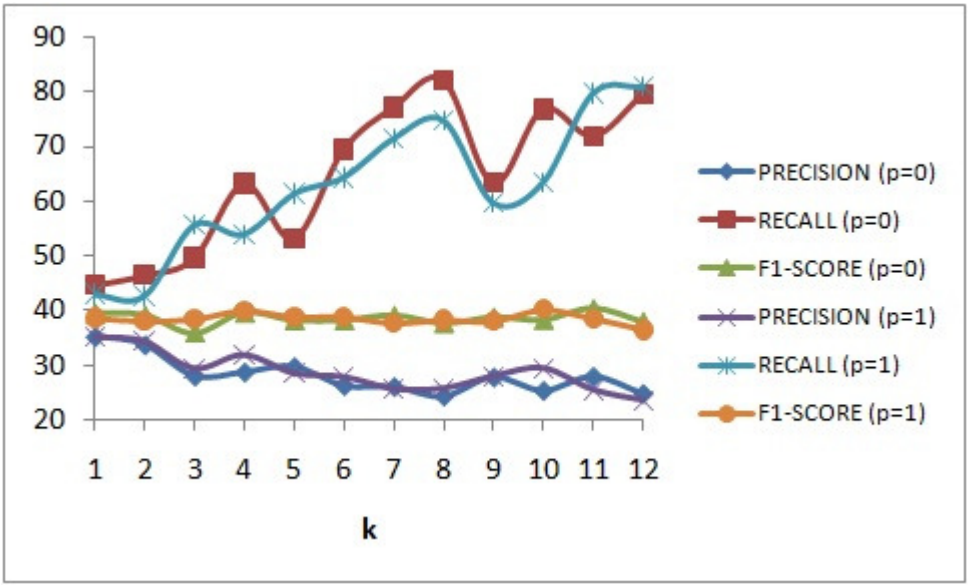

Figure A.4: p vs instance labeling on ProMed-Ans

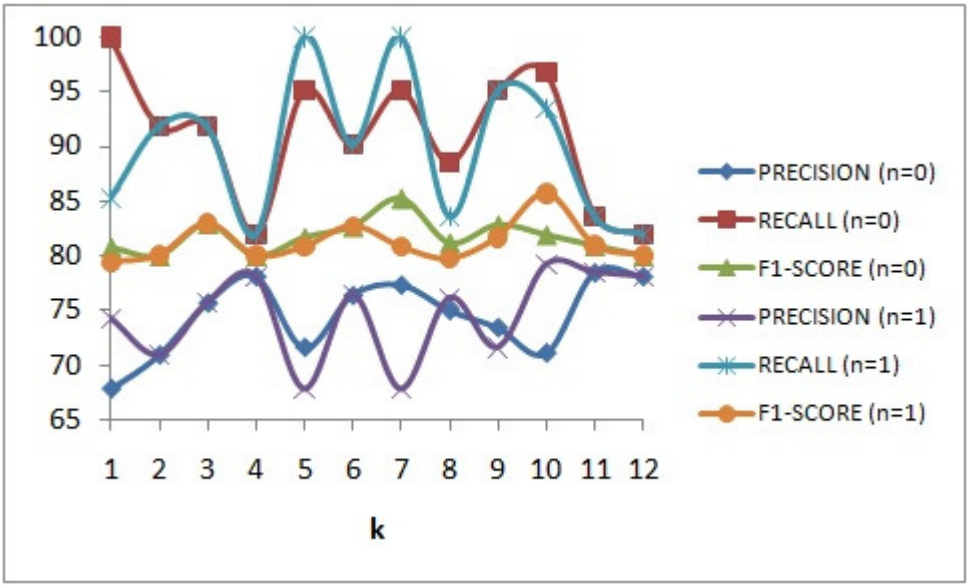

Figure A.5: n vs bag labeling on MUC4-Ans

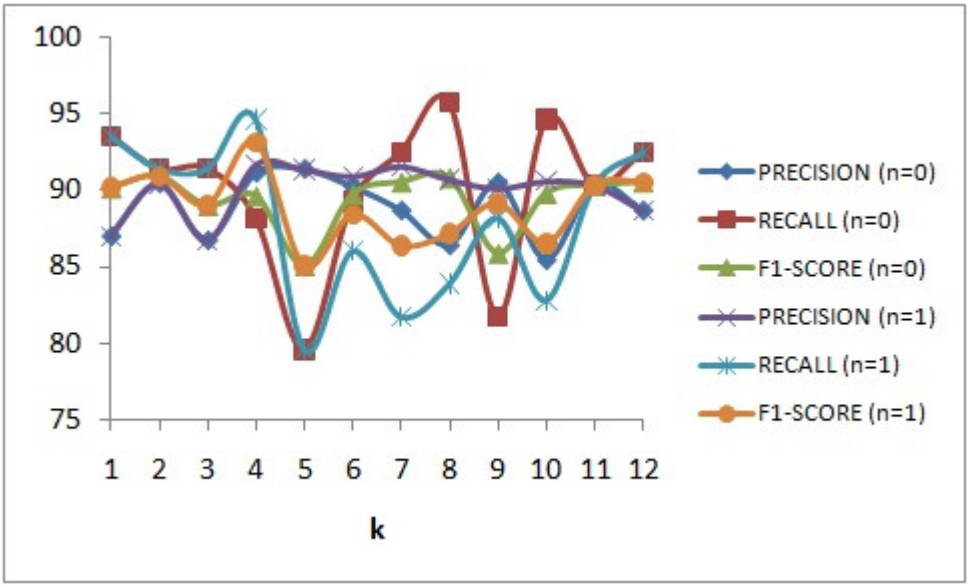

Figure A.6: n vs bag labeling on ProMed-Ans

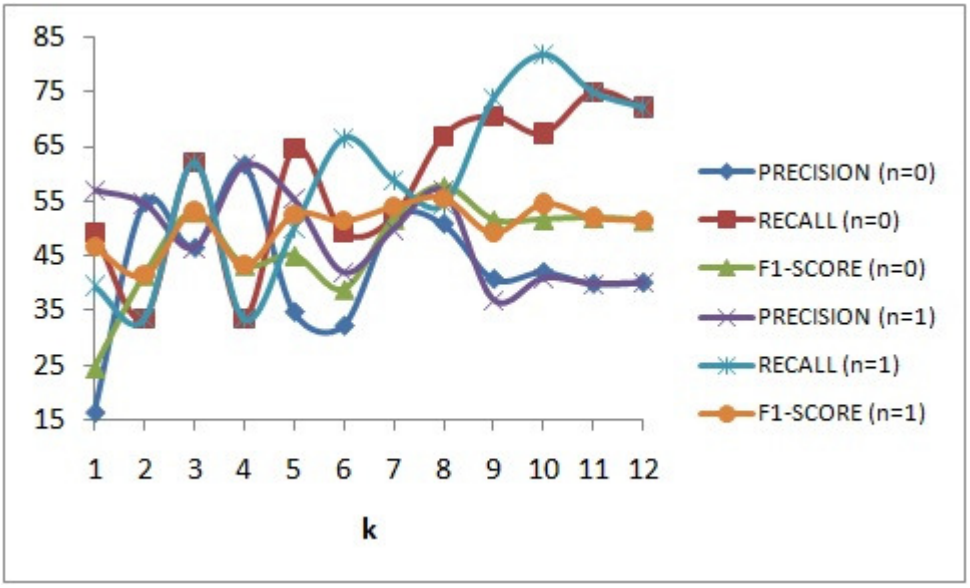

Figure A.7: n vs instance labeling on MUC4-Ans

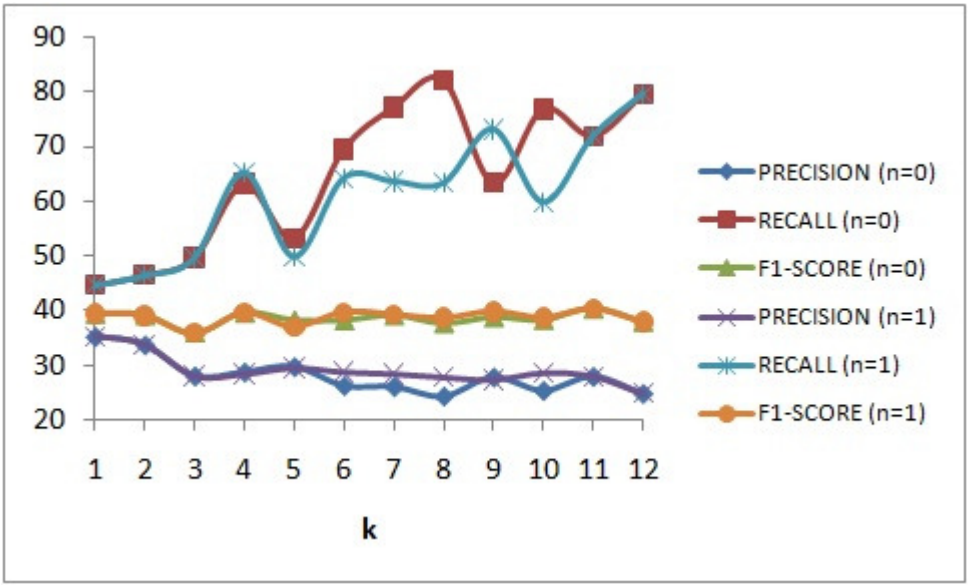

Figure A.8: n vs instance labeling on ProMed-Ans

APPENDIX B

INPUT FORMATS

In this section, we discuss the format in which SVM$^{\text{hmm}}$ and UniverSVM expect the input to be in.

B.1 Format of Input Files for SVM$^{\text{hmm}}$

SVM$^{\text{hmm}}$follows the libsvm[1] input format for most part. Figure B.1 shows a sample input file. The general formatting rules are as follows:

• Each line in the file corresponds to an instance. It begins with a label (an identifier for a class). All labels have to be positive integers. In our work, we used $1, 2$ for negative and positive bags, respectively.

• After the label, there will be a *qid* term to identify which example the instance is a part of. It is the common link between multiple lines of a file.

