Journal of Artificial Intelligence Research 1 (2015) 3-17 Submitted 6/91; published 9/91

Feature Generation by Recursive Induction

Lior Friedman LIORF@CS.TECHNION.AC.IL
Shaul Markovitch SHAULM@CS.TECHNION.AC.IL
Technion-Israel Institute of Technology
Haifa 32000, Israel

Abstract

Induction algorithms have steadily improved over the years, resulting in powerful methods for learning. However, these methods are constrained to use knowledge within the supplied feature vectors. Recently, a large collection of common-sense and domain specific relational knowledge bases have become available on the web. The natural question is how these knowledge bases can be exploited by existing induction algorithms. In this work we propose a novel algorithm for using relational data to generate recursive features. Given a feature, the algorithm recursively defines a new learning task over its set of values, and uses the relational data to construct feature vectors for the new task. The resulting classifier is then added as a new feature. We have applied our algorithm to the domain of text categorization, using large semantic knowledge bases such as Freebase. We have shown that generated recursive features significantly improve the performance of existing induction algorithms.

1. Introduction

In recent decades, we have seen an increasing prevalence of machine learning techniques used in a wide variety of fields such as medical diagnosis, vision, and biology. Most machine learning methods assume a given set of labelled examples, represented by a set of pre-defined features. These methods have proven to be successful when a collection of good, distinguishing features is available. In many real-world applications, however, the given set of features is not sufficient for inducing a high quality classifier.

One approach for overcoming the difficulty resulting from an insufficiently expressive set of features, is to generate new features. There are many existing feature generation algorithms, most of which operating by combining existing features in order to produce new ones. The LFC algorithm (?) combines the original features using logical operators such as $\wedge, \neg$. LFC generates features using the current set of binary features (including generated ones), and requires that any new feature will yield an increase over the original features combined to create it, in terms of information gain. The LDMT algorithm (?) uses linear combinations of the original features to construct more informative ones. This allows for the use of non-binary features in feature generation approaches. The FICUS algorithm (?) presents a general framework for using any set of constructors to combine features. This essentially generalizes the above approach, allowing for a more generalized feature generation framework.

These methods all provide us with ways to enhance the performance of induction algorithms through intelligent combinations of existing features. While this often suffices, there are many cases where merely combining existing features is not sufficient. For this reason, newer approaches aim to incorporate additional knowledge from external sources in order to construct new and informative features. ? (?) ,for example, present a method for generating features that are based on Wikipedia concepts. They created a mapping of words to Wikipedia articles, that serve as semantic concepts, then utilized the distance of words in the given text to those concepts as features. This approach produced positive results, especially in domains where data is sparse, such as text categorization on short texts. ? (?) presents a method for utilizing lexical links between words to generate features. They made use of Roget's Thesaurus as a resource, allowing them to better map words and phrases to their lexical meanings.

In recent years, a new resource in the form of Semantic Linked Data has begun to take form, as part of the Semantic Web project (see survey, ?). Semantic Linked Data contains type-annotated entities covering a multitude of domains and connected using multiple semantically meaningful relations. This resource has led to the creation of several new approaches designed to utilize the new, ontology-based representation of knowledge (?, ?). It should come as no surprise then, that there have been several efforts in utilizing Linked Data for unsupervised feature generation (?, ?). ? (?) devise a theoretical framework for constructing features from linked data. By specifying entity types relevant to the problem at hand, they restrict the space of possible features to a more manageable size, and allow for the creation of a reasonably small amount of features. They also note that this approach tends to lead to highly sparse feature vectors. ? (?) developed FeGeLOD, an automated, fully unsupervised framework that constructs features by using entity recognition techniques to locate semantically meaningful features in the data set, and expand upon those entities using relations within the Semantic Web. They then use feature selection techniques to remove features with a large percentage of missing, identical, or unique[1] values.

Existing feature generation approaches based on Linked Data can offer great benefits, as they can add useful type information, and often add semantic information such as actors playing in a given movie, or the population of a given city. However, existing approaches are unsupervised, and we would also like to be capable of identifying more complex relationships between entities.

In this work, we present a new methodology for generating complex relational features. Our algorithm constructs new learning problems from existing feature values, using relational data, such as the Semantic Web, as the features for the newly constructed problem. Using common induction algorithms, we can then construct a classifier that serves as a feature for the original problem. An important aspect of this approach is that it can be applied recursively within the new learning problem, allowing for an automated, data-driven, exploration of the large space of possible complex features that are difficult to discover using existing feature-generation methods.

1.1 Illustrative Example

Before we delve into the exact description of our algorithm, we would like to illustrate its main ideas using an illustrative example. Suppose we are attempting to identify people with a high risk to be suffering from a certain genetic disease. Assume that the target concept to be discovered is that those at risk are women with ancestors originating from desert areas. To do so, we are given a training sample of sick and healthy people, containing various features, including gender and their full name. Assuming we have no additional information, a base classifier which can be achieved is the one in figure 1. While such a classifier will achieve a low training error, the hundreds of seemingly unrelated surnames will cause it to generalize very poorly.

