

# Recursive Feature Generation for Knowledge-based Induction

**Lior Friedman**

**Shaul Markovitch**

*Technion-Israel Institute of Technology*

*Haifa 32000, Israel*

LIORF@CS.TECHNION.AC.IL

SHAULM@CS.TECHNION.AC.IL

## Abstract

When humans perform inductive learning they often enhance the process with extensive background knowledge. With the increasing availability of well-formed collaborative knowledge bases, the performance of learning algorithms could likewise be significantly enhanced if a way were found to exploit these knowledge bases. In this work we present a novel algorithm for injecting external knowledge into induction algorithms using feature generation. Given a feature, the algorithm defines a new learning task over its set of values, and uses the knowledge base to solve the constructed learning task. The resulting classifier is then used as a new feature for the original problem. We applied our algorithm to the domain of text classification using large semantic knowledge bases. We have shown that the generated features significantly improve the performance of existing learning algorithms.

## 1. Introduction

In recent decades, machine learning techniques have become more prevalent in a wide variety of fields. Most of these methods rely on the inductive approach: given a set of labeled examples, they attempt to locate a hypothesis that is heavily supported by the known data. These methods have proven themselves successful when the number of examples is sufficient, and 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 this difficulty is to generate new features that are combinations of the given ones. For example, the LFC algorithm (Ragavan et al., 1993) combines binary features through the use of logical operators such as  $\wedge$ ,  $\neg$ . Another example is the LDMT algorithm (Brodley & Utgoff, 1995), which generates linear combinations of existing features. Deep Learning methods combine basic and generated features using various activation functions such as sigmoid or softmax. The FICUS framework (Markovitch & Rosenstein, 2002) presents a general method for generating features using any given set of constructor functions.

The above approaches are limited in that they merely combine existing features to make the representation more suitable for the learning algorithm. While this approach often suffices, there are many cases where additional information is required for successful classification. When people perform inductive learning, they usually rely on a vast body of background knowledge to make the process more effective (McNamara & Kintsch, 1996). For example, assume two positive examples of people suffering from some genetic disorder, where the value of the country-of-origin feature is Poland and Romania. Existing induction algorithms, including those generating features by combination, will not be able to generalize over these two examples. Humans, on the other hand, can easily generalize and generate a new feature *Eastern Europe* based on their general background knowledge.

In this work, we present a novel algorithm that uses a similar approach for enhancing inductive learning with background knowledge through feature generation. Our method assumes a given body of knowledge represented in relational form. The algorithm treats feature values as objects and constructs new learning problems, using the background relational knowledge as their features. The resulting classifiers are then used as new generated features. For example, in the above very simple example, our algorithm would have used the *continent* and *region* features of a country, inferred from a geographic knowledge base, to create the new feature that makes generalization possible. Using background knowledge in the form of generated features allows us to utilize existing powerful learning algorithms for the induction process.

We implemented our algorithm in the domain of text classification problems, using Freebase and YAGO2 as our background knowledge bases, and performed an extensive set of experiments to test the effectiveness of our method. Results show that the use of background knowledge significantly improves the classification accuracy of existing learning algorithms.

## 2. Motivation

Before we delve into the detailed description of our algorithm, we would like to illustrate its main ideas using an example. Suppose we are attempting to identify people with a high risk of suffering from a genetic disorder. Assume that the target concept to be discovered is that those at risk are women with ancestors originating from desert areas. To identify patients at risk, we are given a training sample of sick and healthy people, containing various features, including gender and their full name. We call this learning problem  $T_1$ . Assuming we have no additional information, an induction algorithm (a decision tree learner, in this example) would likely produce a result similar to that shown in Figure 1. While such a classifier will achieve a low training error, the hundreds of seemingly unrelated surnames will cause it to generalize poorly.

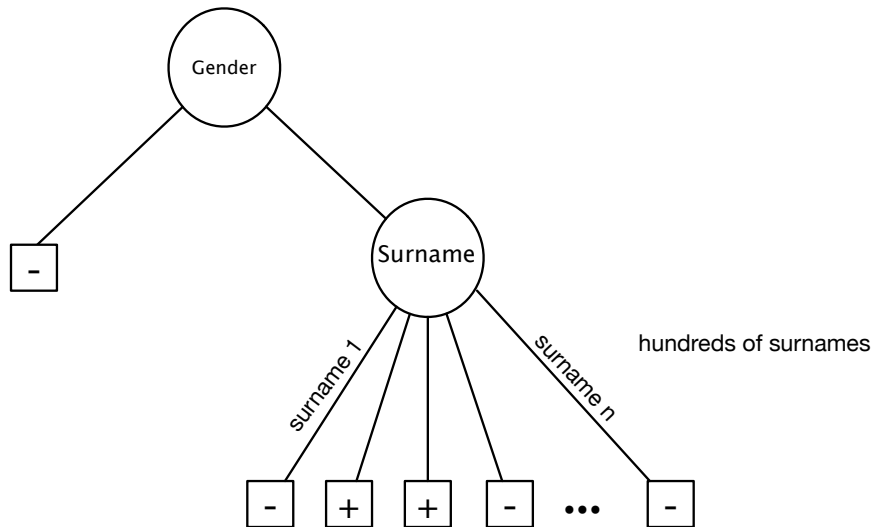


Figure 1: A decision tree for the basic features

The above example illustrates a case where, without additional knowledge, an induction algorithm will yield a very poor result. However, if we assume access to a relational knowledge base connecting surnames to common countries of origin, we can begin to apply our knowledge-based feature generation techniques to the problem, as we can move from the domain of surnames to that of countries. Our algorithm does so by creating a new learning problem  $T_2$ . The training objects for learning problem  $T_2$  are surnames; surnames of people at risk are labeled as positive. The features for these new objects are extracted from the knowledge base. In this case, we have a single feature: the country of origin. Solving the above learning problem through an induction algorithm yields a classifier on surnames that distinguishes between surnames of people at risk and surnames of healthy individuals. This classifier for  $T_2$  can then be used as a binary feature for the original problem  $T_1$  by applying it to the feature value of surname. For example, it can be used as a feature in the node corresponding to people whose gender is female in Figure 1, yielding the tree seen in Figure 2.

This new feature gives us a better generalization over the baseline solution, 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.

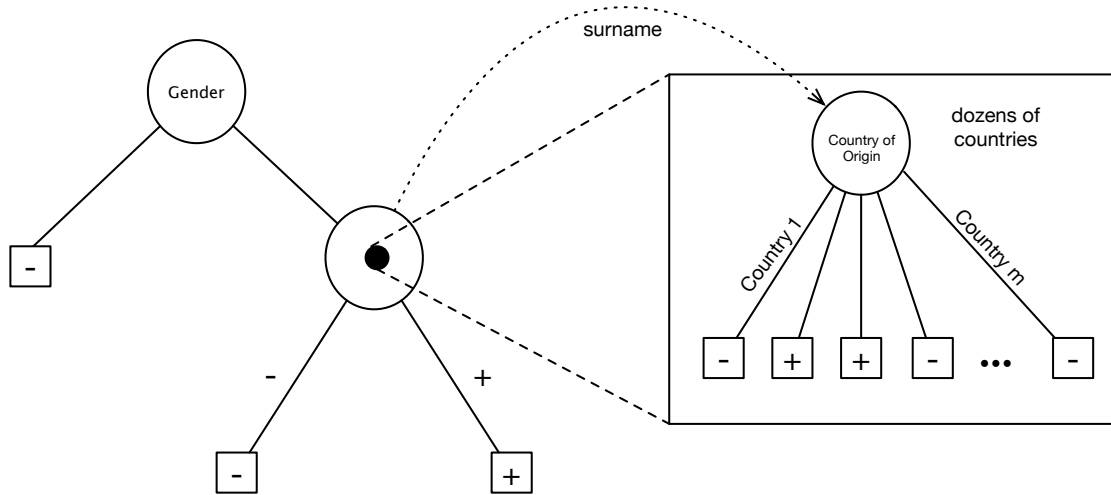


Figure 2: A constructed feature used within a decision tree

If, however, we would have recursively applied our method for solving  $T_2$ , we could have obtained a better generalization. When learning to classify surnames, our method creates a new learning problem,  $T_3$ , with countries as its objects. Countries of surnames belonging to people with high risk are labeled as positive. The knowledge base regarding countries is then used to extract features for this new training set. Applying a standard learning algorithm to  $T_3$  will yield a classifier that distinguishes between countries of origin of people at risk and those not at risk. This classifier will do so by looking at the properties of countries, and concluding that countries with high average temperature and low precipitation, the characteristics of desert areas, are indicative of high risk.

The result of this process, depicted in Figure 3, is a new feature for  $T_2$ , that is, a feature on surnames. This feature is then used to construct a classifier for  $T_2$ , which is used in turn as a feature

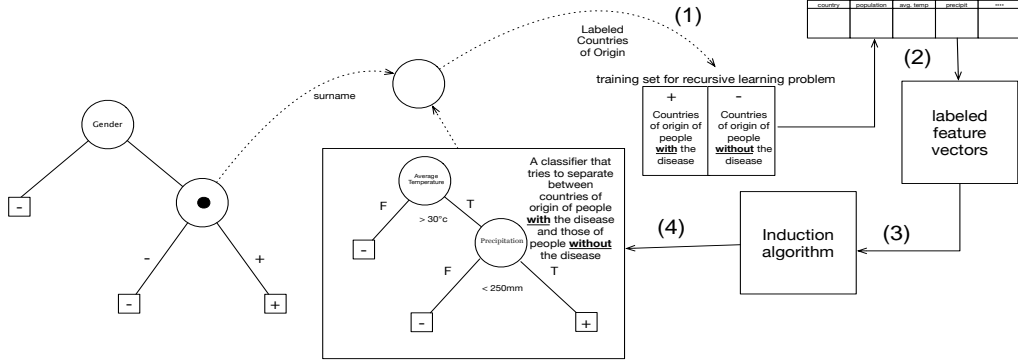


Figure 3: Recursive construction of a learning problem on countries of origin. (1) Creating the objects for the new problem. (2) Creating features using the knowledge base. (3) Applying an induction algorithm. (4) The resulting feature.

for  $T_1$ , yielding a feature on people. This new feature for people, shown in Figure 4, will check whether their surname corresponds to a country of origin that has desert-like characteristics. This example illustrates the major steps of our approach: the construction of new induction problems on feature values using the given knowledge base, the creation of classifiers over the above learning problems, and the use of said classifiers as new features. These steps allow us to accurately capture the target concept using a two-level recursive process.

