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 used to create the new features. 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 (Ragavan, Rendell, Shaw, & Tessmer, 1993) 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 (Utgo & Brodley, 1991) 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 (Markovitch & Rosenstein, 2002) presents a general framework for using any set of constructors to combine features. This essentially generalizes the above approach, allowing for a customizable 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. Gabrilovich and Markovitch (2009) ,for example, present a method for generating features that are based on Wikipedia concepts. They create a mapping of words to Wikipedia articles, that serve as semantic concepts, then utilize 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. Jarmasz (2012) 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, Bizer, Heath, & Berners-Lee, 2009). 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 (Lösch, Bloehdorn, & Rettinger, 2012; Rios, Specia, Gelbukh, & Mitkov, 2014). It should come as no surprise then, that there have been several efforts in utilizing Linked Data for unsupervised feature generation (Cheng, Kasneci, Graepel, Stern, & Herbrich, 2011; Paulheim & Fümkranz, 2012). Cheng et al. (2011) 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. Paulheim and Fümkranz (2012) 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, the unsupervised nature of existing approaches limits us greatly when attempting to identify more complex relationships between entities.

In this work, we present a new supervised 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. This allows us to discover powerful features that an unsupervised approach would have difficulty discovering.

1.1 Illustrative Example

Before we delve into the exact description of our algorithm, we would like to showcase 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

1. Values which only appear in a single example

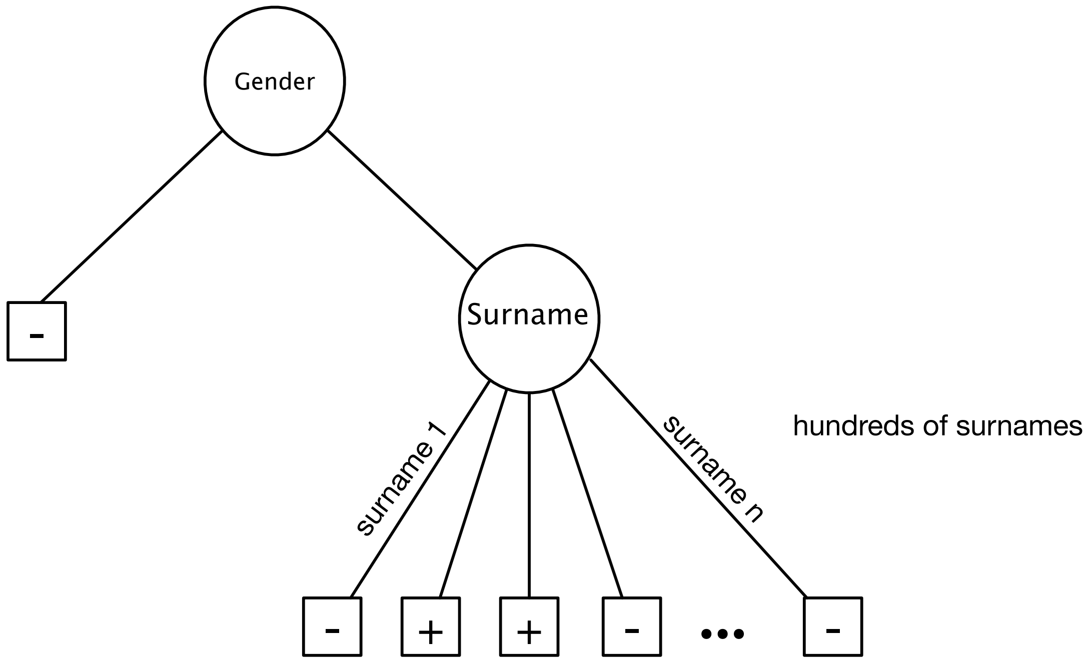


Figure 1: A decision tree for the basic features

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 problem. In the original problem, the node of value 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. Therefore, 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 (See section 2.1).

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 value female in figure 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 result 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 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