However, if we assume that we have access to a relational knowledge base connecting surnames to common countries of origin, we can begin to apply our feature generation technique to the prob-

1. Values which only appear in a single example

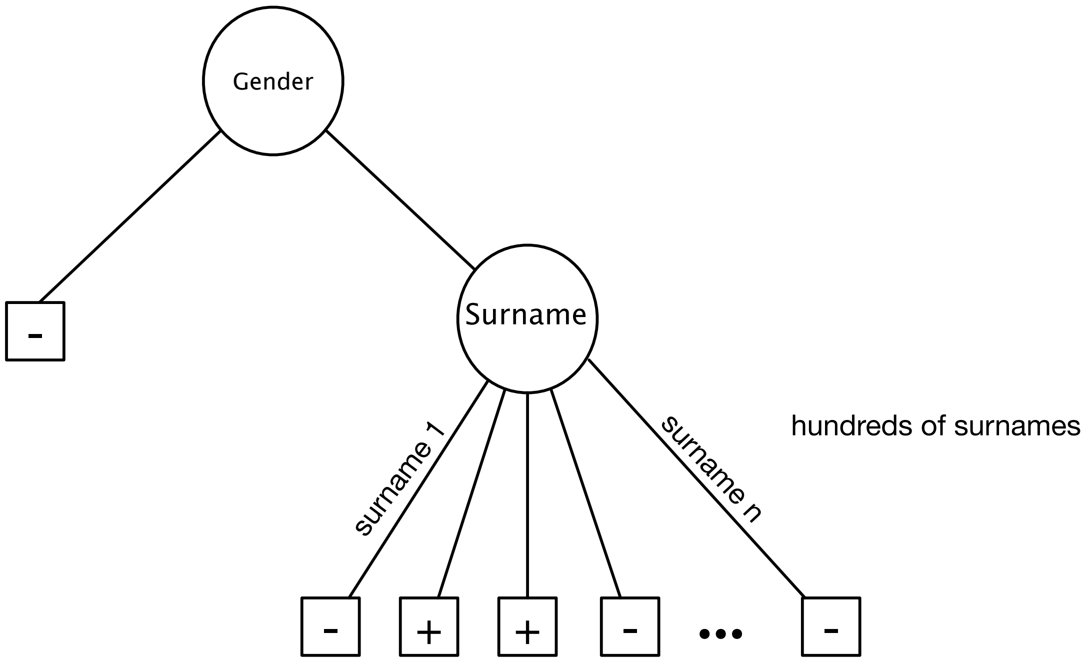

Figure 1: A decision tree for the basic features

lem. In the original problem, the node of female contains objects of type person. Our goal is to separate this set of people to those at high risk and those at low risk. Our recursive method defines a new learning problem with the following training set: The objects are surnames; surnames of people with the disease are labelled as positive. The features for these objects are extracted from the knowledge base.

Solving the above learning problem through an induction algorithm yields a classifier on surnames. This classifier can be used as a binary feature in the original problem. For example, it can be used as a feature in the node of female in the node of 1, yielding the tree seen in figure 2. This new feature gives us a better generalization, as we now abstract the long list of surnames to a short list of countries. This also allows us to capture previously unseen surnames from those countries. However, this is not a sufficient solution, as we have no way of generalizing on previously unseen countries of origin.

Suppose then, that we also have access to a knowledge base of facts about countries, such as average temperature, precipitation, and more (one such knowledge base is DBpedia). Using such a knowledge base, we can recursively apply our method while trying to learn the new problem. We create a new training set, the objects of which are countries of origin, and countries of surnames of people with the disease are labelled as positive. Note that this is a second level of recursion. This training set is given to an induction algorithm using the relational knowledge base about countries to construct features. The result is a classifier that tries to separate between countries of origin of people with the disease and those without the disease. The classifier is used as a feature by the first level recursive algorithm. The whole process is depicted in figure 3. The resulting two-