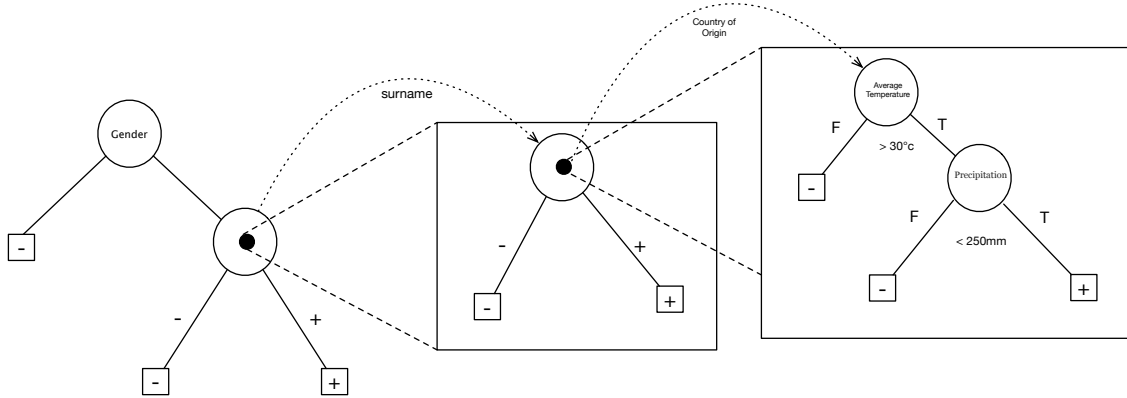


Figure 4: A two-level constructed feature used within a decision tree

### 3. Generating Features through Recursive Induction

In the following sections, we formally define the feature generation problem, present a solution in the form of a simple algorithm that generates features by using relational expansions, and then proceed to describe our main recursive feature generation algorithm.

### 3.1 Problem definition

We begin our discussion with a standard definition of an induction problem. Let  $O$  be a set of objects. Let  $Y = \{0, 1\}$  be a set of labels<sup>1</sup>. Let  $C : O \rightarrow Y$  be a target concept. Let  $S = \{(o_1, y_1), \dots, (o_m, y_m)\}$  be a set of labeled examples such that  $o_i \in O, y_i \in Y, C(o_i) = y_i$ . Let  $F = \{f_1, \dots, f_n\}$  be a *feature map*, a set consisting of several *feature functions*  $f_i : O \rightarrow I_i$ . This definition implies a training set represented by feature vectors:  $S_F = \{(\langle f_1(o_i), \dots, f_n(o_i) \rangle, y_i) | (o_i, y_i) \in S\}$ . A learning algorithm  $L$  takes  $S_F$  as inputs, and outputs a classifier  $h_{S_F} : O \rightarrow Y$ .

**Definition 3.1.** Let  $L(S, F) = h_{S_F}$  be the classifier given as an output by  $L$  given  $\langle S, F \rangle$ . Assuming  $S \sim D$ , the generalization error of a learning algorithm  $L$  is the probability  $Pr(h_{S_F}(x) \neq y)$ , where  $(x, y) \sim D$ .

**Definition 3.2.** A *feature generation algorithm*  $A$  is an algorithm that, given  $\langle S, F \rangle$ , creates a new feature map  $F' = \{f'_1, \dots, f'_l\}, f'_k : O \rightarrow I_k$ .

In order to evaluate the output of a feature generation algorithm  $A$ , we must define its utility. Given  $\langle S, F \rangle$ ,  $A$  generates a feature set  $F'_A$ . Given  $S \sim D$ , a feature set  $F$ , a generated feature set  $F'_A$  and a learning algorithm  $L$ , the utility of  $A$  is  $U(A(S, F)) = Pr(h_{S_F}(x) \neq y) - Pr(h_{S_{F'_A}}(x) \neq y)$ , where  $(x, y) \sim D$ .

Thus, in order for the utility of  $A$  to be positive, the generated feature set  $F'_A$  must yield a lower generalization error than the original feature map  $F$ .

In this work, we assume that, in addition to  $S_F$ ,  $A$  also has access to a set of binary<sup>2</sup> relations  $\mathcal{R} = \{R_1, \dots, R_t\}, R_j : D_j \times D_{j'}$  representing our knowledge base.

**Definition 3.3.** A *knowledge-based feature-generation algorithm*  $A$  is an algorithm that, given  $\langle S, F, \mathcal{R} \rangle$ , creates a new feature map  $F_{\mathcal{R}} = \{f'_1, \dots, f'_l\}, f'_k : O \rightarrow I_k$ .

By using a feature generation algorithm based on a set of relations  $\mathcal{R}$ , we would like to achieve a higher utility than that achieved without  $\mathcal{R}$ . We assume that some feature values are also relation keys, allowing us to apply these relations to our learning problem. More formally, we assume that  $\bigcup_{f_i} I_i \cap \bigcup_{R_j} D_j \neq \emptyset$ .

### 3.2 Expansion-based feature generation

Our first method for knowledge-based feature generation ignores the labels of the original learning problem. The algorithm extends each feature value by all of the tuples it appears in. Let  $f_i$  be an original feature. Let  $R_j : D_j \times D_{j'}$  be a relation such that  $Image(f_i) \subseteq D_j$ . We can generate a new feature function  $f_{i,j} : O \rightarrow D_{j'}$  by composing  $R_j$  onto  $f_i$ , yielding our new feature function  $f_{i,j}(x) = R_j \circ f_i$ . For example, if  $f_i$  lists a person's country of origin, and  $R_j$  maps countries to their continent, then  $f_{i,j}$  lists a person's continent of origin.

In the general case, composing  $R_j$  onto  $F_i$  yields a set of values, meaning that  $f_{i,j}(x) = \{v \in D_{j'} | (f_i(x), v) \in R_j\}$ . To yield a single value, we use an aggregation function on this set. An aggregation function is a function that, given a multi-set as input, outputs a single value. There are many types of aggregators, such as numerical aggregators (i.e. sum), categorical aggregators (i.e. most common member) and binary aggregation functions (i.e. exists). For the experiments

1. We assume binary labels for ease of discussion.

2. If our relations are not binary, we can use projection to create multiple binary relations instead.

described in this paper, we use two aggregation function types: Majority and Any, but in general any other reasonable aggregation function can be used instead. The Majority aggregator, for example, is defined as follows. For each value  $v \in D_{j'}$ , we generate a binary feature function with value 1 only if  $v$  is the majority value:  $\text{Majority}^v(X) = 1 \iff \text{majority}(X) = v$ . The pseudo-code of this algorithm (called *Expander-FG*) is listed in Algorithm 1.

---

**Algorithm 1** *Expander-FG*


---

$\sigma$  - An aggregation function family

```

function GENERATEFEATURES( $S, F, \mathcal{R}$ )
   $generated = \emptyset$ 
  for  $f_i \in F$  do
    for  $R_j \in \mathcal{R}$  such that  $\text{Image}(f_i) \subseteq D_j$  do
      if  $R_j$  is a function then
        add  $\{f_{i,j} = R_j \circ f_i\}$  to  $generated$ 
      else
        add  $F_{i,j}^\sigma = \{\sigma^v(f_{i,j}) | v \in D_{j'}\}$  to  $generated$ 
         $\triangleright R_j$  is a relation  $R_j : D_j \times D_{j'}$ 
  return  $generated$ 

```

---

### 3.3 Recursive feature generation algorithm

One way to extend the *Expander-FG* algorithm described in the previous section is to apply it repeatedly to its own output. Extending the algorithm in this fashion, however, would yield an exponential increase in the number of generated features, making deep connections very difficult to discover. To that end, we propose an alternative knowledge-based feature generation algorithm. Given an input  $\langle S, F, \mathcal{R} \rangle$ , for each feature  $f_i \in F$ ,  $f_i : O \rightarrow I_i$ , our algorithm creates a recursive learning problem whose objects are the values of  $f_i$ , where values associated with positive examples in  $S$  are labeled positive. Features for this generated problem are created using the relations in  $\mathcal{R}$ . Once this new learning problem  $\langle S'_i, F_{\mathcal{R}} \rangle$  is defined, a learning algorithm is used to induce a classifier  $h_i : I_i \rightarrow Y$ . Finally, our algorithm outputs a single generated feature for  $f_i$ ,  $f'_i(x) = h_i \circ f_i = h_i(f_i(x))$ ,  $f'_i : O \rightarrow Y$ . We note that during the induction process of the newly created learning problem, we can apply a feature generation algorithm on  $\langle S'_i, F_{\mathcal{R}}, \mathcal{R} \rangle$ . In particular, we can apply the above method recursively to create additional features. We call this algorithm *FEAGURE* (FEAture Generation Using REcursive induction).