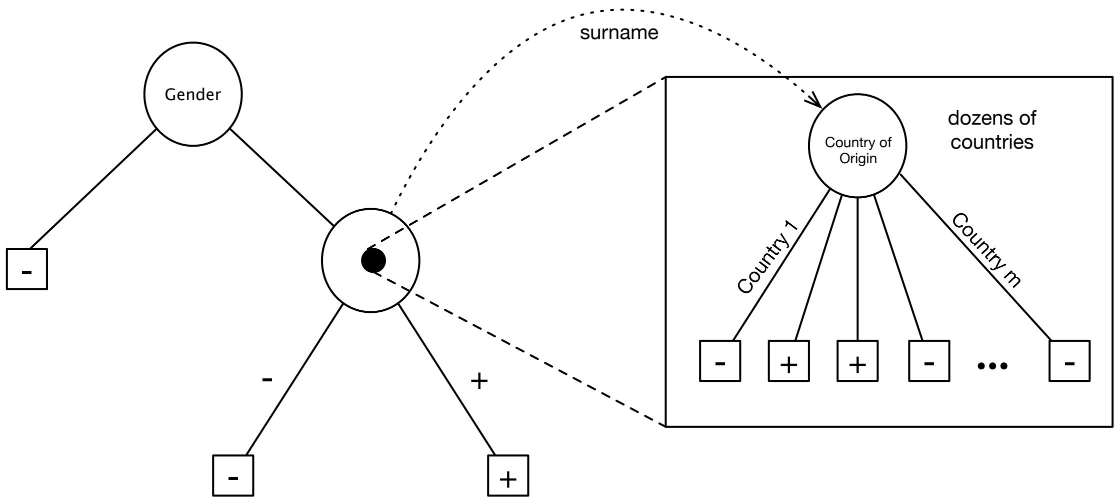


Figure 2: A constructed feature used within a decision tree

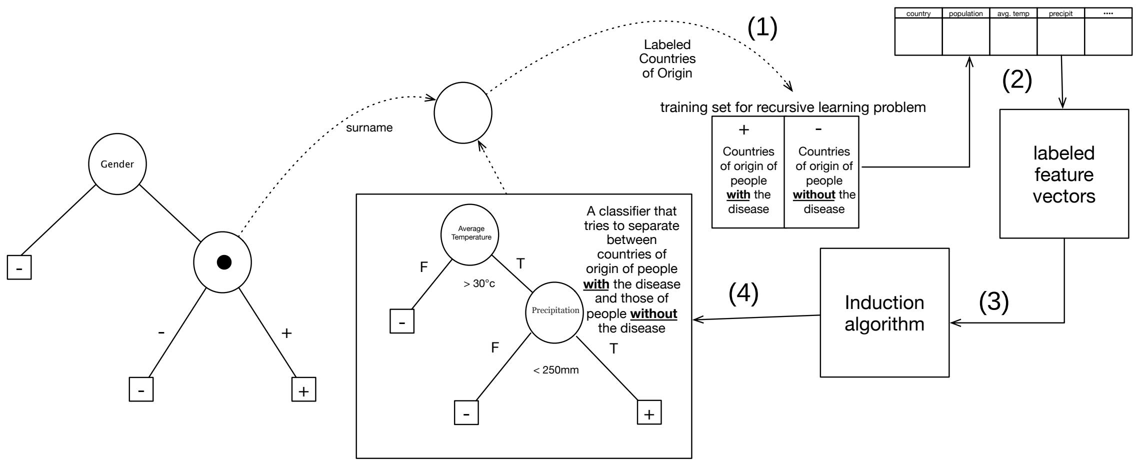

(1)
Labeled Countries of Origin
country
population
avg. temp
precipit
....
(2)
training set for recursive learning problem
+
-
Countries of origin of people with the disease
Countries of origin of people without the disease
labeled feature vectors
surname
Gender
-
-
+
-
+
Average Temperature
> 30°c
F
T
Precipitation
< 250mm
F
T
-
-
+
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.

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

While this is a simple example, it shows a case where additional information and complex features can result in an overall simpler and more general result. It is important to mention that an

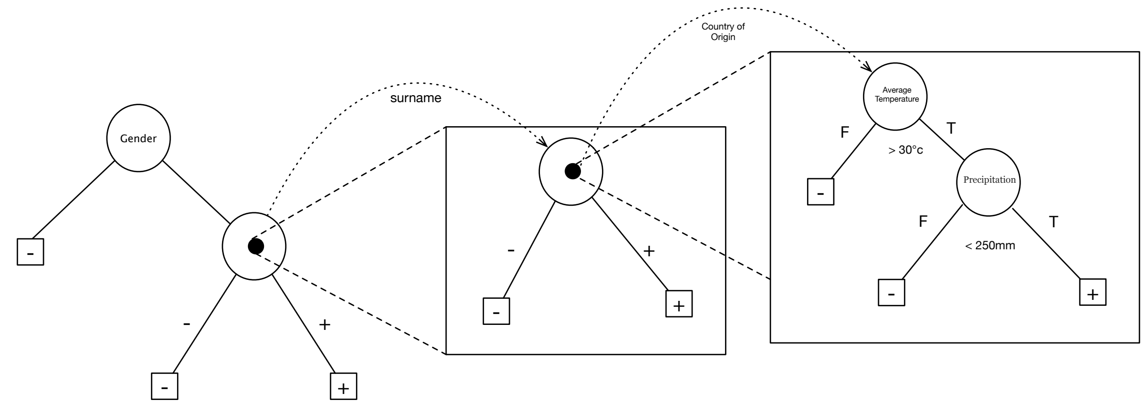


Figure 4: A two-level constructed feature used within a decision tree

unsupervised approach such as FeGeLOD could also solve this problem, but would be forced to generate more features, and utilize complex feature selection to then select relevant features.

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

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. 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 $R_i : K_i \to Dom_i$, and therefore we can treat $\mathcal{R}$ as a set of functions (possibly partial).

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 $h_i : Dom_i \to Y$ that can label feature values of f_i. We can then define a new feature $f'_i(x) = h_i(f_i(x)), f'_i : O \to Y$. We name this algorithm *FEAGURE* (FEAture Generation Using REcursive induction). You can see the full pseudo-code in appendix **??**.

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 $v_i(S) = \{v | (o, y) \in S, f_i(o) = v\}$ the set of feature values for f_i in S. In the intro example, for instance, $v_i(S)$ will be the set of all last names of patients that appeared in the training set. We now formulate a new learning problem with the new training set $S'_i = \{(v, label(v)) | v \in v_i(S)\}$. 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 S'_i, we look for all relations in $\mathcal{R}$ where the domain of the key argument contains v_i: $R(v_i, \mathcal{R}) = \{r \in \mathcal{R} | v_i(S) \subseteq \{x_1 | (x_1, x_2) \in r\}\}$. In the intro example, one such r can be a relation mapping last names to countries of origin. We then use $R(v_i, \mathcal{R})$ as our feature map. With this, we have defined a new learning problem over $S'_i, R(v_i, \mathcal{R})$, which yields our classifier $h_i : Dom_i \to Y$. Note that during the process of learning h_i, we can once again call the feature generation procedure to generate features for the *new* learning problem over S'_i, 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 for a single feature

```
function CREATENEWPROBLEM($f_i$, $S$, $\mathcal{R}$)
    $v_i(S) = \{v | (o, y) \in S, f_i(o) = v\}$
    $S'_i = \{(v, label(v)) | v \in v_i(S)\}$        ▷ $label(v)$ can be, for example, the majority function.
    $R(v_i, \mathcal{R}) = \{r \in \mathcal{R} | v_i(S) \subseteq \{x_1 | (x_1, x_2) \in r\}\}$
    $F_{new} = \{r(v) = c | c \in Range(r), r \in R(v_i, \mathcal{R})\}$
        ▷ These are features from $Dom_i$ to $\{0, 1\}$. For each relevant relation, we use every possible value.
return $S'_i, F_{new}$        ▷ Our new training set and features define a learning problem.
end function
```

We note that while this algorithm deterministically defines a learning problem for a single feature f_i, we can use a variety of known machine learning techniques to create h_i. We also note that we can re-apply our approach for this new problem, by simply picking $f_j \in F_{new}$ and applying algorithm 1.

2.2 From a Learning Problem to a Feature

Let us discuss how to create h_i from the new learning problem $\langle S'_i, F_{new}, R(v, \mathcal{R}) \rangle$. To begin with, we utilized a Decision Tree classifier (Quinlan, 1986) on the new learning problem. While training the decision tree classifier, we measure information gain for all $f \in F_{new}$, and we can also re-apply our approach as discussed above, creating new features to measure against features in F_{new}.