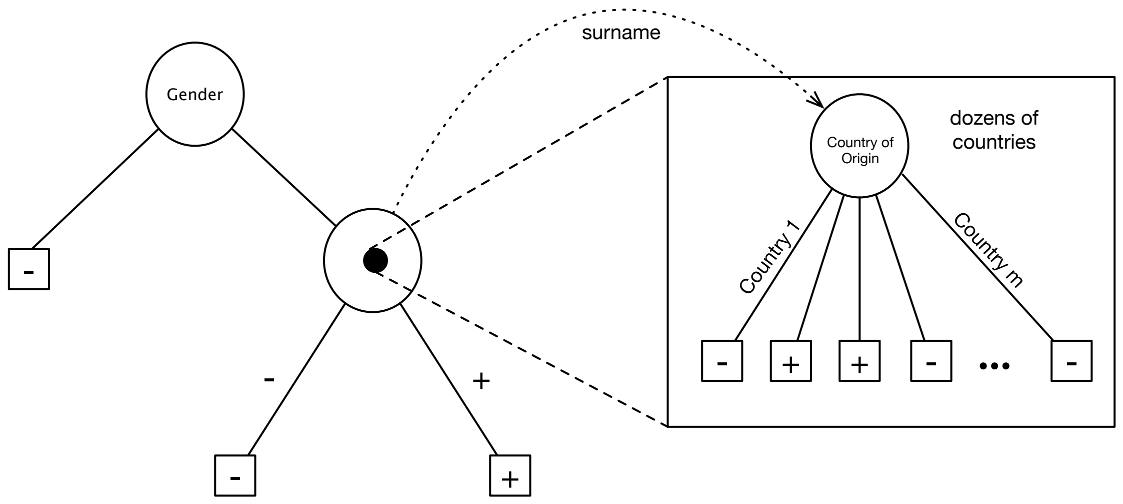

Figure 2: A constructed feature used within a decision tree

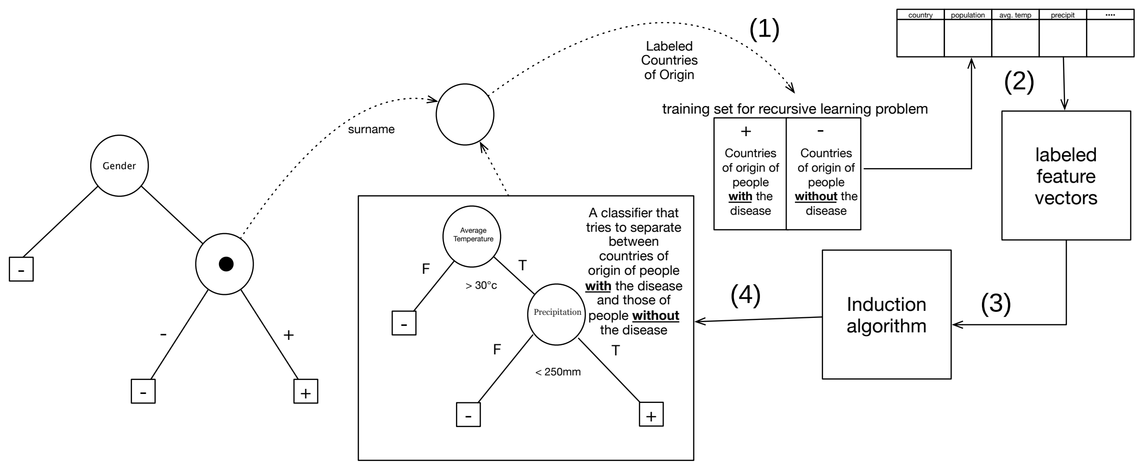
(1)
Labeled Countries of Origin
country
population
avg. temp
precipit
····
(2)
surname
training set for recursive learning problem
+
-
Countries of origin of people with the disease
Countries of origin of people without the disease
labeled feature vectors
Gender
-
-
+
-
+
Average Temperature
F
T
> 30°c
-
Precipitation
F
T
< 250mm
-
+
A classifier that tries to separate between countries of origin of people with the disease and those of people without the disease
(4)
Induction algorithm
(3)

Figure 3: Recursive construction of a learning problem on countries of origin. (1)-Creating the objects for the new problem. (2)-Creating features using the knowledge base. (3)-Applying an induction algorithm. (4)-The resulting feature.

level recursive classifier is depicted in figure 4. This constructed feature allows us to concisely and accurately capture the target concept.

2. Generating Features through Recursive Induction

Let O be a set of objects. Let $Y = \{0, 1\}$ be a set of labels (we assume binary labels for ease of discussion). Let $S = \{(o_1, y_1), \ldots, (o_m, y_m)\}$ be a set of labeled examples such that $o_i \in O, y_i \in Y$. Let $F = \{f_1, \ldots, f_n\}$ be a *feature map*, a set of *feature functions* $f_i : O \to Dom_i$. This

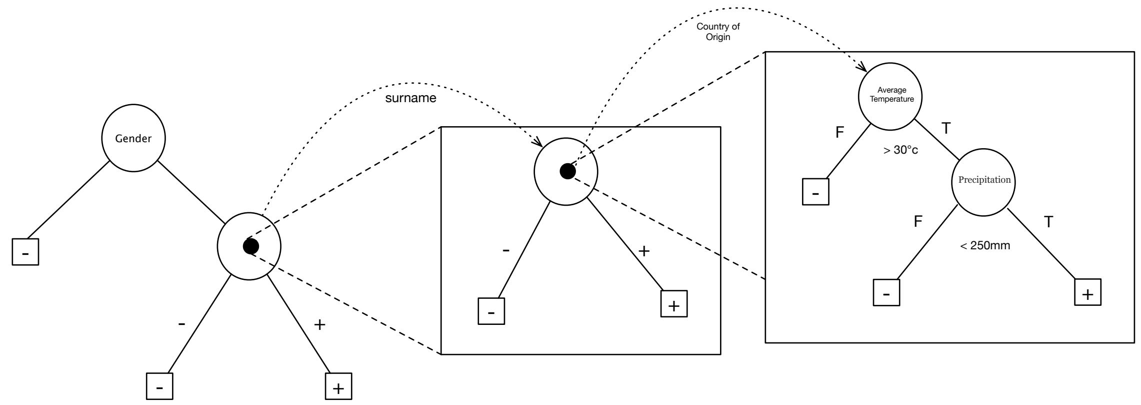

Figure 4: A two-level constructed feature used within a decision tree

definition implies a training set represented by feature vectors: $\{(\langle f_1(o_i), \ldots, f_n(o_i)\rangle, y_i) | (o_i, y_i) \in S\}$.

Given a set of relations $\mathcal{R} = \{R_1, \ldots, R_t\}$ with arity of n_j ($j = 1 \ldots t$), we can assume w.l.o.g that the first argument is a key. For each relation R_j we define $n_j - 1$ new binary relations where each of the first elements is the key belonging to set K_j and the second one is another column from some domain (not necessarily one represented within the existing feature functions). Let $\bar{R} = \{R_1, \ldots, R_k\}$ be such set of binary relations, where R_i is defined over $K_i \times Dom_i$. These relations can thus be seen as functions $R_i : K_i \to Dom_i$.

Definition 2.1. A *supervised feature generation algorithm A* using relations is an algorithm that given $\langle S, F, \mathcal{R}\rangle$, creates a new feature map $F_{\mathcal{R}} = \{f'_1, \ldots, f'_l\}$.

We would like the new hypothesis space, defined over $F_{\mathcal{R}}$, to be one that is both rich enough to provide hypotheses with a lower loss than those in the original space, as well as simple enough that the learning algorithm used will be able to find such good hypotheses given training data.