Given a feature  $f_i$ , we create a recursive learning problem  $\langle S'_i, F_{\mathcal{R}} \rangle$ . Let  $v_i(S) = \{v | (o, y) \in S, f_i(o) = v\}$  be the set of feature values for  $f_i$  in the example set  $S$ . We use  $v_i(S)$  as our set of objects. To label each  $v \in v_i(S)$ , we first examine the labels in the original problem. If there is a single example  $o \in S$  such that  $f_i(o) = v$ , then the label of  $v$  will be the label of  $o$ . Otherwise, we take the majority label of examples whose feature value is  $v$ . We can state this more formally as follows: if  $s_i(v) = \{o | (o, y) \in S, f_i(o) = v\}$  is the set of objects whose feature value is  $v$ , then  $\text{label}(v) = \text{majority}(\{y | (o, y) \in S, o \in s_i(v)\})$ .

To fully define our learning problem, we must specify a feature map over  $v_i(S)$ . In a similar manner to *Expander-FG*, we use the relations in  $\mathcal{R}$  on the elements in the new training set  $S'_i = \{(v, \text{label}(v)) | v \in v_i(S)\}$ . For each  $R_j \in \mathcal{R}$ , if it is relevant to the problem domain, meaning that  $v_i(S) \subseteq D_j$ , we utilize it as a feature by applying it to  $v$ . If  $R_j(v)$  is a set, we use aggregators, as described in the previous section. The result of this process is a generated feature map for  $S'_i$ , denoted as  $F_{\mathcal{R}}$ .

We now have a new induction problem  $\langle S'_i, F_{\mathcal{R}} \rangle$ . We can further extend  $F_{\mathcal{R}}$  by recursively using *FEAGURE*, yielding a new feature map  $F'_{\mathcal{R}}$ . The depth of recursion is controlled by a parameter  $d$ , that is usually set according to available learning resources. We proceed to use a learning algorithm<sup>3</sup> on  $\langle S'_i, F'_{\mathcal{R}} \rangle$  in order to train a classifier, giving us  $h_i : I_i \rightarrow Y$ . We can then use  $h_i$  on objects in  $S$  as discussed above, yielding a new feature  $f'_i(x) = h_i(f_i(x))$ ,  $f'_i : O \rightarrow Y$ .

It is important to note that these new features might over-fit the generated problem, especially if it is small or imbalanced. There are several known ways to combat this issue. For our experiments, we decided to apply the following filters:

- We do not call the learner in cases where the generated learning problem has too few examples.
- We do not call the learner on a generated training set if it is a single set problem.
- We filter out generated features that have an information gain of zero on the (original or generated) training set.

We found these criteria sufficient for disqualifying very poor features without requiring a significant change in our approach.

The full algorithm is listed in Algorithm 2. We note that this feature generation approach creates features that can be used alongside any induction algorithm. Additionally, any induction algorithm can be used to learn  $h_i$ . Due to this generality, the extensive knowledge and literature available for induction methods applies in full. Furthermore, the recursive nature of *FEAGURE* allows for a structured search of the knowledge base, with each new recursive problem modeling a projection of the problem space to the new domain. For example, in the motivating example, we began by learning a problem regarding people at risk, which we projected to a new domain of surnames. We then re-contextualized that problem to the domain of countries, for which we induced a good solution. We proceeded to use this solution to resolve the original learning task concisely and accurately.

---

**Algorithm 2** FEAGURE algorithm
 

---

```

function GENERATEFEATURES( $F, S, \mathcal{R}, d$ )
    for  $f_i \in F$  do
         $S'_i, F_{\mathcal{R}} = \text{CREATENEWPROBLEM}(f_i, S, \mathcal{R}, d)$ 
         $h_i = \text{INDUCTIONALGORITHM}(S'_i, F_{\mathcal{R}})$ 
        add  $f'_i(x) = h_i \circ f_i$  to generated features (unless disqualified)
    return generated features

function CREATENEWPROBLEM( $f_i, S, \mathcal{R}, d$ )
     $v_i(S) = \{v \mid (o, y) \in S, f_i(o) = v\}$ 
     $S'_i = \{(v, \text{majority-label}(s_i(v))) \mid v \in v_i(S), s_i(v) = \{o \mid (o, y) \in S, f_i(o) = v\}\}$ 
     $F_{\mathcal{R}} = \{R_j(v) \mid R_j \in \mathcal{R}, v_i(S) \subseteq D_j\}$ 
    if  $d > 0$  then
         $F_{\mathcal{R}} = F_{\mathcal{R}} \cup \text{GENERATEFEATURES}(F_{\mathcal{R}}, S'_i, \mathcal{R}, d - 1)$ 
    return  $S'_i, F_{\mathcal{R}}$ 
    
```

---

3. For the experiments described in this paper, we used a decision tree learner, but any induction algorithm can be used.

### 3.4 Finding Locally Improving Features

Some generated features may prove very useful for separating only a subset of the training data. Such features may be difficult to identify in the context of the full training data. In the motivating example in Section 2, for instance, examples of male individuals (which are not at risk) are irrelevant to the target concept, and thus mask the usefulness of the candidates for feature generation.

In this subsection, we present an extension of our *FEAGURE* algorithm that evaluates features in local contexts using the divide & conquer approach. This algorithm uses a decision tree induction method as the basis of its divide & conquer process. At each node, the *FEAGURE* algorithm is applied to the given set of features, yielding a set of generated features. Out of the expanded (base and generated) feature set, the feature with the highest information gain measure (Quinlan, 1986) is selected, and the training set is split based on the values of that feature. This feature may or may not be one of the generated features. We continue to apply this approach recursively to the examples in each child node, using the expanded feature set as a baseline. Once a stopping criterion has been reached, the generated features at each node are gathered as the final output. In our case, we stop the process if the training set is too small or if all examples have the same label. The decision tree is then discarded.

In addition to generating features that operate within localized contexts of the original induction problem, our approach offers several advantages for feature generation:

1. Orthogonality: Because all examples with the same value for a given feature are grouped together, any further splits must make use of different features. Due to this and the fact that the features selected in each step have high IG, features chosen later in the process will be mostly orthogonal to previously chosen features. This results in a larger variety of features overall. The feature chosen as a split effectively prunes the search tree of possible features and forces later splits to rely on other features and thus different domains.
2. Interpretability: Looking at the features used at each splitting point gives us an intuitive understanding of the resulting subsets. Because of this, we can more easily understand why certain features were picked over others, which domains are no longer relevant, and so on.
3. Iterative construction: The divide & conquer approach allows for an iterative search process, which can be interrupted if a sufficient number of features were generated, or when the remaining training set is no longer sufficiently representative for drawing meaningful conclusions.

The above advantages give us a strong incentive to utilize this approach when attempting to generate features using *FEAGURE*. We call this new algorithm *Deep-FEAGURE*, as it goes into increasingly deeper local contexts. Pseudocode for this method is shown in Algorithm 3. Through the use of a divide & conquer approach, we can better identify strong, locally useful features that the *FEAGURE* algorithm may have difficulty generating otherwise. In our experiments, we used this approach to generate features.



---

**Algorithm 3** Deep FEAGURE- Divide & conquer feature generation
 

---

minSize: minimal size of a node.

generatedFeatures: A global list of all generated features.

SelectFeature: Method that selects a single feature, such as highest information gain.

```

function DEEPFEAGURE( $S, F, \mathcal{R}, d$ )
    if all examples in  $S$  are of same class  $c$  then
        return leaf( $c$ )
    if  $|S| < \text{minSize}$  then
        return leaf(majority class of  $S$ )
    localGeneratedFeatures=FEAGURE( $S, F, \mathcal{R}, d$ )
    add localGeneratedFeatures to generatedFeatures
     $f = \text{SELECTFEATURE}(S, F \cup \text{localGeneratedFeatures})$ 
     $\text{children} = \emptyset$ 
    for  $v \in \text{Domain}(f)$  do
         $S(f) = \{(o, y) \in S \mid f(o) = v\}$ 
        subTree= DEEPFEAGURE( $S(f), F \cup \text{localGeneratedFeatures}, \mathcal{R}, d$ )
        add subTree to  $\text{children}$ 
    return  $\text{children}$ 
    
```

---

## 4. Empirical Evaluation

We have applied our feature generation algorithm to the domain of text classification.

### 4.1 Application of FEAGURE to Text Classification

The text classification problem is defined by a set of texts  $O$  labeled by a set of categories  $Y$ <sup>4</sup> such that we create  $S = \{(o_i, y_i) \mid o_i \in O, y_i \in Y\}$ . Given  $S$ , the learning problem is to find a hypothesis  $h : O \rightarrow Y$  that minimizes generalization error over all possible texts of the given categories. To measure this error, a testing set is used as an approximation.

In recent years, we have seen the rise of Semantic Linked Data as a powerful semantic knowledge base for text-based entities, with large databases such as Google Knowledge Graph (Pelikánová, 2014), Wikidata (Vrandečić & Krötzsch, 2014) and YAGO2 (Hoffart et al., 2013) becoming common. These knowledge bases represent semantic knowledge through the use of relations, mostly represented by triplets of various schema such as RDF, or in structures such as OWL and XML. These structures conform to relationships between entities such as “born in” (hyponyms), as well as type information (hypernyms).

To use FEAGURE for text classification, we use words as binary features and Freebase and YAGO2 as our semantic knowledge bases. YAGO2 (Hoffart et al., 2013) is a large general knowledge base extracted automatically from Wikipedia, WordNet and GeoNames. YAGO2 contains over 10 million entities and 124 million relational facts, mostly dealing with individuals, countries and events. Freebase (Bollacker, Evans, Paritosh, Sturge, & Taylor, 2008) has been described as “a massive, collaboratively edited database of cross-linked data.” Freebase is constructed as a combination of data harvested from databases such as Wikipedia and data contributed by users. The result is a massive, extensive knowledge base containing 1.9 billion facts.

