Institutionen för datavetenskap
Department of Computer and Information Science

Examensarbete

Semantic word classification and temporal dependency detection on cooking recipes

av

Erik Jonsson

LIU-IDA/LITH-EX-A--15/064--SE

2015-11-16

Linköpings universitet

Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings universitet
581 83 Linköping

Linköpings universitet
Institutionen för datavetenskap

Examensarbete

Semantic word classification and temporal dependency detection on cooking recipes

av

Erik Jonsson

LIU-IDA/LITH-EX-A--15/064--SE

2015-11-16

Handledare: Arne Jönsson
Examinator: Lars Ahrenberg

Abstract

This thesis presents an approach to automatically infer facts about cooking recipes. The analysis focuses on two aspects: recognizing words with special semantic meaning in a recipe such as ingredients and tools; and detecting temporal dependencies between steps in the cooking procedure. This is done by use of machine learning classification and several different algorithms are benchmarked against each other.

Acknowledgements

Appreciation can often be seen as a cliché. Yet, when you find yourself in a situation where you are really in need of support it becomes very natural. This thesis would not have been finished without a lot of patience, understanding and assistance from people around me. For this I am deeply grateful. In particular I would like to extend thanks to my supervisor Arne Jönsson; my examiner Lars Ahrenberg; my industry supervisor Gabriel Falkenberg and my thesis opponent Mikael Silvén.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Today there are many resources on the Internet for finding new cooking recipes, like allrecipes.com [1], cooks.com [2] and BBC Good Food [3]. While the way we view and consume the recipes is very modern, the recipes themselves have not changed much. A recipe generally consists of a list of ingredients, and a description of the cooking process either as running text or a sequential list of steps.

So what is wrong with this approach? For one thing: recipes are not sequential and there is not necessarily only one person cooking. They are also not static. Sometimes depending on the context you might do the preparation steps in a different order. So a dish can in general be prepared in a number of ways, and the steps in a recipe can be reorganized and still be a valid encoding of how to cook same the dish.

Information about this could be useful when trying to optimize a work flow by doing several steps in parallel, or due to time constraints on when certain resources like ovens are available in a larger kitchen. To present all valid orderings of steps the traditional description in running text becomes insufficient and a dependency graph would be better suited.

Another problem is that a level of domain knowledge is assumed of the reader. Today certain tools are perhaps less used, such as mortar and pestle. It would then be beneficial to be able to provide contextual help to the reader in the form of dictionary information, where to buy certain ingredients, perhaps information about other recipes that use a tool to evaluate if a purchase would be worthwhile.

Providing links to more information for keywords such as ingredients is actually often available on many of the web resources. There are also some notable exceptions that incorporate flow charts of how to combine the ingredients into the final dish [4]. So this type of data model already exist to a certain extent, but is often manually annotated.

While manual annotation of dependencies and keywords is sometimes done, converting the large body of cooking knowledge available on the web in the form of recipes would be very labour intensive. It may also be desirable to do this quickly for a new recipe from another source without user involvement. In both cases an automatic solution would be preferable.

This thesis presents an approach to automatically infer facts about a traditional cooking recipe and augment it in two ways. First we detect and categorize keywords such as ingredients, tools and actions. We then construct a dependency graph of the cooking process that encodes all possible sequences of steps to cook the dish.

We cast this as a classification problem both for the case of dependencies and keywords. Machine learning algorithms of several different paradigms are benchmarked and compared for the task.

1.1 Formal problem statement

Take as input a cooking recipe with a list of ingredients and a textual description of the cooking procedure. Each ingredient is separated into name, amount and unit (such as tsp, cl and ounce). The textual description is given as a list of sentences where each sentence is assumed to represent one step of the cooking procedure.
For every word w the system should assign it one class $c \in \{1, 2, 3, 4, 5, 6, 7\}$. Where the integer classes represents the semantic classes: ingredient, tool, action, intermediate product, amount, unit and other.
For any pair of two steps the system should infer whether they depend on each other or not. Construct the dependency graph $G = \langle V, E \rangle$ for the cooking procedure, where V is the set of sentences/instructions. For each pair of instructions $e = (i_1, i_2)$, $e \in E$ *iff* i_1 can not be done before i_2.

It is important that the model does not represent any semantically incorrect ways of cooking a dish. Therefore the system should err on the side of inferring extra dependencies in case of uncertainty.

1.2 Use of classification algorithms

Machine learning is a branch of artificial intelligence that has been used widely in the field of natural language processing. Classification is the task of determining which class an instance belongs to, by looking at previous instances and generalizing.

More specifically in our case the instances are single words for the semantic word classification task.

For the dependency detection task the instances are pairs of steps as defined in section 1.1. The steps are divided into the two classes *depends on* and *does not depend on*.

Chapter 2

Background

2.1 Related work

Case Based Reasoning (CBR) is a technique stemming from the field of artificial intelligence. It has been used for recipe search that gives results that are semantically similar to the search term [5]. The European Conference on Case Based Reasoning (ECCBR) hosted a contest where the main task was to determine which ingredients can be substituted with one another. One entrant called TAAABLE created a semantic wiki for storing the domain knowledge and relationship between actions, tools and recipes [6]. Another entrant emphasized a knowledge-light approach and used WordNet [7] for the semantic relationship between ingredients [8]. In this way the system could make suggestions which ingredients are similar and may be substituted. A very early result using CBR is CHEF which tries to create plans for cooking dishes from the Szechwan cuisine [9].

BakeBot is a robot system that can follow cooking instructions and generate a plan for how to cook a dish, in this case baking cookies [10]. While it is the same domain, all of the projects mentioned in this section are more concerned with ingredients or following one ordering of steps to cook a dish rather than finding possible orderings of the steps.

Asium is a system designed for extracting information from maintenance manuals in the aviation industry, but which was tested on the domain of cooking recipes. This illustrates the possibility to generalize techniques used in one domain for use in another domain [11]. Another result deals with general procedural text by creating a grammar to extract instructions [12]. There have also been attempts to classify what type of dish a recipe describes with features that are acquired with graph mining [13].

Some work has been done to determine in what context a dish is suitable. For example if it is related to a certain holiday. To do this semantic relations (for example which ingredients it contain) are extracted from recipes [14]. Here the focus is also not on ordering of steps. A formal instruction set has been defined to encode how to cook a dish. It is called SOUR CREAM and does include annotations for which steps depend on which [15]. Unfortunately the parsing of an existing recipe to this format was left as future work.

Two papers deal with creating work flows from cooking recipes. The first uses manually created rules to extract relations from the recipe. The second paper uses ideas from named entity recognition and uses machine learning in the form of logistic regression to teach a classifier to recognize which words are ingredients [16] [17].

2.2 Classification algorithms

Classification is a task in machine learning and data mining where the objective is to assign an instance to one of several classes. The decision is made by looking at previous instances that has been keyed with the correct classes. The keyed data is called a training set. The idea is that what is learned by looking at the training set can be generalized to a set of rules, a geometric boundary or some other model that generalizes well and can correctly classify other data sets as well.

There are many different algorithms for classification, each with their own pros and cons. When choosing algorithms our goal was to get representatives of several different paradigms. Other important factors are previous proven results in natural language processing tasks, popularity and ease of implementation. The last point did not play a big role as all algorithms considered were part of the WEKA framework or could be plugged in as in the case with LibSVM.

Data values are extracted from each instance and used to distinguish between the different classes. These values are usually called attributes or features. Throughout this thesis we call them attributes.

Most algorithms are geared towards distinguishing between two classes (binary classification). When there are more than two classes (multiclass classification) the solution is often just to extend the *use of* the binary version. More on this in section 2.2.2.

For an overview of different machine learning techniques the WEKA companion book is recommended [18]. For an overview of natural language processing and use of machine

learning techniques in the field the book Speech and language processing is a good start [19].

2.2.1 Nominal and numerical attributes

A nominal attribute is one which has finitely many distinct values, whereas a numerical attribute is continuous. Some algorithms are more naturally expressed in terms of numerical attributes, others in terms of nominal attributes. When the data has both nominal and numerical attributes a conversion is needed, as algorithms often support only one or the other. This can be done in different ways. For converting from numerical to nominal the domain can be split into intervals where each interval maps to a nominal value. For the opposite direction each nominal value can be mapped to an integer. Each attribute can also be mapped to a set of mutually exclusive binary attributes, one for each nominal value [20].

2.2.2 Multiclass classification

Multiclass classification is classification with more than two possible classes. There are several approaches to this. A simple and common method is one-vs-all where one classifier is trained per class. This classifier is used to distinguish that particular class from the the union of all other classes to reduce the problem to many binary classification problems.

2.2.3 Support Vector Machines

A Support Vector Machine [21] (SVM) is a machine learning algorithm that is often used for regression but can also be used for classification. Used in a classification setting with two classes an SVM views the attributes as dimensions in a vector space and will try to find the maximum margin hyperplane that separates the classes. Maximum margin means it sits right between the two classes, with a maximum distance to both class boundaries. This maximum margin hyperplane is found by looking at instances close to the class boundaries; so called support vectors that give the algorithm its name.

In their original form, SVMs can only handle cases where the classes are linearly separable. This can be alleviated by mapping the data onto a higher dimensional space in which the two classes are separable. An SVM does this only implicitly by using a kernel function. This is less computationally expensive than explicitly calculating the new coordinates [22]. For a more detailed overview there is an excellent tutorial [23].

Machine learning in general has shown very good results in the field of natural language processing as evidenced by the regular CoNLL conferences. SVM in particular has proven useful in a number of classification tasks like spam categorization [24]. Jakub Zavrel et al compare several different classification algorithms for use in natural language processing with SVM scoring the highest [25]. Several other papers show SVM scoring good results in various CoNLL tasks [26] [27] [28].

Using a kernel also has the advantage of modularity. Results can be very different between one kernel and another. In this thesis we use the radial basis function (RBF) kernel and the polynomial kernel. Both are commonly used and have different sets of parameters that determine how they function.

Both kernels have a cost parameter C that determines how hard the algorithm will try to find a perfect fit. It signifies the cost of misclassifying one instance. The epsilon parameter (e) determines the size of the insensitivity zone. In practice a lower epsilon will increase the number of support vectors used and thus affect performance. It will also create a more tightly fit model, similarly to a high cost.

The RBF kernel uses the radial basis function $e^{-\gamma \cdot |u-v|^2}$. Here gamma is another parameter that can be varied. Likewise for the polynomial kernel that uses a polynomial $(\gamma \cdot u' \cdot v + c0)^D$ gamma can be varied as well as the degree of the polynomial.

2.2.4 Decision trees

A family of algorithms for classification solve the problem by building a decision tree. Assuming nominal attributes and binary classification the algorithm can be described as follows:

1. Choose one attribute, and split the instances into groups depending on their value for that attribute. 2. Repeat, forming smaller and smaller subgroups. 3. A group that has only yes or no values need not be split further.

This means that the running time for this algorithm will be $O(n * 2^m)$ where n is the number of instances and m is the number of attributes. The number of attributes can in many practical applications grow very large so it is important to get to case 3 above as quickly as possible.

Different algorithms differ in how they choose what attributes to split on and in what order to achieve this. [29] ch 4.3.

Even though the above example is for binary classification it can be extended to multiclass classification without any changes. In fact taking all classes into account rather

than solving multiple binary classification problems often gives better performance for decision trees [29] ch 4.3.

The C4.5 algorithm is an example of a decision tree algorithm that has shown good results in natural language processing, for example on the 2003 and 2011 shared CoNLL tasks [30] [31]. One feature of C4.5 is that once the decision tree is built it goes back and prunes branches and turns them into leaves if confident enough. This can help against overfitting. There are two parameters that can be changed in C4.5. One is the C parameter which determines the confidence needed to prune a branch. The other is m that signifies the minimum number of instances in each leaf. So that if when branching, one child has less than m instances it is made into a leaf and not branched further.

2.2.5 Covering algorithms

Covering algorithms or rule based algorithms are similar to decision trees. A decision tree can in fact be viewed as a set of rules.

They differ in the fact that they do not handle the attributes in any particular order, and that they do not work on all classes at once in the case of multiclass classification. Rather they try to isolate one class by constructing a set of rules that *cover* all the positive instances.

In the case of Ripple Down Rules the rule base is built incrementally starting with the rule that correctly classify the most instances first. Then more and more rules are added as exceptions to the first rule. In this sense this type of rule based algorithm is even more closely tied to the tree representation as it is actually dependant on‘ order. Ripple Down Rules (Ridor) have been used for lemmatisation [32] and part-of-speech tagging and named entity recognition in Vietnamese [33] [34]. The implementation of Ridor used for our experiments uses Incremental Reduced Error Pruning (IREP). There are a number of parameters that can be changed. The N parameter chooses the number of folds used in the underlying IREP algorithm. The S parameters is the number of shuffles used to randomize the data. The N parameter is the minimal weight of instances within a split. It functions similarly to the m parameter of C4.5.

