Feature Construction Using Theory-Guided Sampling and Randomised Search

Sachindra Joshi[1], Ganesh Ramakrishnan[1], and Ashwin Srinivasan[1,2]

[1] IBM India Research Laboratory, 4-C, Vasant Kunj Institutional Area
New Delhi 110070, India
[2] Dept. of Computer Science and Engineering & Centre for Health Informatics,
University of New South Wales, Sydney
{jsachind,ganramkr,ashwin.srinivasan}@in.ibm.com

Abstract. It has repeatedly been found that very good predictive models can result from using Boolean features constructed by an an Inductive Logic Programming (ILP) system with access to relevant relational information. The process of feature construction by an ILP system, sometimes called "propositionalization", has been largely done either as a pre-processing step (in which a large set of possibly useful features are constructed first, and then a predictive model is constructed) or by tightly coupling feature construction and model construction (in which a predictive model is constructed with each new feature, and only those that result in a significant improvement in performance are retained). These represent two extremes, similar in spirit to filter and wrapper-based approaches to feature selection. An interesting, third perspective on the problem arises by taking search-based view of feature construction. In this, we conceptually view the task as searching through subsets of all possible features that can be constructed by the ILP system. Clearly an exhaustive search of such a space will usually be intractable. We resort instead to a randomised local search which repeatedly constructs randomly (but non-uniformly) a subset of features and then performs a greedy local search starting from this subset. The number of possible features usually prohibits an enumeration of all local moves. Consequently, the next move in the search-space is guided by the errors made by the model constructed using the current set of features. This can be seen as sampling non-uniformly from the set of all possible local moves, with a view of selecting only those capable of improving performance. The result is a procedure in which a feature subset is initially generated in the pre-processing style, but further alterations are guided actively by actual model predictions. We test this procedure on language processing task of word-sense disambiguation. Good models have previously been obtained for this task using an SVM in conjunction with ILP features constructed in the pre-processing style. Our results show an improvement on these previous results: predictive accuracies are usually higher, and substantially fewer features are needed.

1 Introduction

Most machine-learning techniques for constructing models from sample data are feature-based. By this, we mean that they expect data in which objects are

F. Železný and N. Lavrač (Eds.): ILP 2008, LNAI 5194, pp. 140–157, 2008.

encoded as vectors of values for a pre-specified set of attributes or features, with the object of the model often being to predict the value of one of these features using the values of the others.[1] One machine-learning approach that clearly does not fit into this category is Inductive Logic Programming (ILP), which deals instead with learning from instances of objects represented in a relational form. For example, suppose we are concerned with building models for identifying toxic chemicals. An appropriate relational representation may be the atoms in the chemical, their location in 3-dimensional space, the bonds amongst them, the different structural and functional group locations (benzene rings, methyl groups *etc.*) and so on. A feature-based representation, on the other hand, may use the bulk properties of the chemical (its molecular weight, the number of atoms of a particular type *etc.*), some structural features (the presence of fused benzene rings, for example), and so on. ILP systems are able to construct models using the former representation, while methods like decision trees, neural networks, support vector machines require the latter.

Despite an ILP system's ability to handle more complex representations, there has long flourished a somewhat heretical strand of ILP activity that has sought to convert (in some efficient manner) the relational representation into a feature-based one, with a view of using well-established feature-based model construction methods. While a theoretical case could be made for some of this work—positive learnability results for specific forms were shown nearly 15 years ago by Dzeroski *et. al* [16]—the primary motivations have been practical. First, feature-based model constructors are computationally efficient. Second, with appropriate features, they are able to construct significantly complex predictive models. Third, both data analysts and domain experts alike are more familiar with these methods than they are with ILP. This form of model construction, with an ILP system providing some or all of the features, and a feature-based learner constructing the actual model has repeatedly been shown to be remarkably effective (see [2,17,18] for some examples).

If we are indeed committed to using a feature-based learner, then a case can be made for ILP as a natural choice for the automatic construction of new features, if the relational information can be encoded in a logical form. The techniques essentially fall into one of two categories: those that construct features independent of the subsequent model-constructor; and those that inter-leave feature and model-construction. The earliest example of the former is LINUS [1], that, given a set of relations, constructed all features possible within some syntactic restrictions. Essentially a similar idea, but using a much more efficient technique for generating the features is executed by [3]. Also in the same category are approaches like those exemplified by [19], which impose additional semantic constraints on the features (using modern terminology, these constraints effectively place minimum requirements on the support and confidence values). The inter-leaved approach sequentially builds up a set of features, in a manner

[1] The terminology, from statistical modelling, of predicting a dependent variable using a set of independent variables is related, but we avoid it here, since independence amongst the features is often not as apparent.

similar to forward-selection techniques used by a statistical model constructor. That is, features constructed by the ILP engine are given one at a time to the model constructor, and only those that actually improve predictive performance are retained. The k-FOIL system [4] and the SAYU procedure proposed in [5] are recent examples of this. The reader will recognise that the two approaches are similar in spirit to filter and wrapper-based approaches to feature selection [20]: the difference being, of course, that we do not already have the features to select from here.

There is an interesting, third perspective to be gained by accepting the fact that what is being done is actually a search for the best subset of features that can be constructed by an ILP engine. Clearly, for any realistic problem, this space would be too large to search exhaustively. In this paper, we examine instead the use of a well-established method of randomised search for exploring this space. There are three specific advantages we gain from adopting this perspective: (1) Randomised search techniques, although not provably optimal, have been shown to be very effective in searching extremely large searce spaces. We are now able to apply them to the task of feature construction; (2) The pre-processing and inter-leaved forms of feature construction can be seen as special cases of a randomised algorithm searching through the feature-subset space; (3) We are able to understand better some of the steps of feature construction in terms of generating a non-uniform sample of local moves for the randomised search. In this paper, we implement the simplest form of randomised search inspired by the GSAT algorithm [21], and investigate its performance on language processing tasks for which ILP-constructed features (generated in the pre-processing style) have been found to be useful.