---

4. We can assume  $Y = \{0, 1\}$  for ease of analysis.

To apply our approach to the domain of text classification, we perform a few minor adjustments to the *FEAGURE* algorithm:

1. To enable linkage between the basic features and the semantic knowledge bases, we use entity linking software (Hoffart et al., 2011; Milne & Witten, 2013) to transform these words into semantically meaningful entities that are connected to our knowledge base. The result of this process is a set of entities corresponding to each document.
2. Since all of our base features are binary words, we use the set of entities extracted from the text as the set of values  $v_i(S)$ , giving us a single feature.
3. Once we have created a new classifier  $h_i$ , we cannot simply compose it on  $f_i$ , since every example might contain multiple entities. To that end, we apply  $h_i$  on each relevant entity and take the majority vote.
4. Since  $v_i(S)$  contains entities belonging multiple domains, we split it into several subsets according to relation domains and apply the *FEAGURE* algorithm independently to each domain. The result of this process is a generated feature for each such domain.

## 4.2 Methodology

We used two collections of datasets for our evaluation:

**TechTC-100** (Davidov, Gabrilovich, & Markovitch, 2004) is a collection of 100 different binary text classification problems of varying difficulty, extracted from the Open Dictionary project. We used the training and testing sets defined in the original paper. As our knowledge base for this task, we used YAGO2. For entity extraction, we used AIDA (Hoffart et al., 2011), a framework for entity detection and disambiguation.

**OHSUMED** (Hersh et al., 1994) is a large dataset of medical abstracts from the MeSH categories of the year 1991. First, we took the first 20,000 documents, similarly to Joachims (1998). Then, we limited the texts further to medical documents that contain only a title. Due to the relatively sparse size of most MeSH categories, we only used the two categories with the most documents, C1 and C20<sup>5</sup>, yielding a binary learning problem. The result is a dataset of 850 documents of each category, for a total of 1700 documents. We used ten-fold cross-validation to evaluate this dataset. Since the YAGO2 knowledge base does not contain many medical relations, we used Freebase instead. We took the Freebase data dump used by Bast et al. (2014). We used Wikipedia miner (Milne & Witten, 2013) for entity linking.

In our experiments, we compared the performance of a learning algorithm with the features generated by the *Deep-FEAGURE* algorithm, discussed in Section 3.4, to the baseline of the same induction algorithm without the constructed features. In addition, since we could not obtain the code of competitive approaches for relation-based feature generation (such as FeGeLOD, SGLR), we instead compared our algorithm to *Expander-FG*, which we believe to be indicative of the performance of these unsupervised approaches. We used three induction algorithms as benchmarks for measuring the difference in performance: SVM (Cortes & Vapnik, 1995), K-NN (Fix & Hodges Jr, 1951) and CART (Breiman, Friedman, Olshen, & Stone, 1984).

---

5. “Bacterial Infections and Mycoses” and “Immunologic Diseases,” respectively.

### 4.3 Results

Table 1 shows average accuracies across all 10 folds for OHSUMED, as well as the average accuracies for all 100 datasets in techTC-100. When the advantage of our method over the baseline was found to be significant using a pairwise t-test (with  $p < 0.05$ ), we marked the  $p$ -value in parenthesis. Best results are marked in bold. For the TechTC-100 dataset, *FEAGURE* shows a significant improvement over the baseline approach, even though the number of generated features is much lower than that of the *Expander-FG* algorithm, as seen in Table 2. Of particular note are the results for KNN and SVM, where the two-level activation of *FEAGURE* shows statistically significant improvement over *Expander-FG* as well as the baseline accuracy ( $p < 0.05$ ). One notable exception to our good results is K-NN for the OHSUMED dataset. This is likely due to the sensitivity of K-NN to the increase in dimension.

Table 1: Average accuracy over all datasets. The columns specify the feature generation approach, with the baseline being no feature generation. The rows specify the induction algorithm used on the generated features for evaluation.

Dataset	Classifier	Baseline	Expander-FG	FEAGURE	FEAGURE 2-level
OHSUMED	KNN	<b>0.777</b>	0.756	0.769	0.75
	SVM	0.797	0.804	0.816 ( $p < 0.05$ )	<b>0.819</b> ( $p < 0.05$ )
	CART	0.806	0.814	0.809	<b>0.829</b> ( $p < 0.05$ )
TechTC-100	KNN	0.531	0.702 ( $p < 0.001$ )	0.772 ( $p < 0.001$ )	<b>0.775</b> ( $p < 0.001$ )
	SVM	0.739	0.782 ( $p < 0.001$ )	0.796 ( $p < 0.001$ )	<b>0.807</b> ( $p < 0.001$ )
	CART	0.81	0.815	0.814	<b>0.825</b> ( $p < 0.05$ )

Figures 5 and 6 show 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 generated using *FEAGURE*. Therefore, any dataset that falls above the  $y = x$  line marks an improvement in accuracy. The results show a strong trend of improvement, with high ( $> 10\%$ ) improvement being common. We see that for a few of the datasets, accuracy decreases. This decrease in accuracy can be a result of mistakes in the entity extraction and linking process that can cause over-fitting in the generated learning problems.

In their paper on TechTC-100, (Davidov et al., 2004) define a metric called Maximal Achievable Accuracy (MAA). This criterion attempts to assess the difficulty of the induction problem by the maximal ten-fold accuracy over three very different induction algorithms (SVM, K-NN and CART). Figure 7 shows the performance of *FEAGURE* on the 25 hardest datasets in TechTC-100, in terms of the MAA criterion. We call this dataset collection “TechTC-25MAA.” Table 1 shows the accuracies for the 25 hardest datasets in TechTC-100, in terms of the MAA criterion. We call this dataset

Table 2: Average number of generated features for each dataset per fold/dataset, using *FEAGURE*, a 2-level activation of *FEAGURE*, and the Shallow algorithm.

	# Features(FEAGURE)	# Features(FEAGURE 2-level)	# Features(Expander-FG)
OHSUMED	506	732	27162
TechTC-100	614	1477	10997

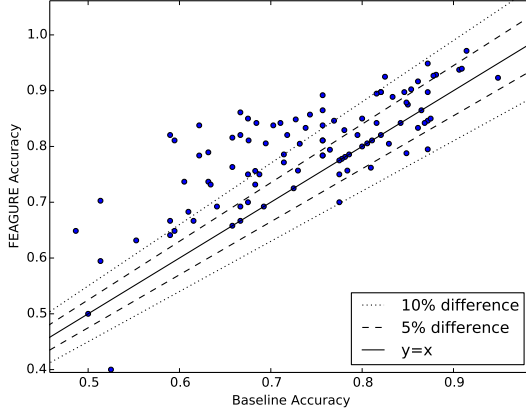


Figure 5: Accuracy of baseline approach compared to one-level activation of *FEAGURE* (SVM). Each point represents a dataset. The dotted lines represent a 5 and 10 percent difference in accuracy

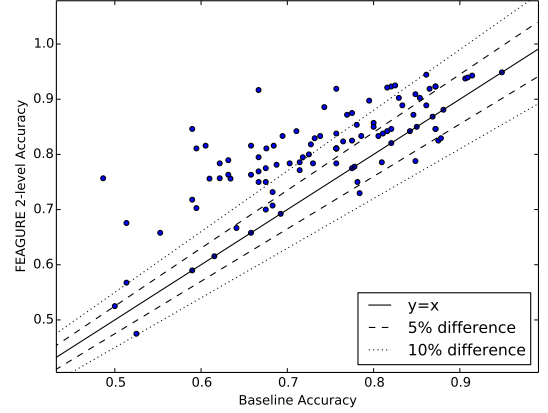


Figure 6: Accuracy of baseline approach compared to two-level activation of *FEAGURE* (SVM). Each point represents a dataset. The dotted lines represent a 5 and 10 percent difference in accuracy

collection “TechTC-25MAA.” These results show a much more pronounced increase in accuracy, and illustrate that we can, in general, rely on *FEAGURE* to yield positive features for difficult classification problems.

Table 3: Average accuracy over the 25 hardest datasets in terms of MAA. The columns specify the feature generation approach, with the baseline being no feature generation. The rows specify the induction algorithm used on the generated features for evaluation. Best results are marked in bold.

Dataset	Classifier	Baseline	Expander-FG	FEAGURE	FEAGURE 2-level
TechTC-25MAA	KNN	0.524	0.723 ( $p < 0.001$ )	<b>0.803</b> ( $p < 0.001$ )	0.795 ( $p < 0.001$ )
	SVM	0.751	0.815 ( $p < 0.001$ )	0.817 ( $p < 0.001$ )	<b>0.829</b> ( $p < 0.001$ )
	CART	0.82	0.839	0.837	<b>0.849</b> ( $p < 0.05$ )
TechTC-100	KNN	0.531	0.702 ( $p < 0.001$ )	0.772 ( $p < 0.001$ )	<b>0.775</b> ( $p < 0.001$ )
	SVM	0.739	0.782 ( $p < 0.001$ )	0.796 ( $p < 0.001$ )	<b>0.807</b> ( $p < 0.001$ )
	CART	0.81	0.815	0.814	<b>0.825</b> ( $p < 0.05$ )

As discussed in Section 3.3, *FEAGURE* creates a generic learning problem as part of its execution. For our main results we learned a decision tree classifier for this new induction problem. We also tested the effects of using K-NN and SVM classifiers instead. We note that this choice is orthogonal to that of the induction algorithm used to evaluate the generated features.

