# Chapter 4

# Rule-based Learning

Theories in theoretical linguistics usually consist of sets of discrete non-statistical rules. However, the models that we have developed in the previous chapters have a different format. It would be interesting to obtain phonotactic models that consist of rules. We could obtain such models by converting our HMMs and our SRN models to sets of rules. However, there are also rule-based learning methods available which generate rule models. Given linguists's preference for rule-based description, we might expect these methods to perform better than the other learning methods we have applied to our problem, learning the structure of monosyllabic words. Therefore we would like to apply rule-based learning methods to this problem and compare their performance with the performances we have obtained in the previous chapters.

In this chapter we will describe a rule-based approach to learning the phonotactic structure of monosyllabic words. In the first section we will sketch the fundamentals of rule-based learning and point out some of its problems. The second section will outline the learning method we have chosen: Inductive Logic Programming (ILP). The learning experiments that we have performed with ILP will be presented in the third section. In the fourth section we will describe our work with more elaborate rule-based models which might perform better than the models used in section three. The final section contains some concluding remarks.

## 1 Introduction to Rule-based Learning

In this section we will present some basic learning theory and evaluate a few rule-based learning algorithms. We will start with the influence of positive and negative examples on the learning process. After this we will explain what kind of output we expect from the learning method. We will conclude the section by presenting some rule-based learning methods and discussing their problems.

## 1.1 Positive versus negative examples

We want to use a symbolic or rule-based learning technique for acquiring a rule model for the phonotactic structure of monosyllabic words. This means that we are looking for a learning method that can produce a rule-based model that can accept or reject strings for a certain language. The model should accept a string when it is a possible word in the language and it should reject it when it is a sequence of characters that cannot occur as a word in the language.

Theory about learning methods that can produce language models like described above is already available. One of the main papers in learning theory was written by E. Mark Gold in the sixties (Gold 1967). In the paper Gold uses a division of languages in different mathematical classes to characterize the type of information that a hypothetical learner requires to learn them. He proves that no infinite language can be learned by using positive examples only. This means that learners cannot be guaranteed to acquire a perfect model of an infinite language without being presented with examples of strings or sentences that do not appear in the language.

The research result of Gold is important for the work we will present in this chapter. In our earlier experiments we have presented the learning algorithms only with positive examples. We want to be able to compare the results of this chapter with the results of the previous chapters in a fair way. This prevents us from using negative examples in the remaining experiments. We want our language models to generate as few errors as possible and therefore it is important to find out if the language we are trying to learn is finite.

In chapter one we have restricted our dataset to monosyllabic words. The longest word in the Dutch orthographic data set contains nine characters. We can imagine that it is possible to construct a word of ten or eleven characters that could qualify to be a monosyllabic Dutch word. However we cannot imagine that it is possible to construct monosyllabic Dutch word of twenty characters or longer.[1] The class of monosyllabic words will be finite for all human languages. Thus it should be possible to acquire a good monosyllabic structure model by using positive examples only. This would not have worked for multisyllabic words since in languages like Dutch and German there does not seem to be a maximum length for words, particularly not for compound words.

[1]In some unusual contexts strings with an arbitrary number of repetitions of the same character, like *aaaah* and *brrrr*, may qualify as a monosyllabic word. The models which we will use will be able to learn these by adopting the following strategy: a sequence of two or more repeated characters indicates that the repetition in that context is allowed for any number of characters.

## 1.2 The expected output of the learning method

In his landmark paper Gold introduces the concept *learning in the limit*. Learners are presented with positive and/or negative examples of a certain domain. They have a partial model of this domain and adjusts this model for every example that they receive. At some moment they will have constructed a perfect model of the domain and future examples will not cause a change of the model. If the learners manage to construct such a perfect model at some point of time then they are said to have learned the training data in the limit.

The learner model put forward by Gold makes no assumptions about the format of the domain model that has to be learned. In case of a finite domain the model may well consist of rules which state that some X is in the domain if and only if X was presented as a positive example. This is the most simple domain model structure. A model structure like that will not perform well in the experiments we are planning to do because it is unable to generalize. In the previous chapters we have presented the learning methods with only a part of the positive data. After this we have tested the domain models with the remaining unseen positive data and some negative data. A simple domain model like described above would reject all the unseen positive test data. This is unacceptable.

In order to avoid this problem the domain model must be able to generalize. Based on positive data only it must be able to decide that the unseen positive test data is correct. In order for the model to be able to do that, we will have to put generalization knowledge in the learning model. The question is what kind of generalization knowledge the model needs to contain. If the knowledge is too restrictive the domain model will reject too much positive test data and if it is too extensive the domain model will accept negative data as well. We will return to this question in section 2.

In the previous two chapters we have worked with Hidden Markov Models and neural network models that represent the structure of monosyllabic words. A disadvantage of these two groups of models is that they hide knowledge in sets of numbers. It is hard to explain the behavior of these models in a way that makes much sense to humans. We believe that the behavior of the models generated by the symbolic learning method should be comprehensible to humans - at least to linguists - in a sensible way. An advantage of this is that explainable models allow finding the cause of possible errors in the model more easily so we will be able to correct them. The behavior of a model consisting of rules can be explained in a sensible way and therefore we want the models generated by the symbolic learning method to consist of rules.

## 1.3 Available symbolic learning methods

There are many symbolic or rule-based learning methods available (Winston 1992). In this chapter we will concentrate on two groups of symbolic learning methods which are currently the most popular ones: lazy learning and decision trees. Learning methods that fall in these classes are generally used for classification tasks (Van den Bosch et al. 1996b).