While most traditional machine learning algorithms yield useful properties such as filtering and regularization, a Decision Tree classifier offers several useful traits with regards to recursive application of our approach:

1. Decomposability: As we will discuss in section 2.3, a Decision Tree classifier is trivial to decompose into multiple Decision Stumps, allowing us to decide between generating a single strong feature or multiple weaker features.

2. Interpertability: Decision trees are generally considered easier to interpret and understand, allowing us to easily grasp which relations are more impactful, and thus which domains are likely to contain additional knowledge which we may not have utilized.

3. Orthogonality: Since features further down in a decision tree are orthogonal to ones already used, we can effectively limit our search space, as relations we have already used will not be picked. This also has a beneficial side effect of increasing variety in generated features.

2.3 Creating Multiple Features

Now that we have discussed how we create a single feature, we must consider how to expand this and generate multiple features. One simple approach would be to simply apply the above technique (algorithm 1) once for each feature. Naturally, this would result in n new features. Even if we apply the above approach recursively, we would still obtain only n new features, though they may be stronger as we have generated new features within each learning problem. Thus, if we wish to generate a larger number of features, we must take a different approach.

Assuming we have utilized a Decision Tree classifier to create h_i, we can decompose it to a set of Decision Stumps by taking each decision node in the tree and turning it into a node. By doing so, we can create between d and 2^d new features for each h_i (d being the depth of the decision tree). Still, each of these features is weaker than the original h_i, since h_i was chosen as a classifier that minimizes training error (subject to regularization) over S_i'.

2.4 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. In addition to the above, usage of a decision tree yields a second benefit - it allows us to generate features one at a time, creating a more ordered search. It is important to note that the features generated in this process can be later used for *any* induction algorithm.

First, we construct a decision tree node, go over the feature set F and evaluate as normal. We then apply algorithm 1 on all features in F, creating our new recursive features. Now, we compare between new and existing features, and pick the best one (We used Information Gain Ratio (Quinlan, 1986)). Next, we create son nodes based on feature values, and apply the process recursively until we reach a depth/node size limit in order to avoid over-fitting. 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 FEAGURE-FEAture Generation Using REcursive induction

minSize- minimal size of a node in the decision tree.
SplitByFeature- a method that splits a training set according to a feature.

```
function ConstructFeaturesInANode($S$, $F$, $\mathcal{R}$)
    f= bestFeature($S$,$F$)
    if $|S|$ <minSize then                                          ▷ avoids over-fitting
        return SplitByFeature($S$, f), $\emptyset$
    end if
    for $f_i \in F$ do
        $S'_i, F_{new}$= CreateNewProblem($f_i$,$S$,$\mathcal{R}$)     ▷ We can apply our algorithm recursively here
        $h_i$= InductionAlgorithm($S'_i$, $F_{new}$)
        if Compare($h_i, f$)$\geq 0$ then                           ▷ for example,information gain ratio
            add $h_i$ to feature pool $F$
        end if
    end for
    $f_{best}$=bestFeature($S$,$F \cup \{h_i\}$)
    let newFeatures be the set of all features added to $F$
    return SplitByFeature($S$, $f_{best}$), newFeatures
end function

function ConstructFeatures($S$, $F$, $\mathcal{R}$)
    currentFeatures= $\emptyset$
    sons, $F_{new}$=ConstructFeaturesInANode($S$, $F$, $\mathcal{R}$)
    currentFeatures= currentFeatures$\cup F_{new}$
    for son in sons do                                              ▷ Recursive decision tree training
        newFeatures=ConstructFeatures(son.$S$,$F$,$\mathcal{R}$)
        currentFeatures= currentFeatures$\cup$newFeatures
    end for
    return currentFeatures
end function
```

3. Applications for Text Categorization

The text Categorization problem is defined by a set of texts X labelled by a set of categories Y such that we create $S = \{(x_i, y_i)|x_i \in X, y_i \in Y\}$. Given S, The learning problem is to create a hypothesis $h : X \rightarrow Y$ which minimizes error (mistaken label). The standard approach to solving this problem is based on the bag-of-words (Wu & Salton, 1981; Salton & McGill, 1983) approach, Using the frequencies of word appearances within the text as features and thus learn h.

As discussed, we have seen the recent rise of Semantic Linked Data as a powerful knowledge base for text-based entities, with large databases such as Google Knowledge Graph (Pelikánová, 2014), Wikidata (Vrandečić & Krötzsch, 2014) and YAGO2 (Hoffart, Suchanek, Berberich, & Weikum, 2013). Naturally, we would like to make use of such semantic data to improve upon the bag-of-words approach, with the implicit assumption that additional Semantic knowledge will allow us to better approximate the relationship between the text and its given label.

Since semantic data utilizes entities, not words and texts as its underlying knowledge components, we must apply some approach which extracts entities from texts. There is a multitude of such methods, such as Named Entity Recognition (NER), Wikification (Bunescu & Pasca, 2006) and Entity Linking (Rao, McNamee, & Dredze, 2013). Using the above approaches, we can convert a text x into a set of entities $e(x)$.

3.1 FEAGURE for Text Categorization

Given a set of texts converted to entities, we can define $f_i(x) = E(x)$ the entities within the text. We then define $\mathcal{R}$ to be the set of relations within our knowledge base. Combining these definitions with algorithm 1 we get $v_i(S) = \cup_{x_i} E(x_i)$ the set of entities for all texts, $R(v_i, \mathcal{R})$ is the set of relations in our knowledge base that apply to all entities, and F_{new} would be a set of functions mapping entity keys to their values with regards to relation r, so $f_r(x) = \{c | \exists e \in E(x) : r(e) = c\}$.

We notice two important difference from the general case that must be discussed. Firstly, the suggested scheme only considers relations that apply to all entities contained within the text. While such relations exist (for example, the IS-A relation which describes entity type), most relations within a standard knowledge base do not apply to all entity types. To combat this, we allow the use of relations that apply only to some entities (based on domains), allowing us to use all relations within the knowledge base. However, as a result of this it is possible that $f_r(x) = \emptyset$, which will, in turn, mean that the resulting h created during feature generation will be inapplicable to x. As a result, h is a partial function $h : X \rightarrow \{0, 1\}$, and when using FEAGURE we will have three child nodes: One for value 1, one for value 0, and a third for value inapplicable.