Table 4 shows the accuracies achieved with these induction algorithms. We see that in general, with the exception of an external K-NN classifier that uses an internal RBF-SVM, replacing the internal tree induction algorithm does not increase the accuracy of the resulting feature map.

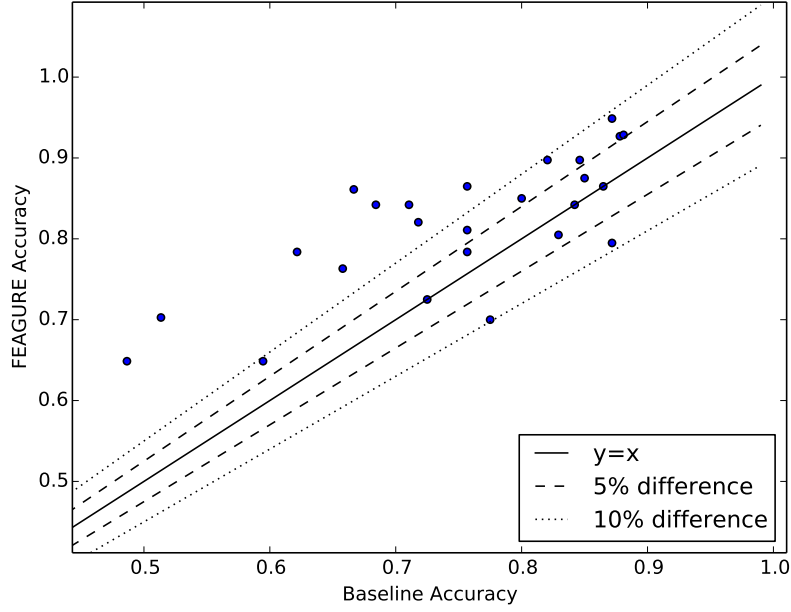


Figure 7: Accuracy of the baseline approach compared to single-level activation of *FEAGURE* (SVM). Displayed are the 25 hardest datasets (those with the lowest MAA)

Table 4: Average accuracy over all datasets. The columns specify the induction algorithm used in *FEAGURE* (SVM with linear or RBF kernel, K-NN). The rows specify the induction algorithm used on the generated features for evaluation. Entries marked with \* show a statistically significant p-value over the baseline accuracy

Dataset	Classifier	Baseline	Expander-FG	Tree	RBF SVM	5-NN
OHSUMED	KNN	0.777	0.756	0.769	<b>0.795*</b>	0.771
	SVM	0.797	0.804	<b>0.816*</b>	0.796	0.788
	CART	0.806	<b>0.814</b>	0.809	0.791	0.787
TechTC-100	KNN	0.531	0.702*	<b>0.772*</b>	0.689*	0.705*
	SVM	0.739	0.782*	<b>0.796*</b>	0.774*	0.774*
	CART	0.81	<b>0.815</b>	0.814	0.81	0.79

#### 4.4 Quantitative Analysis

To better understand the behavior of the *Deep-FEAGURE* algorithm, we measured its output and performed several aggregations over it. We first look at the average number of features considered by *Deep-FEAGURE* at every node: for TechTC-100, out of an average of 36.7 partitions by type that are considered for each feature, an average of 7.3 are expanded into new features by *FEAGURE*. The rest of the generated problems are discarded due to the filtering criteria mentioned in Section 3. We note that there are 46 available relations, and thus we can expect to look at no more than 46 features (if a relation does not apply to a sub-problem, it is not counted in this average).

When the depth of the search tree increases, the number of generated features decreases rapidly, as shown in Figure 8. This is unsurprising, as we know that increased depth will cause generated problems to have a smaller training set, increasing the likelihood of those features to be filtered out. Additionally, we see that the number of available features decreases in a roughly linear manner with depth. This is again unsurprising, as features deeper in the tree cannot easily make use of the same relation multiple times due to the orthogonality trait discussed in Section 3.4.

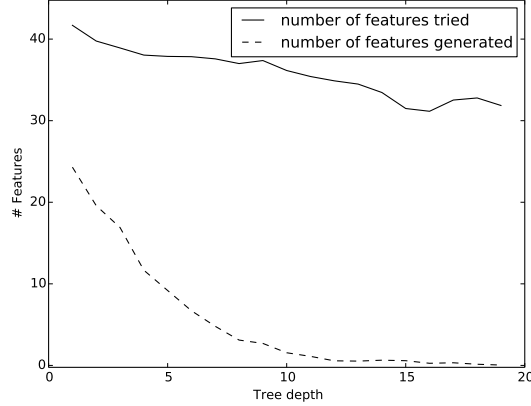


Figure 8: Average number of features tried vs average number of features generated per depth.

Let us now compare the relative size of recursive problems, to the existing induction problems. Figure 9 shows the ratio between the number of examples (size) in the newly constructed induction problem and the size of the original learning problem at various depths of the divide & conquer search process. As depth in the search tree increases, the size ratio increases as well. This is somewhat surprising, as an increase in depth means a smaller induction problem, and thus we would expect a similar size ratio to be maintained. Even more unexpected, however, is that the average **recursive** problem size increases with problem depth. Intuitively, we would have expected the recursive problem size to decrease as problem depth increases. One possible explanation to the unexpected increase in problem size is that as we search smaller problems deeper down the search process, the relations that result in a smaller or roughly similar sized recursive induction problem have already been used, and thus relations that cause a larger size ratio are required.

We also note that these recursive problems do not maintain the same label balance as the original learning problems: one label set is much larger. This is expected, as the divide & conquer search strategy aims to use the label set as a basis for separation. We may therefore wish to make use of various known strategies to reduce the impact of label imbalance in induction problems when using *Deep-FEAGURE*.

Finally, we look at the local information gain of the newly created features. In Figure 10, we see the mean information gain of generated features per depth, compared to the best information gained achieved by a non-recursive feature for that depth. We see that our recursive features tend to have a much higher information gain, especially in the beginning of the search process. We see a decrease in both measures as depth increases, because it is then more difficult to find distinguishing features.

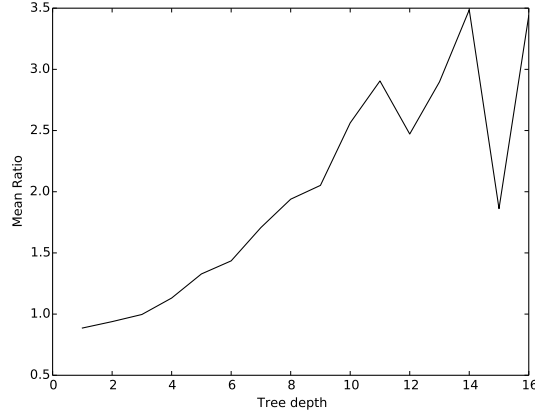


Figure 9: Mean size ratio of newly created recursive problem compared to the original learning problem size (new problem training set size divided by original problem training set size).

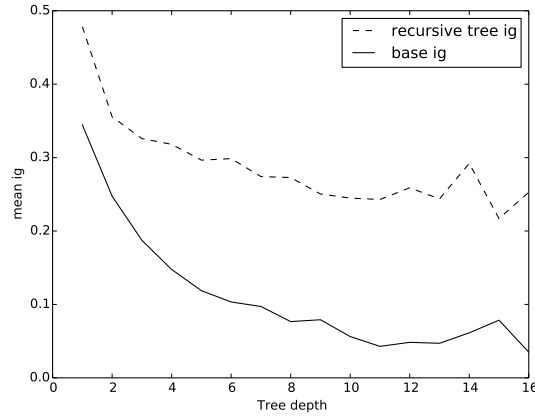


Figure 10: Mean information gain of generated features compared to the best information gain for the original learning problem.

## 4.5 Qualitative Analysis

To better understand the contribution of the features constructed by *FEAGURE*, let us look at a few generated features. We will consider the output of our algorithm on datasets from the TechTC-100 collection.

### 4.5.1 A SIMPLE EXAMPLE

For our first example, let us consider a single-level feature construction. In this example, we are attempting to separate texts regarding Virginia businesses from those discussing locations in Michigan. Early in the learning process, we apply *FEAGURE* to this problem, taking entities from the texts. We create a recursive problem over the subset of entities contained in the domain of the “owns” relation (most of which are organizations). Thus, this new problem (Figure 11) attempts

to separate organizations mentioned in texts regarding Virginia businesses from those mentioned in texts regarding locations in Michigan.

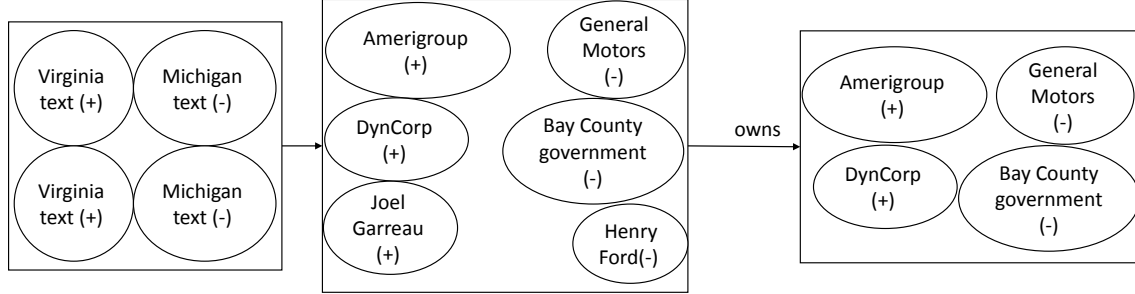


Figure 11: The left box represents the original learning problem. The middle box shows the entities extracted from the original problem. The right box represents the new learning problem containing entities that serve as keys for a specific relation.