• Following the qid part will be the features. Each feature has is represented as a positive integer. The feature and its weight (*a.k.a.* value) will be separated by a colon. The features must appear in ascending order. A feature with 0 weight can be ignored.

In the training set, we give the bag labels to each of the instance labels *i.e.,* all the instances with the same qid will have the same label. In the tune and test set, we give the instance labels because the classifier needs to compare its results with the truth to output the performance metrics (P,R,F).

B.2 Format of Input Files for UniverSVM

UniverSVM also extends the libsvm format. It is described below:

• The number of examples in the file has to be in the first line of the file. Each line following this represents an instance.

• Each line corresponding to an instance begins with $sid : qid : label$ where *sid* is the instance id, *qid* is the example id and *label* is an identifier for its class. Note that the *sid* has to be unique across all the examples, not just the example that it belongs to.

---

[1]LIBSVM is a very popular software among machine learning community for classification and regression. It is available for free download along with a lot of free classification datasets at: `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`.

```
1 qid:1 1:0.35 2:0.78 3:0.34 4:0.02 5:0.24 6:0.12 17:0.59
2 qid:1 1:0.47 2:0.20 3:0.43 4:0.17 5:0.43 6:0.03 18:0.94
1 qid:1 1:0.36 2:0.81 3:0.31 4:0.04 5:0.25 6:0.11 12:0.62
1 qid:2 1:0.37 2:0.91 3:0.24 4:0.08 5:0.27 6:0.10 17:0.70
2 qid:2 1:0.66 2:0.45 3:0.21 4:0.13 6:0.20 9:0.31 12:0.91
1 qid:2 1:0.37 2:0.95 3:0.19 4:0.10 5:0.27 6:0.10 18:0.75
2 qid:2 1:0.66 2:0.43 3:0.24 4:0.16 5:0.29 6:0.30 11:0.92
1 qid:3 1:0.37 2:0.91 3:0.12 4:0.15 5:0.32 6:0.09 17:0.79
1 qid:3 1:0.37 2:0.88 3:0.06 4:0.16 5:0.32 6:0.09 11:0.81
1 qid:3 1:0.49 2:0.30 3:0.27 4:0.15 5:0.29 6:0.57 17:0.97
1 qid:3 1:0.48 2:0.31 3:0.27 4:0.14 5:0.28 6:0.56 19:0.97
1 qid:4 1:0.48 2:0.33 3:0.18 4:0.13 5:0.27 6:0.56 13:0.94
2 qid:4 1:0.49 2:0.25 3:0.42 4:0.23 5:0.50 6:0.04 10:0.98
1 qid:4 1:0.47 2:0.32 3:0.12 4:0.12 5:0.27 6:0.56 17:0.92
2 qid:4 1:0.69 2:0.78 3:0.10 4:0.19 5:0.28 6:0.63 17:0.79
2 qid:5 1:0.67 2:0.45 3:0.09 4:0.04 6:0.22 7:0.41 19:0.88
1 qid:5 1:0.47 2:0.26 3:0.18 4:0.09 5:0.23 6:0.54 17:0.93
2 qid:5 1:0.67 2:0.46 3:0.13 4:0.04 5:0.22 8:0.40 20:0.89
```

