Recursive Feature Generation for Knowledge-based 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 algorithms are constrained to using knowledge within the supplied feature vectors. In recent years, a large collection of both general and domain-specific knowledge bases have become increasingly 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 supervised 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 then solve the constructed learning task. The resulting classifier is then used to create new features for the original problem. We have applied our algorithm to the domain of text categorization, using large semantic knowledge bases such as Freebase. We have shown that generated 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 of these methods rely on the inductive approach: given a set of labelled examples, they attempt to locate a hypothesis that is heavily supported by the known data. These methods have proven themselves successful in cases where there is a sufficient number of examples, 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 the difficulty resulting from an insufficiently expressive set of features, is to generate new features based on existing ones. These feature generation approaches attempt to combine the given feature set in an effort to create better, more discriminatory features with regards to the induction problem at hand. As an example of this, the LFC algorithm (Ragavan et al., 1993) combines binary features through the use of logical operators such as $\wedge, \neg$.

These feature generation methods all provide us with ways to enhance the performance of induction algorithms through intelligent combinations of existing features. While this approach often suffices, there are many cases where merely combining existing features is not sufficient. In such cases, we require knowledge external to the problem in order to solve it effectively. Supposed, for example, that the induction problem at hand is to identify people at risk of the genetic disorder Tay-Sachs [1]. The training set contains examples of people that have developed Tay-Sachs, and examples of people that did not. Since the concept is related to a patient's country of origin, a human expert might look at surnames of former patients, and using his background knowledge regarding them would link them to geographical locations, allowing him to identify potential patients with

1. A recessive genetic disorder prevalent mostly among Ashkenazi Jews

high accuracy. A machine learning algorithm, on the other hand, would do poorly, as it lacks the knowledge required to make that link.

One known method of utilizing external knowledge is known as Deductive Learning (Mitchell, 1982; DeJong & Mooney, 1986). This school of learning methods uses a knowledge base of logical assertions in order to locate a small set of logical conditions that capture the target concept based on a examples from that concept. This approach does so using deduction, rather than induction, as its main tool, allowing us to leverage a logical knowledge base effectively. There are two main concerns that must be addressed for deductive learning to be effective:

1. Extensive knowledge bases comprised of logical assertions are difficult to find and create.
2. This approach forfeits the progress and power gained by the use of well-known and extensively researched inductive methods.

In order to combat these issues, we can make use of the extensive relational knowledge bases that have been constructed as part of the Semantic Web project (see survey, Bizer, Heath, & Berners-Lee, 2009). These knowledge bases contain type-annotated entities covering a multitude of domains that are connected via semantically meaningful relations. Thus, the question is how relational knowledge can be used to enhance existing induction methods. There have been several efforts in utilizing Linked Data for unsupervised feature generation. Cheng et al. (2011) devised a framework for constructing features from linked data. By using intelligent queries, it is possible to construct features that utilize the knowledge base in a meaningful way. 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.

In this work, we build upon these existing approaches in order to present a new supervised methodology for generating complex features based on a relational knowledge base. Our algorithm recursively constructs new learning problems from existing feature values, then uses knowledge bases such as the Semantic Web to construct the features for the new learning problem. Using common induction algorithms, we can then construct a classifier that serves as a feature for the original problem, giving us a complex feature. The contributions of this approach are threefold. First, usage of training data allows for an automated, concept-driven exploration of the large space of possible features, in a manner similar to deductive approaches. Second, this approach can be applied recursively within the constructed learning problem, allowing for easier discovery of complex features. Third, it can effectively utilize one-to-many relations, which are more difficult to use in unsupervised approaches.

2. Motivation

Before we delve into the exact 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 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. We call this learning problem T_1. Assuming we have no additional information, an induction algorithm would likely produce a result similar to that shown in Figure 1. While such a classifier will achieve a low

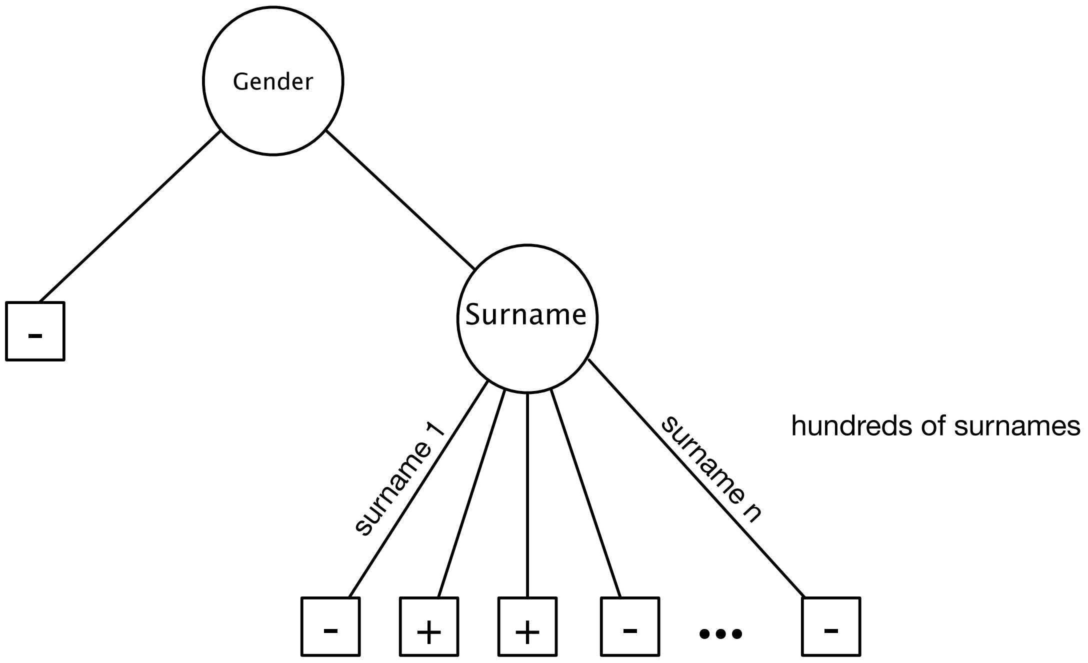


Figure 1: A decision tree for the basic features

training error, the hundreds of seemingly unrelated surnames will cause it to generalize poorly. This phenomenon is known as over-fitting, that is, the model is too adjusted to known data and cannot generalize. We can intuitively see this in Figure 1, as each leaf hypothesis is supported by only a single example.

The above example illustrates that without additional knowledge, an induction algorithm will yield a very poor result. Even if we attempt to use standard feature generation approaches, there is no additional information that combinations of these features would find. 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 knowledge-based feature generation techniques to the problem, as we can move from the domain of surnames to that of countries.

Our goal is to separate this set of people to those at high risk and those at low risk using induction algorithms. We see that the existing feature set is insufficient to perform this task successfully. Therefore, our approach aims to construct new features for the learning task T_1 by recursively creating a new learning problem T_2. Once we have done so, we can use induction methods on T_2 in order to create a classifier that can be used in T_1. We construct T_2 as follows: The training objects are surnames; surnames of people with the disease are labelled as positive. The features for these new objects are extracted from the knowledge base, they are their countries of origin. Solving the above learning problem through an induction algorithm yields a classifier on surnames that distinguishes between surnames of patients with the disease and surnames of healthy individuals. This classifier for T_2 can then be used as a binary feature in the original problem T_1 by applying it

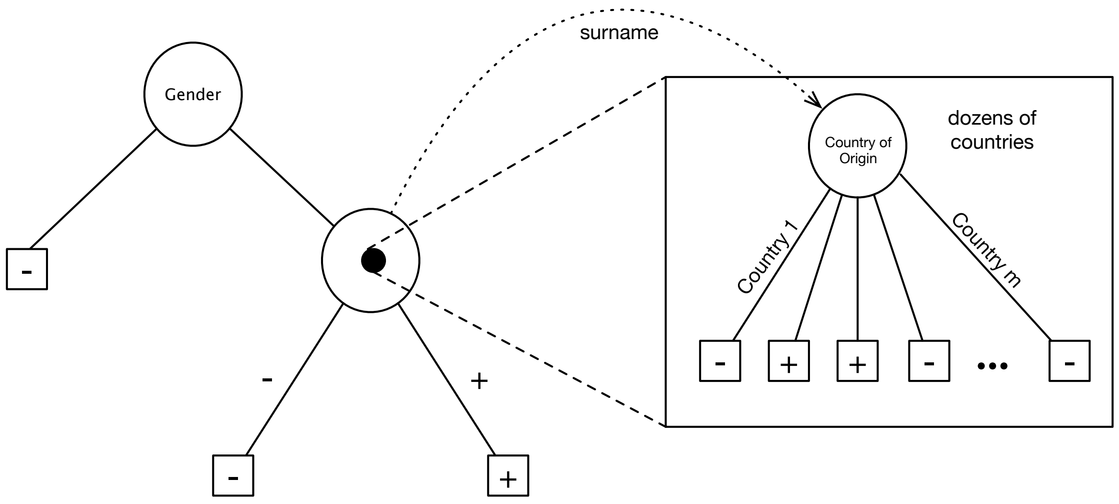


Figure 2: A constructed feature used within a decision tree

to the feature value of surname. 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 over the baseline solution, as we now abstract the long list of surnames to a short list of countries. We can see this as nodes in the classifier for T_2 are derived from multiple surnames. 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, and some countries may not have sufficient representation to induce an accurate classifier.