2.1 Generating a feature

Given an original feature $f_i : O \to Dom_i$, our algorithm will formulate a new learning task trying to separate values in Dom_i appearing in positive examples of the original learning task from those appearing in negative ones. The result of the new learning task will be a classifier $h_i : Dom_i \to \{0, 1\}$ that can label feature values of f_i. We can then define a new binary feature $f'_i(x) = h_i(f_i(x))$. We name this algorithm *FEAGURE* (FEAture Generation Using REcursive induction). You can see the full pseudo-code in appendix A.

In order to explain how we create such h_i, let us consider a single step of the feature generation algorithm. Given a feature f_i, we define the set of all its value in the training set S as $v_i(S) = \{v | (o, y) \in S, f_i(o) = v\}$. In the intro example, for instance, $v_i(S)$ will be the set of all last names of patients. We now formulate a new learning problem with the new training set $\hat{S}_i = \{(v, label(v)) | v \in v_i(S)\}$. The labelling function can be, for example, defined as the majority label: $label(v) = majority(\{y_k | (o_k, y_k) \in S, f_i(o_k) = v\})$.

To define a feature map over the new training set $\hat{S}_i$, we look for all relations in $\bar{R}$ where the domain of the key argument contains v_i: $\mathcal{G}(S, \bar{R}) = \big\{r \in \bar{R} | v_i(S) \subseteq \{x_1 | (x_1, x_2) \in r\}\big\}$. In the intro example, one such r can be a relation mapping last names to countries of origin. We then use $\mathcal{G}$ as a feature map for $\hat{S}_i$. Solving this new learning problem on $\hat{S}_i, \mathcal{G}(S, \bar{R})$ yields our classifier h_i. Note that during the process of learning h_i, we can once again call the feature generation procedure to generate useful features for the *new* task, hence the recursive aspect of the process. We can see this in the intro example, wherein we construct a second-level problem on countries of origin to correctly identify the appropriate conditions that signify high risk countries.

Algorithm 1 Creating a recursive problem

```
function CREATENEWPROBLEM($f_i$, $S$, $\bar{R}$)
    $v_i(S) = \{v|(o, y) \in S, f_i(o) = v\}$
    $\hat{S}_i = \{(v, label(v))|v \in v_i(S)\}$        ▷ $label(v)$ can be, for example, the majority function.
    $F_{new} = \{r(v) = c|c \in Range(r), r \in \mathcal{G}(S, \bar{R})\}$
        ▷ These are features from $Dom_i$ to $\{0, 1\}$. For each relevant relation, we use every possible value.
return $\hat{S}_i, F_{new}$        ▷ Our new training set and features define a learning problem.
end function
```

We note that in order to prevent cycles which provide no new information, we do not allow the use of any relation that is the inverse of a previously used relation. This is not a major concern, as it is often the case that even relations using similar sets of keys will have often have differing targets (so even if $R_i : K \to Dom_i, R_j : K \to Dom_j$, usually Dom_i and Dom_j will be dissimilar).

2.2 Finding Locally improving features

When performing feature generation, we often wish to evaluate the predictive power of generated features. When such evaluation is done in the context of the entire dataset, it is possible to lose the unique benefit of a potentially powerful feature. To address this phenomenon, we decided to apply our recursive feature generation algorithm in a divide-and-conquer approach through the use of a decision tree, thus allowing us to better identify features with a strong local impact which may be difficult to see in a more global context. It is important to note, however, that the features generated in this process can be used for any induction algorithm, not necessarily decision tree induction algorithms. Using decision trees for this divide-and-conquer approach for feature generation offers several advantages:

- Since we repeat this process within inner nodes of the tree, we allow for the discovery of locally relevant relationships which may be difficult to pick up when looking at the entirety of the data. We use pre-pruning based on minimal leaf size to avoid creating features which are poorly supported by actual training data and thus may overfit to the training set.
- Since a decision tree classifier tests features one at a time, we can easily isolate the impact of a specific generated feature.
- We gain an innate way to evaluate and filter generated features: A feature generated in a node will be added to the final result only if it is no less informative than any existing feature for that node.
- Since we use generated features in the constructed tree, we can ensure that features generated afterwards within the tree will be largely orthogonal to it, thus ensuring variety in the generated features.

- Decision trees are relatively interpretable, which leads to features which are easier to understand.

Naturally, we use decision trees as the classifiers for the newly constructed learning problems as well, allowing us to easily repeat the process recursively.

Finally, the features generated during the decision tree training procedure are collected and returned for use as general, black-box features which can be used along with any induction algorithm. The decision tree itself is discarded.

Algorithm 2 Using a decision tree for feature generation and evaluation

```
minLeafSize- minimal size of a node in the decision tree required to apply our method.
SplitByFeature- a method that splits a training set according to a feature.
  function ConstructFeaturesInANode($S$, $F$, $\bar{R}$, minLeafSize)
      f, ig= bestIGRatio($S$,$F$)
      if $|S|$ <minLeafSize then                         ▷ pre-pruning to avoid poor features
          return SplitByFeature($S$, f), $\emptyset$
      end if
      for $f' \in F$ do
          newS, newF= CreateNewProblem($S$,$F$,$\bar{R}$)
          createdFeature= InductionAlgorithm($S$,$F$)     ▷ We can apply our algorithm recursively here
          newFeatureIg= igRatio($S$,createdFeature)
          if newFeatureIg $\geq$ ig then
              add createdFeature to feature pool $F$
          end if
      end for
      fSplit= bestIGRatio($S$,$F$)                                      ▷ Including new features
      let newFeatures be the set of all features added to $F$
      return SplitByFeature($S$, fSplit), newFeatures
  end function
  function ConstructFeatures($S$, $F$, $\bar{R}$, minLeafSize)
      currentFeatures= $\emptyset$
      sons, newFeatures=ConstructFeaturesInANode($S$, $F$, $\bar{R}$, minLeafSize)
      currentFeatures= currentFeatures$\cup$newFeatures
      for son in sons do                                 ▷ Recursive decision tree construction
          newFeatures=ConstructFeatures(son.$S$,son.$F$,$\bar{R}$, minLeafSize)
          currentFeatures= currentFeatures$\cup$newFeatures
      end for
      return currentFeatures
  end function
```