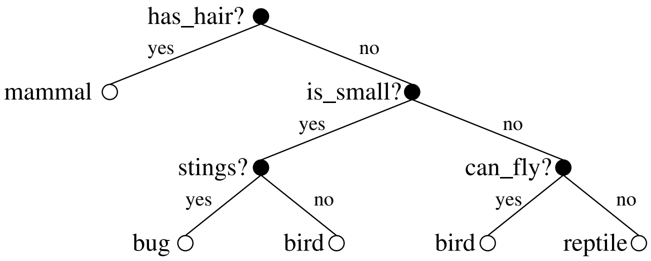


Figure 4.1: An example decision tree for animal classification. A number of questions have to be answered in order to come to a classification. This tree would classify a mosquito as a bug because it does not have hair, is small and stings.

Lazy learning methods or memory-based methods learn the structure of a domain by storing learning examples with their classification (Van den Bosch et al. 1996b). The domain model that results from a lazy learning process is able to generalize by using a predefined distance function. When the domain model is required to give the classification for an unseen domain element then it will use the distance function for finding the stored example that is closest to this unseen example.

The following is a simple application of a lazy learning method to animal classification. Suppose we know the classification of the two animals lion and mosquito. A lion has been classified as a mammal and a mosquito has been classified as a bug. This information is presented as information to the learning algorithm together with features representing the two animals, for example [ big, has_hair , eats_animals] for lion and [ small, can_fly , stings ] for mosquito. The algorithm will compare the features for new animals and classify them either as mammal or bug depending on how many features the new animals have in common with the stored animals. For example, a bat with the features [ small , can_fly , has_hair ] would have been classified as a bug because it has two features in common with mosquito and only one with lion.[2]

Decision tree methods like, for example, C4.5 build a tree describing the structure of the domain (Quinlan 1993). The nodes of the tree consist of questions regarding the values of the elements of the domain. When one wants to know the classification of a certain domain example one will start at the root of the tree and answer the first question with regard to the example. The answer to the question will lead to another node with another question. In this way one will traverse the tree until a leaf node is reached. This leaf node contains the classification of the example.

A decision tree method applied to our animal classification example could result in a tree which contained only one question. There are several alternative questions possible and the method has too few learning examples to make a good choice. A

[2]In our example the classification algorithm assign the same weight to all features but this does not necessarily have to be so.

possible question discriminating the two learning examples could have been *Does it have hair?*. The answer *yes* would result in a mammal classification because a lion has hair and the answer *no* would result in a bug classification. This question would classify bat as a mammal because it has hair.

Both decision tree methods and lazy learning methods have some problems when applied to our learning problem with the constraints we have put on the output from the learning method. First of all these learning methods require learning examples of all possible qualification classes. Our research problem contains two qualification classes: valid strings and invalid strings. However we want to use the same learning input as used in our previous experiments and this means that we want to train the symbolic learning method by using positive examples only. Neither decision tree methods nor lazy learning were designed for training with positive examples only.[3]

A second disadvantage of these two symbolic learning methods is that they do not generate rules. Instead they hide knowledge in a big database of examples combined with a distance function (lazy learning) or a tree with decision nodes (decision trees). There are ways of converting the behavior of decision tree models to comprehensible rules but for the behavior of lazy learning models this will be very difficult. Still we would like to keep our constraint of generating a model that consists of rules and use that to mark these two model structures as a disadvantage.

These two disadvantages make the use of either lazy learning or decision tree learning unacceptable to us. Other rule-based methods suffer from similar problems. Version Spaces require negative examples in order to derive reasonable models ((Winston 1992), chapter 20). C4.5 is a decision tree method and thus it requires negative examples as well (Quinlan 1993). Explanation-Based Learning is able to learn from positive data only by relying on background knowledge (Mitchell et al. 1986). The learning method that we have chosen uses the same approach. We will describe this learning strategy in the next section.

## 2 Inductive Logic Programming

In this section we will describe the symbolic learning method Inductive Logic Programming (ILP). The description has been divided in four parts: a general outline, a part about the background knowledge concept, a part on how ILP can be used in language experiments and a part which relates our models to grammar theory.

## 2.1 Introduction to Inductive Logic Programming

Inductive Logic Programming (ILP) is a logic programming approach to machine learning (Muggleton 1992). The term induction in the name is a reasoning technique

[3] See the section 1.1 of this chapter and section 1.3 of chapter 1 for a motivation about using only positive examples

which can be seen as the reverse of deduction. To explain inductive reasoning we first look at a famous example of deduction:

| | |
|---|---|
| $P_1$ | All men are mortal. |
| $P_2$ | Socrates is a man. |
| DC | Socrates is mortal. |

This example contains two premises $P_1$ and $P_2$. By using these two premises we can derive by deduction that DC must be true. Thus deduction is used to draw conclusions from a theory.

In inductive reasoning we start with facts and attempt to derive a theory from facts. An example of this is the following derivation:

| | |
|---|---|
| $P_1$ | All men are mortal. |
| $P_2$ | Socrates is mortal. |
| IC | Socrates is a man. |

Again we have two premises $P_1$ and $P_2$. By using these two we can derive by induction that IC could be true. An inductive derivation like this one can only be made if the inductive conclusion (IC) and one premise (here $P_1$) can be used for deductively deriving the other premise (here $P_2$). The inductive derivation is not logically sound: that is, the result of the derivation might be wrong. Yet inferences like these are made by humans in everyday life all the time and they are exactly what we are looking for. Inferences like these give us a method for deriving a theory from a set of facts.