Once again, we require additional knowledge in order to capture the target concept of T_2. Therefore, we can recursively apply our method while trying to learn T_2 in an attempt to better locate the target concept and induce a classifier for T_1. We create a new training set, the objects of which are countries of origin, with countries of surnames belonging to people with high risk are labelled as positive. The knowledge base regarding countries is then used to construct features for this new this training set, giving us a recursive learning problem T_3.

We can then use T_3 as an input to an induction algorithm. As a result of this process, we end up with a classifier for T_3, which tries to separate 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 reach the conclusion that countries with high average temperature and low precipitation: both being characteristic of desert areas. This classifier is then used on the country of origin feature in T_2, yielding a new feature for T_2. This new feature can then be used when inducting a classifier for T_2, which, as we have mentioned earlier, is used as a feature for the original problem T_1.

The complete 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, as the construction process is guided by the target concept, in a manner similar to that of deductive approaches. While this is a simple example, it illustrates how an unsolvable learning problem given the original features can be solved via the use of complex additional knowledge and existing induction methods.

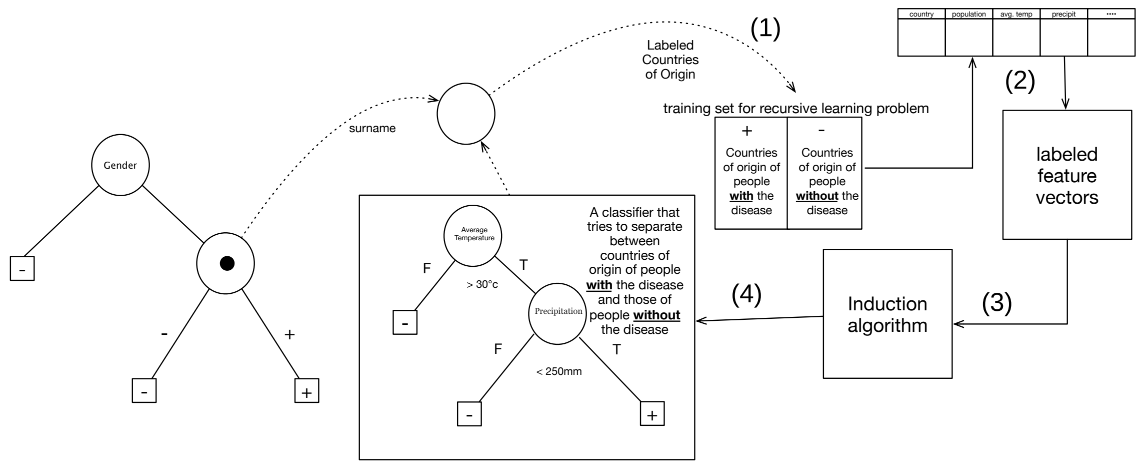

(1)
Labeled Countries of Origin
surname
Gender
training set for recursive learning problem
+
Countries of origin of people **with** the disease
-
Countries of origin of people **without** the disease
country
population
avg. temp
precipit
....
(2)
labeled feature vectors
(3)
Induction algorithm
(4)
A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

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.

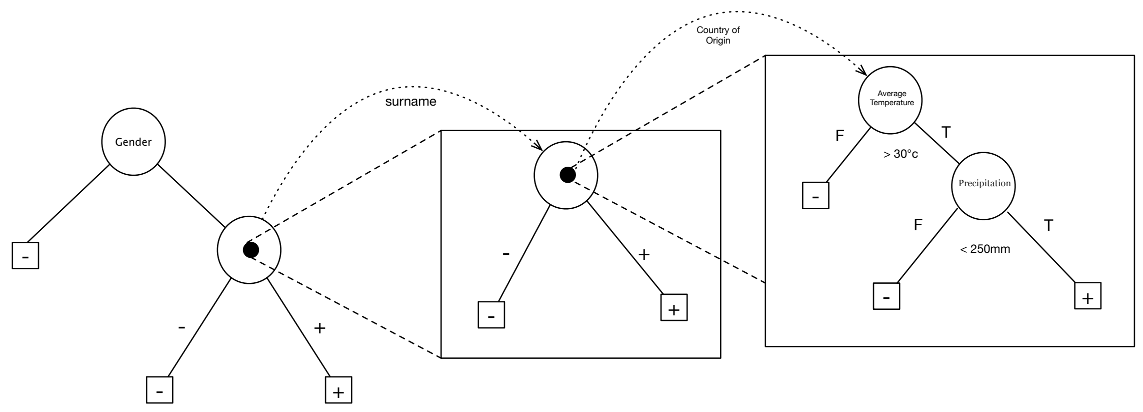


Figure 4: A two-level constructed feature used within a decision tree

3. Generating Features through Recursive Induction

We begin our discussion with the standard definition of an induction problem. Let O be a set of objects. Let $Y = \{0, 1\}$ be a set of labels [2]. Let $C : O \rightarrow Y$ be the target concept. Let $S = \{(o_1, y_1), \ldots, (o_m, y_m)\}$ be a set of labelled examples such that $o_i \in O, y_i \in Y, C(o_i) = y_i$. Let $F = \{f_1, \ldots, f_n\}$ be a *feature map*, a set of *feature functions* $f_i : O \rightarrow I_i$. This definition implies a training set represented by feature vectors: $S_F = \{(\langle f_1(o_i), \ldots, f_n(o_i)\rangle, y_i)|(o_i, y_i) \in S\}$. 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 outputted 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 $(x, y) \sim D$.

2. We assume binary labels for ease of discussion

In this work, we assume that in addition to S_F we also have a set of binary [3] relations $\mathcal{R} = \{R_1, \ldots, R_t\}, R_j : D_j \times D_{j'}$ representing our knowledge base.

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

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 $(x, y) \sim D$.

The implications of this definition are twofold: First, if the classifier induced by L given $\langle S, F \rangle$ yields a low generalization error, there is little to be gained through feature generation. Second, in order for the utility of A to be positive, the generated feature set F'_A must yield lower generalization error than F.

When using a feature generation algorithm using a set of relations $\mathcal{R}$, we would like to achieve a higher utility than that achieved without the use of $\mathcal{R}$. Clearly, in order to do so we must require some connection between $\mathcal{R}$ and our feature set F. To that effect, we add an additional requirement: $\bigcup_{f_i} I_i \cap \bigcup_{R_j} D_j \neq \emptyset$.

3.1 An unsupervised approach to knowledge-based feature generation

To begin with, we shall try to ignore the labels, and define an unsupervised feature generation algorithm based on $\mathcal{R}$. We call this algorithm *Shallow*, as it uses the relations in a shallow manner to create features. Using *Shallow*, we can highlight several important concepts of our approach.

Let us first consider the case where any given relation R_j is a function $R_j : D_j \rightarrow D_{j'}$. In this case, for any given f_i, R_j such that $Image(f_i) \subseteq D_j$, we can compose R_j onto f_i, giving us a function $f_{i,j}(x) = R_j \circ f_i = R_j(f_i(x)), f_{i,j} : O \rightarrow D_{j'}$. 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. Under this condition, we can define the new feature set using a closed formula: $\{f_{i,j} | Image(f_i) \subseteq D_j\}$.