2.2.6 Instance based learning

The previously mentioned algorithms in section 2.2 all create some sort of criteria to distinguish between different classes. It could be that one class goes on one side of a hyperplane and another on the other side, rules or a decision tree. The original keyed data is not used to decide what class an instance should have. Instance based learning

on the other hand does exactly that. To decide on a class for an instance it compares it to the training set.

A common algorithm is *nearest neighbour* that assigns the class of the nearest neighbour in the training set, using the Euclidian distance. K-nearest neighbour (KNN) assigns class based on the k nearest neighbours where k is a parameter. This can be done by majority vote or by assigning weight based on distance or other metrics.

KNN has been used for the 2008 and 2010 shared CoNLL tasks [35] [36] as well as text categorization [37]. The implementation we use is the IBk algorithm in WEKA.

2.2.7 Ensemble learning

Sometimes combining the results of different models gives a better result. There are various ways of doing this. Common alternatives include bagging, boosting and stacking [18] ch 8.1.

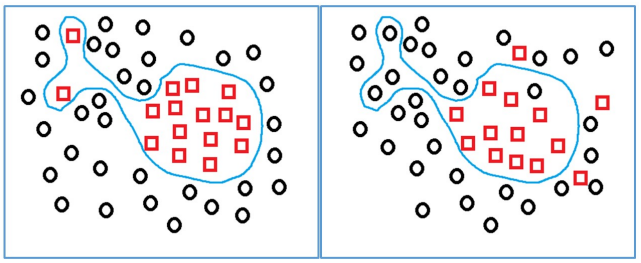
In boosting several different models are trained with different parameters on the same data set. The collection of models are built iteratively, with later models being designed to focus on the areas that earlier models struggled. This is done by giving instances that earlier models misclassified a greater weight in the training. The results of these models are then combined according to some weight.

Carreras et al had a lot of success using AdaBoost for named entity recognition for both the 2002 and 2003 shared CoNLL task [38] [39]. This result was later surpassed by Szarvas et al, also using AdaBoost [30].

2.2.8 Overfitting

In practice the model will never be perfect and the data will have noise. If the model is severely penalized for errors on the training set it may be affected by this noise and become distorted in an attempt to classify the training set correctly. In this case the model will be very good at predicting the training data, but generalize poorly to another data set. Figure 2.1 illustrates this. To the left is the training data. We can see a clear trend with two noisy outliers. Because of a high penalty on not getting a perfect fit on the training data, a poor model that accommodates the noise is created, which does not work well with the actual test data to the right. Here the noise is different so we actually get a worse result than with a less precise model.

FIGURE 2.1: An example of the negative effects of overfitting.



2.3 WordNet

WordNet is a lexical database in English where words with the same meaning are grouped together. Semantic links between these groups are then added, so that a network is formed where similar words that are related semantically or lexically are in close proximity [7]. Such links describe what kind of semantic relation the two groups have, such as meronym, holonym, hypernym or hyponym. There are several different metrics for measuring the distance between two concepts in WordNet. Using just the number of hops is usually not a very good metric, as following a hyperonym link and then a hyponym link is often a much larger semantic distance than following two hyponym links. There are several different types of metrics used for WordNet that take this into account [7] [40] [41] [42] [43] [44] [45].

Chapter 3

Method and implementation

Given a recipe separated into a list of ingredients and a method description in running text, the description is split into discrete steps where each sentence is one step. We aim to solve two problems: Detecting dependencies between steps and annotating each word with a semantic class.

For both problems we view it as a classification problem. A grammar or rule based approach requires a lot of domain knowledge, because there is need for a good grasp of the type of sentences or grammar constructs that occur in the data and their meaning. This is somewhat true for classification as well in the choice of good features, but not to the same extent. This makes it a good choice for a first foray into the field. As noted previously many of the algorithms we try have had good performance on similar tasks like named entity recognition.

The results of the semantic classification of words is used as an attribute in the dependency detection.

3.1 Choice of classification algorithms

We have chosen to measure results of 4 different algorithms. The algorithms were chosen partly based on demonstrated good results in similar natural language processing tasks. Preferably results that are comparable to other algorithms, like submissions to a shared CoNLL task. See chapter 2.2 for a discussion of previous results. Another important factor is to get a spread of algorithms across different paradigms like rule-based or vector space algorithms. To represent linear or vector space algorithms we use SVM, for rule based algorithms Ridor, for decision tree C4.5 and for instance based IBk. Other

considerations include ease of implementation and early results on a small test set with default settings.

3.2 Choice of attributes

The attributes were developed iteratively. The first attributes were chosen by looking at an instance and determining why a human could classify it correctly. What information does a human use? Things like parts of speech, and dictionary lookups are the result of this. Use of WordNet is a way to mimic the semantic knowledge a human has. We know a spatula is a utensil and that a utensil is a tool.

Later specific examples where the algorithm classified something wrong was studied and more attributes were added to address those specific problems. For example: The coreference resolution used from CoreNLP that was used failed to pick up back references properly so a more simple version was used in parallel.

3.3 Semantic class of words

The aim is to assign each word to one of 7 classes, making it a multiclass classification problem. The 7 classes are ingredient, tool, action, intermediate product, unit, amount, other. Tools can be utensils but also resources like an oven. Ingredients are any foodstuff that is an input to the recipe and intermediate product anything that is made by a combination of the ingredients but is not the final dish, for example a dough. Units can be millilitres or cups for example and amount can be a numeral or expressions like *a few*. Other is everything else deemed not to be explicitly related to any of the previous.

For compound expressions like large metal pan we assign the tool class to every word. So in this example large gets tagged as tool, because it is part of a larger expression for a tool.

This partitioning was made with three considerations. First, for what could contextual information be useful? Raw ingredients, utensils and actions fall into this category. Second, what does the raw data look like? Amounts and units fall into this category since they are already separate in the ingredient list in the source data. Note that units also fall into the first category since in the US imperial measurements are used which may be unfamiliar to a European reader for example. And lastly, what data could be useful to our second task of detecting dependencies. Intermediate product is in this category as a backward reference to an intermediate product would be a sure sign that there is a dependency.

3.3.1 Semantic class attributes

These attributes are extracted from each instance (word). For some attributes a context is needed such as a lookup in a dictionary, comparison to the ingredient list or analyzing the recipe as a whole.

3.3.1.1 Part of speech

The part of speech of the current word is extracted with the help of Stanford CoreNLP. This is a nominal attribute with all word classes as possible values.

3.3.1.2 Dictionary

A dictionary is constructed containing information about how often each word occurred in each of the semantic classes. It is constructed by looking at all recipes except the one currently being processed. This feature is expressed as fraction for each semantic class which gives us seven numeric attributes. The lemma of the word is used instead of the actual word to make the data less sparse. When extracting attributes the dictionary is constructed for all recipes but the current. This is so that the model generated should not rely overly on the dictionary and generalize well to new recipes where it has not seen any of the specific words before. For lemmatization CoreNLP was used as above with part of speech.

3.3.1.3 Exists in the ingredient list

This is a binary (nominal) feature. It is true if the inspected word can be found in the ingredient list. Here the lemma of the word is used to make sure that slightly differing mentions still triggers a match. The idea is that ingredients, amounts and units are common in the ingredient list, whereas tools and actions are not.

3.3.1.4 Adjacent words

This is exactly the same as the dictionary and word class attributes, but for the words to the left and right of the current word. The idea is that perhaps ingredients often come after verbs, and units after counting words and similar correlations. So in total 16 attributes.

3.3.1.5 WordNet distance

The distance between the current word and the terms *food, animal, tool*, and *utensil*. The idea is to disambiguate between dead objects like tools and living objects like ingredients. We use the seven metrics of distance described in section 2.3 as well as the raw shortest path distance. So in total this gives us 32 numeric attributes. All of these things are available in the library ws4j through a very convenient API [46]. For extracting features this part is by far the most computationally demanding.

3.4 Dependency detection

For each pair of steps (i_1, i_2) we want to infer whether i_2 depends on i_1 or not. This is an ordered tuple, and chosen so that i_1 is a step that comes before i_2 in the instructions. Note that it never makes sense for i_1 to depend on i_2, because then the original recipe would be invalid. This is a binary classification problem with the two classes *depends on* and *not dependant*.

We chose the sentence level for our steps but a step could also be modeled at the more fine grained level of clauses. This can be useful as a sentence may include several cooking instructions at once. This would require us to extract much deeper semantics of the text, and we first want to see what can be done at this more shallow level.

If the set of steps in a recipe is S, we want to find a dependence relation $R \in S \times S$. This can be seen as a dependency graph $G = \langle V, E \rangle$ where the vertices V is the same set as the set of steps S. If one step depends on another we add an edge to E. To find these edges E given V, we use classification.

For each instance (each step/sentence in the recipe that is) the attributes listed below in section is extracted.

3.4.1 Attributes

These features are for a pair of sentences (A, B), and are used to predict whether B depends on A or not.

3.4.1.1 Semantic class statistics

Certain words are more important in a recipe than others, like ingredients, tools and processes or actions to create the dish. We believe the amount and relation between

numbers of occurrences of these types of words can indicate whether there is a dependence or not. The semantic classes of words that we track are: Ingredients, tools, actions, intermediate products (things like sauces, batter, dough that is made as a step in the cooking process and combines ingredients but is not the final product), amount (like 5, a little), unit (cl, tsp, package) and other.

This is a problem in itself to extract and we treat it in exactly the same way: As a classification problem.

This gives us 7 numeric attributes, one for each type of word that we track. The value of each attribute is the number of occurrences of that word type in the step.

3.4.1.2 Directly preceding

This is a nominal attribute with the values yes and no. If A directly precedes B the value is yes, otherwise no. This is fairly straight forward as steps are often subsequent *because* of a dependency rather than the constraints of the linear format of text.

3.4.1.3 Position

This is a nominal attribute with the possible values begin, end and inside. This means whether the step is the first or last step in the list of instructions, or somewhere in between. Both the first and last step probably differ from the rest. For one thing, the last step may never be depended upon and the first step can not depend on anything. Also easy to extract.

3.4.1.4 Coreference

This is a nominal attribute with the possible values yes and no. If there is a coreference from B to A, then the value is yes, otherwise no. To find coreferences the coreference resolution system in CoreNLP is used. However, because it is like many such systems trained on newspaper articles it does fairly badly on imperative recipe text. Of all the tools used, the coreference resolution system was particularly prone to error and performed well below its standard.

Therefore a simpler metric was also introduced that only measures word matches of ingredients, tools and intermediate products. This is an exact match, but the lemma is used rather than the actual word. The number of matches is counted so this is a numeric attribute.

3.5 Data acquisition

All training data was extracted from allrecipes.com [1] and preprocessed into the format mentioned in section 3.2 and 3.3.1.5. The recipes at allrecipes.com have their ingredients annotated as amount, measurement and foodstuff already. There are other similar sites with more recipes [2] [3].

Only recipes with over 1000 reviews and an average rating higher than 4.5 out of 5 were chosen. By choosing popular recipes we hoped to filter out recipes with a lot of misspellings and incorrect grammar.

The data is split into a training set and a gold standard. 25 recipes were annotated by one person, and serve as training set. As a gold standard to measure how well the model generalizes five recipes were annotated by two people, solving conflicts by consensus. Any conflicts that were recorded between the annotators is regarded as an upper bound on accuracy achievable on the task. An example of the output of this activity can be seen in appendix A.

Instances for both the dependency and semantic word class problems were extracted from all recipes in the training set and gold standard. This gave us 1747 and 813 instances for the word class and dependency problem respectively.

For optimizing algorithm parameters cross validation was used with a 80/20 split on the training set. This means that the values in the parameter optimization section for both problems are averages.

Before and after parameter optimization the effectiveness of every algorithm is measured by varying the size of the training set from 5 to 25 recipes and measuring the results when applied on the gold standard.

3.6 WEKA and LibSVM

For the classification task we use the Waikato Environment for Knowledge Analysis (WEKA) [29]. WEKA implements a host of different machine learning algorithms and wraps many other popular implementations. WEKA is implemented in Java and has a number of useful functions for normalizing data and evaluating results. It also has very rich documentation partly in the form of a book [18]. Other alternatives include Encog [47] and PyML [48].

Using WEKA we can easily test and compare different algorithms performance against each other. We compare one implementation of each paradigm described in section 2.2.

WEKA wraps the library LibSVM which we use as our *Support Vector Machine* implementation. For decision trees we choose the C4.5 algorithm that has shown good performance in somewhat similar tasks.

We use the WEKA implementations of Ridor and IBk for ripple down rules and k-nearest neighbour respectively.

3.7 Stanford CoreNLP

The CoreNLP package from Stanford comes with a number of tools we use for preprocessing and feature extraction. The tools primarily used are the part-of-speech tagger [49] [50] and coreference resolution [51] [52]. CoreNLP is implemented in Java and has state of the art performance on several CoNLL tasks. Alternatives include GATE [53], Lingpipe [54] and NLTK [55].