The rest of the paper is organised as follows. Section 2 formulates feature-construction by an ILP engine as a search process. The use of a randomised local search procedure to execute this search is described in Section 2.1. The basic procedure is usually impractical to use directly. Modifications to the basic randomised procedure in the form of using the errors made by the model-constructor to guide sampling of local moves are in Section 2.2. We examine the possibility of reducing the computational burden further by using models that assign weights to features. This is described in Section 2.3. An empirical evaluation of the approach for the word-sense disambiguation problems is presented in Section 3. Section 5 concludes the paper.

2 Feature Construction as Search

We motivate the approach we propose using the "trains" problem, originally proposed by Ryzhard Michalski. The task, familiar to many readers, is to construct a model that can discriminate between eastbound and westbound trains, using properties of their carriages, and the loads carried (see Fig. 1).

We will assume that the trains can be adequately described by background predicates that will become evident shortly. Further, let us assume that the 10 trains shown in the figure are denoted $t1, t2, \ldots, t10$ and that their

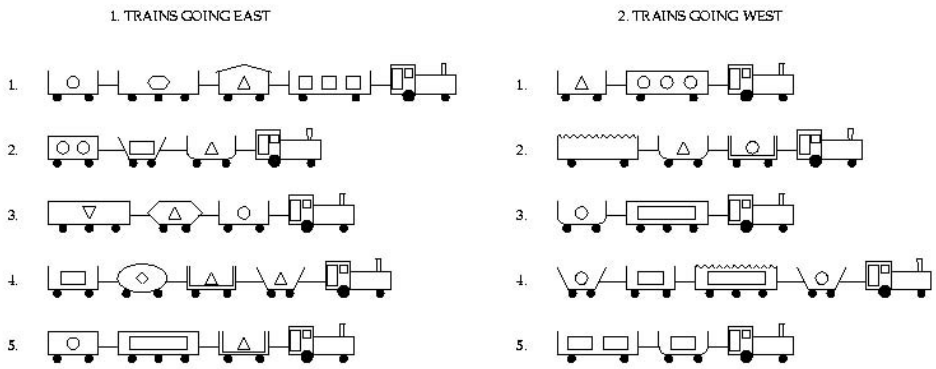

Fig. 1. The trains problem. Trains are classified either as "eastbound" or "westbound". They have open or closed carriages of different shapes, lengths, and so on. The carriages contain loads of different shapes and numbers. The task is to construct a model that, given the description of a train, can predict whether it will be eastbound or westbound.

classifications are encoded (in the Prolog language) as a set of examples of the form: $class(t1, eastbound)$, $class(t2, eastbound)$, ..., $class(t10, westbound)$. As is quite normal in the use of ILP for feature-construction, we will assume features to be Boolean valued, and obtained from some clause identified by the ILP program. For example, Fig. 2 shows five such features, found by an ILP engine, and the corresponding tabular representation of the 10 examples in Fig. 1.

Suppose that these 6 features are the only features that can be constructed by the ILP engine, and further, that it is our task to find the best subset of these that can result in the best model. Clearly, if we simply evaluated models obtained with each of the 63 subsets of the set $\{f1, f2, f3, f4, f5, f6\}$ and return the subset that returned the best model, we would be done. Now, let us consider

$f1 = \begin{cases} 1 \; iff \; has \; short \; and \; closed \; car \\ 0 \; otherwise \end{cases}$

$f2 = \begin{cases} 1 \; iff \; has \; closed \; car \; and \; contains \; triangle \; load \\ 0 \; otherwise \end{cases}$

$f3 = \begin{cases} 1 \; iff \; has \; double-wall \; car \; and \; contains \; triangle \; load \\ 0 \; otherwise \end{cases}$

$f4 = \begin{cases} 1 \; iff \; has \; car \; with \; jagged \; roof \\ 0 \; otherwise \end{cases}$

$f5 = \begin{cases} 1 \; iff \; has \; long \; car \; and \; contains \; rectaglular \; load \\ 0 \; otherwise \end{cases}$

$f6 = \begin{cases} 1 \; iff \; has \; long \; car \; and \; can \; with \; jagged \; roof \\ 0 \; otherwise \end{cases}$

f1	f2	f3	f4	f5	f6	Class
1	1	0	0	0	0	East
1	0	0	0	0	0	East
1	1	0	0	0	0	East
1	0	1	0	0	0	East
1	0	1	0	0	0	East
0	0	0	0	0	0	West
0	0	0	1	0	1	West
0	0	0	0	1	0	West
0	0	0	1	1	1	West
0	0	0	0	0	0	West

Fig. 2. Some Boolean features for the trains problem, and a corresponding tabular representation of the trains in Fig. 1

a more practical situation. Suppose the features that can be constructed by an ILP system are not in the 10s, but in the 1000s or even 100s of 1000s. This would make it intractable to construct models with all possible subsets of features. Further, suppose constructing each feature is not straightforward, computationally speaking, making it impractical to even use the ILP engine to construct all the features in the first place. Are we able to nevertheless able to determine the subset that would yield the best model (which we will now interpret to mean the model with the highest classification accuracy). The problem to be addressed is shown in Fig. 3.

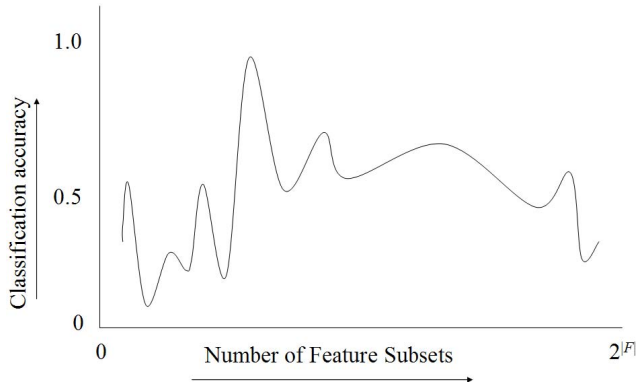

Fig. 3. Identifying the best subset of features for a model-construction algorithm A. The X-axis enumerates the different subsets of features that can be constructed by an ILP engine ($\mathcal{F}$ denotes the set of all possible features that can be constructed by the engine). The Y-axis shows the probability that an an instance drawn randomly using some pre-specified distribution will be correctly classified by a model constructed by A, given the corresponding subset on the X-axis. We wish to identify the subset $k*$ that yields the highest probability, without actually constructing all the features in $\mathcal{F}$.