Now that we have considered the simple case, we can consider more complex cases. For one, we must consider the case where $Image(f_i) \not\subseteq D_j$ but $Image(f_i) \cap D_j \neq \emptyset$. In this case, there are values for which R_j cannot be applied. We can therefore treat R_j as a partial function from $Image(f_i)$ to $D_{j'}$. In order to complete this partial function, let us first mark *undefined* as $\perp$. We can turn R_j into a full function with regards to $Image(f_i)$ as follows: $\tilde{R}_j(x) = \begin{cases} R_j(x) & \text{if } x \in D_j \\ \perp & \text{otherwise} \end{cases}$. The result is a full function $\tilde{R}_j : Image(f_i) \cup D_j \rightarrow D_{j'} \cup \{\perp\}$. We also note that the same process can be applied in cases where R_j has missing values, meaning it is a partial function with regards to D_j

Although there are many relations which are also functions, such a requirement may serve as a limitation. For example, a relation mapping a surnames to their countries of origin, as in section 2, may have multiple countries of origin for a single surname. Similarly, a relation mapping a parent to their children may have many children for the same parent. In cases such as this, composing R_j onto F_i may yield a set of values, requiring an aggregation function to be used should we wish to

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

utilize it effectively. There are many possible aggregation functions that we can apply on such a value set, but we shall discuss three major ones:

- All- Given a set of values, whether they all fulfil a certain predicate $p : D_{j'} \to \{0,1\}$.
- Exists- Given a set of values, whether there exists a value in that set which fulfils a certain predicate p.
- Majority- Given a set of values, whether a majority of them fulfil a certain predicate p.

All of the above require a predicate function $p : D_{j'} \to \{0,1\}$. As an example, we can check whether a constant $c \in D_{j'}$ is within the values of the set by using $p(x) = \begin{cases} 1 & \text{if } x = c \\ 0 & \text{otherwise} \end{cases}$ alongside the "exists" aggregation method.

In some situations, we would like to work with binary features rather than multi-valued ones. In such cases, we can restrict the constructed $f_{i,j}$ by a constant, yielding $f_{i,j,c}(x) = \begin{cases} 1 & \text{if } f_{i,j}(x) = c \\ 0 & \text{otherwise} \end{cases}$. In the more complex cases, we must consider how to handle missing values and multiple values. To simplify, we can handle $\perp$ as false and use the "exists" aggregation condition to handle these cases.

Now that we have considered all possible behaviours of a given relation R_j, we can proceed to define the *Shallow* algorithm. Given a feature f_i, *Shallow* will go over the relations within the knowledge base and attempt to create features $f_{i,j,c} : O \to D_{j'} \cup \{\perp\}$ through the discussed techniques. *Shallow* then gathers all generated features and outputs them. The pseudo-code is given below.

Algorithm 1 *Shallow*: Non-recursive Feature Generation using relations

```
function CREATEFEATURES($S$,$F$, $\mathcal{R}$)
    $F_{result} = \emptyset$
    for $f_i \in F$ do
        for $R_j \in \mathcal{R}$ do
            if $Image(f_i) \cap D_j = \emptyset$ then
                Continue
            else if $R_j : D_j \to D_{j'}$ and $Image(f_i) \subseteq D_j$ then
                $F_{result} = F_{result} \cup \{f_{i,j} = R_j \circ f_i\}$
            else if $R_j : D_j \to D_{j'}$ and $Image(f_i) \cap D_j \neq \emptyset$ then
                $F_{result} = F_{result} \cup \{f_{i,j}(x) = \begin{cases} R_j(f_i(x)) & \text{if } f_i(x) \in D_j \\ \perp & \text{otherwise} \end{cases}\}$
            else                                        ▷ $R_j$ is a relation $R_j : D_j \times D_{j'}$
                $F_{result} = F_{result} \cup \{Aggregate(p, f_{i,j})\}$
    return $F_{result}$
```

We see that this approach allows us to generate features from our knowledge base, regardless of the ways in which it interacts with our existing features. One example of such features is that of semantic type, features making use of the "IS-A" relation to note the entity type. We note that this approach creates a potentially large number of features, and that a given feature may have a large number of values. While this approach does allow us a degree of generalization, it can only utilize our knowledge base shallowly, from base entities to their values. Should we try to apply a chain of relations rather than a single one, we can begin to explore more complex domains. However, we must consider the exponential nature of such a chain. Without any way to guide our search through the knowledge base, we are likely to generate and increasingly large number of relations, many of which will be irrelevant.

3.2 Creating Learning Problems using Relations

Now that we have seen how an unsupervised algorithm would make use of a knowledge base for feature generation, we may begin to properly discuss our approach. To do so, we must understand how our approach creates new induction problems using the knowledge base.

Given an existing feature $f_i : O \to I_i$, our algorithm will formulate a new learning task trying to separate values in $Image(f_i)$ associated with positive examples of the original learning task from those appearing in negative ones. The result of the new learning task will be a classifier $h_i : I_i \to Y$ which we can then use to label elements in I_i, such as those appearing in 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). The full pseudo-code is described in algorithm 2.

In order to explain how we create such h_i, let us first consider the creation of a new learning problem. Once we have done so, we can use any induction algorithm to create h_i. Given a feature f_i, we first 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 use $v_i(S)$ as our set of objects. To define a learning problem, we must consider their labels.

In the simple case, every feature value appears only for a single example in S. In that case, we can simply take the same label, with the result being $label(v) = y$ where $f_i(x) = v, (x, y) \in S$. In the above example, if each patient has a unique surname, each surname would be labelled according to that patient's label. Most features in general, and in machine learning tasks in particular, however, do not have a unique value for each example. In such cases we would have multiple examples in S such that $f_i(x) = v$. Let us call $S(v)$ the set of such examples. In order for us to give a label for v, we must aggregate the labels of objects in $S(v)$ somehow. As in section 3.1, we can consider three major aggregation functions. Using the All aggregation function results in a strict labelling, as a value will be classified as belonging to the target concept only if all examples in which that value has appeared are labelled positive. The result is the consistent label function: $label(v) = \begin{cases} 1 & \text{if } \forall (x, y) \in S(v) : y = 1 \\ 0 & \text{otherwise} \end{cases}$. A more lenient label function is the majority label, taking the Majority aggregation function: $label(v) = majority(\{y|(x, y) \in S(v)\})$. In the intro example, the consistent label would take the set of patients with the given surname, and label that surname as having high risk only if all such patients have the disease. The majority label, in comparison, would check whether most patients with that name have the disease.

We now formulate a new learning problem with the constructed training set $S_i' = \{(v, label(v))|v \in v_i(S)\}$. To fully define our learning problem, we must specify a feature map over $v_i(S) \subseteq Image(f_i)$. Consider a relation R_j such that $D_j \cap v_i(S) \neq \emptyset$. For such R_j, if it is a function, we can simply use it as a feature function. If R_j is a partial function, we simply have a feature with missing values. Finally, if R_j is not a function, we can use aggregation as in section 3.1 to create features.

Once we have considered all relations in $\mathcal{R}$, we can generate a feature map for S_i'. Given R_j, assuming it is relevant to the problem domain, meaning $v_i(S) \cap D_j \neq \emptyset$, we can utilize it as a feature. We then look at all constants $c \in \{R_j(v)|v \in v_i(S) \wedge R_j(v) \neq \perp\}$. For each such c, we add $f_{j,c}(v) = \begin{cases} 1 & \text{if } R_j(v) = c \\ 0 & \text{otherwise} \end{cases}$, $f_{j,c} : I_i \rightarrow \{0, 1, \perp\}$ to our feature map. For example, in the intro example, each constant c is a country of origin, and $f_{j,c}$ will be 1 if the given surname is from that country of origin. The result of this process is a feature map for S_i', $F_{\mathcal{R}}$ which includes all such $f_{j,c}$. We now have a complete induction problem for which we can train a classifier, giving us $h_i : I_i \rightarrow Y$. We can then use this h_i on objects in S as discussed above, giving us $f_i'(x) = h_i(f_i(x)), f_i' : O \rightarrow Y$.

We note that once we have an induction problem over I_i, we can apply our approach recursively on S_i' in order to create additional features. We can see an example of this in section 2, wherein we construct a second-level problem on countries of origin to correctly identify the appropriate conditions that signify high risk countries. For each relation R_j, we take its values, and treat those as a feature $f_j(v) = \{v'|R_j(v) = v'\}$. We can then proceed to construct a learning problem over values f_j in the same manner as before, yielding a classifier $h_j : D_{j'} \rightarrow Y$, which can then be used to create a feature for S_i'.