3.8 Metrics

To measure the accuracy of the models we make use of *precision* (P), *recall* (R) and *F-score*. These metrics compare true positives (tp), false positives (fp) and false negatives (fn). F-score is an aggregate of precision and recall thought to give a good overall estimate of how accurate a model is.

$$\text{Precision (P)} = \frac{tp}{tp + fp} \tag{3.1}$$

$$\text{Recall (R)} = \frac{tp}{tp + fn} \tag{3.2}$$

$$\text{F-score} = 2 \cdot \frac{P \cdot R}{P + R} \tag{3.3}$$

For the semantic class problem we use F-score to get a balance between precision and recall. For the dependency problem we also use measure the recall on the yes instances. It is important that we do not miss any dependencies as that could lead to an incorrect recipe.

3.9 Parameter optimization

Each input parameter was measured for different values to find the best set of parameters for the data set. To avoid combinatorial explosion of different parameters each parameter

was optimized separately and in turn. This means that the best measured value for the first parameter is used when evaluating the second parameter, which is then used to evaluate the third parameter and so on.

3.10 Learning curves

A plot of an algorithms accuracy (measure in F-score or something else) against the size of the training data is called a learning curve. The accuracy is measured both on the training set itself, and on a separate test set. These can be useful for analyzing problems with the implementation like overfitting. In the case of overfitting the algorithm will continue to give good results on the training set as more training data as more is added. But it will give worse results on the test set at high levels of training data since noise will have distorted the model. A typical good result is a slight decrease in accuracy on the training set as more training data is added, and a sharp increase on the test set. This increase on the test set may level out, but not decrease again.

Chapter 4

Results and discussion

The results in this section are split into two sections, one for each subproblem. Each section shows the results of parameter optimization for each algorithm in turn.
For each algorithm we begin by plotting a learning curve. We then show the results of optimization for every parameter and a learning curve for the optimized version.
At the end of each subsection the different algorithms are compared to each other. We then explore the effect of boosting on the most effective algorithm.

4.1 Semantic class of words

In figure 4.1 the learning curves of C4.5 is plotted. The model generalizes well to the test set, the difference between training and test is less than 10%. If there are limitations in the algorithm that prevent a correct classification we would expect the error on training data to increase with more data. As this is not the case it may instead be that the choice of attributes is the limiting factor. More training data brings up the accuracy on the test set, but only slightly so. The main gains were gotten in the first five recipes.

In table 4.1 and 4.2 we see the effect of varying the K and M parameters. The stock parameters turned out to be the best. The model performs better on the cross validation set than the test set even though in both cases it is not trained and tested on the same data set. This could be due to a number of reasons, such as a natural variation in data. It could also be that the training set and thus also the portion serving as cross validation set is labelled by only one person, whereas the test set is labelled with the consensus of two. The effect persists for all variations of the parameters, so it is not explained by the fact that we optimize parameters geared for the cross validation set. Finally it could also be because when choosing a cross validation set, the split is done as a percentage

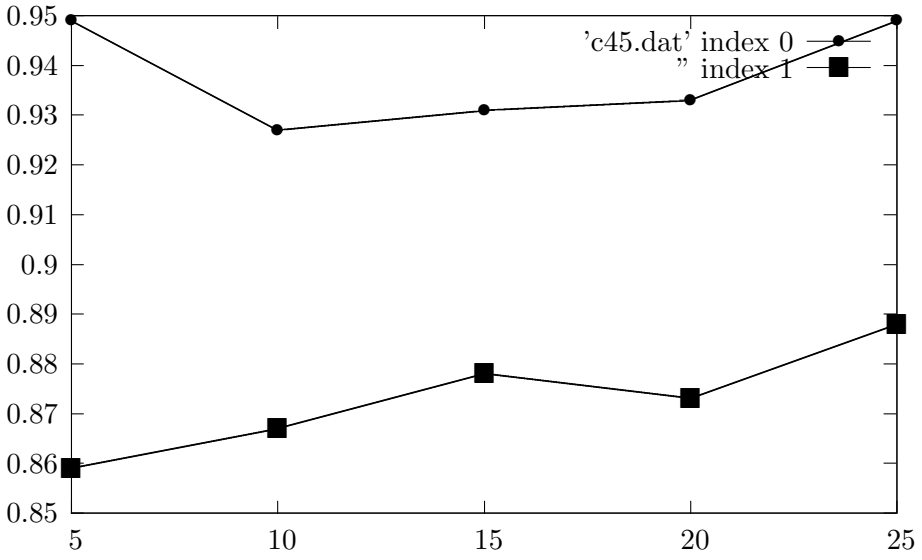


FIGURE 4.1: Learning curve for unoptimized C4.5. Index 0 is the training set and index 1 is the test set.

of instances. Thus a large recipe could potentially be split in two parts where some instances are used for training and some for verifying the same model.

TABLE 4.1: Parameter optimization for C4.5: Varying the C parameter

C parameter	0.05	0.15	0.25	0.35	0.45	0.55	0.65
Avg. F-score	0.905	0.922	0.922	0.914	0.914	0.915	0.915

TABLE 4.2: Parameter optimization for C4.5: Varying the M parameter

M parameter	1	2	3	4	5	6
Avg. F-score	0.919	0.922	0.906	0.899	0.901	0.898

For IBk the results for the training data has even less error as seen in graph 4.2. This is not surprising since it saves all training data and then compares against it.

When optimizing the number of instances to compare to, the K parameter, the stock setting of 1 performed the best. This can be seen in table 4.3 The other parameters does not make sense for K = 1 so for those tests K = 5 is used since it is the second best performer. This also shows how the greedy approach to parameter optimization used here can miss the optimum as a combination of K = 5 and some of the more advanced distance weighing metrics provided the best results according to tables 4.4 and 4.5.

After parameter optimization we can see that the model no longer performs as well on the training data, in graph 4.3. This is expected as it now compares to the five closest instances not just one. More surprising is that it performs less well on the test set

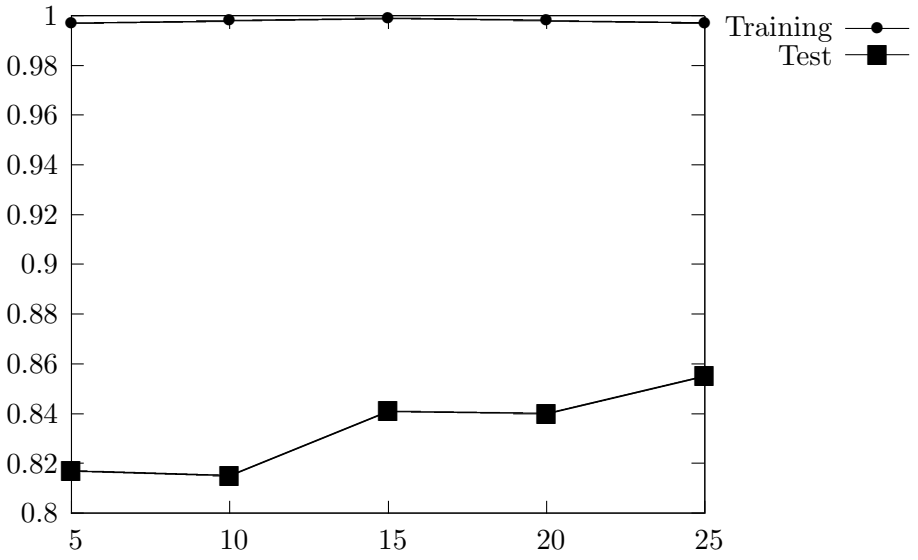


FIGURE 4.2: Learning curve for unoptimized IBk

TABLE 4.3: Parameter optimization for IBk: Varying the K parameter

K parameter	1	2	3	4	5	6	7
Avg. F-score	0.879	0.857	0.864	0.872	0.876	0.868	0.875

TABLE 4.4: Parameter optimization for IBk: Effect of using mean squared, with K = 5

Mean sqrd.	On	Off
Avg. F-score	0.876	0.876

TABLE 4.5: Parameter optimization for IBk: Effect of using different types of distance weighing, with K = 5

Dst. weigh.	none	1/d	1 - d
Avg. F-score	0.876	0.885	0.886

with little training data. However when using all the training data it just edges out the unoptimized version.

For Ridor the trend continues that increased size of the training data does not increase the error on the training set as seen in graph 4.4. This can be because the limiting factor is not the algorithm but the choice of attributes.

For Ridor non-stock settings performed better. The F parameter is the number of folds or the number of iterations when creating rules and exceptions or more specific rules that override more general ones. The default number of folds is three. In table 4.6 we

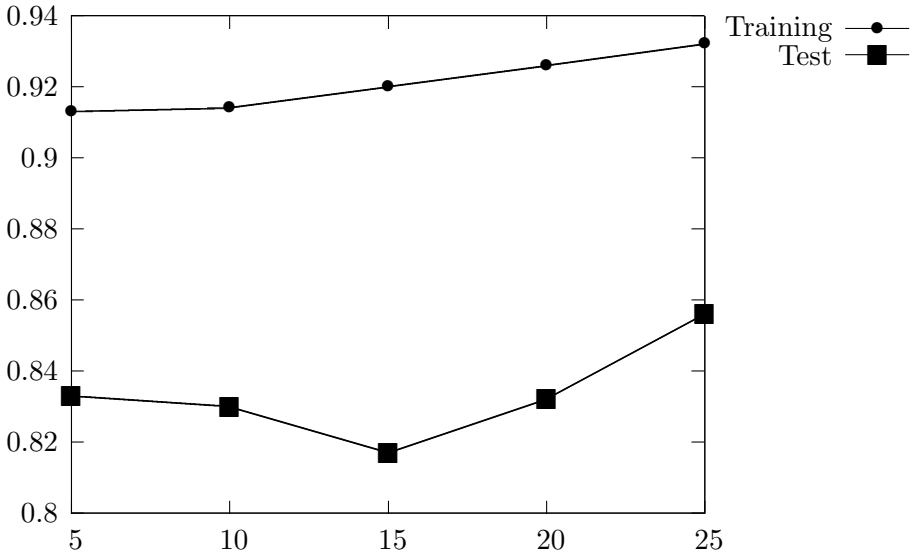


FIGURE 4.3: Learning curve for parameter optimized IBk

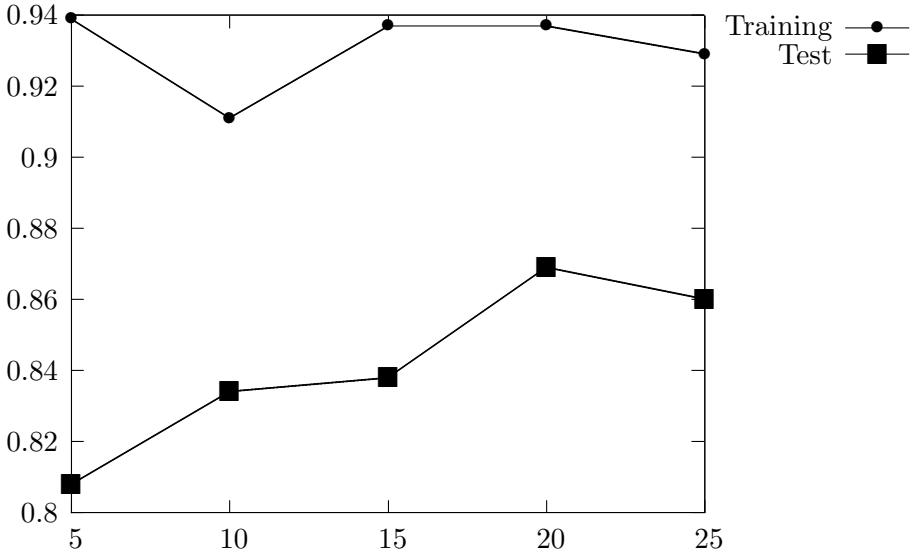


FIGURE 4.4: Learning curve for unoptimized Ridor

see better results with more folds up to a point. The reason that performance decreases at 8 folds can be overfitting as the rules produced are then very precise and may apply to only single instances. More folds increase the time to generate the model.