ILP theory makes a distinction between three types of knowledge namely background knowledge, observations and hypotheses (Muggleton 1992). Background knowledge is initial knowledge that learners have before they start learning. Observations are the input patterns for the learning method with their classifications. The hypotheses are the rules that ILP derives from the training data. The relation between these three knowledge types can be described with two rules:

| | | | |
|---|---|---|---|
| DR: $B \wedge H \vdash O$ | Example: | B | All men are mortal. |
| | | H | Socrates is a man. |
| | | O | Socrates is mortal. |
| | | | |
| IR: $B \wedge O \rightsquigarrow H$ | Example: | B | All men are mortal. |
| | | O | Socrates is mortal. |
| | | H | Socrates is a man. |

These rules contain three symbols: $\wedge$ stands for *and*, $\vdash$ stands for *leads deductively to* and $\rightsquigarrow$ stands for *leads inductively to*. DR represents the deductive rule which states that the observations (O) are derivable from the background knowledge (B) and the hypotheses (H) (Muggleton 1992). The inductive rule IR represents the inductive step that we want to make: derive hypotheses from the background knowledge and the observations.

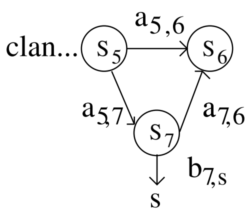


Figure 4.2: A schematic representation of the addition of a character after a valid word in an HMM. The character *s* is appended to word *clan* and as a result of this the path from state $s_5$ via state $s_7$ to $s_6$ will be used during processing. The score assigned by the HMM to the word *clans* will be equal to the score assigned to *clan* multiplied with the factor $(a_{5,7} * b_{7,s} * a_{7,6})/a_{5,6}$.

The inductive rule IR can be used for deriving many different hypotheses sets and the most difficult problem in ILP is to constrain the number of hypotheses sets. Initially only one restriction will be placed on the hypotheses sets: they must satisfy the deductive rule. In our inductive Socrates example the fact *All men are mortal* would be background knowledge and *Socrates is mortal* would be the observation. Then *Socrates is a man* would be a valid hypothesis because we can derive the observation from this hypothesis and the background knowledge. The fact *Socrates is a philosopher* cannot be used in combination with the background knowledge to derive the observation and therefore this fact is not a valid hypothesis.

In our Socrates example we could also have generated the hypothesis *All things named Socrates are man*. This is an example of an alternative valid hypothesis. The facts in the background knowledge and the observations are usually larger in number than in this toy example. The number of valid hypotheses will grow rapidly with any fact we add to the background knowledge and the observations. (Muggleton 1992) suggests four ways for reducing the number of hypotheses: restricting the observations to ground literals,[4] limiting the number of the hypotheses to one, restricting the hypotheses to the most general hypotheses relative to the background knowledge and choosing the hypotheses which can compress the observations as much as possible. In section 2.3 we will discuss these reduction ways and outline which one we will use.

If we want to apply ILP to the problem of acquiring the structure of monosyllabic words then we will have to make two decisions. First we have to decide what we should use as background knowledge. Second we should decide how we are going to use the induction rule for generating monosyllabic word models in practice. We will deal with these two topics in the next two sections.

[4]A ground literal is a logical term without variables. For example the term *man(Socrates)* is a ground literal but *man(x)→mortal(x)* is not because it contains a variable (x).

## 2.2 The background knowledge and the hypotheses

Before we are going to deal with the format and contents of the background knowledge and the hypotheses that are appropriate for our problem we first have to note that there is an important constraint on these bodies of knowledge. We will compare the results of ILP with the results of the previous chapters in which Hidden Markov Models (HMMs) and Simple Recurrent Networks (SRNs) were used. If we want this comparison to be fair then we should take care that no learning method receives more learning input than another. This imposes a constraint on ILP's background knowledge and the format of the hypotheses. There is a danger that we use background knowledge that give ILP an advantage over the other two algorithms. This should be avoided.

In our application of ILP to the acquisition of monosyllabic word models the background knowledge and the hypotheses will contain explicit and implicit knowledge about the structure of syllables. In order to make sure that this knowledge does not give ILP an advantage over the other two learning algorithms we will show how this knowledge is implicitly used in HMMs. This explanation will require some basic HMM knowledge of the reader. You can look back at chapter 2.1 if necessary.

In our ILP models we want to use rules which add a character to a valid word and thus produce another valid word. An example of such a rule is: *When an* s *is added behind a valid word that ends in* n *the resulting word will be valid as well.* Rules of this format are implicitly used in HMMs:

Suppose an HMM is able to process the words *clan* and *clans* and accepts both words. If we ignore all but the most probable path, processing *clan* involves using six states because it contains four visible characters, a begin-of-word character and an end-of-word character. We can label the states in this path with the numbers one to six. The word *clans* can be processed with the same path with one extra state inserted between state five and state six. This extra state will take care of producing the suffix *s* and we will call it state seven (see figure 4.2).