In cases where R_j is a relation, $f_j(v)$ is a set of values- a relation. In such cases, We must discuss both how to extract a new learning problem, and how a classifier is then used for a set of features. To begin with, let us tackle the issue of constructing a learning problem where each feature value is a set. In such cases, we still have the above problem where multiple examples may contain the same value, but a single example has multiple values. A simple solution to this issue is to give each value the same label as that of the example, and proceeding to resolve multiple labels via aggregation as we have discussed previously. Once we have done so, we can construct a new learning problem for f_j and, using FEAGURE, receive a classifier $h_j : D_{j'} \rightarrow Y$. Another question we must ask ourselves is how such a classifier can be applied to the set of values in $f_j(v)$. To illustrate, consider the classifier shown in Figure 5. We see that the classifier can be applied to multiple values of the feature, giving us a set of labels as the label of our example. As we have already discussed how such a set of labels should be resolved when discussing the labelling of the constructed learning problem, we can apply the same solution here, giving us a single feature value for our example in S_i'.

In many feature generation approaches, feature selection methods play a key part in filtering out poor features. In FEAGURE, we can use feature selection both as a filtering mechanism, as well as a search parameter. As a filtering technique, the features generated by FEAGURE are compared to the already existing feature map F. There is a multitude of feature selection techniques that can be applied for this task, but we shall cover three filtering criteria.

1. Maximal Information Gain (IG)- The information gain metric (Quinlan, 1986) compares how well a single feature reduces the amount of information entropy within a given labelled set should we split by it. In essence, it measures how well each feature separates the set based on its labels. Taking the maximal information gain as a metric, we measure whether the constructed feature gives a better separation compared to all existing features in F.

2. Average Information Gain- Just as we can compare our feature to the maximal IG of the feature map, we can compare it to the mean IG for features in F.

3. Sufficient Diversity- IG based approaches measure the separation of the training set. In many feature generation tasks, however, the more correct approach is to generate features which

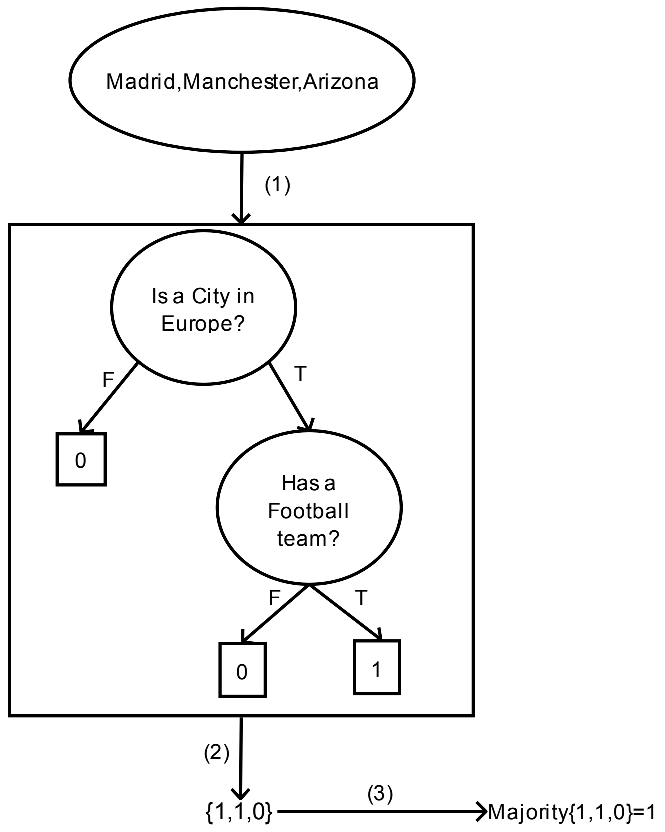


Figure 5: Using a constructed classifier on a set of values: (1)-Feature values are used to construct a learning problem. (2)-The classifier is then applied to each value. (3)-An aggregation function is applied on the results. In this case, the majority function is used to determine the output for the learning problem.

differ from existing features. To do so, we could require that the correlation of the generated feature with features in F be below a certain threshold.

Beyond their use for filtering, criteria that measure the performance of features such as the ones above can be used as a deciding factor in whether or not a constructed learning problem should apply FEAGURE recursively. The idea here is that should the generated feature be close to the threshold, we may wish to continue expanding it in hopes of giving it the necessary boost.

Algorithm 2 FEAGURE-FEAture Generation Using REcursive induction

```
function CONSTRUCTFEATURES($F$, $S$, $\mathcal{R}$)
    for $f_i \in F$ do
        $S_i'$, $F_{\mathcal{R}}$= CREATENEWPROBLEM($f_i$,$S$,$\mathcal{R}$)        ▷ We can apply our algorithm recursively here
        $h_i$= INDUCTIONALGORITHM($S_i'$, $F_{\mathcal{R}}$)
        if COMPARE($h_i$, $F$) then                                       ▷ Compare $h_i$ to $F$
            add $h_i$ to new features
    return new features

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.
    Let $f_{j,c}(v) = \begin{cases} 1 & \text{if } R_j(v) = c \vee c \in R_j(v) \\ \perp & \text{if} R_j(v) = \perp \\ 0 & \text{otherwise} \end{cases}$
    $F_{\mathcal{R}} = \{f_{j,c} | c \in D_{j'}, R_j \in \mathcal{R}\}$
                                                    ▷ We only need $c$ such that $\exists v \in v_i(S) : f_{j,c}(v) = 1$
    return $S_i'$, $F_{\mathcal{R}}$
```

We note that this feature generation approach creates features which 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 modelling a projection of the problem space to the new domain, allowing for a process similar to deductive learning. For example, in the motivating example, we began by learning a problem regarding patients, which we projected to a new domain of surnames. We then re-contextualized that problem to the domain of countries, for which we located a good solution. We then used this solution to resolve the original learning task.

While this approach has many benefits, there are is a potential issue with this approach, namely, that the number of features generated by it is limited to the number of features in the original problem. There are several methods to resolve this issue:

1. Using multiple induction algorithms on the generated learning problem may yield significantly different classifiers which can then be used as unique features, in a manner resembling an ensemble approach.

2. Use of feature selection methods within the generated problem allows for multiple classifiers to be learned by reusing the same feature map. This approach has the potential downside that later features may be significantly worse.

3. Some induction algorithms, the chief of which being decision tree classifier induction methods, yield decomposable classifiers. In the case of decision trees, such a tree classifier can be flattened into a set of decision stumps, each holding a single split of the tree. This approach allows us to turn a single strong feature into multiple weaker ones.

4. By sub-sampling our training set, we can learn features that only apply to a part of the training set, thus potentially giving us different, local features.

3.3 Finding Locally Improving Features

As we have previously discussed, in some cases creating multiple features is superior to generating only a few. Beyond this, in many cases it is far easier to locate powerful features within a localized subset of data which shares common attributes rather than trying to locate the same features in a more global context. In the motivating example, for instance, male patients are irrelevant to the target concept, and thus could potentially cause noise when used for feature generation. There is a multitude of approaches to creating such local features. We shall look at a few of these approaches:

- Sub-Sampling: By sampling a subset of training examples, we can hope to create a good local context for feature generation. An issue with this approach is that there is a huge number of possible subsets, and thus going over all of them is not computationally feasible.
- Clustering: This approach attempts to make use of the feature set to create clusters of similar examples. In some cases, this would indeed yield good local contexts, but this approach is potentially problematic for our algorithm, as it is very likely that similar examples will have the same label, thus creating highly unbalanced learning problems, which would be more difficult to utilize effectively.
- Divide & Conquer: This approach seeks to iteratively create increasingly smaller subsets of the original problem by dividing the problem according to the values of a feature. Most often, the IG metric is used to determine the feature used to split the examples.

Of the above approaches, the Divide & Conquer approach yields numerous advantages we can make use of, namely:

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, it is usually the case that features chosen later in the process will be mostly orthogonal to previously chosen features. This results in a larger variety of features overall. In our approach in particular, the elimination of a feature prunes the search tree and forces later splits to rely on different features and thus different domains.
2. interpertability: 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 we can then easily interrupt if a sufficient number of features were generated, or when the remaining training set is no longer sufficiently representative for drawing meaningful conclusions.
4. Guided by labelling: This approach explicitly uses the labels of the training set to create meaningfully different groups. As a result of this, we can expect that as we go further down, the need for good, distinguishing features will rise, and we can locate stronger features.