TABLE 4.6: Parameter optimization for Ridor: Varying the F parameter

F parameter	2	3	4	5	6	7	8
Avg. F-score	0.884	0.875	0.885	0.893	0.893	0.891	0.865

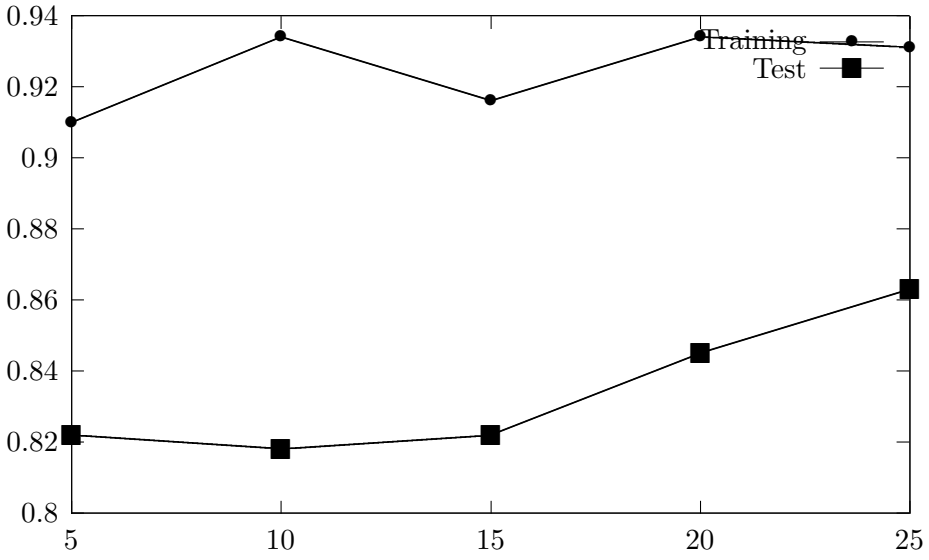


FIGURE 4.5: Learning curve for optimized Ridor

In table 4.7 is the minimum weight of instances in a split. So if a rule would induce a split it will be ignored if the split is too small. This can be useful against overfitting. On the other hand a too high value would make the algorithm unable to create specific rules.

TABLE 4.7: Parameter optimization for Ridor: Varying the N parameter

N parameter	1	2	3	4	5	6
Avg. F-score	0.887	0.903	0.890	0.904	0.882	0.878

In graph 4.6 we can see that the model behaves very similarly on both the training set and the test set. This could be a sign of high bias.

For the cost parameter C of SVM with RBF kernel there is a peak at 20.48 seen in table 4.8. This is a lot higher than the default value of 1.

TABLE 4.8: Parameter optimization for SVM with RBF kernel: Varying the C parameter

C parameter	0.16	0.32	0.64	1.28	2.56	5.12	10.24	20.48	40.96
Avg. F-score	0.58	0.63	0.71	0.727	0.791	0.801	0.803	0.807	0.801

Varying the e parameter not show much change except at very high values. As the f-score is constant in table 4.9 for most values the default value is kept.

For the G parameter a value of 0.008 performs best and is kept. Looking at table 4.10 we can see that a G value of 0.008 is the best choice.

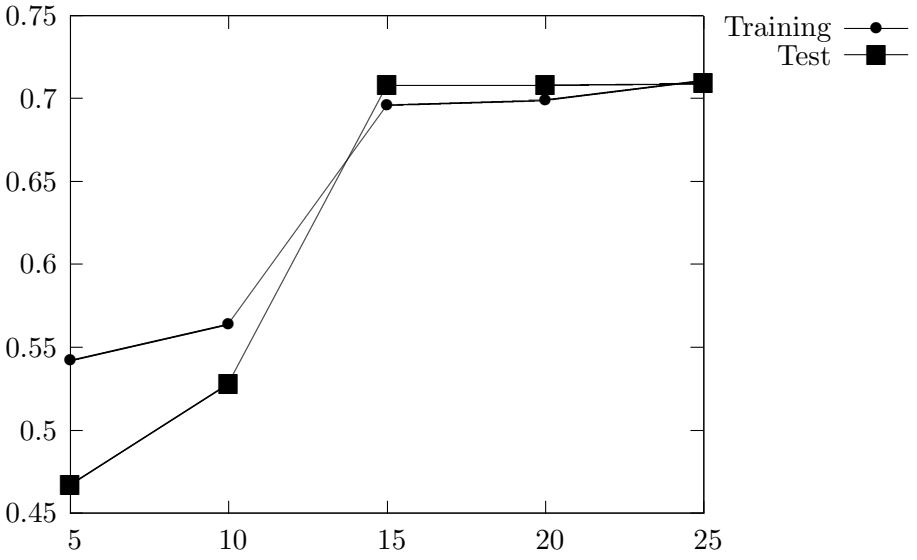


FIGURE 4.6: Learning curve for unoptimized SVM with RBF kernel

TABLE 4.9: Parameter optimization for SVM with RBF kernel: Varying the e parameter

e parameter	0.0001	0.001	0.01	0.1	1	10
Avg. F-score	0.807	0.807	0.807	0.802	0.804	0.004

TABLE 4.10: Parameter optimization for SVM with RBF kernel: Varying the G parameter

G parameter	0	0.001	0.002	0.004	0.008	0.016	0.032	0.064	0.128
Avg. F-score	0.807	0.715	0.76	0.784	0.813	0.81	0.781	0.76	0.755

When optimizing the parameters and making the cost parameter higher we can see in graph 4.7 that the model now performs much better on the training set and slightly better on the test set. This is more like the kind of learning curve we would expect, that there is a higher error rate when using the test set. In other words that the performance does not perfectly generalize to other data. Before optimization the bias dominated, and even when testing against the training set the model could not make the right choices.

Compared to the RBF kernel the polynomial kernel in figure 4.8 is not constricted by bias in the same way. It performs very well on the training set. This performance does not generalize very well to the test set though. Some loss of performance is to be expected, but the slight dip when going from 20 to 25 recipes could be due to overfitting.

Table 4.11 shows that a degree of 3 performs better. This is good since with more free variables varying the other parameters should make a bigger difference. The downside

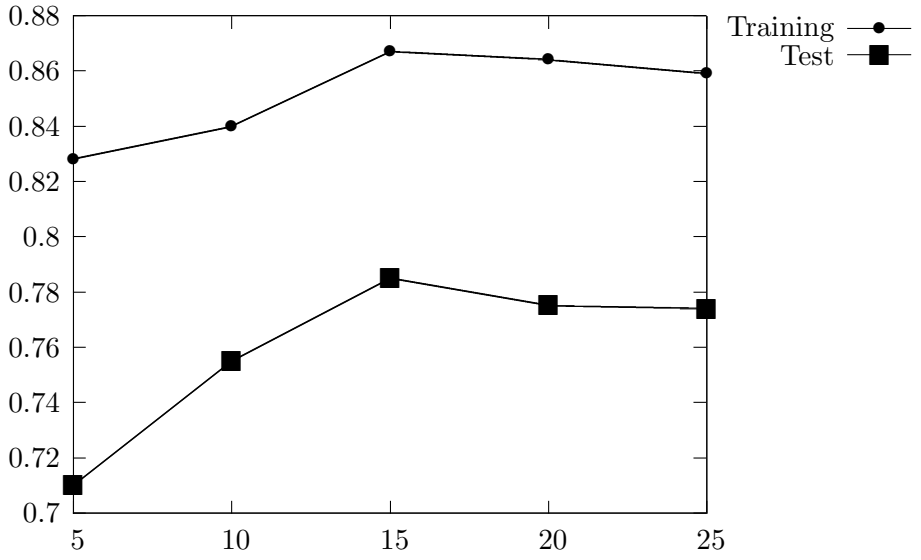


FIGURE 4.7: Learning curve for parameter optimized SVM with RBF kernel.

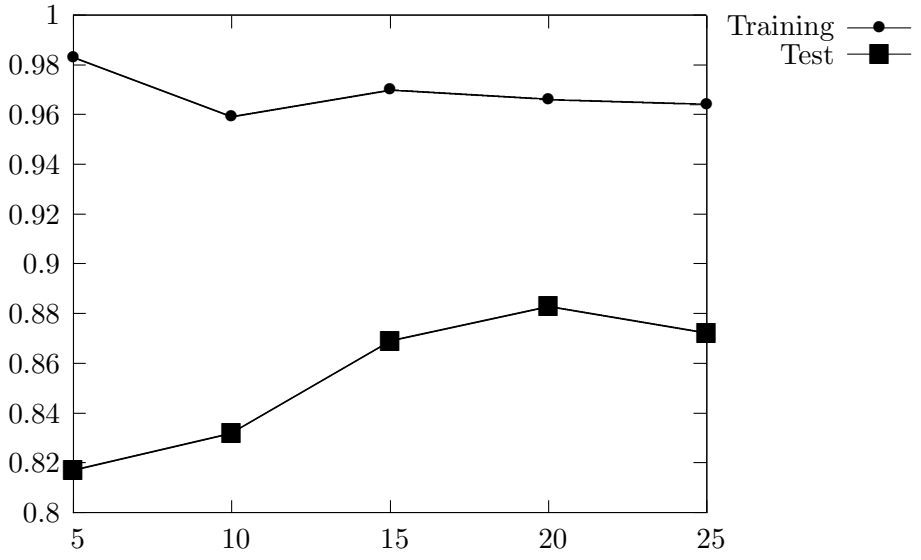


FIGURE 4.8: Learning curve for unoptimized SVM with polynomial kernel.

is that we saw signs of overfitting in the learning curve and a higher degree could make it worse.

TABLE 4.11: Parameter optimization for SVM with polynomial kernel: Varying the D parameter

D parameter	1	2	3	4	5	6	7	8
Avg. F-score	0.575	0.689	0.876	0.844	0.841	0.836	0.809	0.801

Despite the hint of overfitting seen in the learning curve, when optimizing the C parameter we again see in table 4.12 that a higher value performs better.

TABLE 4.12: Parameter optimization for SVM with polynomial kernel: Varying the C parameter

C parameter	0.04	0.08	0.16	0.32	0.64	1.28	2.56	5.12	10.24
Avg. F-score	0.795	0.823	0.86	0.694	0.694	0.879	0.877	0.873	0.863

The epsilon and gamma parameters, depicted in tables 4.13 and 4.14 are best left at their stock values.

TABLE 4.13: Parameter optimization for SVM with RBF kernel: Varying the e parameter

e parameter	0.0001	0.001	0.01	0.1	1	10
Avg. F-score	0.879	0.879	0.879	0.879	0.875	0.004

When looking at figure 4.9 we see that the model now works even better on the training data but the slope when going from 20 to 25 recipes on the test set is even more pronounced. In this case this form of parameter optimization did not work very well and rather enhanced the problem.

TABLE 4.14: Parameter optimization for SVM with polynomial kernel: Varying the G parameter

G parameter	0	0.004	0.008	0.016	0.032	0.064	0.128	0.256	0.512	1.024	2.048
Avg. F-score	0.879	0.783	0.851	0.879	0.863	0.857	0.858	0.869	0.872	0.869	0.791

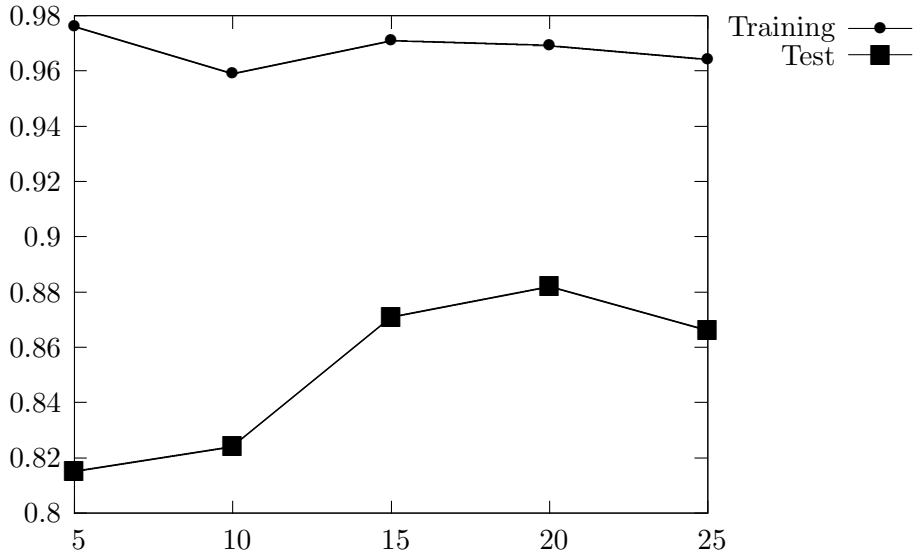


FIGURE 4.9: Learning curve for parameter optimized SVM with polynomial kernel.

We tried boosting the optimized versions of all algorithms with AdaBoost. Results vary as can be seen in table 4.15. Boosting has a clearly positive effect on the bias limited RBF kernel, but actually decreases F-score for C4.5. The reason for this can be that C4.5 is already not limited by bias, and boosting actually introduces more overfitting. It can also be because boosting handles many classes badly. It requires that the base model can classify all classes at above 50 percent and for harder classes like intermediate product this could be a problem. For the final evaluation of the algorithms, the non booster version was used.