Readers will recognise this as somewhat similar to the problem addressed by a randomised procedure for distribution-estimation like Gibb's sampling. There, if the $\mathcal{F}$ features are given (or at least can be enumerated), then it is possible to converge on the best-performing subset without examining the entire space of $2^{|\mathcal{F}|}$ elements. Clearly, if we are unable to generate all possible features in $\mathcal{F}$ beforehand, we are not in a position to use these methods. Instead, we resort a randomised local search procedure inspired by a randomised procedure for checking satisfiability of Boolean formulae.

2.1 A Randomised Local Search Procedure

Randomised local search procedures are some of the most effective methods proposed for addressing the hard problem of determining the satisfiability of propositional formulae. The basic search procedure embodied in a technique like GSAT [21] is straightforward (Fig. 4, taken from [6]).

(Here R and M represent the number of random restarts and moves allowed.)

This kind of search procedure has been adapted successfuly to the ILP problem of identifying a set of clauses that, along with some background knowledge, entail a set of examples [15]. Here, we examine its use for feature construction (Fig. 5).

Existing techniques for feature construction can be re-cast as special cases of the procedure in Fig. 5, with appropriate values assigned to R and M; and definitions of a starting point (Step 3a) and local moves (Step 3(f)i). For example, "LINUS-inspired" methods that use an ILP engine to construct a large number of features independent of the model constructor can be seen as an instance of the randomised procedure with $R = 1$ and $M = 0$. Some additional constraints may be imposed on the "starting subset" selected in Step 3a, which could be encapsulated in the distribution used for random selection. Any further selection amongst features in this subset are then upto the model-constructor. On the other hand, SAYU-like procedures can be emulated with $R = 1$ and large values of M; along with restrictions that force the search to start from an empty subset, and local moves only to add a single feature at a time.

```
1. currentbest:= 0 ("0" is some conventional default answer)
2. for i = 1 to R do begin
   (a) current:= randomly selected starting point
   (b) if current is better than currenbest then currentbest:= current
   (c) for j = 1 to M do begin
          i. next:= best local move from current
         ii. if next is better than currenbest then currentbest:= next
        iii. current:= next
   (d) end
3. end
4. return currentbest
```

Fig. 4. A basic randomised local search procedure

```
1. bestfeatures:= {}
2. bestaccuracy:= 0.0
3. for i = 1 to R do begin
   (a) currentfeatures:= randomly selected set of features
   (b) currentmodel:= model constructed with currentfeatures
   (c) accuracy:= estimated accuracy of currentmodel
   (d) if accuracy > bestaccuracy then begin
          i. bestfeatures:= currentfeatures
         ii. bestaccuracy:= accuracy
   (e) end
   (f) for j = 1 to M do begin
          i. nextfeatures:= best local move from currentfeatures
         ii. nextmodel:= model constructed with nextfeatures
        iii. accuracy:= estimated accuracy of nextmodel
         iv. if accuracy > bestaccuracy then begin
                A. bestfeatures:= nextfeatures
                B. bestaccuracy:= accuracy
          v. end
         vi. currentfeatures:= nextfeatures
   (g) end
4. end
5. return bestfeatures
```

Fig. 5. The basic randomised local search procedure, adapted to the task of feature construction

We consider now the more general procedure shown. In this, R and M can take on any value from the set of natural numbers (including 0). In addition, the starting subset is assumed to be drawn using some distribution that need not be known; and a local move is one that either adds a single new feature to the existing subset of features, or drops a feature from the existing subset. We are now immediately confronted with two issues that make it impractical to use the procedure as shown. First, we have the same difficulty that prevented us from using an enumerative technique like a Gibb's sampler: generating the local neighbourhood requires us to obtain all possible single-feature additions. Second, for each local move, we need to construct a model, which can often be computationally expensive. We address each of these in turn.

2.2 Reducing Local Moves Using Theory-Guided Sampling

In this section, we consider a modification of the search procedure in Fig. 5 that results in only examining a small sample of all the local moves possible before deciding on the next move in Step 3(f)i. Ideally, we are interested in obtaining a sample that, with high probability, contains the best local move possible. Assuming there are no ties, and that the number of possible local moves is very large, it would clearly be undesirable to select the sample using a uniform distribution over local moves. We propose instead a selection that uses the errors made by the model-constructor to obtain a sample of local moves. As a result, features in the local neighbourhood that are relevant to the errors are more likely to be selected. In some sense, this is somewhat reminiscent of boosting methods: here, instead of increasing the weights of examples incorrectly classified, the representation language is enriched in a way that is biased to classify these examples correctly on subsequent iterations.

Recall that at any point, a local move from a feature-subset F is obtained by either dropping an existing feature in F or adding a new feature to F. We are specifically concerned with the addition step, since in principle, all possible features that can be constructed by the ILP engine could be considered as candidates. We curtail this in the following ways. First, we restrict ourselves to samples of features that are relevant to examples misclassifed by the model-constructor using the current set of features F (by "relevant" we mean those that are TRUE for at least one of the examples in error). Second, in our implementation, we restrict ourselves to a single new feature for each such example (by "new", we mean a feature not already in F).[2] The altered procedure is in Fig. 6.

2.3 Reducing Models Constructed Using Feature Weights

Step 3(f)v although now considering fewer moves, still has to construct a model for each move before deciding on the best one. For some model-construction

[2] In the implementation, we select this feature using its discriminatory power given the original set of examples.