The HMM will assign a different probability to the two words. We can encode the difference with the formula:

$$P(clans) = P(clan) * (a_{5,7} * b_{7,s} * a_{7,6})/a_{5,6}$$

in which P(W) is the score assigned by the HMM to the word W, $a_{i,j}$ is the probability of moving from state $i$ to state $j$, and $b_{i,c}$ is the probability that state $i$ produces character $c$. The link from state five to six ($a_{5,6}$) has been replaced by the links from state five to seven ($a_{5,7}$) and state seven to six ($a_{7,6}$). Furthermore the probability of producing character $s$ in state seven has to be taken into account ($b_{7,s}$). Figure 4.2 contains a schematic representation of the change in the HMM processing phase. The probabilities of the words may be different but the HMM accepts both words. This could have been explained if we knew that the factor $(a_{5,7} * b_{7,s} * a_{7,6})/a_{5,6}$ is equal to one. We don't know if that is true but we will assume that it is true.[5]

[5]Here we have assumed that we have a perfect HMM; one that assigns the score one to all valid strings

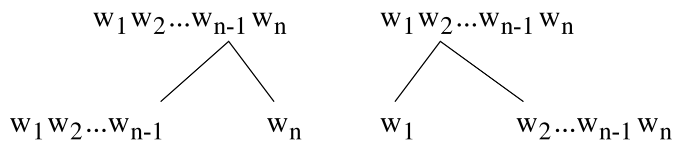


Figure 4.3: A tree representation of the background knowledge suffix rule (left tree) and the background knowledge prefix rule (right tree). Word W=$w_1w_2...w_{n-1}w_n$ is a valid word if its final character $w_n$ is a valid suffix for words ending in its penultimate character $w_{n-1}$ and if $w_1w_2...w_{n-1}$ is a valid word. Word W is a valid word if its initial character $w_1$ is a valid prefix for words starting with its second character $w_2$ and if $w_2...w_{n-1}w_n$is a valid word.

If we assume that the factor is equal to one then we can add $s_7$ to any path for words that end in *n* and create new words that end in *ns*. So we can translate the assumption to a rule that states that from every valid word which ends in *n* one can make another valid word by adding a suffix *s* to the word. Our motivation for choosing a context string of length one is that we have used context strings of the same length in the HMM chapter. The final version of the assumption is exactly the rule which we started with: *When an* s *is added behind a valid word that ends in* n *the resulting word will be valid as well.*

Now we have derived a rule which is being used implicitly in the bigrams HMMs we have worked with in chapter 2. This means that we can use the rule in the ILP model. The rule could be used as a hypothesis. However we want our hypotheses to be as simple as possible and therefore we will split the rule in a background knowledge part and a hypothesis part. The background knowledge part has the following format:

> BACKGROUND KNOWLEDGE SUFFIX RULE
> Suppose there exists a word W=$w_1...w_{n-1}w_n$ ($w_1...w_n$ are $n$ characters) and a suffix hypothesis SH($w_{n-1}$,$w_n$).
> In that case the fact that W is a valid word will imply that $w_1...w_{n-1}$ is a valid word and vice versa.

Now the SUFFIX HYPOTHESIS for this specific case would be defined as SH(*n*,*s*) which states that an *s* is a valid suffix for words ending in *n*. This hypothesis format regards a character in a context containing one neighboring character. This implies that we are working with character bigrams just like in our HMM experiments.

The convention in the ILP literature is to represent knowledge in Prolog-like rules. These rules have the format A←B,C which means that A is true when both B and C are true. The capital characters can be replaced by predicates like abcd(X) which

---

and the score zero to all invalid strings. In practice there will be some deviation in the string score. The factor $(a_{5,7} * b_{7,s} * a_{7,6})/a_{5,6}$ will nearly always be smaller than one but this problem was fixed in our HMMs by score correction for length (section 3.4 of chapter 2) so that it does not affect our assumption.

stands for the proposition *X has property abcd*. We will include a similar notation in our definitions of background knowledge. The background knowledge suffix rule can be represented with two rules: valid($w_1...w_{n-1}$)←valid($w_1...w_{n-1}w_n$),SH($w_{n-1}$,$w_n$) and valid($w_1...w_{n-1}w_n$)←valid($w_1...w_{n-1}$),SH($w_{n-1}$,$w_n$).

Now that we have derived one background rule we can build other background rules in a similar way. We need a rule which adds a character to the front of a word, for example: a valid word that starts with *p* can be converted into another valid word if we put an *s* in front of it. Again we will split this rule into a general part for the background knowledge and a specific part which will be a hypothesis. The background part of this prefix rule has the following format:

> BACKGROUND KNOWLEDGE PREFIX RULE
> Suppose there exists a word W=$w_1w_2...w_n$ and a prefix hypothesis PH($w_1$,$w_2$). In that case the fact that W is a valid word implies that $w_2...w_n$ is a valid word and vice versa.
> Rule notation: valid($w_2...w_n$)←valid($w_1w_2...w_n$),PH($w_1$,$w_2$)
> valid($w_1w_2...w_n$)←valid($w_2...w_n$),PH($w_1$,$w_2$)

And the PREFIX HYPOTHESIS for this specific case can be specified as PH(*s*,*p*) which states that an *s* is a valid prefix for words starting with *p*.

The two rules can be used for explaining why complex words are valid based on the fact that basic words are valid. We need a rule that specifies what specific basic words will be valid. This rule will be very simple: a word will be valid when it has been defined as a valid basic word. Again we will divide this rule in two parts: a background knowledge part and a hypothesis part. The definition of the background part is:

> BACKGROUND KNOWLEDGE BASIC WORD RULE
> The existence of a basic word hypothesis BWH(W) implies that word W is a valid word.
> Rule notation: valid(W)←BWH(W).

And an example of a BASIC WORD HYPOTHESIS is BWH(*lynx*) which states that *lynx* is a valid word.

This concludes the derivation of the background knowledge and the format of the hypotheses. By starting from an HMM that processed a word we have derived the format of three hypotheses and three background knowledge rules. In the background knowledge we have defined that words can be regarded as a nucleus to which prefix characters and suffix characters can be appended. This knowledge is implicitly available in HMMs so by using it in or ILP rules we have not supplied the ILP algorithm with knowledge that was not available to the learning algorithms used in the previous chapters.