TABLE 4.15: Effects of boosting on the optimized models, F-score

Algorithm	C4.5	IBk	Ridor	SVM RBF	SVM poly
Without boosting	0.888	0.856	0.863	0.774	0.866
With AdaBoost	0.879	0.863	0.876	0.858	0.874

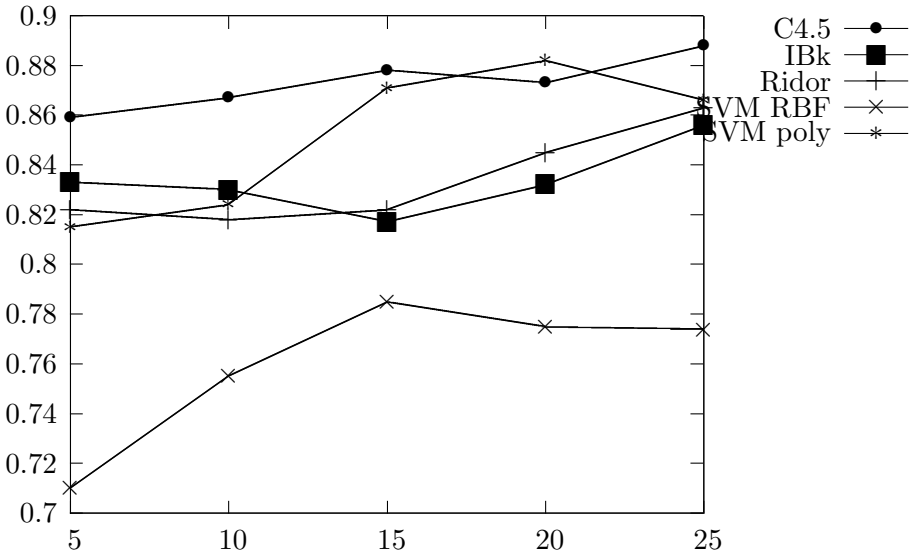


FIGURE 4.10: Side by side comparison of algorithms.

In table 4.16 the results when using only one group of attributes at a time is listed. As we can see the two types of attributes that give the most effect is dictionary and WordNet. A reason that the dictionary is very effective is that the domain is limited compared to for example news paper articles which is often used as a test in natural language processing. Another form of baseline that can be considered is just always labeling as the most common class: other. In the test set 204 out of 368 instances are of the class other so always guessing 'other' would give an F-score of 0.713.

TABLE 4.16: Effects of boosting on the optimized models, F-score

Attribute	Word class	Dictionary	In ingredients	Neighbours	WordNet
F-score	0.693	0.856	0.483	0.708	0.828

TABLE 4.17: Confusion matrix of semantic word classes, 1 = ingredient, 2=tool, 3=action, 4=intermediate product, 5=unit, 6=amount, 7=other. Horizontal axis is what class an instance was classified as, vertical axis the actual class.

	1	2	3	4	5	6	7
1	41	0	1	0	0	0	1
2	1	19	0	0	0	0	3
3	1	0	41	1	0	0	5
4	13	0	0	4	0	0	4
5	0	0	0	0	17	0	0
6	0	0	0	0	0	12	0
7	2	1	4	0	0	0	197

Table 4.17 shows a confusion matrix of classification done by the C4.5 algorithm with default parameters when applied on the gold standard set. The most common error is that intermediate products are classified as ingredients.

4.2 Dependencies between steps

For the dependency task we again measure the F-score. But we also measure specifically the recall of yes-instances. The reasoning behind this is that if you miss a no instance, the recipe generated is still usable but has less room for optimization. If you on the other hand miss a yes-instance it is possible to create a non functional recipe.

Table 4.11 shows the learning curve for C45 with both F-score and yes-recall. We can see that the yes-recall does improve with more training data but flattens out at 20 and 25 recipes.

For parameter optimization the choice is fairly straight forward as C of 0.15 and M of 2 gives the best performance both in F-score and yes-rating. See tables 4.18 and 4.19

TABLE 4.18: Parameter optimization for C4.5: Varying the C parameter

C parameter	0.05	0.15	0.25	0.35	0.45	0.55	0.65
Avg. F-score	0.887	0.901	0.894	0.879	0.872	0.861	0.861
Yes recall	0.703	0.757	0.73	0.649	0.622	0.622	0.622

TABLE 4.19: Parameter optimization for C4.5: Varying the M parameter

M parameter	1	2	3	4	5	6	7	8
Avg. F-score	0.897	0.901	0.894	0.871	0.871	0.904	0.882	0.882
Yes recall	0.676	0.757	0.73	0.703	0.703	0.703	0.703	0.703

However when analyzing the results in the learning curves in **??** we can see that while the F-score improved slightly the yes-recall actually decreased.

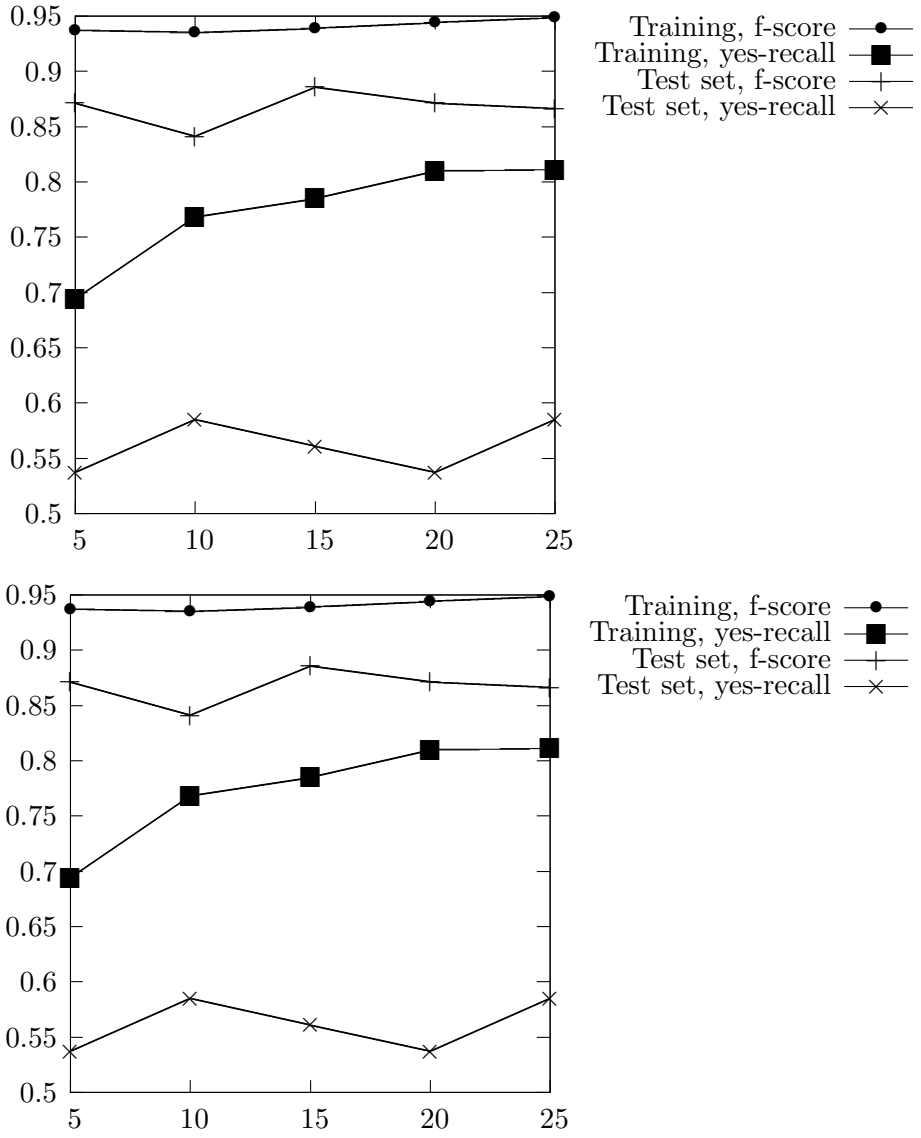


FIGURE 4.11: Learning curve for unoptimized C4.5

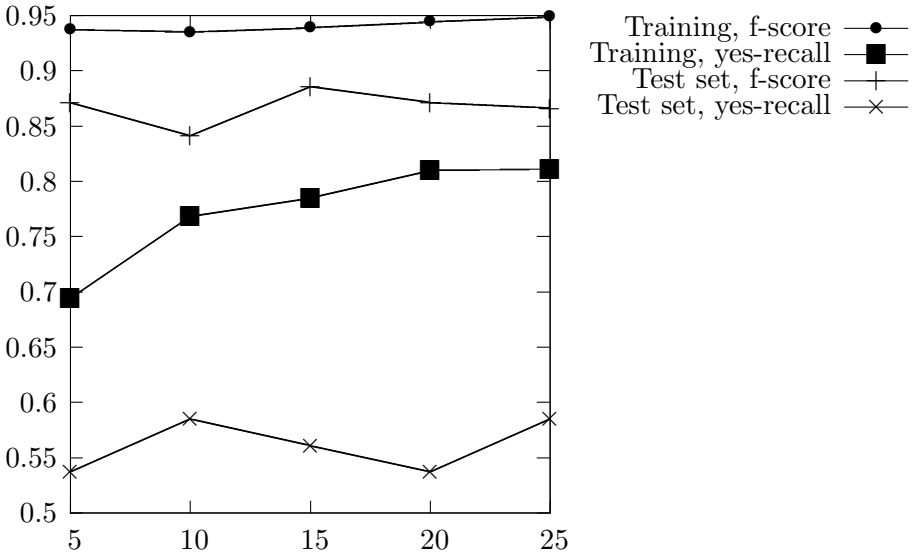


FIGURE 4.12: Learning curve for optimized C4.5

For IBk the learning curve for the training set is again a flat line, seen in graph 4.13. This is because all data is saved and the default setting of K is 1. The performance on the test set improves slightly with data for F-score and more substantially for yes-recall.

A higher K value of 5 gives a better performance, see table 4.20. however there is no effect either way of using mean squared or distance weighing as seen in tables 4.21 and 4.22.

TABLE 4.20: Parameter optimization for IBk: Varying the K parameter

M parameter	1	2	3	4	5	6	7
Avg. F-score	0.885	0.867	0.890	0.883	0.906	0.888	0.892
Yes recall	0.649	0.757	0.649	0.730	0.730	0.730	0.676

TABLE 4.21: Parameter optimization for IBk: Effects of using mean squared.

Mean squared	On	Off
Avg. F-score	0.906	0.906
Yes recall	0.730	0.730

TABLE 4.22: Parameter optimization for IBk: Effects of using distance weighing.

Mean squared	None	1/d	1-d
Avg. F-score	0.906	0.906	0.906
Yes recall	0.730	0.730	0.906

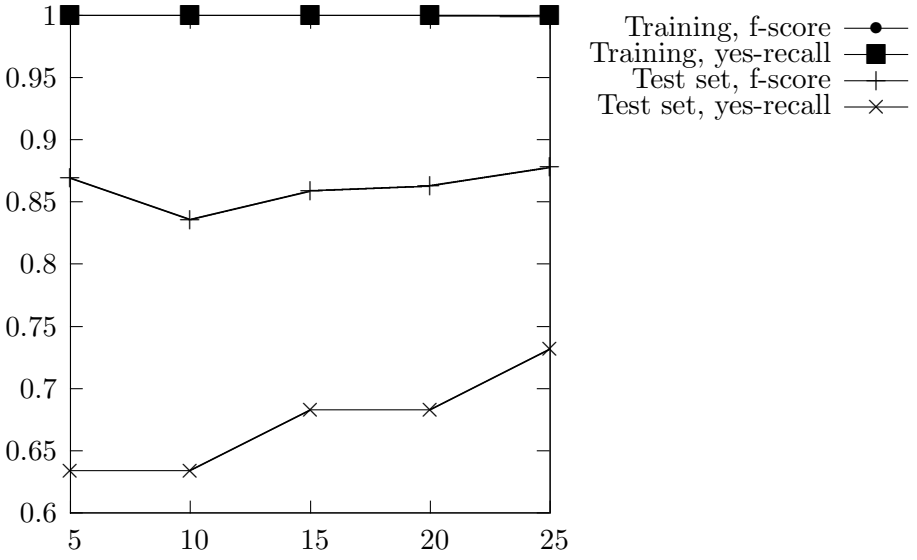


FIGURE 4.13: Learning curve for unoptimized IBk

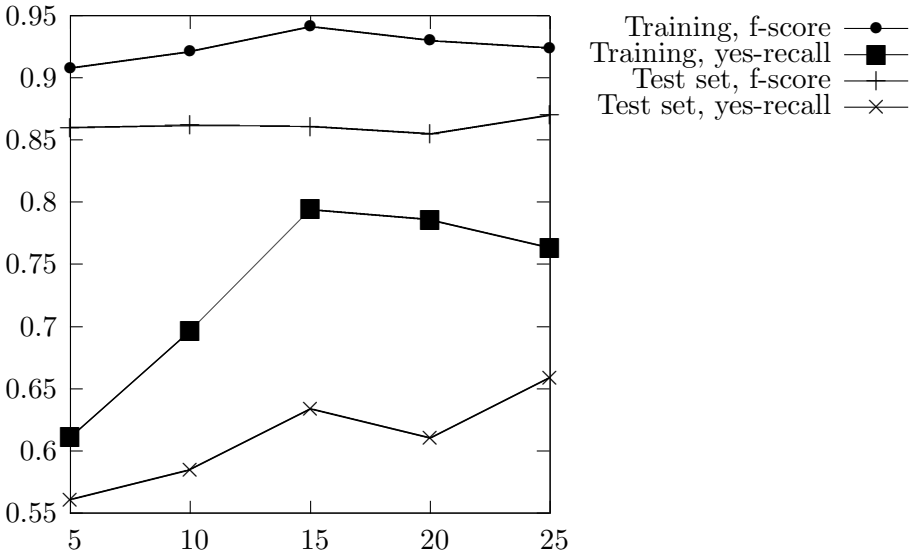


FIGURE 4.14: Learning curve for optimized IBk

For the optimized version in graph 4.14 it is no longer a flat line for the training set. This is because we choose a K of 5. Since we now look at the 5 closest neighbours there is a larger error. There is no effect of distance weighing and enabling mean squared.