1. bestfeatures:= {}
2. bestaccuracy:= 0.0
3. for i = 1 to R do begin
   (a) currentfeatures:= randomly selected set of features
   (b) currentmodel:= model constructed with currentfeatures
   (c) accuracy:= estimated accuracy of currentmodel
   (d) if accuracy > bestaccuracy then begin
      i. bestfeatures:= currentfeatures
      ii. bestaccuracy:= accuracy
   (e) end
   (f) for j = 1 to M do begin
      i. F^-:= set of feature subsets obtained by dropping a feature from currentfeatures
      ii. F_{new}:= sample of new features that are TRUE for errors made by currentmodel
      iii. F^+:= set of feature subsets obtained by adding a feature in F_{new} to currentfeatures
      iv. localmoves:= $F^- \cup F^+$
      v. nextfeatures:= best subset in localmoves
      vi. nextmodel:= model constructed with nextfeatures
      vii. accuracy:= estimated accuracy of nextmodel
      viii. if accuracy > bestaccuracy then begin
         A. bestfeatures:= nextfeatures
         B. bestaccuracy:= accuracy
      ix. end
      x. currentfeatures:= nextfeatures
      xi. currentmodel:= nextmodel
   (g) end
4. end
5. return bestfeatures

Fig. 6. The randomised local search procedure for feature construction, modified using theory-guided sampling of local moves

methods, this may also be computationally too expensive. We examine the possibility of using feature weights to reduce the computational burden further.

The principal purpose of constructing and evaluating models in the local neighbourhood is to decide on the best next move to make. This will necessarily involve either an addition of a new feature to the existing set of features, or the deletion of an existing feature from the current set of features. That is, we are looking to find the best new feature to add, or the worst old feature to drop (given the other features in the set, of course). Correctly, we would form models with each old feature omitted in turn from the current set and each new feature added in turn to the current set. The best model would then determine the next move. Using a model-constructor that assigns weights to features allows us to adopt the following sub-optimal procedure instead. First, we find the feature with the lowest weight in the current model: this is taken to be the worst old feature. Next, we construct a single model with all features (old and new). Let us call this "extended model". The best new feature is taken to be the new feature with the highest weight in the extended model. The procedure in Fig. 6 with this further modification is shown in Fig. 7. It is evident that the number of additional models constructed at any point in the search space is now reduced to just 3: the price we pay is that we are not guaranteed to obtain the same result as actually performing the individual additions and deletions of features.

It is the randomised local search procedure in Fig. 7, with one small difference, that we implement and evaluate empirically in this paper. The difference arises from the comparison of models: in the procedure shown, this is always done using

1. bestfeatures:=
2. bestaccuracy:= 0.0
3. for i = 1 to R do begin
   (a) currentfeatures:= randomly selected set of features
   (b) currentmodel:= model constructed with currentfeatures
   (c) accuracy:= estimated accuracy of currentmodel
   (d) if accuracy > bestaccuracy then begin
      i. bestfeatures:= currentfeatures
      ii. bestaccuracy:= accuracy
   (e) end
   (f) for j = 1 to M do begin
      i. F_{new}:= sample of new features that are TRUE for errors made by currentmodel
      ii. extendedmodel:= model constructed using currentfeatures and F_{new}
      iii. f_{worst}:= feature in currentfeatures with lowest weight in currentmodel
      iv. f_{best}:= feature in F_{new} with highest weight in extendedmodel
      v. F^-:= set with feature subset obtained by dropping f_{worst} from currentfeatures
      vi. F^+:= set with feature subset obtained by adding f_{best} to currentfeatures
      vii. localmoves:= $F^- \cup F^+$
      viii. nextfeatures:= best subset in localmoves
      ix. nextmodel:= model constructed with nextfeatures
      x. accuracy:= estimated accuracy of nextmodel
      xi. if accuracy > bestaccuracy then begin
         A. bestfeatures:= nextfeatures
         B. bestaccuracy:= accuracy
      xii. end
      xiii. currentfeatures:= nextfeatures
      xiv. currentmodel:= nextmodel
   (g) end
4. end
5. return bestfeatures

Fig. 7. The randomised local search procedure for feature construction, modified using theory-guided sampling of local moves and the use of feature-weights to reduce model construction

estimated accuracies only. In our implementation, if estimated accuracies for a pair of models are identical, then the model using fewer features is prefered (that is, comparisons are done on the pair (A, F) where A is the estimated accuracy of the model and F is the number of features used in the model).

One final point is worth clarifying. This concerns how the procedure in Fig. 7 is to avoid over-fitting the data. There are three principal ways in which we see this can be achieved: (1) Estimated accuracies of the model constructed, if unbiased, should give the procedure a way of halting before over-fitting; (2) The feature-constructor—here an ILP learner—can ensure that features have some minimal support (in the data mining sense of the word); and (3) The model constructor can perform some appropriate trade-off of fit-versus-complexity to avoid over-fitting.

3 Empirical Evaluation

Our objective is to evaluate empirically the effectiveness of the approach based on randomised search for feature construction. Specifically, we intend to compare the performance of following two kinds of feature-construction techniques on word-sense disambiguation problems in language processing, for which ILP-based feature construction methods have been previously studied in the literature [2]:

1. *Features constructed prior to model construction.* An ILP system first constructs all interesting features that are consistent with the constraints provided by the background knowledge. Models are then constructed using these features. Any feature selection that may be necessary is done prior to model construction.
2. *Features constructed using randomised local search and theory-guided sampling.* This is the procedure we have described in this paper (specifically, the one in Fig. 7). Models are constructed using the same model construction procedure as above, but no separate step of feature-selection is done.

3.1 Materials

Data and Background Knowledge. We use two benchmark data sets for Word Sense Disambiguation (WSD) problem introduced in [2] to investigate the effectiveness of our proposed method. WSD is an important problem that needs to be solved in many natural language tasks such as machine translation, information retrieval, speech and text processing and so on. Although complete descriptions of the data are available in [2], we describe them briefly here for completeness.