## 2.3 Deriving hypotheses

Now that we have defined the background knowledge and the format of the hypotheses we should explain how an ILP algorithm can derive the hypotheses from the learning input. This means that we must make explicit how the inductive rule IR defined in section 2.1 will be realized in practice. As we have explained there are many sets of hypotheses that can be derived by this rule. The most important problem in using this inductive rule is cutting down the number of acceptable hypothesis sets without removing the interesting ones.

We will derive three inference rules based on the background knowledge format we have described in the previous section. This derivation will be based on an example. We will try to use ILP for generating a model which describes the three words *clan*, *clans* and *lans*. These will be our observations and the background knowledge will be as described in section 2.2. Our task is to derive suffix hypotheses, prefix hypotheses and basic word hypotheses for these observations.

In this particular example the words can be explained by each other in combination with appropriate suffix hypotheses and prefix hypotheses. If this had not been the case we would have been forced to define the validity of the three words with three basic word hypotheses without being able to add any other hypotheses. We will consider the recognition of basic word hypotheses as the initial step of the ILP algorithm. This initial step can be made explicit with the following inference rule:

> BASIC WORD INFERENCE RULE
> If word W is a valid word then we will derive the basic word hypothesis BWH(W). All observed words are valid words.
> Rule notation: $\dfrac{\text{valid(W)}}{\text{BWH(W)}}$

In our example this inference rule will produce the hypotheses BWH(*clan*), BWH(*clans*) and BWH(*lans*). These hypotheses can be used in combination with the background knowledge basic word rule to prove that the three words are valid. By using the basic word inference rule we have derived a model that explains the learning input data. This model is an example of the simple domain model we have discussed in section 1.2. The problem of that model was that it will reject all unseen data because it is unable to generalize. This means that we should restructure the model to enable to generalize. Two other inference rules will take care of that.

We can prove that *clans* is a valid word by using the basic word hypothesis BWH(*clans*) and the background knowledge basic word rule. The fact that this word is valid could also be explained by the fact that *lans* is valid in combination with the background prefix rule and a prefix hypothesis PH(*c*,*l*). The latter hypothesis does not exist in our present model and we want to be able to derive it. The hypothesis is not necessary for explaining the valid word but it will add generalization possibilities to the model. It is very important that the final model is able to generalize and therefore we will include in it all prefix hypotheses and all suffix hypotheses which can be used for explaining valid words.

Prefix hypotheses can be derived with the following inference rule:

> PREFIX HYPOTHESIS INFERENCE RULE
> If W=$w_1w_2...w_n$ is a valid word and $w_2...w_n$ is a valid word as well then we will derive the prefix hypothesis PH($w_1$,$w_2$).
> Rule notation: $\dfrac{\text{valid}(w_1w_2...w_n),\text{valid}(w_2...w_n)}{\text{PH}(w_1,w_2),\text{valid}(w_2...w_n)}$

Since both *clans* and *lans* are valid words we can use this inference rule to derive the prefix hypothesis PH(*c*,*l*). This hypothesis can in turn be used in combination with the background knowledge prefix rule and the fact that *clan* is valid to prove that *lan* is valid. The latter word was not among the observations so this is an example of the generalization capabilities of an extra prefix hypothesis. Note that the background knowledge prefix rule can be used in two directions: with PH(c,l) and BWH(lan) we can prove that *clan* is valid and with PH(c,l) and BWH(clan) we can prove that *lan* is valid.

The suffix hypothesis inference rule has a similar format as the prefix hypothesis inference rule:

> SUFFIX HYPOTHESIS INFERENCE RULE
> If W=$w_1...w_{n-1}w_n$ is a valid word and $w_1...w_{n-1}$ is a valid word as well then we will derive the suffix hypothesis SH($w_{n-1}$,$w_n$).
> Rule notation: $\dfrac{\text{valid}(w_1...w_{n-1}w_n),\text{valid}(w_1...w_{n-1})}{\text{SH}(w_{n-1},w_n),\text{valid}(w_1...w_{n-1})}$

In our example model we can use this suffix hypothesis inference rule with either of the valid word pairs *clan* and *clans* or *lan* and *lans* for deriving the suffix hypothesis SH(*n*,*s*). After having removed redundant basic word hypotheses, our final model for the observations *clan*, *clans* and *lans* will consist of the background knowledge defined in the previous section in combination with one basic word hypothesis BWH(*lan*), one prefix hypothesis PH(*c*,*l*) and one suffix hypothesis SH(*n*,*s*).

Now we have defined a method for deriving hypotheses for a specific format of background knowledge and observations. The derivation method uses only one of the four hypotheses space reduction methods mentioned in (Muggleton 1995): the observations are ground literals. We did not limit the number of derivable hypotheses to one. On the contrary we want to derive as many valid hypotheses as possible because they will improve the generalization capabilities of the final model. The derived hypotheses were neither restricted to the most general hypotheses nor to the ones that compressed the observations as much as possible. Instead of that we have limited the inference rules in such a way that one rule can only derive one particular hypotheses as the explanation of a single item of data or a pair of data. The inference process is deterministic and we will accept all the hypotheses it derives.