The second major difference we must consider is the possibility of having multiple entities for which the resulting classifier applies, potentially yielding multiple classifications for x [2]. 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

4. Empirical Evaluation

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

4.1 Methodology

For evaluation, we used two datasets:

1. **TechTC-100** (Davidov, Gabrilovich, & Markovitch, 2004) - A collection of 100 different binary text categorization problems of varying difficulty, extracted from the Open Dictionary project. On each dataset, we used the Stanford Named Entity Recognizer (Finkel, Grenager, & Manning, 2005) for entity recognition. We then performed stopword elimination using NLTK (Bird, Klein, & Loper, 2009) and performed stemming using the Porter Stemmer (Van Rijsbergen, Robertson, & Porter, 1980). As our knowledge base, we used **YAGO2** (Hoffart et al., 2013), omitting any relations with literal data such as dates or geographic coordinates. For each relation within YAGO2, we created both the relation itself as well as the inverse relation, and then removed reverse relations that map very few values to very many [3].

2. For example, a text mentioning multiple countries, each classified differently by h
3. For example, the inverse of the “has gender” relation maps “male” and “female” to all to all existing people within the YAGO2 database.

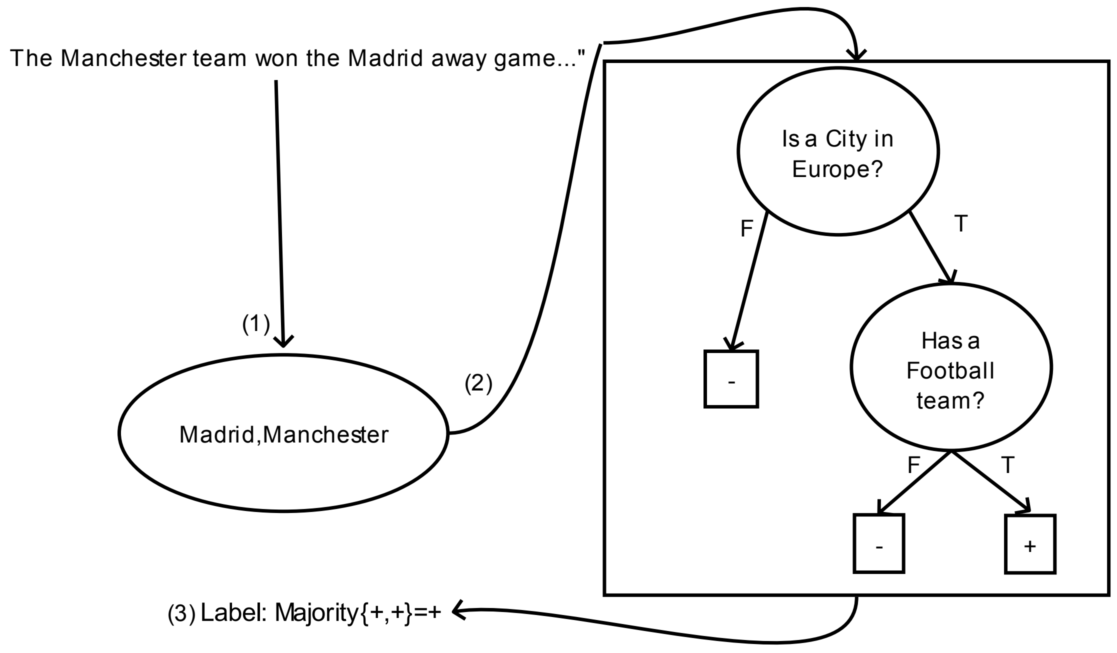

The Manchester team won the Madrid away game..."
(1)
Madrid,Manchester
(2)
Is a City in Europe?
F
T
-
Has a Football team?
F
T
-
+
(3) Label: Majority{+,+}=+

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.

This dataset collection served as a powerful benchmark, as it contains small multiple text classification problems of varying difficulties. We used YAGO2 as our knowledge base as it contains a great deal of "common knowledge" - facts regarding countries, events and individuals.

2. **OHSUMED** (Hersh, Buckley, Leone, & Hickam, 1994) - A dataset of medical abstracts from the MeSH categories of the year 1991. Similarly to Joachims (1998), we use only the first 20,000 documents. Furthermore, we limit ourselves to medical documents that only contain a title (that is, there is no available abstract). On each document, stopword elimination and simple stemming was used (due to the medical nature of the texts, the porter stemmer performed poorly). For entity extraction, we used Wikipedia Miner (Milne & Witten, 2013). In addition, we only take two categories, C1 and C20 [4] as a binary learning problem. We then applied ten-fold cross validation to rigorously test our approach. As our knowledge base, we used Freebase data dump used by Bast, Bäurle, Buchhold, and Haußmann (2014), taking only entities matching our extracted entities. This effectively limits our search space without sacrificing anything in the way of accuracy (unless we apply our algorithm recursively).

   This dataset served as a test case representing a larger, more specialized learning problem, where domain-specific knowledge is required. Thus, Freebase which contains scientific facts, including medical facts, was more suitable. We also experimented with a different entity extraction method.

4. Bacterial Infections and Mycoses, Immunologic Diseases

4.2 Experiment Parameters

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 within a generated learning problem. As discussed in section 2.3, we then flattened the resulting feature trees. We compared the new set of features (our new features combined with the original features) to the binary bag-of-words features, as well as to a simpler, non-recursive feature generation algorithm, described in algorithm 3. This algorithm creates a set of features indicating whether a word appears in the text that appears in a certain relation with a specific value. For example, we could generate a feature that checks whether a word in the text appears in relation "IS-A" with value "Disease".