2.3 Applications for Text Categorization

In order to apply our feature generation technique to the domain of text categorization, we make the standard assumption that our basic features are the words that appear in the texts, as in the bag-of-words (?, ?) approach. We can therefore take words which also exist as entities within the relational database as feature values v_i for the sake of our approach. To successfully do so, we naturally turn to *Named Entity Recognition* (NER) approaches in order to better locate such entitles within the text.

The resulting features will locate sets of words which are connected in the relational domain. This allows us to better generalize compared to the standard bag-of-words method, and find potentially complex semantic relationships, especially when using Semantic Linked Data as our relational database.

An important side effect of this usage is that, since each label in the semantic data defines a relation and texts may contain words from different contexts, there are often multiple relations which apply and thus $\mathcal{G}(S, \bar{R})$ is often large. For example, a text referring to William Shakespeare's play of "Hamlet" may connect to relations on people (for William Shakespeare himself), on theatre plays (for "Hamlet"), and possibly locations (for the country of Denmark, where the play is set). In order to make the process more computationally tractable, especially if it is performed recursively, we choose a sub-sample of $\mathcal{G}(S, \bar{R})$ for which we construct new learning problems. For the initial level of recursive activation we do so uniformly. In later levels we perform a weighted sampling on applicable relations in $\bar{R}$ based on the mean information-gain ratio of features within the current level: We take all features in current level for some relation $r \in \bar{R}$, measure the mean information-gain ratio, and use this as the weight for picking r as a target for recursive activation of the FEAGURE algorithm.

A second relevant difference to note is that generated feature may be non-applicable to a specific object, as it may be the case that none of the entities within the text come from the relation's key set [2]. Therefore, we decided to have generated features yield a third value for cases where the generated feature does not apply.

A final difference is that it may be the case that multiple words in the text are entities of the same domain [3]. In such cases, a constructed feature may return differing values for different entities within the text. To best resolve this, in cases where a feature is applicable to multiple unique words within the text, we return a majority vote of the classification returned by the feature as applied to each such word, as seen in figure 5

2.4 Discussion and Analysis of the FEAGURE algorithm

We note that the process of re-labeling and constructing recursive problems as detalied in section 2.1 offers some unique contributions:

- By moving our domain space when constructing a new problem, we essentially look at the problem from another perspective, which allows for the discovery of complex relationships.
- The process of re-labeling allows noise reduction and emphasizes more general trends within the data that may be harder to otherwise notice.
- We can exploit the power of existing, well-developed learning algorithms when we create a classifier, and possibly use different ones as we take recursive steps.

Furthermore, we note that FEAGURE can locate locally useful features which may be difficult to identify when looking at the dataset as a whole.

The resulting features of the FEAGURE algorithm

In terms of runtime performance, while FEAGURE has a large overhead, we note two major things: Firstly, the critical section of re-labeling and constructing recursive problems can be parallelized for different recursive problems within a decision tree node. Secondly, we note that existing

2. In fact, it is possible that the text contains no entities at all
3. For example, a text mentioning multiple countries

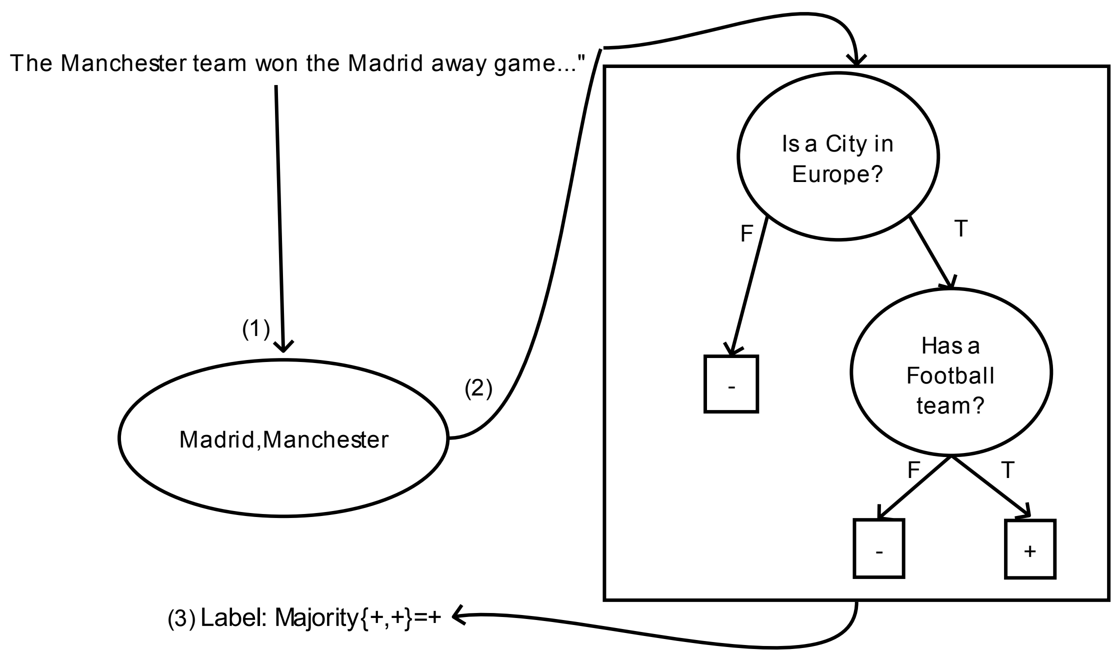
The Manchester team won the Madrid away game..."
(1)
Madrid,Manchester
(2)
Is a City in Europe?
F
T
-
Has a Football team?
F
T
-
+
(3) Label: Majority{+,+}=+