Figure B.1: Input format for SVM$^{\text{hmm}}$

• Features (along with their weights) must follow the metadata about the instance just like SVM$^{\text{hmm}}$.

A sample input file for UniverSVM can be seen in Figure B.2. UniverSVM has an additional flexibility with the process of training a classifier. One could put the training and test sets in the same input file and specify the split of train and test splits using *id-files*. Each id-file contains a list of example ids (as used in the input file). Effectively, one could train and test a classifier by specifying the input file and two id-files (train split, test split). We did not use this functionality because we tuned the classifier parameters before testing.

```
5
01:1:1 1:0.35 2:0.78 3:0.34 4:0.02 5:0.24 6:0.12 17:0.59
02:1:2 1:0.47 2:0.20 3:0.43 4:0.17 5:0.43 6:0.03 18:0.94
03:1:1 1:0.36 2:0.81 3:0.31 4:0.04 5:0.25 6:0.11 12:0.62
04:2:1 1:0.37 2:0.91 3:0.24 4:0.08 5:0.27 6:0.10 17:0.70
05:2:2 1:0.66 2:0.45 3:0.21 4:0.13 6:0.20 9:0.31 12:0.91
06:2:1 1:0.37 2:0.95 3:0.19 4:0.10 5:0.27 6:0.10 18:0.75
07:2:2 1:0.66 2:0.43 3:0.24 4:0.16 5:0.29 6:0.30 11:0.92
08:3:1 1:0.37 2:0.91 3:0.12 4:0.15 5:0.32 6:0.09 17:0.79
09:3:1 1:0.37 2:0.88 3:0.06 4:0.16 5:0.32 6:0.09 11:0.81
10:3:1 1:0.49 2:0.30 3:0.27 4:0.15 5:0.29 6:0.57 17:0.97
11:3:1 1:0.48 2:0.31 3:0.27 4:0.14 5:0.28 6:0.56 19:0.97
12:4:1 1:0.48 2:0.33 3:0.18 4:0.13 5:0.27 6:0.56 13:0.94
13:4:2 1:0.49 2:0.25 3:0.42 4:0.23 5:0.50 6:0.04 10:0.98
14:4:1 1:0.47 2:0.32 3:0.12 4:0.12 5:0.27 6:0.56 17:0.92
15:4:2 1:0.69 2:0.78 3:0.10 4:0.19 5:0.28 6:0.63 17:0.79
16:5:2 1:0.67 2:0.45 3:0.09 4:0.04 6:0.22 7:0.41 19:0.88
17:5:1 1:0.47 2:0.26 3:0.18 4:0.09 5:0.23 6:0.54 17:0.93
18:5:2 1:0.67 2:0.46 3:0.13 4:0.04 5:0.22 8:0.40 20:0.89
```

Figure B.2: Input format for UniverSVM

APPENDIX C

MODIFICATIONS TO SVM^{hmm} CODE

In this section, we briefly describe the changes we have made to SVM^{hmm}. Listing all the changes will be too much since we wrote at least 1000 lines of code for this project.

SVM^{hmm} comes as a `svm_hmm.tar.gz` file. When you extract it using the command `tar xvzf svm_hmm.tar.gz`, you will get a directory named `svm_hmm`. You may enter this directory and see its structure using the command "`tree .`" (Figure C.1).

The directory svm_light contains the code to perform optimization given the QP problem whereas svm_struct contains the code to perform cutting plane optimization for structured inputs. Files in the parent directory *i.e.,* svm_hmm, contain API which can be modified to use SVM^{struct} for a different kind of structure. By default, with SVM^{hmm} we get HMM versions of the API.