Algorithm 3 Non-recursive Feature Generation using relations

```
function CREATEFEATURESFORTEXT($S$,$F$, $\mathcal{R}$)
    $F_{result} = \emptyset$
    for $r \in R$ do
        Let $f_{r,c}(x) = \begin{cases} 1 & \text{if } \exists f_i \in F : r(f_i(x)) = c \\ 0 & \text{otherwise} \end{cases}$ ▷ Indicator function for a word appearing in the text
such that it is the key of relation $r$ with value $c$
        Let $f_r(S) = \cup_{x \in S} f_r(x) = \{c | \exists x \in S, f_i \in F : r(f_i(x)) = c\}$ ▷ All values of $r$ with regards to $F$
(our texts)
        $F_{result} = F_{result} \cup \{f_{r,c}(x) | c \in f_r(S)\}$
    end for
    return $F_{result}$
end function
```

In order to avoid over-fitting, we limit both tree depth and node size, proportionately to the size of the training set. We also limit these parameters accordingly within the generated recursive problems, as they also utilize a decision tree as their classifier. To decide whether a tree classifier was sufficiently beneficial, we compared its information gain ratio (including the value of inapplicable) to the information gain ratio of the feature with highest information gain ratio. We then allowed a 50% negative penalty parameter to encourage picking generated features assuming they were better than 50% times the best information gain for that decision tree node.

For labelling objects within constructed learning problems, we use the majority label - The label of an object within the constructed problem will be the label corresponding to the majority of texts containing the entity that was the relation key. Similarly, we use the majority label when deciding on the output of the constructed classifier- In cases where our classifier outputs multiple labels for a single text, we simply take the label corresponding to the majority. We can see an example of this in figure 5.

4.2.1 EVALUATION

Once we generate our feature set, we proceed to test it by learning a classifier on a training set and measuring its accuracy on a testing set. We made use of three well-known learning algorithms for this task: SVM (Cortes & Vapnik, 1995), K-NN (Fix & Hodges Jr, 1951) and CART (Breiman, Friedman, Stone, & Olshen, 1984). For parameters, for SVM we used a linear kernel and a regular-

Table 1: Average Number of Generated Features

	# Features(FEAGURE)	# Features(FEAGURE 2-level)	# Features(Competing)
OHSUMED	506.2	732	27162.2
TechTC-100	63.3	65.06	5004.66

Table 2: Average accuracy

	Baseline	Competing	FEAGURE	FEAGURE 2-level
OHSUMED KNN	**0.777**	0.756	0.769	0.75
OHSUMED SVM	0.797	0.804	0.816	**0.819 ($p < .05$)**
TechTC-100 KNN	0.527	0.523	0.533 ($p < .05$)	**0.557 ($p < .0001$)**
TechTC-100 SVM	0.688	0.694	**0.707 ($p < .0001$)**	0.701 ($p < .01$)

ization parameter $C = 10$, for K-NN we used $K = 3$, and for CART we limited node size according to the size of the training set.

For TechTC-100, we ran our algorithm on all 100 datasets, use the pre-defined training and testing sets to measure accuracy, then ran a paired t-test on the accuracies of our feature set versus both the baseline (no features) and algorithm 3. For OHSUMED, we used 10-fold cross-validation to divide our dataset to ten sets of training and testing data. We then proceeded to run our algorithm, measured its accuracy, and ran a paired t-test on the resulting accuracies.

4.3 Main Results

Table 2 shows average accuracies across all 10 folds for OHSUMED, as well as the average accuracies for datasets in techTC-100, using 80% of all available (chosen by highest information gain ratio). Statistical significance over baseline is shown in parenthesis. We also note that results in bold are significantly better compared to the competing non-recursive approach discussed in algorithm 3. For the TechTC-100 dataset, we see a significant improvement over the baseline and competing approaches, even though fewer features are generated than the competing algorithm. We also see that for SVM, the two-level application gives poorer results than the single level FEAGURE algorithm. This can be attributed to over-fitting effects resulting from a combination of mistakes in entity extraction as well as the small dataset size. Due to the nature of linear SVM, such a feature will have significant impact on performance.

Analysis of the results for the OHSUMED datasets show that for K-NN, we do not achieve an improvement over the baseline approach (and neither does the non recursive relation-based algorithm). For SVM, we see a significant improvement over the baseline approach, with a single level application achieving a 2.4% increase in accuracy, and a two-level application giving a total of 2.8% improvement. This is much better than the 1% improvement offered by a non-recursive feature generation algorithm (algorithm 3).

Before we look at varying parameters of our algorithm, let us analyze the results on TechTC-100 to better understand the impact of our approach on datasets of varying difficulty. Figure 6 shows the accuracies for datasets in techTC-100 using a SVM classifier, The x axis represents the baseline accuracy without feature generation, and the y axis represents the accuracy using our new feature set using FEAGURE. Therefore, any dataset that falls above the $y = x$ line is an improvement in

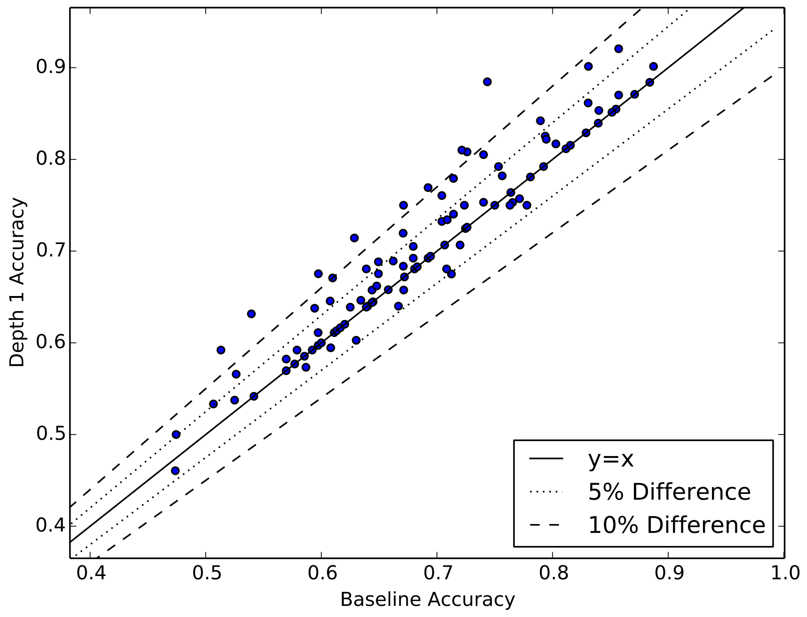


Figure 6: Accuracy of baseline approach compared to single activation of FEAGURE (SVM). Each point represents a dataset. The dotted lines represent a 5 and 10 percent difference in accuracy

accuracy. We use this visualization technique to illustrate our results on the techTC-100 dataset collection.