Monolingual task. This data consist of the 32 verbs from the SENSEVAL-3 competition [7]. SENSEVAL[3] is a joint evaluation effort for WSD and related tasks. We use all the verbs of the English lexical sample task from the competition. The number of examples for each verb varies from 40 to 398 (average of 186). The number of senses varies from 3 to 12 with an average of 7 senses. The average accuracy of the majority class is about 55%. The benchmark identifies 66% of the data that can be used for model construction. The rest are used for testing models.

Bilingual task. This task consists of 7 highly ambiguous verbs in machine translation from English to Portuguese. The sample corpus comprises around 200 English sentences for each verb extracted from a corpus of fiction books. In that corpus, the number of translations varies from 5 to 17, with an average of 11 translations. The average accuracy of the majority class is about 54%.

In [2], 9 categories of background predicates were introduced. Of these, the category B0 consists of some simple hand-crafted features. The background predicates specifically of relevance to an ILP-based feature constructor are in categories B1–B8. These consist of: (B1) the local context of the verb in a sentence; (B2) lemmas of 5 content words to the right and left of the verb, lemmatized by MINIPAR [9]; (B3) part-of-speech tags of 5 content words to the right and left of the verb (obtained using MXPOST [10]); (B4) subject and object syntactic relations with respect to the verb, obtained from MINIPAR parses; (B5) collocations with respect to the verb, of the form: 1st preposition to the right, 1st

[3] www.senseval.org

and 2nd words to the left and right, 1st noun, 1st adjective, and 1st verb to the left and right; (B6) verb restrictions, in terms of the semantic features of their arguments in the sentence, extracted using LDOCE [11]; (B7) dictionary definitions, being a relative count of the overlapping words in dictionary definitions of each of the possible translations of the verb and the words; and (B8) phrasal verbs possibly occurring in a sentence.

Fig. 8 tabulates the number of facts in each of B1–B8 for the two disambiguation tasks.

Background	Monolingual Task	Bilingual Task
B1	61175	4244
B2	33969	0
B3	33969	4870
B4	6523	1598
B5	66092	10514
B6	1308	1922
B7	1591	150
B8	0	418
All	204627	23716

Fig. 8. Ground facts comprising definitions for B1–B8. B8 does not appear in the monolingual task, since SENSEVAL-3 data do not consider senses of the verbs occurring in phrasal verbs. B2 was not included in the bilingual task because it was considered to be covered by B5, since we had shorter sentences than in the monolingual task.

Algorithms. We distinguish here between three implementations that, together, comprise the randomised local search approach: (1) The implementation of the procedure in Fig. 7). This is implemented here in the Prolog language, with some associated executable files; (2) The ILP implementation used to construct features using B1–B8. We use the ILP system Aleph [6]; and (3) The model constructor. We use a linear SVM: the specific implementation used is the one provided in the WEKA toolbox called SMO.[4]

3.2 Method

Our method is straightforward:

> For each verb in each task (that is, 32 verbs in the monolingual task and 7 verbs in the bilingual task):

[4] `http://www.cs.waikato.ac.nz/~ml/weka/`. We obtain weights for features by using the support vector machine with linear kernel. Specifically, let $x_i = [x_{i1}, x_{i2}, \ldots, x_{in}]$ represent an n-dimensional input vector. In the case of linear kernel, the predictor function can be described as: $prediction(x_i) = sgn[b + w^T x_i]$, where $w = \sum_i \alpha_i x_i$ and b is the interceptor. The linear classifier categorizes a new data point x to the positive class if linear combination $w_1x_1+w_2x_2+\ldots, w_nx_n$ is above a given threshold and to the negative class if the linear combination is below the threshold. We use the absolute value $|w_j|$ as the weight of the j^{th} feature. These weights have also been used for the feature selection problem in [13].

1. Construct a set of good features using the ILP engine. Select a subset of these as being relevant for the prediction task. Construct a model using the feature-subset selected. Call this the "model with pre-specified features" (or PreSpec model).
2. Identify a set of features using the randomised local-search procedure in Fig. 7. Obtain a model using these features. Call this the "model with features using randomised search" (or RandSearch model).
3. Compare the performance of the PreSpec model against the performance of the RandSearch model.

The following details are relevant:

(a) We use the same "training-test" splits of the data as those used in [2]. Performance will be measured by the accuracy of prediction on the test set (that is, the percentage of test examples whose sense is predicted correctly).
(b) The ILP learner constructs a set of definite clauses in line with the usual specifications for predictive ILP systems (see [14] for a statement of these). Positive examples for the ILP learner are provided by the correct sense (or translation in the bilingual case) of verbs in a sentence. Negative examples are generated automatically using all other senses (or translations). The translation of a clause into a Boolean feature is straightforward: the feature corresponding to a definite clause is a function that returns the value "true" for an example if and only if the body of the definite clause is true, given the background knowledge. The ILP engine is constrained to return only a single feature that discriminates best between the positive and negative examples. On each iteration of Fig. 7, the ILP engine is forced to construct features that are true for errors made by the current model.
(c) Although we are able to generate PreSpec models using the randomised procedure in Fig. 7 (by simply setting $R = 1$ and $M = 0$), we will take the models reported in [2] to denote the PreSpec models. We also consider a slight variation by performing some amount of experimentation to determine (locally) optimal settings for two parameters: the C parameter used by the linear SVM and F, the number of features to be selected. We determine the appropriate setting by systematic variation of the C parameter over the range 0.0001 to 10000 in multiple of 10s and F over the same range examined in [2]. The predictive accuracy with each (C, F) setting is estimated and the values that yield the best results are used to construct the final model (the predictive accuracy estimate is obtained using an average over 5 repeats of predictions on 20% of the training data sampled to form a "validation" set). We refer to these models as PreSpec* models.
(d) For RandSearch models, we use a value of $R = 10$ and $M = 5$ for the number of random restarts and iterations of local moves. We are not able to offer domain-independent guidelines on values for these parameters. In the context of using a similar approach in search for clauses [15], short periods of local search with many random restarts were found to be effective. While the values we have selected here are arbitrary, some more principled approach

may be possible by a systematic empirical exploration of reasonable values. We also consider a variant that performs no local search, and returns a set of features, of the same size as those obtained by RandSearch, but constructed randomly. We call this procedure "Random". Both cases require a "start point", which is provided by features obtained from (repeated, if $R > 1$) random samples of 30 examples.