C.1 svm_struct_api.c

This file contains the logic for loss function and executing structure-specific inference algorithms. For SVM^{hmm} this would be viterbi algorithm.

C.1.1 loss()

The following two changes are made in loss function. We implement MISP function in the stub left for "`-l 0`".

• Take model as an additional parameter.

• If it is training phase, use MISP (Figure 3.5) instead of hamming loss. If it is testing phase, use hamming loss.

C.1.2 viterbi_forward_order1()

If it is testing phase and a bias is provided (see Section 4.1.3), modify the transition probabilities as required. There may be boilerplate code to obtain the values of bias from the command line arguments through the main function.

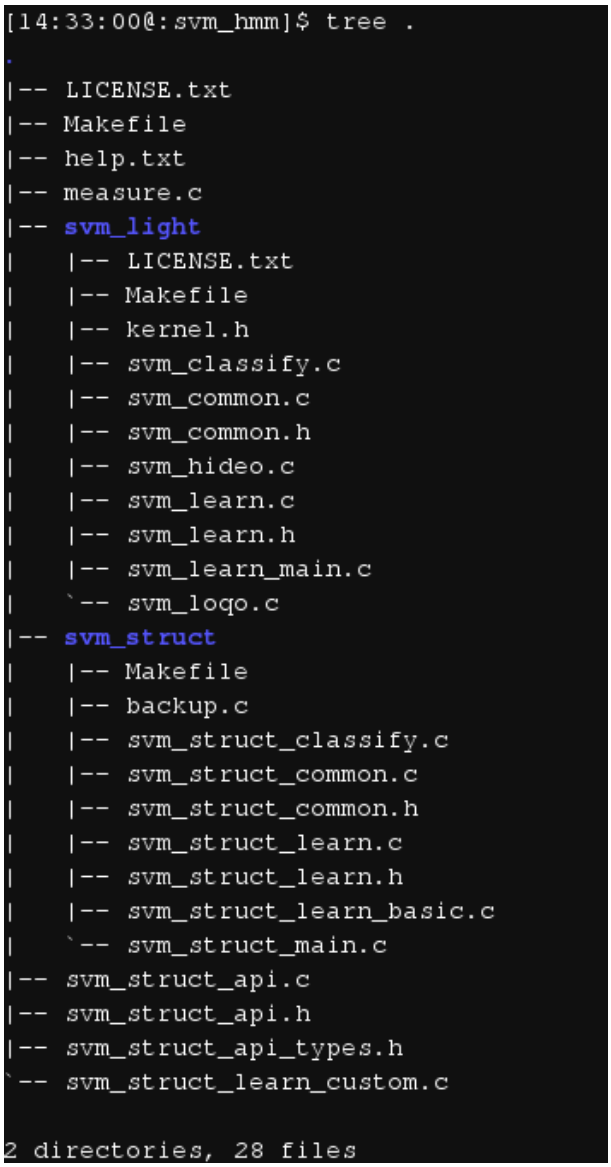

Figure C.1: Directory structure of SVM$^{\text{hmm}}$

C.2 svm_struct/svm_struct_learn.c

This file contains the logic to re-solve the QP optimization problem with the new most violating constraints found in each iteration. Here, we made the following changes:

• If it is a positive bag on which we are finding the most violated constraint, then we check if this constraint defies the "norm" of cutting plane optimization (slack of this constraint is smaller than the slack of working set). If yes, then we replace the previous constraint on this bag with this constraint. In fact, our implementation keeps *exactly* one constraint for each bag.

• If we finish α number of iterations, we stop the cutting plane optimization and write the model.

C.3 Miscellaneous

The files svm_struct_main.c and svm_struct_classify.c contain code to parse the command line options and store them in appropriate data structures, for svm_hmm_learn binary (learning module of the classifier) and svm_hmm_classify binary (prediction module of the classifier). They also set the default values to various data structures required for the execution of SVM$^{\text{hmm}}$. We modified these to add additional command line arguments (α, bias for transition probabilities *etc.*).

For MUC4-Hum and ProMed-Hum datasets, we needed to segment the sentences. For this, we used `Lingua::EN::Sentence` package from `http://search.cpan.org/~shlomoy/Lingua-EN-Sentence-0.25/lib/Lingua/EN/Sentence.pm`.