The results for show a general trend of improvement, with high ($> 5\%$) or very high ($> 10\%$) improvement being common. We also see multiple datasets where no features are generated, with few datasets showing a degrade in accuracy. This can be attributed to two major causes:

- Errors in the entity extraction process may lead to the creation of misleading entities and thus features. For instance, the word "One" may be interpreted as the entity "One (Metallica song)".
- Over-fitting within the feature generation algorithm may create features that appear to have high information gain while generalizing poorly to test data.

Figure 7 shows the 25 hardest datasets in TechTC-100, in terms of lowest accuracy achieved using the full feature set in the original paper of Gabrilovich and Markovitch (2004). We see that for most of these, we achieve improvement, with a 5% or higher increase being common. We also see several datasets where no features are generated (and thus the accuracy is the same) and three datasets that show a minor degrade in accuracy. This results illustrates that we can, in general, rely on FEAGURE to yield positive recursive features on difficult classification problems.

4.4 Using Non-Tree classifiers with FEAGURE

As we have discussed in section 2.1, algorithm 1 creates a generic learning problem. In section 2.2 we discuss several reasons to use a decision tree on this problem. In this section, we will show

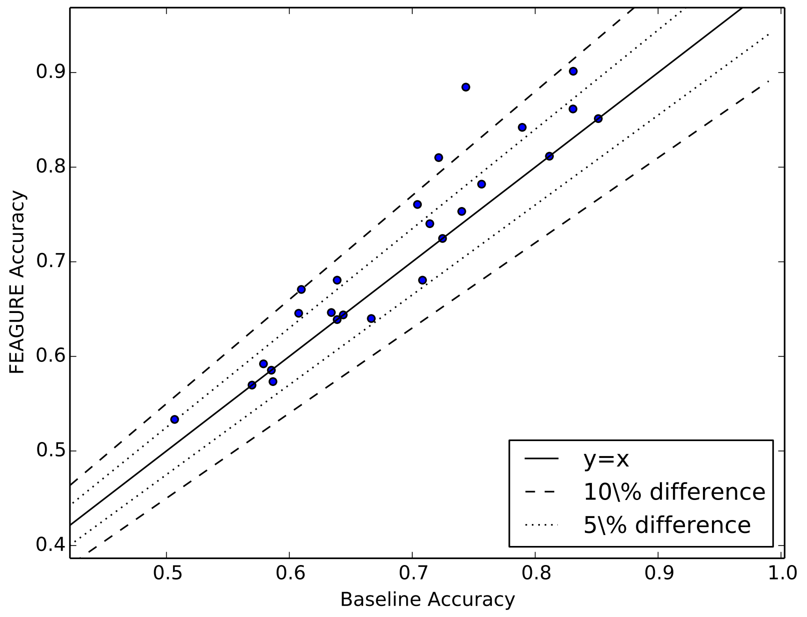


Figure 7: Accuracy of baseline approach compared to single activation of FEAGURE (SVM). 25 Hardest datasets (lowest accuracy)

Table 3: Average Number of Generated Features

	# Features(Tree)	# Features(Linear SVM)	# Features(RBF SVM)	# Features(3-NN)
OHSUMED	506.2	24.7	34.5	14.6
TechTC-100	63.3	12.04	8.95	4.25

the effects of other classifiers on these recursive problems. We have chosen to try both K-NN and SVM, with the following parameters: For K-NN, we used $K = 3$. For SVM we used $C = 10$, with both Linear and a Radial Basis Function (RBF) kernels. We only construct classifiers for the full problem, and not for any child nodes, unlike algorithm 2.

Table 3 shows the average number of generated features. We see that these approaches generate far fewer features on average. This should come as no surprise, as we cannot flatten constructed recursive trees in order to generate additional features, nor do we run the learning algorithms on smaller problems. Table 4 shows the accuracies achieved by these approaches. We see that while in general, the results are poorer than those achieved by our approach, we do see a clear improvement in accuracy, in many cases granting better accuracy than that achieved by the non-recursive feature generation approach (algorithm 3).

4.5 Effect of Constructed Feature Evaluation on Accuracy

In this section we look at the effect of using a different evaluation method for generated features. In algorithm 2, we use a comparison function which compares the constructed feature to the best

Table 4: Average accuracy

	Baseline	Tree	Linear SVM	RBF SVM	3-NN
OHSUMED KNN	0.777	0.769	0.760	**0.795**	0.756
OHSUMED SVM	0.797	**0.816**	0.798	0.788	0.796
TechTC-100 KNN	0.527	**0.533**	0.532	0.526	0.527
TechTC-100 SVM	0.688	**0.707**	0.695	0.692	0.696

Table 5: Average Number of Generated Features

	FEAGURE	FEAGURE 2-level	Average IG	Average IG 2-level
OHSUMED	506.2	732	1270.6	1371.4
TechTC-100	63.3	65.06	234.43	244.05

non-relational feature. It may be the case, however, that we should compare our feature in some way to all existing features. One way to do so is to use the mean Information Gain Ratio of features in F. Should our new feature score lower than the average, we filter it out.

Table 5 shows the amount of features generated by FEAGURE. We see that the amount is significantly greater, indicating that this is a more lax requirement than the one we used (over half of the best information gain ratio). Table 6 shows the accuracies attained through this change. For the TechTC-100 dataset collection, we see a general increase in accuracy, especially for single level application. For the OHSUMED dataset, we cannot detect a meaningful improvement, and for K-NN see a degrade in accuracy. These results indicate that for small problems, generating additional features is preferable, whereas for problems with many examples, a stronger filtering mechanism is more useful.