Due to the above advantages, we have a strong incentive to utilize the divide & conquer approach when attempting to create additional features using FEAGURE. This new algorithm, which we shall name *Deep-FEAGURE*, begins by taking the entire training set and applying FEAGURE to it. Once features have been generated, any generated features which have performed well are kept, and the feature with the highest information gain (potentially the one generated by FEAGURE) is used to split the training set into groups according to its values. For each of these new, smaller training sets we then repeat this process until we either receive a consistent training set, meaning all examples have the same label, or the size of the training set becomes so small that it is wholly unrepresentative of the complete problem. Once we have finished this process, only the generated features are kept, and returned as the output of Deep-FEAGURE.

A major potential issue that must be considered in any induction approach, and is especially problematic for divide & conquer approaches, is that of over-fitting, meaning that the result is too adjusted to seen data, and cannot generalize for previously unseen data. In the extreme case, an induction method may yield a classifier that simply memorizes the known data. This classifier will yield perfect results for seen data, but is worthless for unseen data. To combat this issue, we consider several steps:

- Limiting the minimal size of the training set allows us to control for unrepresentative subsets, which is a major factor in over-fitting.
- Feature selection inherent to FEAGURE limits the potential for over-fitting, as generated features must be beneficial to enter the feature pool. It is important to mention, however, that a very strict feature selection criterion may have an opposite effect, as any feature that pass it must be very well fitted to the training data.
- Limiting the depth of search by refusing to further split beyond a certain depth is another way to prevent unrepresentative subsets. Of particular notice, the number of relations in our knowledge base gives us a natural depth limit due to the orthogonal nature of this approach. Essentially, once a certain relation has been used, it is unlikely that another feature down the tree will make use of it again, as we have already split the training set according to values of that domain.
- Well-known induction approaches often acknowledge the issue of over-fitting, and offer multiple techniques to minimize it. Should we use search trees, for example, flattening can be used to create a set of weaker features, which is less likely to over-fit than a single complex decision tree.

Algorithm 3 Deep FEAGURE- Divide & Conquer Feature Generation

```
minSize- minimal size of a node.
SplitByFeature- a method that splits a training set according to a feature.
  function ConstructFeaturesInANode($S$, $F$, $\mathcal{R}$)
    if $|S|$ <minSize then
      return
    newFeatures=FEAGURE($S$, $F$, $\mathcal{R}$)
    $f_{best}$=bestFeature($S$,$F\cup$ newFeatures)
    return SplitByFeature($S$, $f_{best}$), newFeatures

  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
      newFeatures=ConstructFeatures(son.$S$,$F$,$\mathcal{R}$)
      currentFeatures= currentFeatures$\cup$newFeatures
    return currentFeatures
```

3.4 Application of FEAGURE for Text Categorization

A particually interesting problem domain is that of *Text Categorization*. The text categorization problem is defined by a set of texts O labelled by a set of categories Y [4] such that we create $S = \{(o_i, y_i)|o_i \in O, y_i \in Y\}$. Given S, The learning problem is to find a hypothesis $h : O \to Y$ which minimizes generalization error over all possible texts of the given categories. To measure this error, a testing set is used as an approximation. The standard approach to solving this problem is based on the bag-of-words (Wu & Salton, 1981; Salton & McGill, 1983) approach, which uses the frequencies of word appearances (without regards to order) within the text as features.

Another way to describe this approach is that it creates a single feature, the titular bag-of-words, whose value is the set of all unique words in the text. As we are well aware, however, not all words have semantic meaning. This observation makes usage of knowledge-based approaches non-trivial, as most knowledge bases rely on semantically meaningful entities rather than raw words. To resolve this issue, we can make use of techniques such as Named Entity Recognition (NER), Wikification (Bunescu & Pasca, 2006) and Entity Linking (Rao, McNamee, & Dredze, 2013). These methods as well as similar techniques allow us to link words in the text with semantically meaningful entities. They do so by identifying keywords, linking related concepts, and other supervised and unsupervised techniques. Using these techniques, we can move to a *bag-of-entities*, that is, a model of the semantically meaningful entities within the text, without regard to order.

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) 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", as well as type information. Naturally, we would like to make use of such semantic data to

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

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 category.

A major topic to consider when trying to apply FEAGURE to this domain is the fact that the bag-of-entities approach inherently creates features with many values. Because of this, special care must be taken to both the construction of recursive problems (a single example may have multiple applicable values) and the application of the resulting classifier on the document (several entities may apply). Naturally, many entities within a given document will not apply to a given relation, and thus yield a value of $\perp$. To combat this, we only give a document the value of $\perp$ if it contains no entities which are relevant to the classifier.

4. Empirical Evaluation

In this section, we discuss our experimental methodology, detail the datasets and knowledge bases we utilize and display our main results.

4.1 Methodology

As we have discussed in section 3, in order to evaluate a Feature Generation algorithm, we must first define an induction problem in which to use it. Once we have done so, we activate the algorithm on the training set in order to generate new features. Finally, once we have generated our new feature set, we proceed to test it by learning a classifier on the training set and measuring its accuracy against a testing set [5], compared to the baseline approach without feature generation. We made use of two well-known learning algorithms for this task: SVM (Cortes & Vapnik, 1995) and K-NN (Fix & Hodges Jr, 1951)[6]. The use of multiple induction algorithms allows us to reduce the impact of the specific induction approach in evaluating the performance of the generated features.

4.1.1 DATASETS

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) to create the bag-of-entities. We then performed stopword elimination using NLTK (Bird, Klein, & Loper, 2009) and stemming using the Porter Stemmer (Van Rijsbergen, Robertson, & Porter, 1980) on remaining words in hopes of finding additional entities and removing noise. We then proceeded to evaluate our approach by measuring accuracy based on the training and testing sets defined in the original paper for each dataset. Additionally, since Gabrilovich and Markovitch (2004) have demonstrated that aggressive feature selection leads to better accuracy for this dataset, we tested our feature set both without feature selection, as well as with a 5% feature selection threshold, meaning only the 5% of features with the highest IG will be selected.

5. This testing set is treated as a sample of the object space O
6. For SVM we used a linear kernel and a regularization parameter $C = 10$; for K-NN we used $K = 3$

As our knowledge base for this task, we used **YAGO2** (Hoffart et al., 2013). YAGO2 (Yet Another Great Ontology) is a large knowledge base extracted automatically from Wikipedia, WordNet and GeoNames. Its accuracy was manually evaluated on a sample of facts, yielding above 95% accuracy. YAGO2 contains over 10 million entities and 124 million facts, mostly dealing in individuals, countries and events. In order to fully make use of this knowledge base, we performed some processing on it. Specifically, we omitted relations with literal data such as dates or geographic coordinates, and created inverse relations of all relations except ones that map very few values to very many [7].

This dataset collection served as a powerful benchmark, as it contains several small multiple text classification problems of varying difficulties. This gives us a representative cross-section of text categorization tasks on which broader conclusions could be drawn. We used YAGO2 as our knowledge base as it contains a great deal of general knowledge.

2. **OHSUMED** (Hersh et al., 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 contain only a title (that is, there is no available abstract). On each document we used stopword elimination and a simple stemming scheme (due to the medical nature of the texts, there was no need for complex stemming) in order to remove unnecessary noise. Once the documents have been cleaned of stopwords and stemmed, we used Wikipedia Miner (Milne & Witten, 2013) for entity extraction. Due to the relatively sparse size of most MeSH categories (since not all documents contain only a title), we only used the two categories with the most documents, "Bacterial Infections and Mycoses" and "Immunologic Diseases" [8], yielding a binary learning problem. We then applied ten-fold cross validation on the resulting document set to rigorously test our approach and allow us to measure statistical significance.

   As our knowledge base, we used **Freebase**. Freebase 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. We used the same free-base data dump used by Bast et al. (2014), taking only entities relevant to our domain. This smaller dump of the Freebase knowledge base contains approximately 49 million entities and 242 million facts regarding multiple domains. Data regarding the medical domain is far more sparse, containing roughly a hundred thousand facts in total.