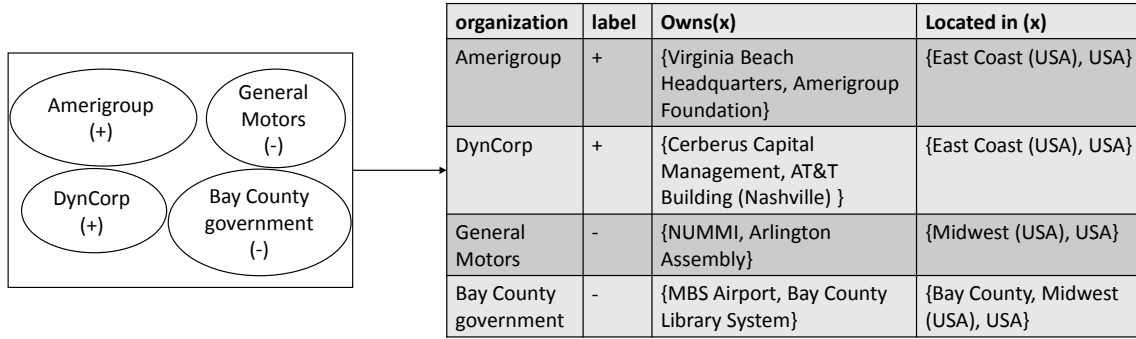


Figure 12: Recursive induction problem creation process: Features are created using the relational knowledge base.

At this stage, our algorithm has constructed a new learning problem, whose objects are organizations, labeled by the original learning problem. In the next step, the algorithm uses applicable relations, the “Owns” and “Located in” relations in this case, to convert the objects to feature vectors. Since an organization can own more than one property, and can be located in more than one location, these two features have multi-valued attributes. Figure 12 shows the resulting labeled feature vectors. This new learning problem is fed as input to an induction algorithm (in this case, a decision tree learner), resulting in the classifier shown in Figure 13. This classifier is used as a binary feature in the original domain, representing the concept of organizations located in the East Coast region of the United States. This is a powerful feature, as Virginia is an East Coast state, whereas Michigan is not. Therefore, texts regarding Virginia businesses are significantly more likely to link to organizations located on the East Coast than are texts regarding Michigan. We note that this particular feature was indeed generated by our algorithm and was later used by our external induction algorithm due to its high information gain.



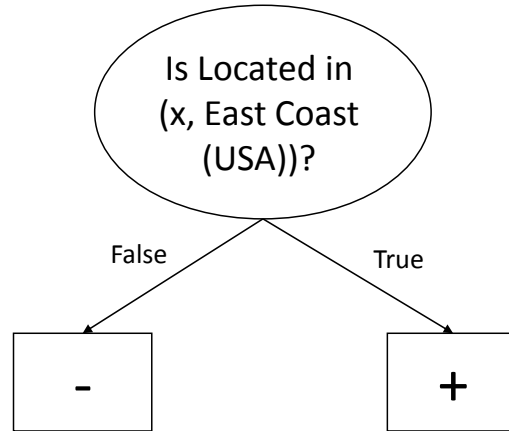


Figure 13: Feature constructed by *FEAGURE* for entities (organisations) in the “owns” relation.

#### 4.5.2 A MORE COMPLEX EXAMPLE

Our next example shows a slightly more complex feature within a single-level activation of *FEAGURE*. Here, we are attempting to separate texts that pertain to locations in California from those that pertain to locations in Alabama. Once again, entities are extracted, and this time, we take the subset of entities in the domain of the “lives in” relation. The resulting entities are people labeled according to the original tagging (Figure 14). To create a classifier for this problem, our algorithm uses all applicable relations (“Type”, “Citizen of” and “Died in”) to create features for the newly created induction problem. This process is demonstrated in Figure 15.

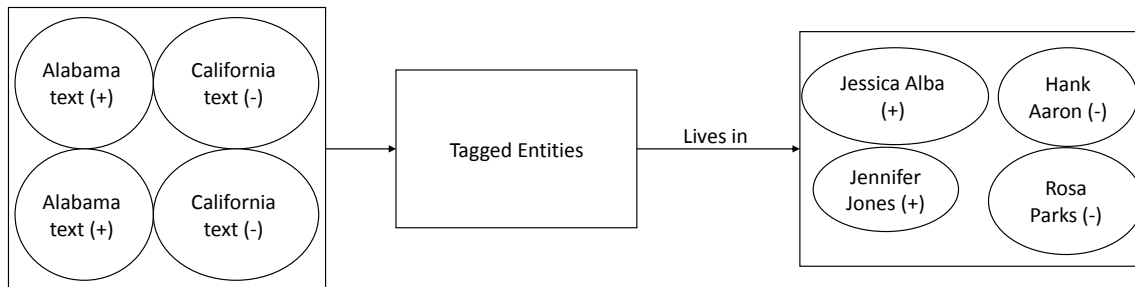
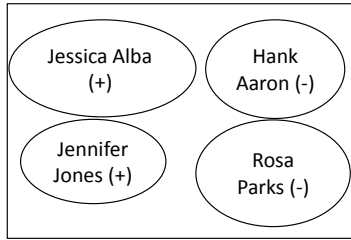


Figure 14: Entities are extracted and labeled. These entities are the new example set. The left box shows the original learning problem. The middle box represents entities extracted from the problem. The right box shows the new learning problem containing a subset of entities in a single domain.

Once this recursive learning problem has been constructed, our algorithm induces a decision tree classifier for it, yielding the recursive feature described in Figure 16. This feature is then applied as a binary feature for people in the text, and can be described as “is this person an actor, director or model?”. This feature is quite useful, as a person living in California is much more likely to belong to one of these categories than someone living in Alabama. This feature shows how type



Person	label	Type(x)	Citizen of (x)	Died in (x)
Jessica Alba	+	{wordnet_person, wordnet_actor, wordnet_model}	USA	N/A
Jennifer Jones	+	{wordnet_person, wordnet_actor}	USA	California
Hank Aaron	-	{wordnet_person, wordnet_baseball_player}	USA	N/A
Rosa Parks	-	{wordnet_person, wikicategory: civil_rights_activist}	USA	Michigan

Figure 15: Applicable relations are used as feature function for the new example set.

relations (also known as “is-a” relations or hypernyms) can be used to effectively group together several types of celebrity figures, namely actors, directors and models.

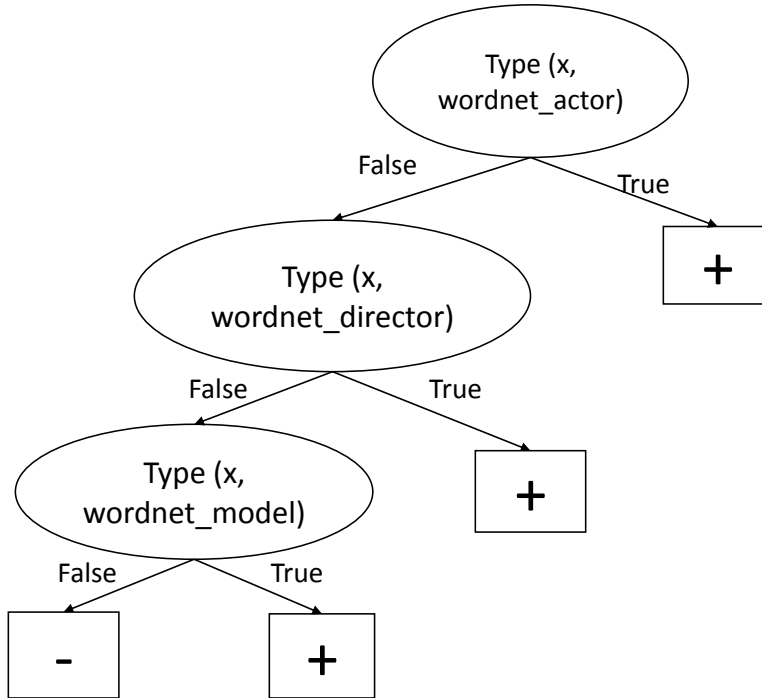


Figure 16: Feature constructed by FEAGURE for entities (people) in the “lives in” relation.

#### 4.5.3 RECURSIVE APPLICATION OF *FEAGURE*

In this example, texts refer either to locations in and around Texas, or to locations in and around New York. As before, entities are extracted and labeled. This time, locations are our entities, with the “Located in” relation as our domain (Figure 17). Applicable relations are used to then create a

new induction problem. *FEAGURE* uses the “Located in” and “Happened in” relations as features for this problem, as shown in Figure 18.

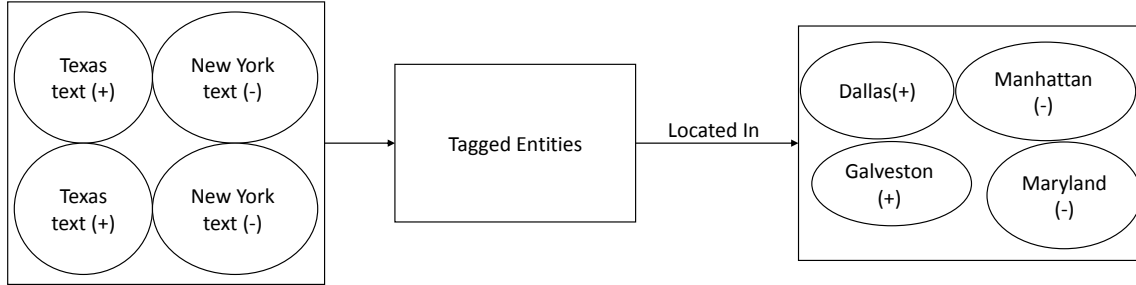


Figure 17: Entities are extracted from the text, and entities in the “Located in” relation are used as labeled objects.