4.6 Effect of Search Tree Depth on Accuracy

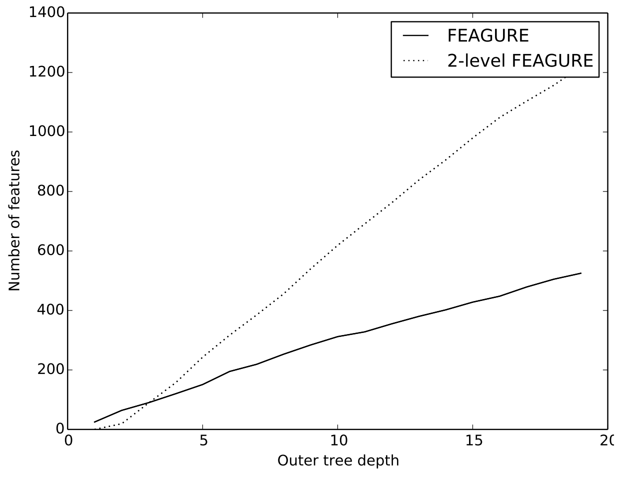
In this section, we look at the usefulness of searching local problems for additional features. To do so, we run FEAGURE while limiting the depth of the search tree in algorithm 2. We compare our two datasets side-by-side, without feature selection.

Figure 8 shows the number of generated features per depth. For both datasets, we see a linear increase, with a two-level application showing a faster increase. In TechTC-100, we see that after a certain depth, there is no additional gain. This is due to the small size of the learning problems.

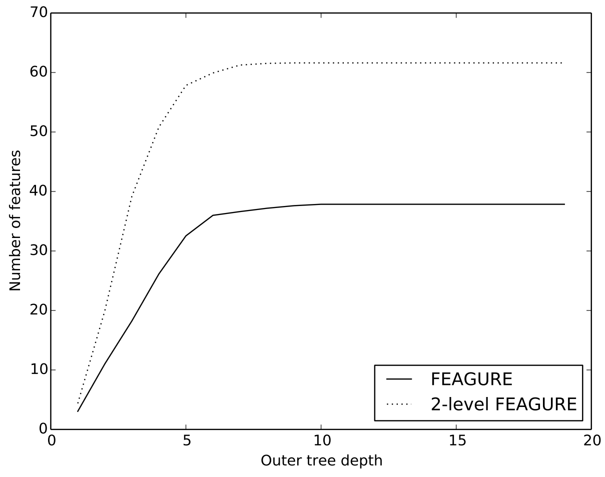
Figure 9 shows the mean accuracy of a SVM classifier for increasing depths. The results for K-NN classifiers are similar their general trend. In TechTC-100, we see a trend of increasing accuracy,

Table 6: Average accuracy

	Baseline	FEAGURE	FEAGURE 2-level	Average IG	Average IG 2-level
OHSUMED KNN	**0.777**	0.769	0.75	0.68	0.687
OHSUMED SVM	0.797	0.816	**0.819**	0.817	0.815
TechTC-100 KNN	0.527	0.533	0.557	0.557	**0.558**
TechTC-100 SVM	0.688	0.707	0.701	**0.723**	0.678

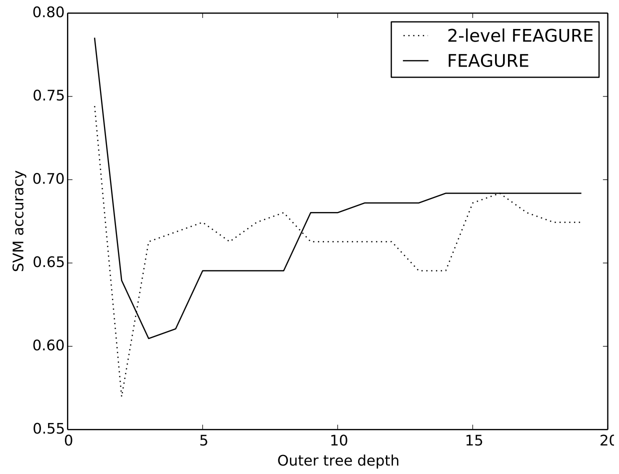


(a) OHSUMED

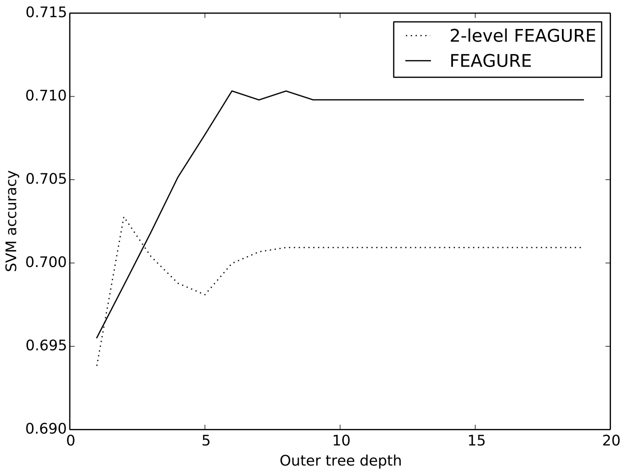


(b) TechTC-100

Figure 8: Mean number of features generated



(a) OHSUMED



(b) TechTC-100

Figure 9: Mean SVM Accuracy

up to a saturation point. This is to be expected, as adding largely orthogonal features would yield improvement until a maximal depth is achieved.

For OHSUMED, however, we see a sharp fall in accuracy, followed by a recovery. This is caused by the orthogonality of features down the tree combined with errors in entity extraction causing features immediately following the first few tree nodes to separate on mistakenly extracted entities.

5. Related Work

One of the earliest methods of utilizing relational information is *Inductive Logic Programming(ILP)* (Quinlan, 1990; Muggleton, 1991), 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 (Davis, Burnside, Dutra, Page, Ramakrishnan, Shavlik, & Costa, 2007), 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 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* (Kramer & Frank, 2000), 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 (Van Laer & De Raedt, 2001) and SGLR (Popescul & Ungar, 2007) 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* (LeCun, Bottou, Bengio, & Haffner, 1998; Bengio, 2009) as a feature generation technique. Good examples of this can be seen in FEX (Plötz, Hammerla, & Olivier, 2011) and DBN2 (Kim, Lee, & Provost, 2013). 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.

6. 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 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 (Bunescu & Pasca, 2006) in general and Entity Linking (Rao et al., 2013) 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 classification problems. We have seen in our experiments that primitive entity matching techniques may lead to mistakes which can cascade into poor features.