(e) Comparison of performance will be done using the Wilcoxon signed-rank test. The test is a non-parametric test of the null hypothesis that there is no significant difference between the median performance of a pair of algorithms. The test works by ranking the absolute value of the differences observed in performance of the pair of algorithms. Ties are discarded and the ranks are then given signs depending on whether the performance of the first algorithm is higher or lower than that of the second. If the null hypothesis holds, the sum of the signed ranks should be approximately 0. The probabilities of observing the actual signed rank sum can be obtained by an exact calculation (if the number of entries is less than 10), or by using a normal approximation. We are interested in all cases in the the directional hypothesis that the performance of RandSearch models are better than those of PreSpec models.

4 Results and Discussion

Fig. 9 and 10 tabulate the comparative performance of the PreSpec, PreSpec*, Random and RandSearch models on the two disambiguation tasks. Also included is the performance of a classifier that simply predicts the most frequent sense of the verb (as assessed on the training set). The average number of features used in each case is in Fig. 11. We note at the outset that there appears to be little to choose between "Random" and "Majority Class", and between "PreSpec" and "PreSpec*". Therefore, in what follows, we will not include any further discussion of either Random or PreSpec*. With this caveat, the principal details in the tabulations are these: (1) Majority Class performs worse than the other procedures; (2) For both tasks, the accuracies of the PreSpec models are usually lower than the RandSearch models. Discarding ties, the PreSpec has the highest accuracy for 10 of the 32 verbs for the monolingual task and for 1 of the 7 verbs for the bilingual task; and (3) RandSearch models use substantially fewer features than PreSpec models.

We turn now to the question of whether the differences in accuracies observed between the models are in fact significant. The probabilities calculated by using the Wilcoxon test are shown in Fig. 12.[5] The tabulations further support the position that both PreSpec and RandSearch models are significantly better than a simple majority class guesser. RandSearch is significantly better on the bilingual task, and there is reasonable evidence to believe that it is better on the

[5] These were obtained from the program kindly provided by Richard Lowry at `http://faculty.vassar.edu/lowry/wilcoxon.html`.

Verb	Senses	Accuracy				
		Majority Class	PreSpec	PreSpec*	RandSearch	Random
activate	5	82.46 ±3.56	83.33 ±3.49	82.45 ±3.56	**92.98** ±2.39	82.46 ±3.56
add	6	45.80 ±4.35	82.44 ±3.32	**83.21** ±3.27	74.81 ±3.79	46.56 ±4.36
appear	3	44.70 ±4.33	71.21 ±3.94	71.21 ±3.94	**87.12** ±2.92	43.94 ±4.32
ask	6	27.78 ±3.99	50.00 ±4.45	53.17 ±4.44	**60.32** ±4.36	29.37 ±4.06
begin	4	59.74 ±5.59	**74.03** ±5.00	72.73 ±5.07	71.43 ±5.15	59.74 ±5.59
climb	5	55.22 ±6.08	83.58 ±4.53	**86.57** ±4.17	82.09 ±4.68	55.22 ±6.08
decide	4	67.74 ±5.94	77.42 ±5.31	**80.64** ±5.02	77.42 ±5.31	67.74 ±5.94
eat	7	**88.37** ±3.46	87.21 ±3.60	**88.37** ±3.46	**88.37** ±3.46	**88.37** ±3.46
encounter	4	50.77 ±6.20	72.31 ±5.55	72.30 ±5.55	**73.85** ±5.45	36.92 ±5.99
expect	3	74.36 ±4.94	**92.31** ±3.02	**92.31** ±3.02	89.74 ±3.44	74.36 ±4.94
express	4	69.09 ±6.23	**72.73** ±6.01	67.27 ±6.32	70.91 ±6.12	69.09 ±6.23
hear	7	46.88 ±8.82	**65.63** ±8.40	62.5 ±8.55	40.62 ±8.68	46.88 ±8.82
lose	9	52.78 ±8.32	**58.33** ±8.22	52.7 ±8.32	47.22 ±8.32	52.78 ±8.32
mean	7	52.50 ±7.90	70.00 ±7.25	**75.0** ±6.85	**75.00** ±6.85	52.50 ±7.90
miss	8	33.33 ±8.61	33.33 ±8.61	36.66 ±8.80	**56.67** ±9.05	33.33 ±8.61
note	3	38.81 ±5.95	**88.06** ±3.96	**88.06** ±3.96	82.09 ±4.68	56.72 ±6.05
operate	5	16.67 ±8.78	77.78 ±9.80	72.22 ±10.55	**88.89** ±7.41	38.89 ±11.49
play	12	46.15 ±6.91	53.85 ±6.91	**55.77** ±6.89	**55.77** ±6.89	46.15 ±6.91
produce	6	52.13 ±5.15	67.02 ±4.85	65.96 ±4.88	**77.66** ±4.30	52.13 ±5.15
provide	6	85.51 ±4.24	89.86 ±3.63	86.96 ±4.05	**91.30** ±3.39	82.61 ±4.56
receive	9	88.89 ±6.05	88.89 ±6.05	88.89 ±6.05	**92.59** ±5.04	88.89 ±6.05
remain	3	78.57 ±4.90	87.14 ±4.00	85.71 ±4.18	**95.71** ±2.42	78.57 ±4.90
rule	5	50.00 ±9.13	83.33 ±6.80	83.33 ±6.80	**90.00** ±5.48	40.00 ±8.94
smell	7	40.74 ±6.69	**77.78** ±5.66	75.92 ±5.82	74.07 ±5.96	40.74 ±6.69
suspend	7	35.94 ±6.00	57.81 ±6.17	56.25 ±6.20	**68.75** ±5.79	39.06 ±6.10
talk	9	72.60 ±5.22	73.97 ±5.14	73.97 ±5.14	**75.34** ±5.04	72.60 ±5.22
treat	9	28.07 ±5.95	47.37 ±6.61	**57.89** ±6.53	49.12 ±6.62	28.07 ±5.95
use	5	71.43 ±12.07	**92.86** ±6.88	**92.86** ±6.88	**92.86** ±6.88	71.43 ±12.07
wash	12	67.65 ±8.02	**73.53** ±7.57	61.76 ±8.33	58.82 ±8.44	67.65 ±8.02
watch	7	**74.51** ±6.10	**74.51** ±6.10	72.54 ±6.24	**74.51** ±6.10	**74.51** ±6.10
win	7	44.74 ±8.07	60.53 ±7.93	57.89 ±8.00	**63.16** ±7.83	42.11 ±8.01
write	8	26.09 ±9.16	34.78 ±9.93	39.13 ±10.17	**52.17** ±10.42	34.78 ±9.93
Mean	7	55.31	71.97	71.63	74.10	56.06
Median	6	52.31	74.03	72.63	74.90	52.31