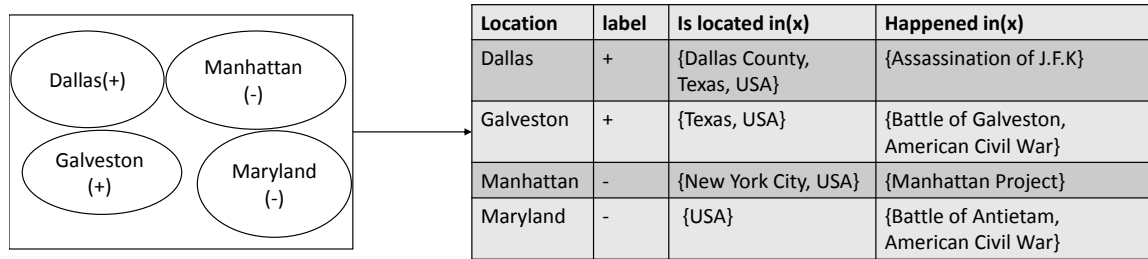


Figure 18: Construction of a recursive learning problem based on the “Located in” relation. Applicable relations are used to create a feature set for the newly constructed example set.

This feature set is not sufficient for creating a good classifier for our learning problem. Therefore, the algorithm calls *FEAGURE* recursively to try and generate new features for the new induction problem. The values of the feature “Happened in” are events. These events are used as objects for a recursive learning problem (Figure 19). We use the “Type” relation as a feature, relying on hypernyms to classify events (Figure 20). The resulting classifier (a decision tree induction algorithm was used) is shown in Figure 21, and can be interpreted as “is this event a battle or conflict?”.

Once we have generated this classifier on events, we can use it as a binary feature. *FEAGURE* uses this new feature to expand the constructed induction problem on locations, shown in Figure 18. This feature is applied to a location through a majority vote over events that happened in that location. The result is a feature for locations representing the concept “were most notable events in this location battles/conflicts?” Finally, a decision tree learner is used on the expanded feature set to learn a classifier on locations to be used as a feature for our original learning problem. The new classifier for locations is shown in Figure 22. It can be described as “is this location located in Texas, or the site of battles or conflicts?”. Texts mentioning locations in and around Texas are more likely to link to locations that correspond to the output of this classifier. We note that this feature was generated by *FEAGURE*, and was later used by our external induction algorithm due to its high information gain.

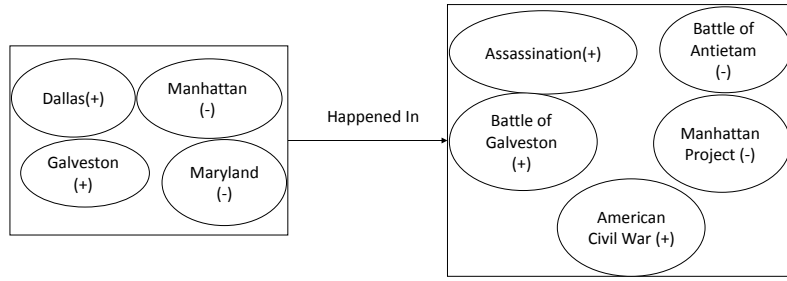


Figure 19: Construction of a second level recursive learning problem based on the “Happened in” relation. Feature values are treated as objects and labeled according to the labels of the problem on locations.

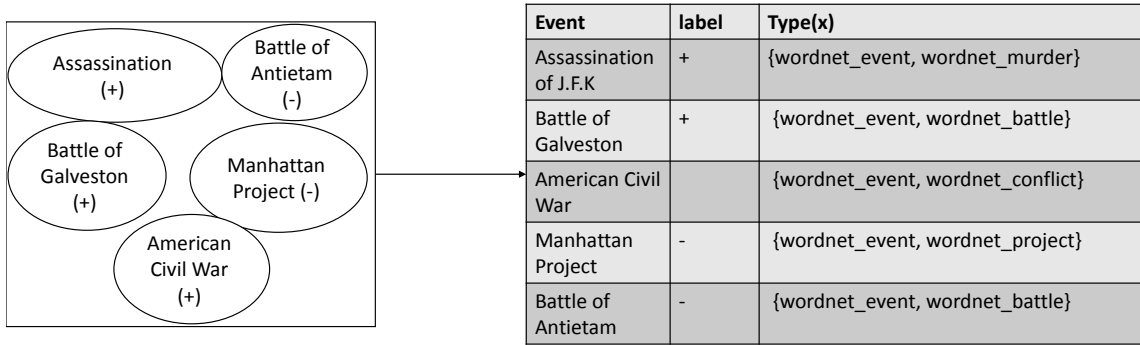


Figure 20: Construction of a recursive learning problem based on the “Happened in” relation. Once the example set has been created, applicable relations are used to create a feature set for the newly constructed induction problem.

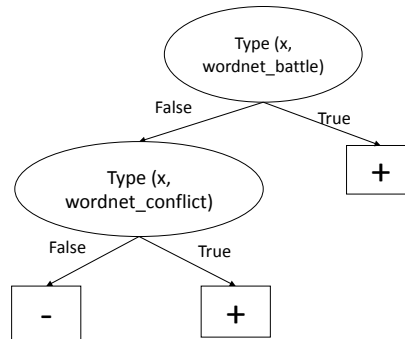


Figure 21: Recursive feature constructed by *FEAGURE* for entities in the “happened in” relation. This feature operates on events, and can be used in a classifier on locations.

To conclude, we see that in general, good features created by *FEAGURE* are those that are sufficiently broad in scope to group together a sizeable subset of similar entities. To this end, hypernyms can be very useful, as they group together entities in a taxonomy.

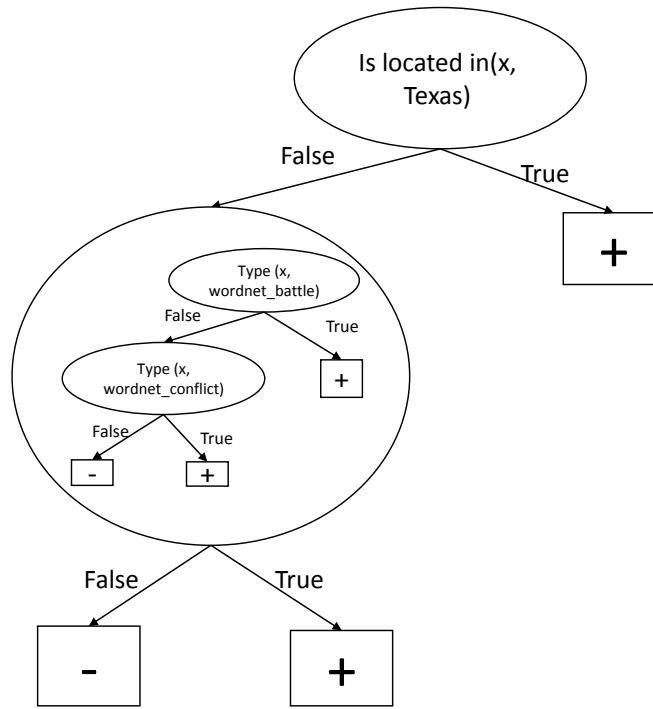


Figure 22: Final generated feature constructed by *FEAGURE* for entities in the “located in” relation. The feature in the left branch is the recursive feature constructed by applying *FEAGURE* to the new learning problem.

## 5. Related Work

Many feature generation methodologies have been developed to search for new features that better represent the target concepts. There are three major approaches for feature generation: tailored methods, combinational techniques and algorithms utilizing external knowledge.

Tailored approaches (Sutton & Matheus, 1991; Hirsh & Japkowicz, 1994) are designed for specific problem domains and rely on domain-specific techniques. One such example is the bootstrapping algorithm (Hirsh & Japkowicz, 1994), designed for the domain of molecular biology. The algorithm represents features as nucleotide sequences whose structure is determined by existing background knowledge. The algorithm uses an initial set of feature sequences, produced by human experts, and uses a domain-specific set of operators to change them into new sequence features. Such special-purpose algorithms may be effectively tailored for a given domain, but have proven difficult to generalize to other domains and problems.

Combinational feature generation techniques are domain-independent methods for constructing more descriptive features by combining existing features in various ways. The LDMT algorithm (Brodley & Utgoff, 1995), for example, performs feature construction in the course of building a decision-tree classifier. At each created tree node, the algorithm constructs a hyperplane feature through linear combinations of existing features in a way likely to produce concise, relevant hyperplanes. The LFC algorithm (Ragavan et al., 1993) combines binary features through the use of logical operators such as  $\wedge$ ,  $\neg$ . The FICUS algorithm (Markovitch & Rosenstein, 2002) allows the

use of any combinational feature generation technique, based on a given set of constructor functions. Recent work by Katz et al. (2016) uses a similar approach.

*Deep Learning* (Rumelhart et al., 1986; LeCun et al., 1998) is another major class of combinational feature generation approaches. Here, the activation functions of the nodes can be viewed as feature schemes, which are instantiated during the learning process by changing the weights.

One limitation of combinational approaches is that they merely combine existing features to make the representation more suitable for the learning algorithm. Our *FEAGURE* algorithm belongs to a third class of approaches that inject additional knowledge into the existing problem through the feature generation process.

Propositionalization approaches (Kramer & Frank, 2000; Cheng et al., 2011) rely on relational data to serve as external knowledge. They use a combination of operators to create first-order logic predicates connecting existing data and relational knowledge. Cheng et al. (2011) devised a generic propositionalization framework using linked data via relation-based queries.

FeGeLOD (Paulheim & Fümkrantz, 2012) also uses linked data to automatically enrich existing data. FeGeLOD uses feature values as entities and adds related knowledge from a relational knowledge base as new features to the existing examples. By doing so, FeGeLOD extends the feature set.