We will not use the two general purpose ILP software packages available: Golem and Progol (Muggleton 1995). These impose restrictions on either the background knowledge or the observations which we cannot impose on our problem. Golem tries to limit the number of acceptable hypotheses by restricting the background knowledge to ground knowledge. This means that there are no variables allowed in the Golem background knowledge. In order to be able to process background knowledge that has been derived using clauses containing variables, Golem provides a method for converting parts of that knowledge to variable-free clauses. Our background knowledge contains clauses with variables. Since we are working with finite strings the background knowledge can in principle be converted to variable free rules. However this will make it so large that it will be unwieldy to work with in practice. The background knowledge cannot be converted to a usable format that is acceptable for Golem and this makes Golem unusable for our data.

Progol does not have the background knowledge restriction of allowing only variable-free clauses. However the Progol version that we have evaluated required both positive and negative examples in order to generate reasonable output. Muggleton has described a theoretical extension to enable Progol to learn from positive data only (Muggleton 1995). As far as we know this extension has not been added to the software yet.[6] Because we only want to supply positive examples to our learning problem Progol was inadequate for tackling this problem.

## 2.4 The hypothesis models and grammar theory

The models that we will derive for our data consist of three types of rules: basic word hypotheses, prefix hypotheses and suffix hypotheses. It is theoretically interesting to find out to which grammar class these models belong. In this section we will show that the behavior of our learning models can be modeled with regular grammars. These grammars generate exactly the same class of languages as finite state automata.

Regular grammars consist of rules that have the format $S \rightarrow xA$ or $A \rightarrow y$ (Hopcroft et al. 1979). Here $A$ and $S$ are non-terminal character while $x$ and $y$ is a terminal character. The grammars generate strings by applying a sequence of rules usually starting with a start non-terminal $S$. For example, with the two rules presented here we could change the start token $S$ in $xA$ with the first rule and successively in $xy$ with the second rule. The result is a string of terminal symbol and our grammar has generated this string.

Words in our rule-based model are built by starting from a nucleus and subsequently adding prefix characters and suffix characters. We will derive an equivalent regular grammar which builds words by starting with the first character and subsequently adding extra characters or character strings until the word is complete. The regular grammar will make use of non-terminal symbols to simulate context dependencies within the word. Each character $x_i$ will have two corresponding non-terminal symbols: a prefix symbol $\mathrm{P}_{x_i}$ and a suffix symbol $\mathrm{S}_{x_i}$.

In our earlier examples we have used the word *clans*. This word will be build in our rule-based model as follows: start with the basic word *lan*, add the character *c* before it to obtain *clan* and add the character *s* behind it to obtain *clans*. In the

[6] A new version of Progol which allowed learning with positive training data only was released in 1997.

equivalent regular grammar we will start with $cP_l$, replace $P_l$ with $lanS_n$ to obtain $clanS_n$, replace $S_n$ with $sS_s$ to obtain $clansS_s$ and replace $S_s$ with the empty string to obtain *clans*.

There are a number of constraints on the intermediate strings that the regular grammar produces in derivations. First, the intermediate strings will always consist of a sequence of terminal symbols (*a*, *b*, *c* etc.) followed by one optional non-terminal symbol ($\mathrm{P}_{x_i}$ or $\mathrm{S}_{x_i}$). Second, the symbol $\mathrm{P}_{x_i}$ can only be replaced with an intermediate string that starts with $x_i$. Third, when an intermediate string ends in $x_iS_{x_j}$ then $x_i$ and $x_j$ will always be the same character. Fourth, the symbol $\mathrm{S}_{x_i}$ can only be replaced with a string of the format $x_jS_{x_j}$ or with the empty string.

In the previous section we have derived a small rule-based model containing three hypotheses: BWH(*lan*), PH(*c*,*l*) and SH(*n*,*s*). We will define conversion rules which can be used for converting this model to a regular grammar containing the rules:

| | | |
|---|---|---|
| BWH(*lan*) | $\mathrm{S} \rightarrow lan\mathrm{S}_n$ | (1) |
| | $\mathrm{P}_l \rightarrow lan\mathrm{S}_n$ | (2) |
| | $\mathrm{S}_n \rightarrow \mathrm{e}$ | (3) |
| PH(*c*,*l*) | $\mathrm{S} \rightarrow c\mathrm{P}_l$ | (4) |
| SH(*n*,*s*) | $\mathrm{S}_n \rightarrow s\mathrm{S}_s$ | (5) |
| | $\mathrm{S}_s \rightarrow \mathrm{e}$ | (6) |

(*e* stands for the empty string) This grammar produces exactly the same strings as the original rule based model: *lan*, *clan*, *lans* and *clans*. For example, we can generate the string *clans* by applying rule (4) to the start symbol *S* (result $c\mathrm{P}_l$) after which we successively apply rules (2) ($clan\mathrm{S}_n$), (5) ($clans\mathrm{S}_s$) and (6) to obtain the target string *clans*.

Now that we have seen an example for the regular grammar at work we can define the necessary rules for converting the hypotheses to regular grammar rules. We will start with a conversion rule for suffix hypotheses:

> SUFFIX HYPOTHESIS CONVERSION RULE
> A suffix hypothesis $\mathrm{SH}(\mathrm{x}_i,\mathrm{x}_j)$ is equivalent to the set of regular grammar rules $\{\ \mathrm{S}_{x_i} \rightarrow \mathrm{x}_j\mathrm{S}_{x_j}\ ,\ \mathrm{S}_{x_j} \rightarrow \mathrm{e}\ \}$.