Fig. 9. Estimates of accuracies of disambiguation models on the monolingual task. "Senses" refers to the numbers of possible senses of each verb. The column labelled "Majority class" gives the accuracy of models that simply predict the most common sense of each verb (as estimated from the training data).

Verb	Translations	Accuracy				
		Majority class	PreSpec	PreSpec*	RandSearch	Random
come	11	50.30±7.62	76.74±6.44	75.56 ±6.41	**86.67** ±5.07	55.56 ±7.41
get	17	21.00±6.70	40.54±8.07	**64.1** ±7.68	51.28 ±8.00	20.51 ±6.47
give	5	88.80±4.81	95.35±3.21	**100** ±0.00	97.78 ±2.20	97.78 ±2.20
go	11	68.50±6.78	78.72±5.97	77.55 ±5.96	**83.67** ±5.28	71.43 ±6.45
look	7	50.30±7.45	82.22±5.70	76.59 ±6.17	**82.98** ±5.48	59.57 ±7.16
make	11	70.00±7.25	75.00±6.85	78.57 ±6.33	**80.95** ±6.06	73.81 ±6.78
take	13	28.50±8.24	**60.00**±8.94	56.25 ±8.77	56.25 ±8.77	12.50 ±5.85
Mean	11	53.91	72.65	75.51	77.08	55.88
Median	11	50.30	76.74	76.60	82.98	59.57

Fig. 10. Estimates of accuracies of disambiguation models on the bilingual task. "Translations" refers to the numbers of possible translations of each verb into Portuguese.

Model	Avg. Features Used	
	Monolingual	Bilingual
Majority class	0	0
PreSpec	250	500
RandSearch	29	27

Fig. 11. Average numbers of features required to construct models. The values for PreSpec are those reported in [2] after feature-selection was performed.

	Majority class	PreSpec	RandSearch
Majority class	–	–	–
PreSpec	< 0.001, 0.010	–	–
RandSearch	< 0.001, 0.010	0.08, 0.05	–

Fig. 12. Probablities of observing the differences in accuracies for the monolingual and bilingual tasks, under the null hypothesis that median accuracies of the pair of algorithms being compared are equal. Each entry consists of a pair of probability estimates, corresponding to the mono and bilingual tasks. The alternate hypothesis is each case is that the column-model performance is better than the row-model performance. Thus a value of 0.08 in the (RandSearch,PreSpec) entry means that there is an 8% chance that the performance of the two models are actually the same, given the performance observed on the monolingual task.

monolingual one as well (with the usual caveat that probabilities from repeated cross-comparisons should be interpreted with caution).

For the monolingual task, we are also able to add to the comparisons reported in [2], based on average accuracies for each verb. This is shown in Fig. 13. The algorithms in this table use model construction methods that include bagged decision trees (the Syntalex family), Naive Bayes (CLaC1), maximum entropy modelling (CLaC2), a multiclass perceptron (MC-WSD), and ILP. It is evident that the RandSearch models are comparable to the state-of-the-art in the field.

Models	Accuracy
Majority class	55.31
Syntalex-1	67.00
Syntalex-2	66.50
Syntalex-3	67.60
Syntalex-4	65.30
CLaC1	67.00
CLaC2	66.00
MC-WSD	72.50
ILP	69.15
ILP-assisted	71.97
RandSearch	74.10

Fig. 13. Comparative average accuracies of the best models reported for the SENSEVAL-3 competition. All rows except the last are as in [2]. There "ILP" refers to using an ILP engine to construct rules for disambiguation (as opposed to using a feature-based learner); and "ILP-assisted" is what we have called PreSpec here.

The superior performance of RandSearch here suggests that adequate models for the WSD tasks can be constructed with fairly small numbers of features. We note however that this may not always be the case: depending on the background knowledge available to the ILP learner, adequate models may require very large numbers of features. In this case, we believe PreSpec-like models may perform better. The distinction between PreSpec and RandSearch models is somewhat illusory: we can clearly obtain the former using the latter with an R value of 1 and an M value of 0. A related question is whether the local search is beneficial at all. We have some evidence for this: on the 32 verbs in the mono-lingual task, not performing a local search yielded a better model only on 6 verbs (there were 10 ties, and local search gave better models on the remaining 16 verbs).

5 Concluding Remarks

In this paper we investigate the applicability of using a randomised search technique for feature construction using ILP. The search method we propose can be seen as a gneralisation of some of the existing approaches to using ILP to extend a feature-based representation. A direct implementation of randomised approach is, however, computationally expensive, and we have described a number of additional modifications for practical use. Chief amongst these is the use of the errors made by a model constructed using an existing set of features to select amongst future moves in the search. Results from an empirical evaluation on some standard datasets in language processing are promising: we find predictive accuracies of models constructed are usually higher and use substantially fewer features. In cases where we are able to compare against the state-of-the-art, we find average prediction accuracies are higher than those reported earlier.