REFERENCES

[1] Andrews, S., Tsochantaridis, I., and Hofmann, T. Support vector machines for multiple-instance learning. In *Advances in Neural Information Processing Systems 15* (2003), MIT Press, pp. 561–568.

[2] Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*, 1st ed. 2006. corr. 2nd printing ed. Springer, October 2007.

[3] Brander, A. W., and Sinclair, M. C. A comparative study of k-shortest path algorithms. In *Proceedings of 11th UK Performance Engineering Workshop* (1995), pp. 370–379.

[4] Bunescu, R. C., and Mooney, R. J. Multiple instance learning for sparse positive bags. In *ICML '07: Proceedings of the 24th International Conference on Machine Learning* (New York, NY, USA, 2007), ACM, pp. 105–112.

[5] Cohen, J. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement 20*, 1 (1960), 37.

[6] Collins, M. Discriminative reranking for natural language parsing. In *Computational Linguistics* (2000), Morgan Kaufmann, pp. 175–182.

[7] Collins, M. Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10* (Stroudsburg, PA, USA, 2002), EMNLP '02, Association for Computational Linguistics, pp. 1–8.

[8] Collins, M. *Parameter Estimation for Statistical Parsing Models: Theory and Practice of Distribution-free Methods.* Kluwer Academic Publishers, Norwell, MA, USA, 2004, pp. 19–55.

[9] Collins, M., and Roark, B. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics* (Stroudsburg, PA, USA, 2004), ACL '04, Association for Computational Linguistics.

[10] Dietterich, T. G., Lathrop, R. H., and Lozano-Pérez, T. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence 89*, 1-2 (1997), 31–71.

[11] Eppstein, D. Bibliography on k shortest paths and other "k-best solutions" problems. `http://www.ics.uci.edu/eppstein/bibs/kpath.bib`.

[12] Gärtner, T., Flach, P. A., Kowalczyk, A., and Smola, A. J. Multi-instance kernels. In *ICML '02: Proceedings of the Nineteenth International Conference on Machine Learning* (San Francisco, CA, USA, 2002), Morgan Kaufmann Publishers Inc., pp. 179–186.

[13] Grishman, R., and Sundheim, B. Message understanding conference-6: a brief history. In *Proceedings of the 16th Conference on Computational Linguistics - Volume 1* (Stroudsburg, PA, USA, 1996), COLING '96, Association for Computational Linguistics, pp. 466–471.

[14] Hasan, F., UzZaman, N., and Khan, M. Comparison of different pos tagging techniques (n-gram, hmm and brill's tagger) for bangla. In *Advances and Innovations in Systems, Computing Sciences and Software Engineering*, K. Elleithy, Ed. Springer Netherlands, 2007, pp. 121–126.

[15] Hu, M., and Liu, B. Mining and summarizing customer reviews. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2004), KDD '04, ACM, pp. 168–177.

[16] hua Zhou, Z., and ling Zhang, M. Multi-instance multi-label learning with application to scene classification. In *Advances in Neural Information Processing Systems 19* (2007).

[17] Joachims, T., Finley, T., and Yu, C.-N. Cutting-plane training of structural svms. *Machine Learning 77* (2009), 27–59.

[18] Keeler, J. D., Rumelhart, D. E., and Leow, W.-K. Integrated segmentation and recognition of hand-printed numerals. In *NIPS-3: Proceedings of the 1990 Conference on Advances in Neural Information Processing Systems 3* (San Francisco, CA, USA, 1990), Morgan Kaufmann Publishers Inc., pp. 557–563.

[19] Lafferty, J., McCallum, A., and Pereira, F. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning* (2001), pp. 282–289.

[20] Li, X.-L., Liu, B., and Ng, S.-K. Negative training data can be harmful to text classification. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing* (Stroudsburg, PA, USA, 2010), EMNLP '10, Association for Computational Linguistics, pp. 218–228.

[21] Liu, B., Dai, Y., Li, X., Lee, W. S., and Yu, P. S. Building text classifiers using positive and unlabeled examples. In *Proceedings of the 3rd IEEE International Conference on Data Mining* (Washington, DC, USA, 2003), ICDM '03, IEEE Computer Society, pp. 179–.