Figure 5: Using a constructed feature in text: (1)-Relevant entities are extracted. (2)-Each entity is sent to the constructed feature. (3)-A majority vote between responses is performed.

approaches such as the ones described by ? (?) and ? (?), as well as propositionalization approaches perform a chain of join operations ($\bowtie$) between existing feature values and external knowledge bases, which require similar overheads in terms of performance.

3. Empirical Evaluation

In this section, we discuss our experimental methodology and display our main results.

3.1 Methodology

As a knowledge base, we used YAGO2 (?), omitting any relations with literal data such as dates or geographic coordinates. This served as a "common knowledge" database. We also made limited use of type information within the database, opting to simply use it as an additional relation. In addition, we stripped semantic information regarding entities, such as "TV series" or "fictional character". For each relation within YAGO2, we created both the relation itself as well as the inverse relation, and then removed inverse relations that map very few values to very many, such as the inverse of the "has gender" relation, which maps "male" and "female" to all to all existing people within the YAGO database.

We used the Stanford Named Entity Recognizer (?) to allow entity matching. We then performed stopword elimination using NLTK (?) , removed punctuation marks and performed stemming using the Porter Stemmer (?).

We ran our feature generation algorithm for both a depth of one, creating a recursive learning problem for the original problem, and a depth of two, creating recursive learning problems for the original problem and the first order recursive problem.

We compared the new set of features (our new features combined with the original features) to the binary bag-of-words features at various feature selection levels. We used information gain ratio as the feature selection criterion.

3.2 Results

Figure 6 shows the accuracies achieved on all 100 datasets, using a SVM (?) classifier (Linear kernel, $C = 100$), comparing the accuracy of the baseline using only the original features to a combined pool of original and generated features (with a depth limit of two). In both cases, feature selection was performed, leaving only 20% of the features. The $y = x$ line represents a case where there was no change in performance, and any point above that line represents a dataset upon which we have attained an improvement. In addition, we show lines representing a 5% and 10% difference in accuracy. Results for a feature selection level of 10% are similar, and are also statistically significant $(p < 0.05)$. Results for comparing depth one and the baseline approach show similar results and were thus omitted for the sake of brevity.

We see that the results are quite positive, with many more datasets showing an advantage to the algorithm using generated features. this advantage is also statistically significant improvement $(p < 0.005)$ using a two-tailed paired t-test.

One can not ignore there are many datasets where there was no change in performance. A closer look at these datasets reveals that the cause for this is that no features were generated. Further analysis reveals that this occurs due to a large number of highly sparse recursive learning problems. Since we only sample some of the possible problems to attempt feature generation, this is a factor in the amount of features generated in general, as well as part of the reason why no features are generated for some datasets. Experiments in lowering the amount of sampled relations show a decrease in both the number of generated features and the mean increase in accuracy.

In order to test the versatility of our approach, we also tried to look at a very different type of classifier, and therefore used a K-Nearest Neighbours (?) classifier $(K = 1)$. As before, we performed feature selection, leaving only 20% of the features. The results are shown in figure 7. We see a similar trend to that of SVM, albeit at a lower base accuracy level. These results also show a statistically significant improvement $(p < 0.01)$ using a two-tailed paired t-test.

4. Related Work

One of the earliest methods of utilizing relational information is *Inductive Logic Programming(ILP)* (?, ?), which induces a set of first-order formulae that define a good separation of the given training set. Following this work, Relational Learning - techniques designed to utilize relational databases, have become increasingly prevalent. One such technique is that of View Learning (?), which generated new tables from existing ones, effectively performing feature generation for relational methods. Unlike View Learning and other Relational Learning methods, our approach constructs a new learning problem in a new domain using the existing problem as a guide. Due to this distinction, we essentially try to re-think about our problem from a new direction, rather than trying to fit increasingly less connected entities to the original problem. Furthermore, the ability of our approach to locate locally beneficial features which may be difficult to otherwise detect is invaluable. Finally, we