There are a number of ways in which the work here can be improved and extended. We list the main limitations here under three categories. On the theoretical front, it is evident that we have not provided any guarantees of optimality on the feature-subset constructed. While this is typical of randomised methods of the type proposed here, it would nevertheless be useful to obtain some performance bounds, however loose.

On the implementation front, our implementation is based on the simplest kind of randomised search (GSAT). Better methods exist and need to be investigated (for example, WalkSat). Further, we could consider other neighbourhood definitions for the local search such as adding or dropping upto k features. Of course, we are not restricted to use SVMs, or even the specific variant of SVM here, as our model constructor. Our experiments on the monolingual task here suggest that there is no significant difference between a "1-norm" SVM and the standard approach we have used here, but other model construction techniques may yield better results.

On the application front, we have evaluated our search procedure on a specific set of problems (namely, those concerned with word-sense disambiguation); and against a specific kind of feature-construction (in which all features are constructed before model construction). Clearly, testing on other datasets is

desirable. We also need to extend the comparative study to include methods like kFOIL [4] that perform "dynamic propositionalisation" (that is, generate features incrementally).

These limitations notwithstanding, we believe there is sufficient evidence to believe that the randomised approach used here could provide an interesting way to interleave the construction of features and their associated models. We note that when used in conjunction with a statistical model constructor (as we have done here), we are effectively performing a form of statistical relational learning. This aspect of the search-based approach needs to be investigated further.

Acknowledgements

The authors would like to thank Lucia Specia for generously providing the data used in the tasks here. Jimmy Foulds, at the University of Waikato, helped beyond the call of duty in getting an executable version of code that constructed models using a 1-norm SVM within WEKA.

References

1. Lavrac, N., Dzeroski, S., Grobelnik, M.: Learning nonrecursive definitions of relations with LINUS. Technical report, Jozef Stefan Institute (1990)
2. Specia, L., Srinivasan, A., Ramakrishnan, G., Nunes, G.V.: Word Sense Disambiguation Using Inductive Logic Programming. In: Inductive Logic Programming, pp. 409–423. Springer, Heidelberg (2007)
3. Zelezny, F.: Efficient Construction of Relational Features. In: Proceedings of the 4th Int. Conf. on Machine Learning and Applications, pp. 259–264. IEEE Computer Society Press, Los Angeles (2005)
4. Landwehr, N., Passerini, A., Raedt, L.D., Frasconi, P.: kFOIL: Learning Simple Relational Kernels. In: Gil, Y., Mooney, R. (eds.) Proc. Twenty-First National Conference on Artificial Intelligence (AAAI 2006) (2006)
5. Davis, J., Ong, I., Struyf, J., Burnside, E., Page, D., Costa, V.S.: Change of representation for statistical relational learning. In: Proc. IJCAI 2007 (2007)
6. Srinivasan, A.: The Aleph Manual (1999), `http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/`
7. Mihalcea, R., Chklovski, T., Kilgariff, A.: The SENSEVAL-3 English Lexical Sample Task. In: SENSEVAL-3: Third International Workshop on the Evaluation of Systems for Semantic Analysis of Text, Barcelona, pp. 25–28 (2004)
8. Specia, L., Nunes, M.G.V., Stevenson, M.: Exploiting Parallel Texts to Produce a Multilingual Sense-tagged Corpus for Word Sense Disambiguation. In: RANLP 2005, Borovets, pp. 525–531 (2005)
9. Lin, D.: Principle based parsing without overgeneration. In: 31st Annual Meeting of the Association for Computational Linguistics, Columbus, pp. 112–120 (1993)
10. Ratnaparkhi, A.: A Maximum Entropy Part-Of-Speech Tagger. In: Empirical Methods in NLP Conference, University of Pennsylvania (1996)
11. Procter, P.(ed.): Longman Dictionary of Contemporary English. Longman Group, Essex (1978)

12. Parker, J., Stahel, Password, M.: English Dictionary for Speakers of Portuguese. Martins Fontes, São Paulo (1998)
13. Brank, J., Grobelnik, M., Milic-Frayling, N., Mladenic, D.: Feature Selection Using Linear Support Vector Machines Technical report, Micorsoft Research, MSR-TR-2002-63
14. Muggleton, S., De Raedt, L.: Inductive Logic Programming: Theory and Methods. J. Log. Program 19/20, 629–679 (1994)
15. Zelezny, F., Srinivasan, A., Page Jr., C.D.: Randomised restarted search in ILP. Machine Learning 64(1-3), 183–208 (2006)
16. Dzeroski, S., Muggleton, S., Russell, S.J.: PAC-Learnability of Determinate Logic Programs. In: COLT 1992, pp. 128–135 (1992)
17. King, R.D., Karwath, A., Clare, A., Dehaspe, L.: Genome scale prediction of protein functional class from sequence using data mining. In: KDD 2000, pp. 384–389 (2000)
18. Ramakrishnan, G., Joshi, S., Balakrishnan, S., Srinivasan, A.: Using ILP to Construct Features for Information Extraction from Semi-structured Text. In: Blockeel, H., Ramon, J., Shavlik, J., Tadepalli, P. (eds.) ILP 2007. LNCS (LNAI), vol. 4894, pp. 211–224. Springer, Heidelberg (2008)
19. Srinivasan, A., King, R.D.: Feature construction with Inductive Logic Programming: a study of quantitative predictions of biological activity aided by structural attributes. Data Mining and Knowledge Discovery 3(1), 37–57 (1999)
20. Kohavi, R., John, G.H.: Wrappers for Feature Subset Selection. Artif. Intell. 97(1-2), 273–324 (1997)
21. Selman, B., Levesque, H.J., Mitchell, D.G.: A New Method for Solving Hard Satisfiability Problems. In: AAAI 1992, pp. 440–446 (1992)