   This dataset served as a test case for a larger, more specialized learning problem, where domain-specific knowledge is required due to the significantly shorter text length. Thus, Free-base, which contains a large body of medical knowledge, was more suitable than a knowledge base containing general knowledge. Knowledge bases specializing in medical knowledge were considered, but ultimately contained less facts.

4.1.2 EXPERIMENT PARAMETERS

We tested the following parameters:

7. For example, the inverse of the "has gender" relation maps "male" and "female" to all to all existing people within the YAGO2 database.
8. C1 and C20, respectively

1. Recursion level: 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 generated problems. This parameter allowed us to test the effect of both first and second order application of our feature generation approach.
2. Tree depth: As discussed in section 3.3, we use Deep-FEAGURE within a decision tree. In order to avoid over-fitting, we limit both tree depth and node size, proportionately to the size of the training set. We experimented with the parameter of maximal tree depth to test its effect on the accuracy of the resulting features.
3. Induction algorithm: In section 3.2, we define FEAGURE without specifying the induction algorithm used to generate features within the new learning problem. We ran most of our experiments using a decision tree induction algorithm for that purpose, flattening the result to yield additional features. We also experimented with the use of other induction algorithms, namely SVM and K-NN.
4. Feature filtering technique: In section 3.2, we use a comparison function to decide whether a generated feature should be added to the feature map. We experimented with both a comparison based on maximal Information Gain (IG) and one based on average IG, both taking into account all existing features in F.

4.2 Main Results

Table 2 shows average accuracies across all 10 folds for OHSUMED, as well as the average accuracies for all 100 datasets in techTC-100. Statistical significance over the baseline accuracy (without feature generation) is shown in parenthesis. Best results are marked in bold. For the TechTC-100 dataset, we see a significant improvement over the baseline and shallow approaches, even though the number of generated features is much lower than the one achieved by the shallow 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 three major factors:

- 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.
- Small dataset size may lead to small, unrepresentative learning problems.

Analysis of the results for the OHSUMED datasets show that for K-NN, the shallow algorithm performs similarly to the baseline approach, meaning the features generated by shallow were not used by the induction algorithm. For FEAGURE, we see a degrade in accuracy, possibly due to the masking effect inherent to K-NN classifiers, meaning that impact of a single distinguishing feature may be masked by other features. For SVM, we see a significant improvement over the baseline approach, with a single level application achieving an average of 3.2% increase in accuracy, and a two-level application giving a total average of 2.4% improvement. This is somewhat surprising, as we see that the deeper feature generation approach yields a lesser improvement. This may be due to similar effects of over-fitted classifiers for the two-level usage of FEAGURE.

Table 1: Average number of generated features for each dataset, using FEAGURE, a 2-level activation of FEAGURE, and the Shallow algorithm.

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

Table 2: Average accuracy over all datasets. The columns specify feature generation approach, with baseline being no feature generation. The rows specify the induction algorithm used on the generated features for evaluation. TechTC-100 has an additional entry for 5% feature selection, meaning only 5% of all feature are used.

Dataset	Classifier	Baseline	Shallow	FEAGURE	FEAGURE 2-level
OHSUMED	KNN	0.766	**0.766**	0.761	0.744
	SVM	0.789	0.789	**0.814** ($p < 0.01$)	0.808
TechTC-100	KNN	0.526	0.526	0.533 ($p < .05$)	**0.552** ($p < .001$)
	SVM	0.694	0.694	**0.712** ($p < 0.001$)	0.691
TechTC-100 (5%)	KNN	0.613	0.592	**0.629** ($p < 0.001$)	0.619
	SVM	0.693	0.692	**0.707** ($p < .001$)	0.704 ($p < 0.05$)

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

The results show a general trend of improvement, with high ($> 5\%$) or very high ($> 10\%$) improvement being common. We also see that for 24 datasets, no improvement is seen. Of these, 4 have no generated features, and 20 have few of them, and those have not contributed to the performance of the classifier. In a small subset of the datasets, we see a degrade in accuracy, with 4 of the 100 datasets showing a degrade of over $> 5\%$.

In their paper on TechTC-100, Davidov et al. (2004) define a metric known as Maximal Achievable Accuracy (MAA). This criterion attempts to define the difficulty of the induction problem by finding the maximal accuracy among three induction algorithms (SVM, K-NN and CART [9]). Intuitively, a dataset with low MAA can be considered harder than one with a high MAA, since a known induction algorithm can achieve a higher accuracy for it. Figure 7 shows the 25 hardest datasets in TechTC-100, in terms of the MAA criterion. We see that for 7 of these, there is no change in accuracy, either due to the lack of features or due to lack of contribution in the final classifier. For 11 of the 25 datasets, we see a minor improvement in accuracy. For 6 of them, we see an improvement of 5% or above, with the highest being over 20%. Of the 25 datasets, only a single dataset shows a degrade in accuracy. This results illustrates that we can, in general, rely on FEAGURE to yield positive features for difficult classification problems.

9. Breiman, Friedman, Stone, and Olshen (1984)

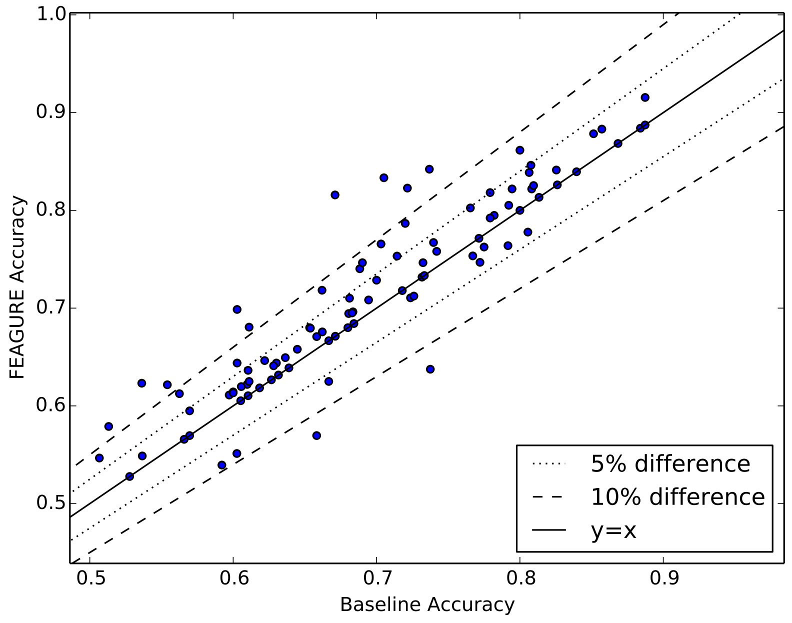


Figure 6: Accuracy of baseline approach compared to single activation of FEAGURE (SVM). Each point represents a dataset. The dotted lines represent a 5 and 10 percent difference in accuracy

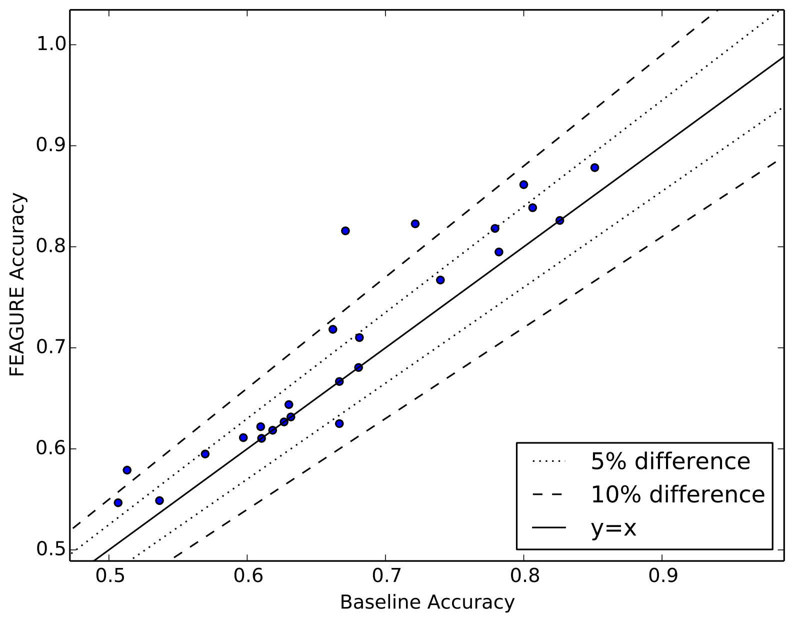


Figure 7: Accuracy of baseline approach compared to single activation of FEAGURE (SVM). Displayed are the 25 hardest datasets (meaning they have the lowest MAA)

Table 3: Average accuracy over all datasets. The columns specify the induction algorithm used in FEAGURE (Decision Tree, SVM with linear or RBF kernel, K-NN). The rows specify the induction algorithm used on the generated features for evaluation. TechTC-100 has an additional entry for 5% feature selection, meaning only 5% of all features are used.