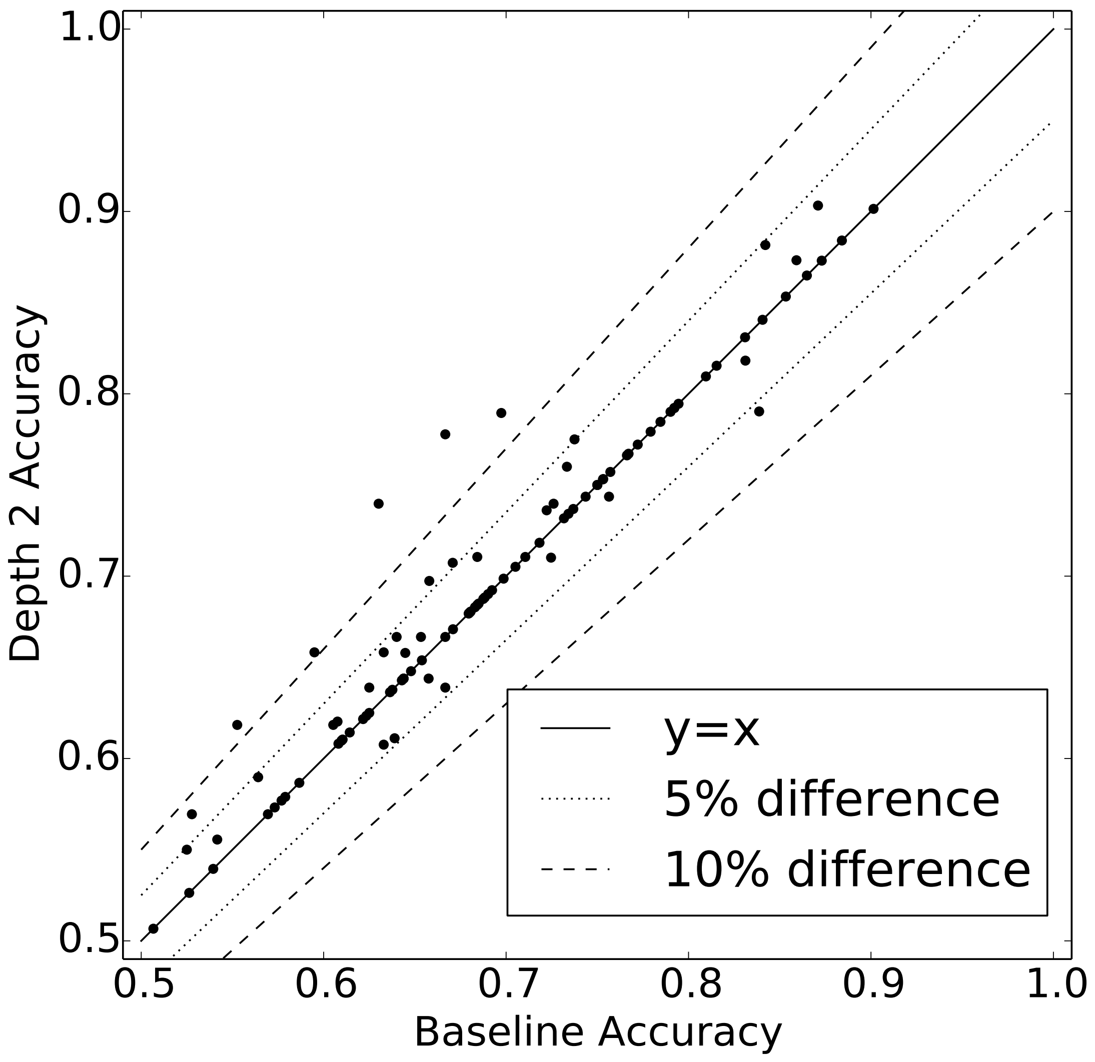

Figure 6: Accuracy (using SVM) of baseline approach compared to the depth two feature generation approach. Each point represents a dataset. The dotted lines represent a 5 and 10 percent difference in accuracy