When looking at 4.15 we see a big bump for yes-recall at 15 recipes. This has been observed in other algorithms as well but is more pronounced here. The reason for this could be overfitting or since only yes-recall is so majorly affected it could be that the recipes added when going from 10 to 15 are similar to the previous ones.

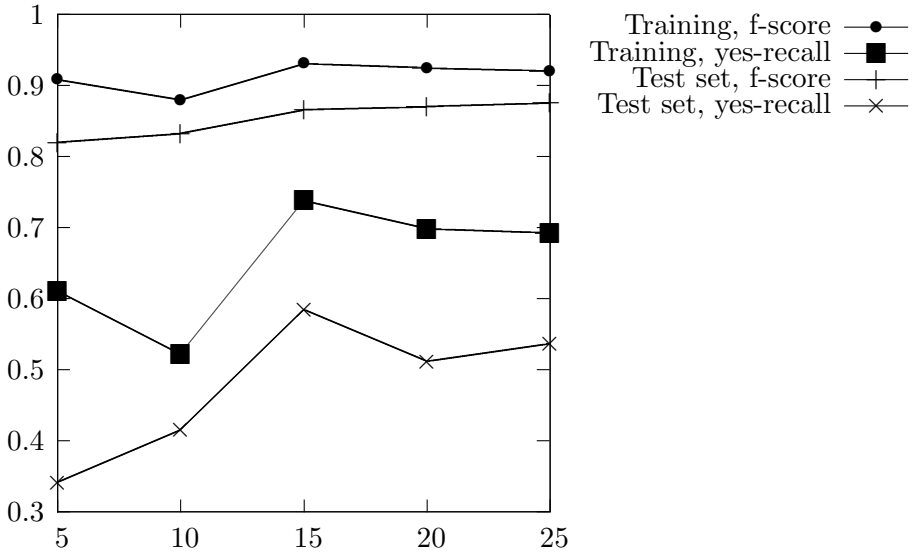


FIGURE 4.15: Learning curve for unoptimized Ridor

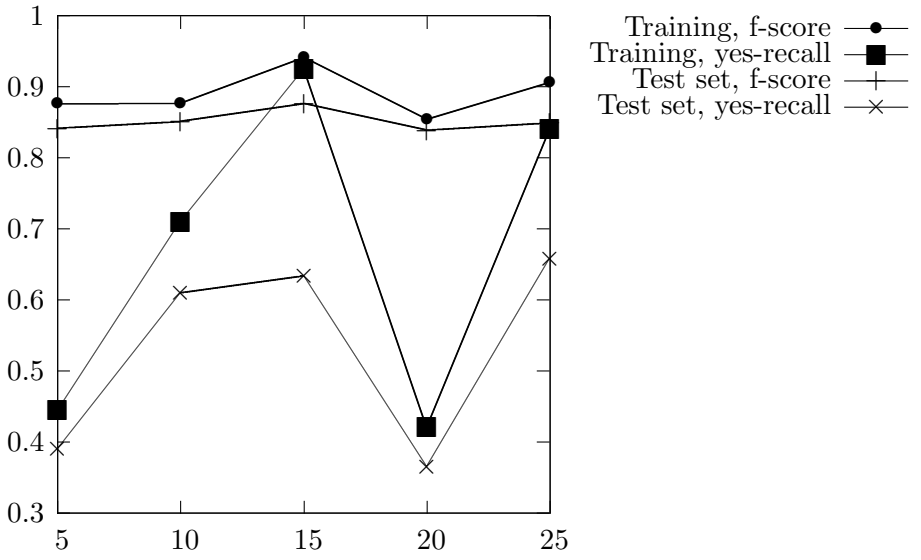


FIGURE 4.16: Learning curve for optimized Ridor

We choose F of 7 as seen in 4.23. 6 gives a higher F-score but as noted before we want to prioritize yes-rating. For S and N we choose 1.

TABLE 4.23: Parameter optimization for Ridor: Effects of changing the F parameter.

F parameter	2	3	4	5	6	7	8	9	10
Avg. F-score	0.858	0.890	0.864	0.835	0.891	0.875	0.858	0.876	0.858
Yes recall	0.514	0.649	0.514	0.649	0.784	0.811	0.514	0.595	0.514

For both the S and N parameters a value of 1 is chosen as seen in tables 4.24 and 4.25.

TABLE 4.24: Parameter optimization for Ridor: Effects of changing the S parameter.

S parameter	1	2	3	4	5
Avg. F-score	0.875	0.829	0.837	0.873	0.731
Yes recall	0.811	0.432	0.676	0.649	0.135

TABLE 4.25: Parameter optimization for Ridor: Effects of changing the N parameter.

N parameter	1	2	3	4	5
Avg. F-score	0.875	0.875	0.875	0.874	0.839
Yes recall	0.811	0.811	0.811	0.589	0.432

After optimization the bump around 15 recipes is even more extreme, and a loss at 20. Shown in figure 4.16

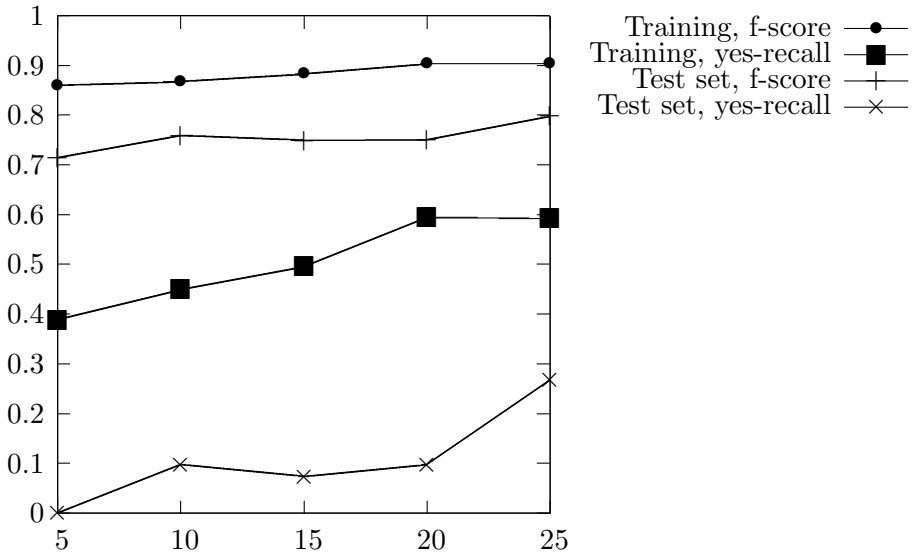


FIGURE 4.17: Learning curve for unoptimized SVM with RBF kernel.

The increase in training yes-recall for SVM with RBF kernel in graph 4.17 is peculiar. As there are more instances to take into account the yes-recall increases. For the training set this normally gets harder. We have no explanation for this effect.

In table 4.26 we can see that the C parameter gives better results, but levels out at 5.12. Varying the e parameter does not have much effect except worse results at very high values. Finally a G parameter of 0.004 increases the yes-recall substantially. This is displayed in tables 4.27 and 4.28.

The optimized version gets better results shown in figure 4.18. The performance on the test set for low amounts of training data is actually worse. This is a little bit peculiar. It could be because all the parameter optimization is done with the full 25 recipes of training data and that it utilizes some information that is missing in the first recipes.

TABLE 4.26: Parameter optimization for SVM with RBF kernel: Effects of changing the C parameter.

C parameter	0.16	0.32	0.64	1.28	2.56	5.12	10.24	20.48	40.96	81.92
Avg. F-score	0.674	0.674	0.688	0.794	0.796	0.823	0.823	0.823	0.816	0.823
Yes recall	0	0	0.027	0.297	0.378	0.486	0. 486	0.486	0.459	0.486

TABLE 4.27: Parameter optimization for SVM with RBF kernel: Effects of changing the e parameter.

e parameter	0.0001	0.001	0.01	0.1	1	10
Avg. F-score	0.823	0.823	0.823	0.823	0.81	0.674
Yes recall	0.486	0.486	0.486	0.486	0.459	0

TABLE 4.28: Parameter optimization for SVM with RBF kernel: Effects of changing the G parameter.

G parameter	0.0	0.004	0.008	0.016	0.032	0.064	0.128	0.256	0.512
Avg. F-score	0.823	0.828	0.854	0.85	0.844	0.813	0.804	0.752	0.698
Yes recall	0.486	0.378	0.703	0.622	0.541	0.486	0.405	0.162	0.054

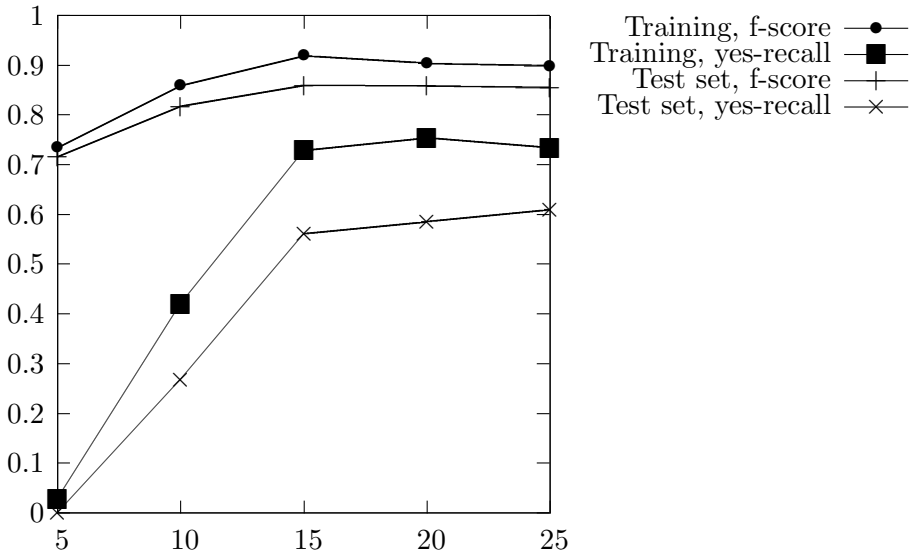


FIGURE 4.18: Learning curve for optimized SVM with RBF kernel.

For SVM with polynomial kernel the yes-recall gets worse when going to 25 recipes. A possible case of overfitting. For degree we choose 3, even though it has worse performance in table 4.29. A higher degree gives the other parameters a higher impact and therefore we try it to explore more options.

The cost parameter C is set to 0.64 as other settings have worse effects. This is depicted in table 4.30. Again high values of e gives bad results, shown in table 4.31. The G parameter is best left at the stock setting (symbolized as 0.0 in table 4.32). When looking at the optimized curves in graph 4.20 we see that the results are now better for low amounts of training data, but actually slightly worse for larger amounts. This can be due to the greedy and therefore imperfect nature of our parameter choice, but could also be because we choose a higher degree despite the results. A higher degree would increase overfitting rather than decrease it as the unoptimized learning curves indicated a need of. And it is consistent with getting better results for little training data and detoriating with more data.

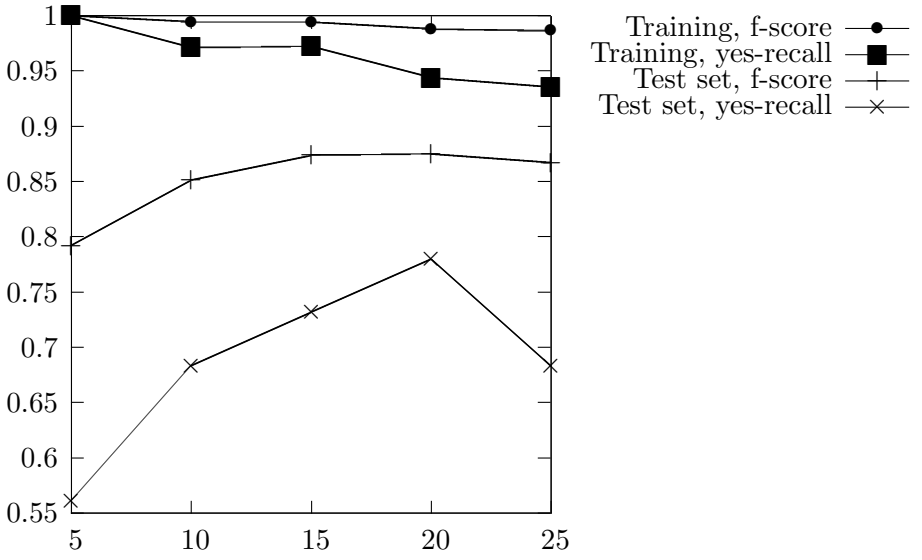


FIGURE 4.19: Learning curve for unoptimized SVM with polynomial kernel.