Dataset	Classifier	Tree-FEAGURE	Linear SVM	RBF SVM	3-NN
OHSUMED	KNN	0.761	0.756	**0.783**	0.744
	SVM	**0.814**	0.795	0.796	0.788
TechTC-100	KNN	**0.533**	0.531 ($p < 0.05$)	0.527	0.528
	SVM	**0.712**	0.705 ($p < 0.005$)	0.698 ($p < 0.05$)	0.697
TechTC-100 (5%)	KNN	**0.629** ($p < 0.001$)	0.625 ($p < 0.05$)	0.61	0.617
	SVM	**0.707** ($p < .001$)	0.697	0.7 ($p < 0.05$)	0.699 ($p < 0.05$)

4.3 Using Non-Tree classifiers with FEAGURE

As we have discussed in section 3.2, algorithm 2 creates a generic learning problem as part of its execution. In this section, we will test the effects of using various induction algorithms 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.

Table 3 shows the accuracies achieved by using these induction algorithms. We see that while in general, the results are slightly poorer than those achieved by our approach, they are roughly comparable, and in some cases perform noticeably better. One particular such case is that of K-NN for the OHSUMED dataset, which was previously the only case where no improvement in accuracy was achieved. Using an SVM classifier to generate features rather than tree-based features, we see an improvement in accuracy for this case. This may be due to the nature of RBF-based classifiers, which look at distance between examples, similarly to K-NN. Looking at the number of features generated, we see that on average, these approaches generate far fewer features. This should come as no surprise, as the tree-based classifier uses flattening to create additional features.

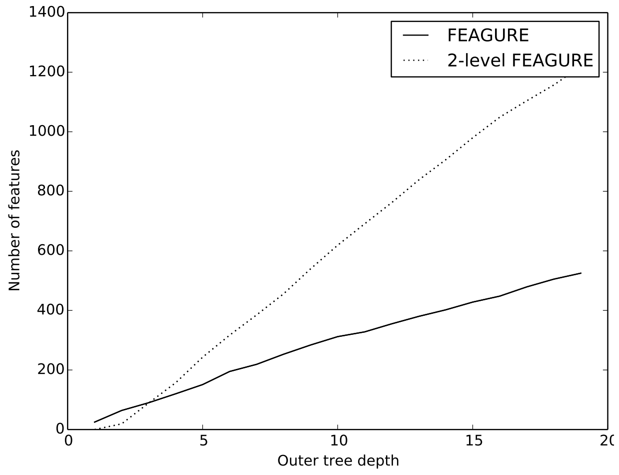
4.4 The Effects of Feature Filtering on Accuracy

In this section we look at the effect of using a different evaluation method for generated features. In algorithm 3, we use a comparison function which compares the constructed feature to the existing feature map. For the most part, we compared the IG Ratio of the constructed feature to the maximal IG Ratio of features in F to do so. An alternative method to this is to use the mean IG Ratio of features in F. Should our new feature score lower than the average, we filter it out.

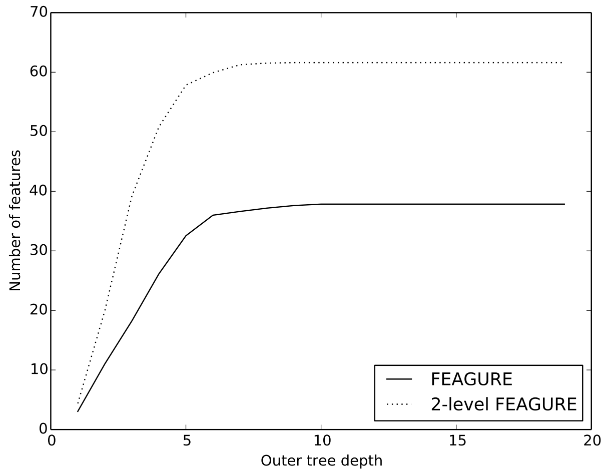
Table 4 shows the accuracies attained through this change. For the TechTC-100 dataset collection, we see a noticeable increase in accuracy, especially for single level application. For the OHSUMED dataset, however, we cannot detect a meaningful improvement, and for K-NN see a significant degrade in accuracy. Looking at he number of features generated reveals that using the average IG method for filtering yields roughly three times as many features for TechTC-100 (compared to using maximal IG ratio). For the OHSUMED dataset, the number of features generated is roughly twice as much as that seen when using maximal IG, and that number is already quite high.

Table 4: Average accuracy across all datasets. The columns specify feature filtering approach, being either the average information gain (IG) or the maximal. The rows specify the induction algorithm used on the generated features for evaluation. TechTC-100 has an additional entry for 5% feature selection, meaning only 5% of all features are used.

Dataset	Classifier	Max-IG	Average-IG	Max-IG 2-level	Average-IG 2-level
OHSUMED	KNN	**0.761**	0.667	0.744	0.686
	SVM	**0.814**	0.811	0.808	0.81
TechTC-100	KNN	0.533	0.553 ($p < 0.001$)	0.552	**0.555** ($p < .001$)
	SVM	0.712	**0.728** ($p < 0.001$)	0.691	0.653
TechTC-100 (5%)	KNN	0.629	**0.646** ($p < 0.001$)	0.619	0.594
	SVM	0.707	**0.724** ($p < .001$)	0.704	0.707 ($p < 0.05$)



(a) OHSUMED



(b) TechTC-100

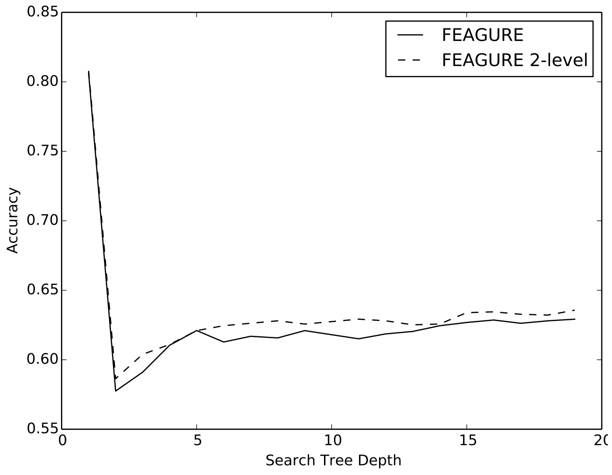
Figure 8: Mean number of features generated

These results suggest that for small problems, generating additional features may be preferable, whereas for ones with many examples, a stronger filtering mechanism is more advantageous.

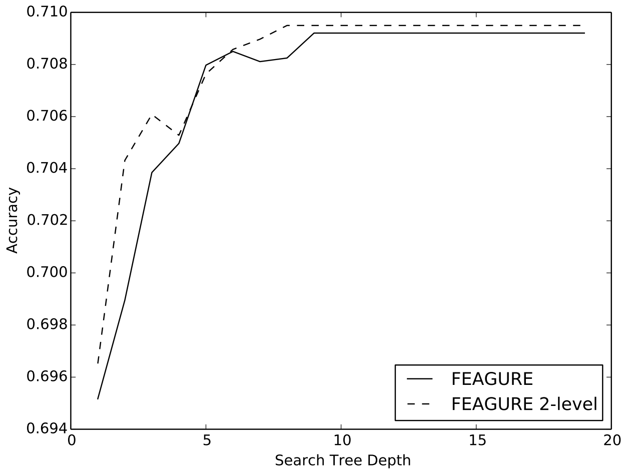
4.5 The Effect of Search Tree Depth on Accuracy

In this section, we test at the usefulness of searching local problems for additional features. To do so, we ran FEAGURE while limiting the depth of the search tree in algorithm 3. We compare our two datasets side-by-side.