[22] M. Chang, V. Srikumar, D. G., and Roth, D. Structured output learning with indirect supervision. In *Proceedings of the International Conference on Machine Learning (ICML)* (2010).

[23] Maron, O., and Lozano-Pérez, T. A framework for multiple-instance learning. In *NIPS '97: Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10* (Cambridge, MA, USA, 1998), MIT Press, pp. 570–576.

[24] McCallum, A., Freitag, D., and Pereira, F. C. N. Maximum entropy markov models for information extraction and segmentation. In *Proceedings of the 17th International Conference on Machine Learning* (San Francisco, CA, USA, 2000), ICML '00, Morgan Kaufmann Publishers Inc., pp. 591–598.

[25] Mohri, M. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics 7* (January 2002), 321–350.

[26] Mohri, M., and Riley, M. An efficient algorithm for the n-best-strings problem. In *Proceedings of the International Conference on Spoken Language Processing 2002 (ICSLP'02)* (2002).

[27] Nilsson, D., and Goldberger, J. Sequentially finding the n-best list in hidden markov models. In *17th International Joint Conference on Artificial Intelligence (IJCAI'01)* (2001), pp. 1280–1285.

[28] Orio, N., and Dechelle, F. Score following using spectral analysis and hidden markov models. In *Proceedings of the International Computer Music Conference* (2001), pp. 125–129.

[29] Patwardhan, S., and Riloff, E. Effective information extraction with semantic affinity patterns and relevant regions. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)* (Prague, Czech Republic, June 2007), Association for Computational Linguistics, pp. 717–727.

[30] Patwardhan, S., and Riloff, E. A unified model of phrasal and sentential evidence for information extraction. In *EMNLP '09: Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing* (Morristown, NJ, USA, 2009), Association for Computational Linguistics, pp. 151–160.

[31] Plamondon, R., and Srihari, S. Online and off-line handwriting recognition: a comprehensive survey. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 22*, 1 (Jan. 2000), 63 –84.

[32] Rabiner, L. R. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE 77*, 2 (Feb 1989), 257–286.

[33] Sarawagi, S., and Gupta, R. Accurate max-margin training for structured output spaces. In *Proceedings of the 25th International Conference on Machine Learning* (New York, NY, USA, 2008), ICML '08, ACM, pp. 888–895.

[34] Sinz, F., and Roffilli, M. Universvm, 2010. `http://mloss.org/software/view/19/`.

[35] Taskar, B., Guestrin, C., and Koller, D. Max-margin markov networks. In *Neural Information Processing Systems* (2003).

[36] Tsochantaridis, I., Hofmann, T., Joachims, T., and Altun, Y. Support vector machine learning for interdependent and structured output spaces. In *ICML '04: Proceedings of the 21st International Conference on Machine Learning* (New York, NY, USA, 2004), ACM, p. 104.

[37] Vapnik, V. N. *Statistical Learning Theory*. Wiley-Interscience, Sept. 1998.

[38] Xu, X., and Li, B. Evaluating multi-class multiple-instance learning for image categorization. In *Proceedings of the 8th Asian Conference on Computer Vision - Volume Part II* (Berlin, Heidelberg, 2007), ACCV'07, Springer-Verlag, pp. 155–165.

[39] Yu, H., Zuo, W., and Peng, T. A new pu learning algorithm for text classification. In *MICAI 2005: Advances in Artificial Intelligence*, vol. 3789 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2005, pp. 824–832.

[40] Zhang, Q., and Goldman, S. A. Em-dd: An improved multiple-instance learning technique. In *Advances in Neural Information Processing Systems* (2001), MIT Press, pp. 1073–1080.

[41] Zhang, Q., Goldman, S. A., Yu, W., and Fritts, J. Content-based image retrieval using multiple-instance learning. In *ICML '02: Proceedings of the 19th International Conference on Machine Learning* (San Francisco, CA, USA, 2002), Morgan Kaufmann Publishers Inc., pp. 682–689.

[42] Zhu, L., Zhao, B., and Gao, Y. Multi-class multi-instance learning for lung cancer image classification based on bag feature selection. *Fourth International Conference on Fuzzy Systems and Knowledge Discovery 2* (2008), 487–492.