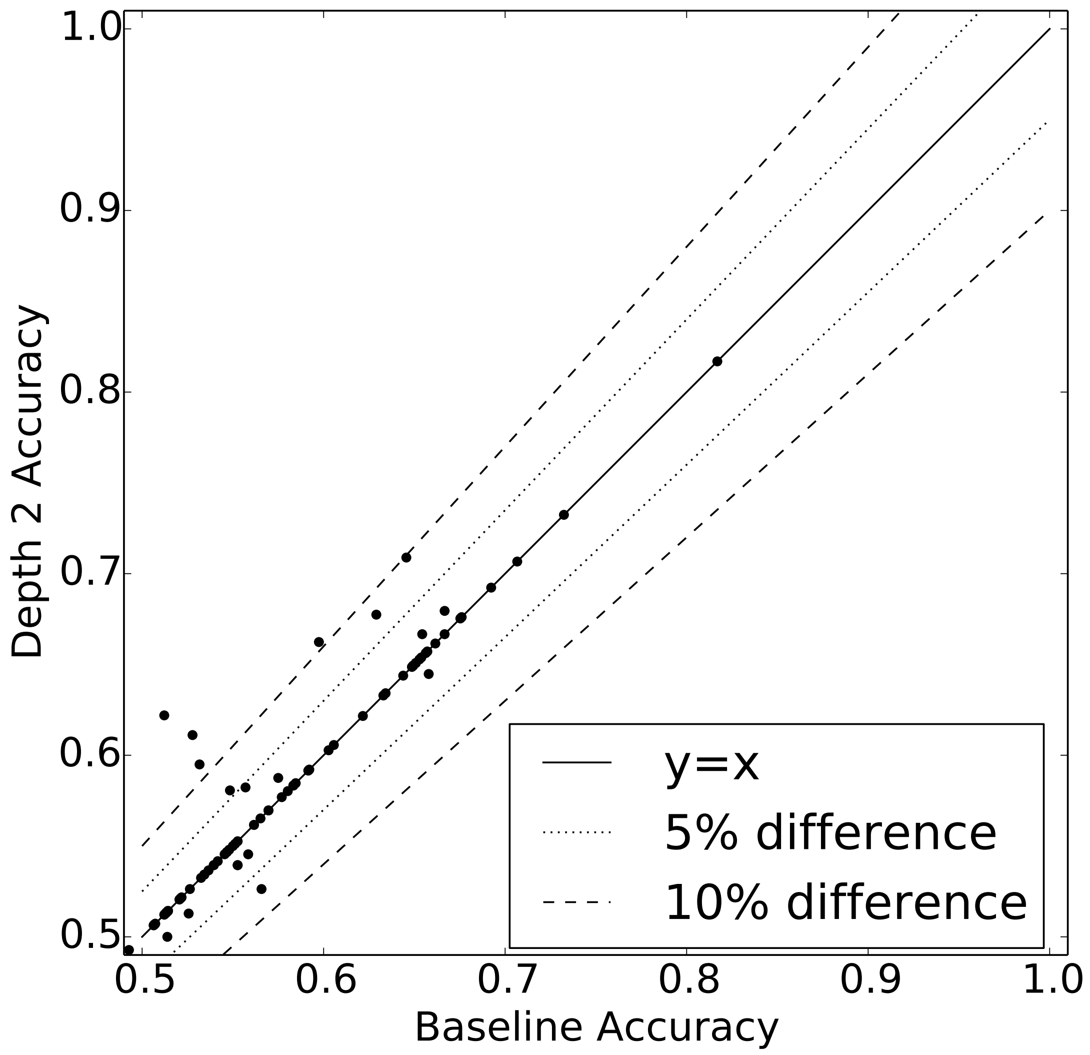

Figure 7: Accuracy (using KNN) of baseline approach compared to the depth two feature generation approach. Each point represents a dataset

note that the use of decision trees during the feature generation process allows for a natural method by which to filter out features even before applying traditional feature selection mechanisms.

One major attempt at adding relational knowledge to traditional induction algorithms was *propositionalization* (?), which allows an unsupervised creations of first-order predicates which can then be used as features for propositional methods. A major setback of this process is that it generates an impractically large number of features, most of which irrelevant. To this end, *upgrade* methods such as ICL (?) and SGLR (?) were suggested, where instead of creating predicates a-priori, feature generation is performed during the training phase, in a more structured manner. While upgrade approaches bear some similarities to our approach, there are several critical differences, the key of which is the ability to more easily locate complex features through the use of existing induction algorithms.

Recently, there has been a strong trend of utilizing *Deep Learning* (?, ?) as a feature generation technique. Good examples of this can be seen in FEX (?) and DBN2 (?). These methods essentially form combinations and transformations on pre-defined features in a semi-supervised manner, thus yielding new, more predictive features. Our approach can be seen as a variation on the same concept, with the transformations being the construction of recursive learning problems, and the combinations being the classifiers constructed in the new problem domain. Our approach differs from those rooted in Deep Learning in both the use of relational data to enrich the feature space, and in that we allow more complex combinations in general.

5. Conclusions

We presented a novel new approach to feature generation using relational data, based on constructing and solving new learning problems in the relational domain. Our feature generation algorithm constructs powerful and complex features, and can identify locally useful features as well as general trends. These features can be used along with any traditional machine learning algorithm. While we focused on applying this approach to text categorization problems, it is important to note that it is applicable to any classification problem where feature values are categorical and have semantic meanings, such as drug names, cities and so on. Although there is no simple way to generalize this approach to numeric features, we believe there are numerous domains where feature values do have semantic meaning, even excluding text-based domains. While our approach only generates binary features, aggregation based techniques such as those used in SGLR apply here as well. We furthermore note that when solving non-binary classification problems, we can construct categorical features with, at most, the same amount of values as there are labels in the dataset.

A major source for potential improvement is the subject of matching values to semantic objects. Given the major advances in the field of Wikification (?) in general and Entity Linking(?) in particular, these approaches can be used to link the initial text to entities within the semantic data without losing information, which may lead to better results, especially for text calssification problems.

Another major potential avenue for improvement is to use the techniques discussed section 2.1 as a way to directly label entities within the semantic graph, which yields a Collective Classification (?) problem in the semantic domain. Solving this learning problem in turn allows us to label any entity within the semantic net, which gives us a potential feature to apply to the original problem domain.

A. FEAGURE pseudocode

Algorithm 3 FEAGURE

$S = \{(o_i, y_i)\}$- set of labeled objects. We will mark Obs the objects and y the appropriate labels.
F- A set of feature-functions.
R- A set of relations.
d- Maximum depth.
min-leaf- Minimum Leaf Size. For clarity we assume this is constant for all recursive levels.

```
function FEAGURE(S, F, R, d, min-leaf)
    if $|S| \leq min - leaf$ then
        return Leaf                                        ▷ We pick features until the tree is too small
    end if
    nf, feature= RI-CHOOSEFEATURE(S, F, R, d, min-leaf)
    OUTPUT(nf)
    return SPLITTREEBYFEATURE(feature, S)
end function
function RI-CHOOSEFEATURE(S, F, R, d, min-leaf)
    if $d = 0$ then
        return CONSTRUCTCLASSIFIER(S,F)                    ▷ Can use any traiditional learning algorithm
    end if
    addedFeatures= $\{\}$
    while resources not exhausted, new relevant features exist do
        feature= PICKFEATURE(S,F)                          ▷ Weighted sampling based on information gain
        newObs= feature(Obs)
        newY= LABEL(S,newObs)                              ▷ In our example, by majority vote
        newF= $\{R_i(newObs)|R_i \in R, newObs \cap Dom_i \neq \emptyset\}$   ▷ A feature function per relevant relation
        S'= (newObs,newY)
        newClassifer= FEAGURE(S', newF, R, $d - 1$, min-leaf)
        f= function(x): return newClassifer(feature(x))
        if INFOGAINRATIO(f, S) $\geq$ BESTINFOGAINRATIO(F, S) then
            addedFeatures= addedFeatures$\cup\{f\}$        ▷ A feature passed our threshold, so we add it
        end if
    end while
    return addedFeatures, FINDBESTFEATUREBYINFOGAIN(F$\cup$addedF, S)
end function
```