The suffix hypothesis $\mathrm{SH}(\mathrm{x}_i,\mathrm{x}_j)$ allows appending $x_j$ to a word that has final character $x_i$. In the regular grammar model an intermediate string of which the final terminal symbol is $x_i$ will initially be followed by the non-terminal $S_{x_i}$. Thus we can simulate the suffix hypothesis by creating a rule which replaces this non-terminal symbol with $\mathrm{x}_j\mathrm{S}_{x_j}$. The $S_{x_j}$ non-terminal symbol in the added string makes possible the addition of extra characters. However, if we want to create proper words that end in $x_j$ we need to be able to remove the non-terminal symbol from the intermediate string. Therefore we have included the second rule which replaces $S_{x_j}$ with the empty string.

The following conversion rule can be used for prefix hypotheses:

> PREFIX HYPOTHESIS CONVERSION RULE
> A prefix hypothesis $\mathrm{PH}(\mathrm{x}_i,\mathrm{x}_j)$ is equivalent to the set of regular grammar rules $\{\ \mathrm{S} \rightarrow \mathrm{x}_i\mathrm{P}_{x_j}\ ,\ \mathrm{P}_{x_i} \rightarrow \mathrm{x}_i\mathrm{P}_{x_j}\ \}$

The prefix hypothesis PH($x_i$,$x_j$) will allow placing character $x_i$ before a word that starts with the character $x_j$. The regular grammar works the other way around: it will append the character $x_j$ to a prefix string which ends in $x_i$.[7] We can tell that an intermediate string is a prefix string by examining the final token. If that token is a prefix non-terminal symbol then the string is a prefix string and otherwise it is not.

While converting the prefix hypothesis to the regular grammar we have to distinguish two cases. First, the added character $x_i$ can be the first character of the word. This case has been taken care of by the first rule which replaces the starting symbol S by $x_i P_{x_j}$. The symbol $P_{x_j}$ makes sure that the next character of the string will be the character $x_j$. Note that according to our second intermediate string format constraint we can only replace $P_{x_j}$ that starts with the character $x_j$.

The second possible case is that the added character $x_i$ is a non-initial character of the word. In that case we will add this character to a prefix string which contains final symbol $P_{x_i}$. The second rule will replace this symbol by $x_i P_{x_j}$. Again we add $\mathrm{P}_{x_j}$ to make sure that the next character will be $x_j$.

The third conversion rule can be used for basic word hypotheses:

> BASIC WORD HYPOTHESIS CONVERSION RULE
> A basic word hypothesis BWH($x_i$...$x_j$) is equivalent to the set of regular grammar rules { $\mathrm{S} \rightarrow \mathrm{x}_i...\mathrm{x}_j\mathrm{S}_{x_j}$ , $\mathrm{P}_{x_i} \rightarrow \mathrm{x}_i...\mathrm{x}_j\mathrm{S}_{x_j}$ , $\mathrm{S}_{x_j} \rightarrow \mathrm{e}$ }

The basic word hypothesis BWH($x_i$...$x_j$) defines that the string $x_i...x_j$ is valid. It is not trivial to add this hypothesis to the regular grammar because of the differences in processing between the grammar and the rule-based models. In the grammar the basic word will be added to a prefix string. An extra suffix non-terminal needs to be added to the string as well.

Just as with the prefix hypothesis conversion we need to distinguish between two cases of basic word hypothesis processing when we want to convert the hypothesis to regular grammar rules. First the basic word can be the initial part of the word we are building. In that case no prefix rules will be used for building the word. The first rule takes care of this case. It replaces the starting symbol S with the string $x_i...x_j S_{x_j}$. The suffix non-terminal $S_{x_j}$ has been included in this string to allow that other characters can be added to the string by using suffix rules. It belongs to the previous terminal symbol $x_j$ and this is in accordance with the third constraint on the format of the intermediate strings. Since the basic word on its own is valid as well we need the possibility to remove the non-terminal symbol $S_{x_j}$ from the word. That is why the third rule has been included here. It can be used for replacing the non-terminal symbol with the empty string.

If the basic word is not the initial part of the word then we will add the basic word to a prefix string. This string will contain a final prefix non-terminal symbol which specifies that the next character needs to be $x_i$. The second rule takes care of replacing

[7] Actually things are a little bit more complex than this. In the regular grammar the prefix hypothesis PH($x_i$,$x_j$) will be modeled by rules that add the character $x_i$ in an $x_i$ context and allow the next character of the intermediate string to be $x_j$.

this symbol $P_{x_i}$ by $x_i ... x_j S_{x_j}$. Again we include the non-terminal suffix symbol $S_{x_j}$ to have the possibility for adding extra suffix characters to the word. If no suffix characters are necessary then third rule can be used for removing the symbol again.

We have presented an algorithmic method for converting a model expressed in hypotheses to a regular grammar. The existence of such a method proves that the behavior of our models can be simulated with regular grammars and finite state automata.

# 3 Experiments with Inductive Logic Programming

In this section we will describe our initial experiments with Inductive Logic Programming (ILP). First we will outline the general setup of these experiments. After that we will present the results of applying ILP to our orthographic and phonetic data. That presentation will be followed by a description of the application of ILP to the same data but while starting the learning process from basic phonotactic knowledge. The section will be concluded with a discussion of the results of the experiments.

## 3.1 General experiment setup

We have used the three inference rules described in section 2.3 for deriving a rule-based model for our orthographic data and a model for our phonetic data. The input data for the learning algorithm consisted of the background knowledge rules described in section 2.2 and observation strings which were the words in the training corpus. The background knowledge rules contain information about the structure of words. However we have shown in section 2.2 that this information is implicitly present in HMMs so encoding it in the background knowledge rules does not give ILP an advantage over HMMs.