TABLE 4.29: Parameter optimization for SVM with RBF kernel: Effects of changing the degree.

Degree	1	2	3	4	5	6	7	8	9	10
Avg. F-score	0.862	0.899	0.894	0.854	0.853	0.853	0.866	0.869	0.869	0.869
Yes recall	0.757	0.703	0.73	0.703	0.676	0.676	0.73	0.676	0.676	0.676

TABLE 4.30: Parameter optimization for SVM with polynomial kernel: Effects of changing the C parameter.

C parameter	0.04	0.08	0.16	0.32	0.64	1.28	2.56	5.12	10.24	20.48
Avg. F-score	0.864	0.879	0.875	0.888	0.894	0.871	0.864	0.869	0.872	0.85
Yes recall	0.595	0.649	0.758	0.73	0.73	0.703	0.676	0.676	0.73	0.622

TABLE 4.31: Parameter optimization for SVM with polynomial kernel: Effects of changing the e parameter.

e parameter	0.001	0.01	0.1	1	10	
Avg. F-score	0.894	0.894	0.894	0.888	0.908	0.674
Yes recall	0.73	0.73	0.73	0.73	0.784	0

TABLE 4.32: Parameter optimization for SVM with polynomial kernel: Effects of changing the G parameter.

G parameter	0.0	0.008	0.016	0.032	0.064	0.128	0.256	0.512	1.024	2.048
Avg. F-score	0.908	0.743	0.839	0.873	0.882	0.872	0.864	0.864	0.864	0.864
Yes recall	0.784	0.162	0.486	0.649	0.703	0.73	0.676	0.676	0.676	0.676

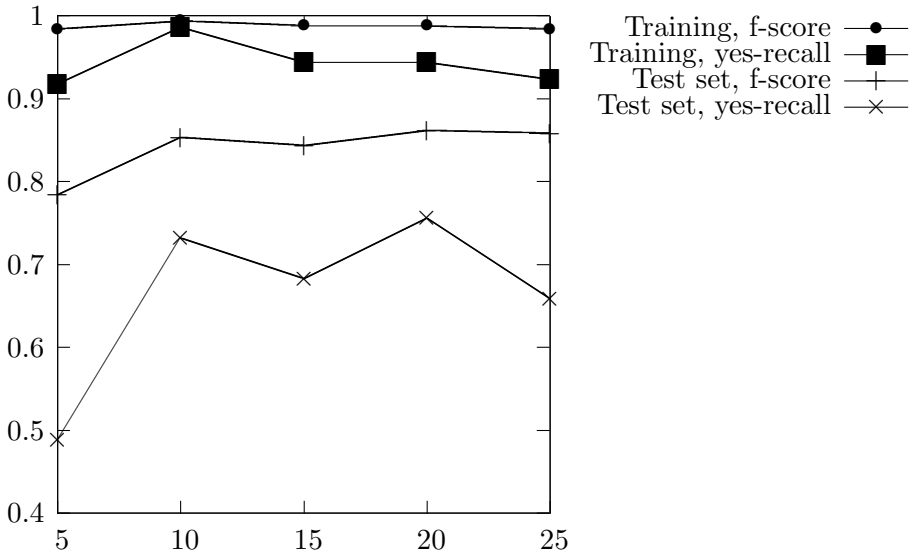


FIGURE 4.20: Learning curve for optimized SVM with polynomial kernel.

4.2.1 Side by side evaluation

C4.5 wins out when it comes to F-score as seen in figure 4.21. However 3 out of the 5 algorithms perform well even at a small amount of training data. This suggests that adding more training data is not likely to benefit any algorithm. It may be capped due to lack of information in the chosen attributes. Both Ridor and C4.5 exhibit a peak at 15 recipes. This could as discussed before mean that those 5 recipes added are very similar to the 5 before it, or that we have some overfitting. If it is overfitting it would be a possible area for improvement.

The performance on yes-rating of all the algorithms are fairly similar (shown in figure 4.22) at the maximum amount of learning data. They differ more in the learning process before that, with two algorithms not gaining much from more training data. This suggests that again, more training data or better chosen arguments will not improve the score further. C4.5 does better with less training data, suggesting a possible room for improvement by eliminating overfitting, especially since we can see tendencies toward that in the f-score.

4.2.2 Effecs of boosting

For dependency detection the effects of boosting are slightly positive in terms of f-score for some of the algorithms and negative or neutral for yes-rating. Given our applications focus on yes-rating, boosting does not seem like a worthwhile pursuit.

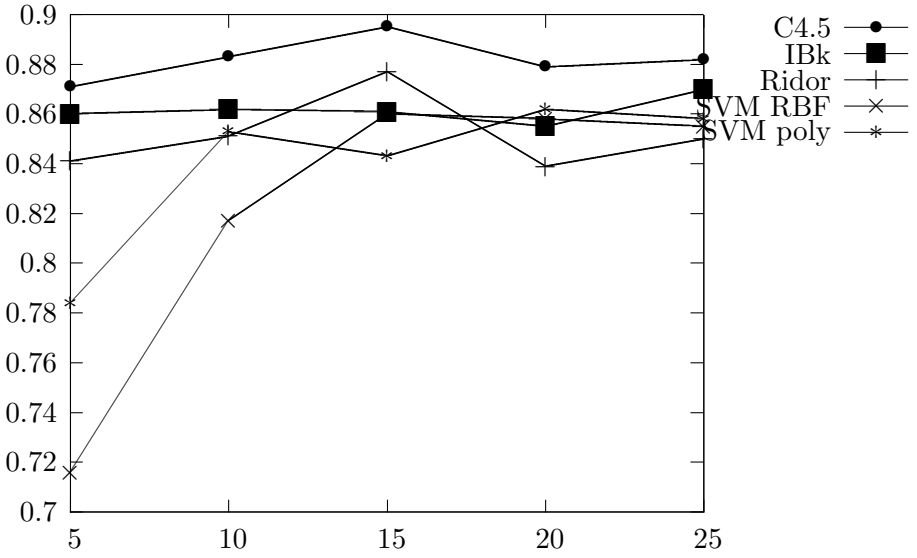


FIGURE 4.21: Side by side comparison of learning algorithms, f-score.

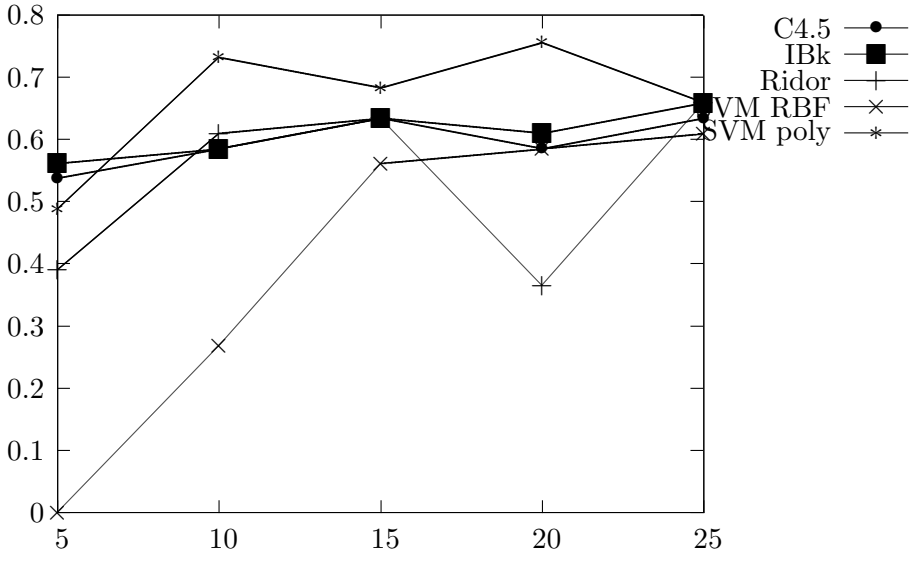


FIGURE 4.22: Side by side comparison of learning algorithms, yes-recall.

TABLE 4.33: Effects of boosting on the optimized models, F-score

Algorithm	C4.5	IBk	Ridor	SVM RBF	SVM poly
Generic f-score 0.882	0.870	0.850	0.885	0.855	0.858
Generic yes-r 0.634	0.659	0.659	0.61	0.61	0.659
Boosted f-score 0.883	0.815	0.868	0.868	0.868	0.84
Boosted yes-r 0.585	0.61	0.61	0.61	0.649	0.561

Chapter 5

Conclusion

We have tested 5 different algorithms, and systematically optimized their parameters. We used a greedy method for optimization. While it is not optimal, and in some cases actually yielded a worse result than the default parameters we do not believe there is much more performance to be gained through improved parameter optimization. This is because most algorithms hit roughly the same ceiling. This also shows that the choice of algorithm does not seem to be the biggest obstacle for greater performance. In some of our test results we saw some signs of overfitting, but not enough for it to be the main factor holding every algorithm back.

Instead we believe that the attributes do not carry enough information to go beyond this level of performance. Especially the task of detecting dependencies is hard while staying at the level of isolated words. More attributes that stem from analysing the sentence structure and the semantics of the text are needed.

The results of the semantic word classification task are possibly good enough to be used in an application. The biggest problem is differentiating between an intermediate product and an ingredient. This could be because of our approach to tokenization. In the case of an ingredient 'chicken filet' and an intermediate product 'chicken broth' both contain the word chicken we will have an overlap between the classes. Again, some more information about the whole sentence would help in this scenario. We get some of this from the adjacent words attributes since if chicken occurs close to a word we know is closely associated with intermediate products (like broth) we can draw the conclusion that it too should be classified as an intermediate product. Another problem is the domain itself. Sometimes what is an intermediate product in one recipe is bought ready-made as an ingredient in another.

For dependency detection the picture is more grim. It suffers more because we mostly have local information. In the case of coreference resolution the algorithm we used was trained on newspaper articles and performed very poorly on the recipe data. Our replacement only looked at exact matches of a word so it did not give true coreference resolution. None of the algorithms got a high enough score on yes-recall to create functional recipes on its own. Too many important dependencies are missed.

The choice of metrics can also play a role. There is some criticism against the use of F-score in classification problems [56]. This is because recall (which is part of F-score) is not as relevant when you are classifying all instances and no instance is more important that another. However, in the dependency problem the yes instances are more important. And in the semantic word class problem the Other class is less important and is also the largest class.

5.1 Future work

As stated, the coreference resolution did not work well because it was trained on other types of data. Also the word classifier in Stanford CoreNLP had some problems. Training these tools on recipe texts could increase performance of both the semantic word classification and dependency detection.

In the word classification task the biggest problem is differentiating ingredients from intermediate products since some words like 'chicken' can be in both types of phrases. To remedy this the Viterbi algorithm [57] or similar could be used. So that in a phrase 'chicken broth' the two words are not allowed to have different classes. It must either fully be an intermediate product or it must fully be a an ingredient.

The parameter optimization could be improved. The current approach is very greedy. Due to combinatorial explosion it is probably not feasible to test the algorithms with many parameters exhaustively. But perhaps doing a more exhaustive search for a smaller subset of parameters. Of course, one would then have to determine which parameters are more important.

Lastly, since the results of this thesis seem to be limited mostly by the information carried by the attributes we recommend going beyond the local approach used here. Analysing the sentence structure to figure out what kind of instruction each sentence encodes. The imperative nature of recipe instructions could probably be exploited in some way, since typical sentence structures will be repeated very often compared to other domains such as newspaper articles.

Appendix A

An annotated recipe

```
1. Preheat oven to 350 degrees F (175 degrees C).[0]
2. Lightly grease a 9 inch pie pan.[0]
3. Heat oil in a large skillet over medium-high heat.[0]
4. Add onions and cook, stirring occasionally, until onions are soft.[3]
5. Stir in spinach and continue cooking until excess moisture has evaporated.[4]
6. In a large bowl, combine eggs, cheese, salt and pepper.[0]
7. Add spinach mixture and stir to blend.[5,6]
8. Scoop into prepared pie pan.[7,2]
9. Bake in preheated oven until eggs have set, about 30 minutes.[8,1]
10. Let cool for 10 minutes before serving.[9]

Preheat\3 oven\2 to\7 350\5 degrees\6 F\6 -LRB-\7 175\5 degrees\6 C\6 -RRB-\7 .\7
Lightly\7 grease\3 a\7 9\5 inch\6 pie\2 pan\2 .\7
Heat\3 oil\1 in\7 a\7 large\2 skillet\2 over\7 medium-high\5 heat\7 .\7
Add\3 onions\1 and\7 cook\3 ,\7 stirring\3 occasionally\7 ,\7 until\7 onions\1
are\7 soft\7 .\7
Stir\3 in\7 spinach\1 and\7 continue\7 cooking\3 until\7 excess\7 moisture\4 has
\7 evaporated\7 .\7
In\7 a\7 large\2 bowl\2 ,\7 combine\3 eggs\1 ,\7 cheese\1 ,\7 salt\1 and\7 pepper
\1 .\7
Add\3 spinach\4 mixture\4 and\7 stir\3 to\7 blend\3 .\7
Scoop\3 into\7 prepared\7 pie\2 pan\2 .\7
Bake\3 in\7 preheated\7 oven\2 until\7 eggs\1 have\7 set\7 ,\7 about\7 30\5
minutes\6 .\7
Let\3 cool\3 for\7 10\5 minutes\6 before\7 serving\3 .\7
```

Bibliography

[1] Allrecipes, April 2013. URL http://www.allrecipes.com.