Figure 8 shows the number of generated features per maximal 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. Due to the small size of the learning problems, beyond that depth the remaining training sets are too small to continue feature generation without significant risk of over-fitting, and thus FEAGURE stops its execution.



(a) OHSUMED



(b) TechTC-100, with feature selection

Figure 9: Mean SVM Accuracy across all datasets

Figure 9 shows the mean accuracy of a SVM classifier for increasing depths. The results for K-NN classifiers are similar in their general trend. For TechTC-100, we see a trend of increasing 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 and no additional features are generated.

For OHSUMED, however, we see a sharp fall in accuracy, followed by a slow recovery. This effect can be attributed to the amount of features generated. As we have seen when looking at the average IG compared to the maximal in section 4.4, more features do not equate better accuracy. As we increase the search depth, however, the local nature of the problems alongside the orthogonality of new features allows us to locate a sufficient number of good features to improve accuracy once more.

In both cases, we see that the two level application of Deep-FEAGURE yields slightly better accuracy on average.

5. Related Work

Feature generation approaches were found useful for a wide variety of learning problems (Markovitch & Rosenstein, 2002; Ragavan et al., 1993; Utgo & Brodley, 1991). These feature generation approaches search for new features that better represent the target concept than existing features. To that end, there are three major approaches for feature generation: tailored approaches, combinational techniques and methods utilizing external knowledge. Tailored approaches (Sutton & Matheus, 1991; Hirsh & Japkowicz, 1994) are designed for specific problem domains and rely on domain-specific background knowledge for feature construction. One such example is the bootstrapping algorithm (Hirsh & Japkowicz, 1994), which was designed for the domain of molecular biology. The algorithm represents features as nucleotides 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 alter 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 aim to locate more descriptive features by combining existing features in various ways, without regards to the specific problem domain. The LDMT algorithm (Utgo & Brodley, 1991), 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. Another major class of combinational feature generation approaches is that of *Deep Learning* (LeCun et al., 1998; Bengio, 2009; Plötz et al., 2011; Kim et al., 2013). Deep learning techniques seek to create features through complex combinations of existing features, using a deep structure of linear and non-linear functions. Doing so allows for a wide variety of possible combinational features. This technique bears some similarities to our approach, in that it allows for the creation of complex features. Unlike deep learning methods, our approach utilizes external knowledge to further improve the generated features. Furthermore, our use of induction algorithms allows for a more generic combination of features.

In contrast to combinational approaches, some feature generation algorithms, including FEAGURE, seek to inject additional knowledge into the existing learning problem. These approaches seek to do so almost exclusively through the use of relational data. Within this wide family of knowledge-based approaches, we can see a distinction between unsupervised and supervised approaches. Unsupervised knowledge-based feature generation techniques aim to incorporate additional knowledge from external sources without consideration to the labels supplied by a given training set. They do so in several differing ways:

- Concept based approaches such as ESA (Gabrilovich & Markovitch, 2009) ,for example, present a method for generating features that are based on semantic concepts. These approaches rely on complex techniques to create a mapping between existing data and the semantic concepts which serve as external knowledge. This mapping is then used on the given problem, enriching the data with external concepts.

- Propositionalization (Kramer & Frank, 2000) approaches are a class of feature generation algorithms that rely on relational data to serve as external knowledge. They operate by using a combination of operators in order to create first-order logic predicates connecting existing data and relational knowledge. Cheng et al. (2011) devise a generic framework to do so using linked data via relation-based queries, and offer some insights into taxonomy-based features by creating features that list the taxonomic class relationships of a given entity. The *Shallow* algorithm described in section 3.1 is another example of propositionalization algorithms.

- Data mining approaches such as FeGeLOD (Paulheim & Fümkranz, 2012) aim to directly utilize linked data in order to automatically enrich existing data, yielding additional information on existing entities within the data. FeGeLOD uses feature values as entities and adds related knowledge to the example, thus creating additional features for that example. FeGeLOD then employs feature selection to filter out poor, non-representative features.

These knowledge-based techniques serve as a powerful tool for feature generation based on knowledge. Despite this fact, the unsupervised nature of these approaches tends to lead to a shallow exploration of the knowledge base, as deep unguided search is computationally expensive, and leads to a potentially large number of features. As a result of this, many of these approaches rely on feature selection to filter out irrelevant features.

Unlike unsupervised approaches, some methods, FEAGURE included, seek to utilize a labelled training set in order to perform a more guided search through the space of possible features based on external knowledge. These techniques can trace their source to *Inductive Logic Programming (ILP)* (Quinlan, 1990; Muggleton, 1991), a supervised approach that induces a set of first-order logical formulae with the purpose of achieving a good separation of the given training set. ILP methods do so by 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 the refinement operator. Such an operator essentially refines formulae by adding conditions to the formula or substituting a variable with a constant. As a result of this, the conditions become increasingly specific. *Upgrade* methods such as ICL (Van Laer & De Raedt, 2001) and SGLR (Popescul & Ungar, 2007) can be seen as the supervised equivalent of propositionalization methods. Instead of creating predicates a-priori, feature generation is performed during the training phase, in a more structured manner, allowing for complex features to be considered. While upgrade approaches bear some similarities to our approach, there are several critical differences, the key of which being the ability to more easily locate complex features through the use of existing induction algorithms.

Another type of supervised, knowledge-based, feature generation approach is *Relational Learning* techniques. These techniques are designed to utilize relational databases and expand their knowledge. One such technique is that of View Learning (Davis et al., 2007). View learning generates new relational tables from the existing relational knowledge, based on labelled data. It does so by applying ILP on relational data to construct new, simple, relational tables, then using a Bayesian network induction algorithm, learns how these relations can be combined more effectively 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 labelled data and background knowledge. Unlike our approach, these views attempt to fit new knowledge to existing examples, whereas our approach re-frames the learning problem within a new context.

FEAGURE itself is a supervised knowledge-based feature generation approach. It focuses on creation of recursive induction problems, in a manner reminiscent of view learning. Unlike view learning, the recursive construction of induction problems allows for complex exploration of features, which can then be combined in a more general way through the use of existing induction algorithms. The features of recursive learning problems created by FEAGURE during the feature generation process are based on the ILP refinement operators.

6. Conclusions

In this paper, we presented FEAGURE, a supervised feature generation algorithm using knowledge bases. Our approach is based around the concept of injecting traditional induction algorithms with external knowledge through the use of features constructed in a supervised manner based on the target concept. The algorithm does so through the creation of recursive learning problems based on existing features and the knowledge base, which is then given as input to an induction algorithm. The output of this process is a classifier that is then turned into a feature for the original problem.

In section 3, we discussed the feature generation setting, looked at he ways in which an unsupervised approach can be used for feature generation given a complex relational knowledge base, and explained our approach in detail. Additionally, we looked at the domain of text categorization, and considered how our problem can be applied to it through the use of the bag-of-entities approach.

Empirical evaluation of our approach on the text categorization learning problem confirmed that complex knowledge-based feature generation significantly improves classifier accuracy. Experiments on the TechTC-100 dataset show an improvement over a wide range of problems of varying difficulty, with problems considered difficult yielding a proportionally higher improvement.

The main strength of approach is its ability to use existing induction approaches in a meaningful way on the knowledge base, allowing for the creation of complex features. These features can then be used by any induction algorithm, giving us a powerful, generic approach.

One limitation of our approach is that even more so than unsupervised feature generation algorithms, we are dependant on a correct link between feature values and entities within the knowledge base. Mistaken links can lead to incorrect features within the constructed learning problems that may cause the classifier learned to represent a false hypothesis.

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 entities can be extracted from feature values, such as drug names, cities and so on.

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.

DeJong, G., & Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine learning*, *1*(2), 145–176.

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.

Hirsh, H., & Japkowicz, N. (1994). Bootstrapping training-data representations for inductive learning: A case study in molecular biology. In *AAAI*, pp. 639–644. Citeseer.

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.

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

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.

Mitchell, T. M. (1982). Generalization as search. *Artificial intelligence*, *18*(2), 203–226.

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.

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

Sutton, R. S., & Matheus, C. J. (1991). Learning polynomial functions by feature construction.. In *ML*, pp. 208–212.

Utgo, P. E., & Brodley, C. E. (1991). Linear machine decision trees. Tech. rep., Tech. Rep. COINS 91-10, University of Massachusetts, Amherst, MA, USA.

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.