Another major potential avenue for improvement is to use the labelling techniques discussed section 2.1 as a way to directly label entities within the semantic graph, which yields a Collective Classification (Kajdanowicz, Kazienko, & Janczak, 2013) problem in the semantic domain. Solving this learning problem in turn allows us to label any entity within the semantic net, essentially yielding a labelled semantic net trained on our problem. We can then use this net to label new objects within the original problem context through a combination of entity extraction, usage of the semantic labels and label consolidation techniques.

References

Bast, H., Bäurle, F., Buchhold, B., & Haußmann, E. (2014). Easy access to the freebase dataset. In *Proceedings of the companion publication of the 23rd international conference on World wide web companion*, pp. 95–98. International World Wide Web Conferences Steering Committee.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and trends® in Machine Learning*, *2*(1), 1–127.

Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with Python.* " O'Reilly Media, Inc.".

Bizer, C., Heath, T., & Berners-Lee, T. (2009). Linked data-the story so far. *International journal on semantic web and information systems*, *5*(3), 1–22.

Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and regression trees*. CRC press.

Bunescu, R. C., & Pasca, M. (2006). Using encyclopedic knowledge for named entity disambiguation.. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, Vol. 6, pp. 9–16.

Cheng, W., Kasneci, G., Graepel, T., Stern, D., & Herbrich, R. (2011). Automated feature generation from structured knowledge. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pp. 1395–1404. ACM.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning*, *20*(3), 273–297.

Davidov, D., Gabrilovich, E., & Markovitch, S. (2004). Parameterized generation of labeled datasets for text categorization based on a hierarchical directory. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 250–257. ACM.

Davis, J., Burnside, E., Dutra, I., Page, D., Ramakrishnan, R., Shavlik, J., & Costa, V. S. (2007). 17 learning a new view of a database: With an application in mammography. *STATISTICAL RELATIONAL LEARNING*, 477.

Finkel, J. R., Grenager, T., & Manning, C. (2005). Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pp. 363–370. Association for Computational Linguistics.

Fix, E., & Hodges Jr, J. L. (1951). Discriminatory analysis-nonparametric discrimination: consistency properties. Tech. rep., DTIC Document.

Gabrilovich, E., & Markovitch, S. (2004). Text categorization with many redundant features: using aggressive feature selection to make svms competitive with c4. 5. In *Proceedings of the twenty-first international conference on Machine learning*, p. 41. ACM.

Gabrilovich, E., & Markovitch, S. (2009). Wikipedia-based semantic interpretation for natural language processing. *Journal of Artificial Intelligence Research*, 443–498.

Hersh, W., Buckley, C., Leone, T., & Hickam, D. (1994). Ohsumed: An interactive retrieval evaluation and new large test collection for research. In *SIGIR94*, pp. 192–201. Springer.

Hoffart, J., Suchanek, F. M., Berberich, K., & Weikum, G. (2013). YAGO2: a spatially and temporally enhanced knowledge base from Wikipedia. *Artificial Intelligence*, *194*, 28–61.

Jarmasz, M. (2012). Roget's thesaurus as a lexical resource for natural language processing. *arXiv preprint arXiv:1204.0140*.

Joachims, T. (1998). *Text categorization with support vector machines: Learning with many relevant features*. Springer.

Kajdanowicz, T., Kazienko, P., & Janczak, M. (2013). Collective classification techniques: an experimental study. In *New Trends in Databases and Information Systems*, pp. 99–108. Springer.

Kim, Y., Lee, H., & Provost, E. M. (2013). Deep learning for robust feature generation in audiovisual emotion recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 3687–3691. IEEE.

Kramer, S., & Frank, E. (2000). Bottom-up propositionalization.. In *ILP Work-in-progress reports*.

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

Lösch, U., Bloehdorn, S., & Rettinger, A. (2012). Graph kernels for RDF data. In *The Semantic Web: Research and Applications*, pp. 134–148. Springer.

Markovitch, S., & Rosenstein, D. (2002). Feature generation using general constructor functions. *Machine Learning*, *49*(1), 59–98.

Milne, D., & Witten, I. H. (2013). An open-source toolkit for mining wikipedia. *Artificial Intelligence*, *194*, 222–239.

Muggleton, S. (1991). Inductive logic programming. *New generation computing*, *8*(4), 295–318.

Paulheim, H., & Fümkranz, J. (2012). Unsupervised generation of data mining features from linked open data. In *Proceedings of the 2nd international conference on web intelligence, mining and semantics*, p. 31. ACM.

Pelikánová, Z. (2014). Google knowledge graph..

Plötz, T., Hammerla, N. Y., & Olivier, P. (2011). Feature learning for activity recognition in ubiquitous computing. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, Vol. 22, p. 1729.

Popescul, A., & Ungar, L. H. (2007). 16 feature generation and selection in multi-relational statistical learning. *STATISTICAL RELATIONAL LEARNING*, 453.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, *1*(1), 81–106.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine learning*, *5*(3), 239–266.

Ragavan, H., Rendell, L., Shaw, M., & Tessmer, A. (1993). Complex concept acquisition through directed search and feature caching. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-1993)*, pp. 946–951.

Rao, D., McNamee, P., & Dredze, M. (2013). Entity linking: Finding extracted entities in a knowledge base. In *Multi-source, Multilingual Information Extraction and Summarization*, pp. 93–115. Springer.

Rios, M., Specia, L., Gelbukh, A., & Mitkov, R. (2014). Statistical relational learning to recognise textual entailment. In *Computational Linguistics and Intelligent Text Processing*, pp. 330–339. Springer.

Salton, G., & McGill, M. J. (1983). Introduction to modern information retrieval..

Utgo, P. E., & Brodley, C. E. (1991). Linear machine decision trees. Tech. rep., University of Massachusetts at Amherst.

Van Laer, W., & De Raedt, L. (2001). How to upgrade propositional learners to first order logic: A case study. In *Machine Learning and Its Applications*, pp. 102–126. Springer.

Van Rijsbergen, C. J., Robertson, S. E., & Porter, M. F. (1980). *New models in probabilistic information retrieval*. Computer Laboratory, University of Cambridge.

Vrandečić, D., & Krötzsch, M. (2014). Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, *57*(10), 78–85.

Wu, H., & Salton, G. (1981). A comparison of search term weighting: Term relevance vs. inverse document frequency. *Special Intrest Group on Information Retrieval (SIGIR) Forum*, *16*(1), 30–39.