[2] Cooks.com recipe search, April 2013. URL http://www.cooks.com.

[3] Bbc good food, April 2013. URL http://www.bbcgoodfood.com/.

[4] Cooking for engineers, April 2013. URL http://www.cookingforengineers.com.

[5] Norman Ihle, Régis Newo, Alexandre Hanft, Kerstin Bach, and Meike Reichle. Cookiis - A Case-Based Recipe Advisor. In Sarah Jane Delany, editor, *Workshop Proceedings of the 8th International Conference on Case-Based Reasoning*, pages 269–278, Seattle, WA, USA, July 2009.

[6] Alexandre Blansché, Julien Cojan, Valmi Dufour-Lussier, Jean Lieber, Pascal Molli, Emmanuel Nauer, Hala Skaf-Molli, and Yannick Toussaint. TAAABLE 3: Adaptation of ingredient quantities and of textual preparations. In *18h International Conference on Case-Based Reasoning - ICCBR 2010, "Computer Cooking Contest" Workshop Proceedings*, Alessandria, Italy, 2010. URL http://hal.inria.fr/inria-00526663.

[7] Graeme Hirst and David St-Onge. Lexical chains as representations of context for the detection and correction of malapropisms. In Christiane Fellbaum, editor, *WordNet: An Electronic Lexical Database*, pages 305–332. MIT Press, 1998.

[8] Qian Zhang, Rong Hu, Brian Mac Namee, and Sarah Jane Delany. Back to the future: Knowledge light case base cookery. In Martin Schaaf, editor, *ECCBR Workshops*, pages 239–248, 2008. URL http://dblp.uni-trier.de/db/conf/ewcbr/eccbr2008w.html#ZhangHND08.

[9] Kristian J. Hammond. Chef: A model of case-based planning. In Tom Kehler, editor, *AAAI*, pages 267–271. Morgan Kaufmann, 1986. URL http://dblp.uni-trier.de/db/conf/aaai/aaai86-1.html#Hammond86.

[10] M. Bollini, J. Barry, and D. Rus. Bakebot: Baking cookies with the pr2. In *IROS PR2 Workshop*, September 2011. URL http://www.iros2011.org/

WorkshopsAndTutorialsProceedings/FW9/mbollini_mit.pdf. Extended abstract.

[11] David Faure and Claire Nédellec. Knowledge acquisition of predicate argument structures from technical texts using machine learning: the system asium, 1999.

[12] Ziqi Zhang, Philip Webster, Victoria Uren, Andrea Varga, and Fabio Ciravegna. Automatically extracting procedural knowledge from instructional texts using natural language processing. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, may 2012. European Language Resources Association (ELRA). ISBN 978-2-9517408-7-7.

[13] Aryana Tavanai. Learning relational patterns mined from temporally rich textual data. Master's thesis, University of Leeds, 2011.

[14] Michael Wiegand, Benjamin Roth, and Dietrich Klakow. Data-driven knowledge extraction for the food domain. In Jeremy Jancsary, editor, *Proceedings of KONVENS 2012*, pages 21–29. ÖGAI, September 2012. URL http://www.oegai.at/konvens2012/proceedings/07_wiegand12o/.

[15] Dan Tasse and Noah A. Smith. Sour cream: Toward semantic processing of recipes, 2008. Report of preliminary work.

[16] Reiko Hamada, Ichiro Ide, Shuichi Sakai, and Hidehiko Tanaka. Structural analysis of cooking preparation steps in japanese. In Kam-Fai Wong, Dik Lun Lee, and Jong-Hyeok Lee, editors, *IRAL*, pages 157–164. ACM, 2000. ISBN 1-58113-300-6. URL http://dblp.uni-trier.de/db/conf/iral/iral2000.html#HamadaIST00.

[17] Shinsuke Mori, Tetsuro Sasada, Yoko Yamakata, and Koichiro Yoshino. A machine learning approach to recipe text processing. In Amélie Cordier and Emmanuel Nauer, editors, *Proceedings of the Cooking with Computers workshop (CwC)*, pages 29–35, 2012.

[18] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, Amsterdam, 3 edition, 2011. ISBN 978-0-12-374856-0.

[19] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2009. ISBN 0131873210.

[20] Jesús Maudes, JuanJ. Rodríguez, and César García-Osorio. Cascading for nominal data. In Michal Haindl, Josef Kittler, and Fabio Roli, editors, *Multiple Classifier Systems*, volume 4472 of *Lecture Notes in Computer Science*, pages 231–240. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-72481-0. doi: 10.1007/978-3-540-72523-7_24. URL http://dx.doi.org/10.1007/978-3-540-72523-7_24.

[21] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. ISSN 0885-6125. doi: 10.1007/BF00994018. URL http://dx.doi.org/10.1007/BF00994018.

[22] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. *Annals of Statistics*, 36:1171–1220, 2008. URL http://arxiv.org/pdf/math/0701907.

[23] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[24] H. Drucker, S. Wu, and V.N. Vapnik. Support vector machines for spam categorization. *Neural Networks, IEEE Transactions on*, 10(5):1048–1054, 1999. ISSN 1045-9227. doi: 10.1109/72.788645.

[25] Jakub Zavrel, Sven Degroeve, Anne Kool, Walter Daelemans, and Kristiina Jokinen. Diverse classifiers for nlp disambiguation tasks comparison, optimization, combination, and evolution, 2000.

[26] Taku Kudo and Yuji Matsumoto. Chunking with support vector machines. In *Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies*, NAACL '01, pages 1–8, Stroudsburg, PA, USA, 2001. Association for Computational Linguistics. doi: 10.3115/1073336.1073361. URL http://dx.doi.org/10.3115/1073336.1073361.

[27] Yaoyong Li, Kalina Bontcheva, and Hamish Cunningham. Svm based learning system for information extraction. In *In Proceedings of Sheffield Machine Learning Workshop, Lecture Notes in Computer Science*. Springer Verlag, 2005.

[28] Tomohiro Mitsumori, Masaki Murata, Yasushi Fukuda, Kouichi Doi, and Hirohumi Doi. Semantic role labeling using support vector machines. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 197–200, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W/W05/W05-0629.

[29] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: An update. *SIGKDD*

Explorations, 11(1):10–18, 2009. URL http://www.sigkdd.org/explorations/issues/11-1-2009-07/p2V11n1.pdf.

[30] György Szarvas, Richárd Farkas, and András Kocsor. A multilingual named entity recognition system using boosting and c4.5 decision tree learning algorithms. In Ljupco Todorovski, Nada Lavrac, and Klaus P. Jantke, editors, *Discovery Science*, volume 4265 of *Lecture Notes in Computer Science*, pages 267–278. Springer, 2006.

[31] Emili Sapena, Lluís Padró, and Jordi Turmo. Relaxcor participation in conll shared task on coreference resolution. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*, CONLL Shared Task '11, pages 35–39, 2011. ISBN 9781937284084. URL http://dl.acm.org/citation.cfm?id=2132936.2132939.

[32] Matjaz Jursic, Igor Mozetic, Tomaz Erjavec, and Nada Lavrac. Lemmagen: Multilingual lemmatisation with induced ripple-down rules. *J. UCS*, 16(9):1190–1214, 2010.

[33] Dat Quoc Nguyen, Dai Quoc Nguyen, Son Bao Pham, and Dang Duc Pham. Ripple down rules for part-of-speech tagging. In *Proceedings of the 12th International Conference on Computational Linguistics and Intelligent Text Processing - Volume Part I*, pages 190–201, 2011.

[34] Dat Ba Nguyen and Son Bao Pham. Ripple down rules for vietnamese named entity recognition. In *ICCCI (1)*, pages 354–363, 2012. ISBN 978-3-642-34629-3.

[35] Roser Morante, Walter Daelemans, and Vincent Van Asch. A combined memory-based semantic role labeler of english. In A. Clark and K. Toutanova, editors, *Proceedings of the 12th Conference on Computational Natural Language Learning*, pages 208–212, Manchester, UK, 08/2008 2008. ACL, ACL. ISBN 978-1-905593-48-4. URL http://www.clips.ua.ac.be/papers/2008/20812.pdf.

[36] Richárd Farkas, Veronika Vincze, György Móra, János Csirik, and György Szarvas. The conll-2010 shared task: Learning to detect hedges and their scope in natural language text. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning — Shared Task*, pages 1–12, 2010. ISBN 978-1-932432-84-8.

[37] Li Baoli, Lu Qin, and Yu Shiwen. An adaptive k-nearest neighbor text categorization strategy. 3(4):215–226. ISSN 1530-0226.

[38] Xavier Carreras, Lluís Màrquez, and Lluís Padró. Named entity extraction using adaboost. In *Proceedings of the 6th Conference on Natural Language Learning - Volume 20*, pages 1–4. Association for Computational Linguistics, 2002.

[39] Xavier Carreras, Lluís Màrquez, and Lluís Padró. A simple named entity extractor using adaboost. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, pages 152–155. Association for Computational Linguistics, 2003.

[40] C. Leacock and M. Chodorow. Combining local context and wordnet similarity for word sense identification. In Christiane Fellfaum, editor, *MIT Press*, pages 265–283, 1998.

[41] Satanjeev Banerjee and Ted Pedersen. An adapted lesk algorithm for word sense disambiguation using wordnet. In *Proceedings of the Third International Conference on Computational Linguistics and Intelligent Text Processing*, pages 136–145, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-43219-1.

[42] Zhibiao Wu and Martha Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32Nd Annual Meeting on Association for Computational Linguistics*, ACL '94, pages 133–138. Association for Computational Linguistics, 1994.

[43] Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1*, pages 448–453. Morgan Kaufmann Publishers Inc., 1995. ISBN 1-55860-363-8, 978-1-558-60363-9.

[44] J.J. Jiang and D.W. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. In *Proc. of the Int'l. Conf. on Research in Computational Linguistics*, pages 19–33, 1997.

[45] Dekang Lin. An information-theoretic definition of similarity. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 296–304. Morgan Kaufmann Publishers Inc., 1998. ISBN 1-55860-556-8.

[46] Hideki Shima. Ws4j, april 2013. URL https://code.google.com/p/ws4j/.

[47] Encog machine learning framework, April 2013. URL http://www.heatonresearch.com/encog.

[48] Pyml - machine learning in python, April 2013. URL http://pyml.sourceforge.net/.

[49] Kristina Toutanova and Christopher D. Manning. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics - Volume 13*, EMNLP '00, pages 63–70, Stroudsburg,

PA, USA, 2000. Association for Computational Linguistics. doi: 10.3115/1117794.1117802. URL http://dx.doi.org/10.3115/1117794.1117802.

[50] Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 173–180, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics. doi: 10.3115/1073445.1073478. URL http://dx.doi.org/10.3115/1073445.1073478.

[51] Karthik Raghunathan, Heeyoung Lee, Sudarshan Rangarajan, Nathanael Chambers, Mihai Surdeanu, Dan Jurafsky, and Christopher Manning. A multi-pass sieve for coreference resolution. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 492–501, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=1870658.1870706.

[52] Heeyoung Lee, Yves Peirsman, Angel Chang, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. Stanford's multi-pass sieve coreference resolution system at the conll-2011 shared task. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*, CONLL Shared Task '11, pages 28–34, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. ISBN 9781937284084. URL http://dl.acm.org/citation.cfm?id=2132936.2132938.

[53] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, Valentin Tablan, Niraj Aswani, Ian Roberts, Genevieve Gorrell, Adam Funk, Angus Roberts, Danica Damljanovic, Thomas Heitz, Mark A. Greenwood, Horacio Saggion, Johann Petrak, Yaoyong Li, and Wim Peters. *Text Processing with GATE (Version 6)*. 2011. ISBN 978-0956599315. URL http://tinyurl.com/gatebook.

[54] Lingpipe, April 2013. URL http://alias-i.com/lingpipe/.

[55] Natural language toolkit, April 2013. URL http://nltk.org/.

[56] M. Sokolova, N. Japkowicz, and S. Szpakowicz. Beyond accuracy, f-score and roc: a family of discriminant measures for performance evaluation. *AI 2006: Advances in Artificial Intelligence*, pages 1015–1021, 2006.

[57] G. D. Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, March 1973. ISSN 0018-9219. doi: 10.1109/proc.1973.9030. URL http://dx.doi.org/10.1109/proc.1973.9030.