While these unsupervised approaches allow us to utilize external knowledge, any attempt to construct deep connections and relationships within the knowledge base will result in an exponential increase in the number of generated features. To avoid this issue, *FEAGURE* and other supervised approaches use the presence of labeled examples to better generate deeper features.

Most supervised methods can trace their source to *Inductive Logic Programming (ILP)* (Muggleton, 1991), a supervised approach that induces a set of first-order logical formulae to separate the different categories of examples in the training set. ILP methods do so by starting from single relation formulae and adding additional relational constraints using the knowledge base, until formulae that separate the training set into positive and negative examples are found. To that end, these approaches make use of a *refinement operator*. When applied on a relational formula, this operator creates a more specialized case of that formula. For example, given the logical formula  $BornIn(X, Y)$ , where  $X$  is a person and  $Y$  is a city, one possible refinement is the formula  $BornIn(X, Y) \wedge CapitalOf(Y, Z)$ , where  $Z$  is a country. The result is a logical formula that considers a more specific case. Additionally, we can look at a refinement that restricts by a constant, turning  $BornIn(X, Y)$  into, for example,  $BornIn(X, United\ States)$ . This refinement process continues until a sufficient set of consistent formulae is found.

*Relational Learning* techniques attempt to approach the problem of expanding knowledge from a different angle. These techniques are designed to expand an existing relational database. One such technique is View Learning (Davis et al., 2007). View learning uses labeled data to generate new relational tables from the existing relational knowledge. It begins by applying ILP on relational data to construct new, simple, relational tables. Then, using a Bayesian network induction algorithm, it learns ways to combine these relations into a single, more complex table. This approach bears similarity to our approach in that it can effectively utilize background knowledge to draw conclusions using the combination of labeled data and background knowledge. Unlike our approach, these constructed tables attempt to fit new knowledge to existing examples, whereas our approach re-frames the learning problem within a new context.

The *dynamic feature generation* approach used by the SGLR algorithm (Popescul & Ungar, 2007) can be seen as the supervised equivalent of propositionalization methods. Instead of creating

first-order predicates a priori, feature generation is performed during the training phase, allowing for complex features to be considered by performing a best-first search on possible candidates. This process allows SGLR to narrow the exponential size of the feature space to a manageable number of candidates. While this supervised approach overcomes the exponential increase in features that unsupervised approaches suffer from, the space of generated features that it searches is significantly less expressive than that of our approach. Through the use of a recursive induction algorithm, our approach automatically locates relationships and combinations that we would not otherwise consider when using other approaches.

Kernel-based knowledge injection techniques aim to make use of powerful machine learning algorithms such as SVM by decoupling the knowledge base from the induction algorithm. This is done through the use of a kernel function that acts as a similarity function between examples in the problem domain. Lösch et al. (2012), for example, devised an approach that attempts to learn graph kernels within the relational domain. This process creates strong kernel functions that look at the neighbourhood of an entity in search of similarities between entities. This approach fundamentally differs from our own. Kernel methods attempt to create a filter through which entity similarity can be measured. In our approach, we attempt to locate the target concept through the use of different domains. This distinction is especially relevant to classification tasks, where many possible domains can apply for each example, and similarity may be difficult to find when considering the entire example set.

Another known method of utilizing external knowledge is Explanation-based Learning (Mitchell et al., 1986; DeJong & Mooney, 1986), which uses a knowledge base of logical assertions to locate a small set of logical conditions that capture the target concept through deduction on examples from that concept. This approach uses deduction, rather than induction, as its main tool, allowing a logical knowledge base to be leveraged effectively. However, due to the difficulty in creating good knowledge bases of logical assertions, this approach has shown limited success.

Terziev (2011) shows an interesting approach to supervised feature generation. He proposes a decision tree based approach, where in each node of the tree, feature expansion is done using a method similar to that of FeGeLOD, with an entropy-based criterion to decide whether further expansion is required. This technique bears several similarities to *Deep-FEAGURE* (Algorithm 3). However, the feature expansion process is unsupervised, and the resulting feature must be a decision tree, restricting the generality of the approach.

Since we have focused on the problem of text classification, we also discuss a few text-based approaches for feature generation. Linguistic methods such as those described in a study by Moschitti and Basili (2004) attempt to use part-of-speech and grammar information to generate more indicative features than the bag-of-words representation of texts. Concept-based approaches are the most similar to our own. Two examples of such methods are Explicit Semantic Analysis (ESA) (Gabrilovich & Markovitch, 2009), which generates explicit concepts from Wikipedia based on similarity scores, and Word2Vec (Mikolov et al., 2013), which generates latent concepts based on a large corpus. Both approaches provide us with a feature set of semantic concepts. These concepts can be used alongside our approach, allowing us to induce over them.

## 6. Conclusions

When humans use inductive reasoning to draw conclusions from their experiences, they use a vast amount of general and specific knowledge. This use of knowledge allows us to reach better, more

general conclusions. In this paper we introduced a novel methodology for enhancing existing learning algorithms with background knowledge represented by relational knowledge bases. The algorithm works by generating complex features induced using this external knowledge. It constructs recursive learning problems based on existing features and the knowledge base. These newly constructed problems are given as input to an induction algorithm. The output of this process is a collection of classifiers that are then turned into features for the original induction problem.

An important strength of our approach is its generality. The features generated by *FEAGURE* can be used by any induction algorithm, allowing us to inject external knowledge in a general, algorithm independent manner. One potential limitation of our approach is that it requires features with meaningful values. Nonetheless, for a wide range of problems, feature values do have meaning. For these problems, we can apply one of several general and domain-specific knowledge bases, depending on the problem.

In recent years, we have seen an increase in the number of available knowledge bases. These knowledge bases include general knowledge bases such as Freebase, YAGO, Wikidata and the Google Knowledge Graph, and more domain-specific knowledge bases, from the British Geographical Survey (BGS) linked dataset, containing roughly one million geological facts regarding various geographical formations, through biological databases such as Proteopedia, composed of thousands of pages regarding biological proteins and molecules, to entertainment-focused databases such as IMDB, containing millions of facts on movies, TV series and known figures in the entertainment industry. With the recent surge in well-formed relational knowledge bases, and the increasing prevalence of strong learning algorithms for a wide variety of tasks, we believe our approach can take the performance of existing machine learning techniques to the next level.



## References

- Bast, H., Baurle, 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.
- Bollacker, K. D., Evans, C., Paritosh, P., Sturge, T., & Taylor, J. (2008). Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1247–1250. AcM.
- Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Wadsworth.
- Brodley, C. E., & Utgoff, P. E. (1995). Multivariate decision trees. *Machine Learning*, 19(1), 45–77.
- Cheng, W., Kasneci, G., Graepel, T., Stern, D. H., & 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.
- Debnath, A. K., Lopez, d. C. R., Debnath, G., Shusterman, A. J., & Hansch, C. (1991). Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity.. *Journal of medicinal chemistry*, 34(2), 786–797.
- DeJong, G., & Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine learning*, 1(2), 145–176.
- Fix, E., & Hodges Jr, J. L. (1951). Discriminatory analysis-nonparametric discrimination: consistency properties. Tech. rep., DTIC Document.
- Gabrilovich, E., & Markovitch, S. (2009). Wikipedia-based semantic interpretation for natural language processing. *Journal of Artificial Intelligence Research*, 34, 443–498.
- Hersh, W. R., Buckley, C., Leone, T. J., & Hickam, D. H. (1994). OHSUMED: an interactive retrieval evaluation and new large test collection for research. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 192–201. Springer.
- Hirsh, H., & Japkowicz, N. (1994). Bootstrapping training-data representations for inductive learning: A case study in molecular biology. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pp. 639–644.
- 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.

- Hoffart, J., Yosef, M. A., Bordino, I., Fürstenau, H., Pinkal, M., Spaniol, M., Taneva, B., Thater, S., & Weikum, G. (2011). Robust disambiguation of named entities in text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 782–792. Association for Computational Linguistics.
- Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pp. 137–142. Springer.
- Katz, G., Shin, E. C. R., & Song, D. (2016). Exploreskit: Automatic feature generation and selection. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pp. 979–984. 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.
- McNamara, D. S., & Kintsch, W. (1996). Learning from texts: Effects of prior knowledge and text coherence. *Discourse processes*, 22(3), 247–288.
- Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119.
- Milne, D., & Witten, I. H. (2013). An open-source toolkit for mining wikipedia. *Artificial Intelligence*, 194, 222–239.
- Mitchell, T. M., Keller, R. M., & Kedar-Cabelli, S. T. (1986). Explanation-based generalization: A unifying view. *Machine learning*, 1(1), 47–80.
- Moschitti, A., & Basili, R. (2004). Complex linguistic features for text classification: A comprehensive study. In *European Conference on Information Retrieval*, pp. 181–196. Springer.
- Muggleton, S. (1991). Inductive logic programming. *New generation computing*, 8(4), 295–318.
- Paulheim, H., & Fümkrantz, 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..
- Popescul, A., & Ungar, L. H. (2007). 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.
- 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.
- Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error-propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1*, Vol. 1, pp. 318–362. MIT Press, Cambridge, MA.

- Sutton, R. S., & Matheus, C. J. (1991). Learning polynomial functions by feature construction.. In *ML*, pp. 208–212.
- Terziev, Y. (2011). Feature generation using ontologies during induction of decision trees on linked data..
- Vrandečić, D., & Krötzsch, M. (2014). Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10), 78–85.
- Wale, N., & Karypis, G. (2006). Acyclic subgraph based descriptor spaces for chemical compound retrieval and classification. Tech. rep., DTIC Document.