The format of the training corpora in these experiments is slightly different from the previous two chapters. In HMMs it was necessary to add an end-of-word token and a start-of-word token to each word. In these experiments both word boundary tokens have been omitted. Processing the data with these tokens would also have been possible but it would have required more complex background suffix rules and more complex inference rules. We have chosen to keep our rules as simple as possible. Processing data with word boundary tokens and more complex rules will lead to models with the same explanatory power as processing data without end-of-word tokens and simple suffix rules. The choice of dropping the word boundary tokens here has no influence on our goal to make the three learning model experiments as comparable as possible.

The ILP hypotheses inference algorithm will process the data in the following way:

1. Convert all observations to basic word hypotheses.
2. Process all basic words, one at a time. We will use the symbol W for the word being processed and assume that W is equal to the character sequence $w_1w_2...w_{n-1}w_n$. We will perform the following actions:
   (a) If $w_2...w_{n-1}w_n$ is a valid word then derive the prefix hypothesis PH($w_1$,$w_2$) and remove the basic word hypothesis for W.
   (b) If $w_1w_2...w_{n-1}$ is a valid word then derive the suffix hypothesis SH($w_{n-1}$,$w_n$) and remove the basic word hypothesis for W.
   (c) If the prefix hypothesis PH($w_1$,$w_2$) exists then derive the basic word hypothesis BWH($w_2...w_{n-1}w_n$) and remove the basic word hypothesis for W.
   (d) If the suffix hypothesis SH($w_{n-1}$,$w_n$) exists then derive the basic word hypothesis BWH($w_1w_2...w_{n-1}$) and remove the basic word hypothesis for W.
3. Repeat step 2 until no new hypotheses can be derived.

Steps 1, 2(a) and 2(b) are straightforward applications of the inference rules for basic words, prefix hypotheses and suffix hypotheses which were defined in section 2.3. The steps 2(c) and 2(d) are less intuitive applications of the background knowledge rules for prefixes and suffixes (see section refsec-ch4-background) in combination with the basic word inference rule. In the background knowledge suffix rule we have defined that $w_1...w_{n-1}$ will be a valid word whenever $w_1...w_{n-1}w_n$ is a valid word and a suffix hypothesis SH($w_{n-1}$,$w_n$) exists. This is exactly the case handled by step 2(d) and because of the fact that $w_1...w_{n-1}$ is a valid word we may derive BWH($w_1...w_{n-1}$) by using the basic word inference rule. Step 2(c) can be explained in a similar way.

The steps 2(c) and 2(d) will be used to make the basic words as short as possible. This is necessary to enable the algorithm to derive all possible prefix and suffix hypotheses. Consider for example the following intermediate configuration hypotheses set:

BWH(yz)
BWH(yx)
SH(y,z)

By applying step 2(d) we can use SH(y,z) and BWH(yz) to add the basic word hypothesis BWH(y) and remove BWH(yz). On its turn this new basic word hypothesis in combination with BWH(yx) can be used for deriving the suffix hypothesis SH(y,x). In this example shortening a basic word has helped to derive an extra suffix hypothesis. We cannot guarantee that the new hypothesis will be correct. However, since we

| data type | rounds | number of hypotheses basic word | prefix | suffix | accepted positive strings | rejected negative strings |
|---|---|---|---|---|---|---|
| orthographic | 4 | 30 | 347 | 327 | 593 (98.8%) | 379 (63.2%) |
| phonetic | 3 | 54 | 383 | 259 | 593 (98.8%) | 517 (86.2%) |

Figure 4.4: The performance of the ILP algorithm and the models generated by this algorithm. The ILP algorithm converts the training strings in models that contain approximately 700 rules. The models perform well on the positive data (more than 98% was accepted) but poorly on the negative data (rejection rates of 63% and 86%).

do not have negative data available which can be used for rejecting new hypotheses we cannot do anything else than accept all proposed hypotheses.

The ILP hypotheses inference algorithm will repeat step 2 until no more new hypotheses can be derived. We will call each repetition of step 2 a training round and list the number of required training rounds in the experiment results.

## 3.2 Handling orthographic and phonetic data

We have used ILP to derive a rule-based model for our training data of 5577 orthographic strings. We used the hypothesis inference algorithm of the previous section which was based on the inference rules defined in section 2.3. The background knowledge consisted of the three background knowledge rules defined in section 2.2 and the training words were used as observations. The algorithm required four training rounds before the hypothesis set stabilized. The final model consisted of 30 basic word hypotheses, 347 prefix hypotheses and 327 suffix hypotheses (see figure 4.4).

The final rule-based model was submitted to the same tests as the models which were generated in the previous chapters. It was tested by making it evaluate 600 positive test strings which were not present in the training data and 600 negative test strings. The model performed well on the positive test data. It accepted 593 words (98.8%) and rejected only 7: *d*, *fjord*, *f's*, *q's*, *schwung*, *t's* and *z*. The rule-based model achieved a lower score on the negative test data. It was only able to reject 379 of the 600 strings (63.2%) Some examples of accepted invalid strings are *bswsk*, *kwrpn*, *ntesllt*, *rdtskise* and *ttrpl*.

After having applied the ILP algorithm for orthographically encoded words we have used it for deriving a rule-based model for our phonetic data. The algorithm started with 5084 observations and required three training rounds before the hypothesis set stabilized. The final model consisted of 54 basic word hypotheses, 383 prefix hypotheses and 259 suffix hypotheses (see table 4.4). It performed equally well on the correct test words as the orthographic model accepting 593 words (98.8%) and rejecting only 7 words: *fjord [fjɔrt]*, *fuut [fyt]*, *square [skwɛ:r]*, *squares [skwɛ:rs]*, *schmink*

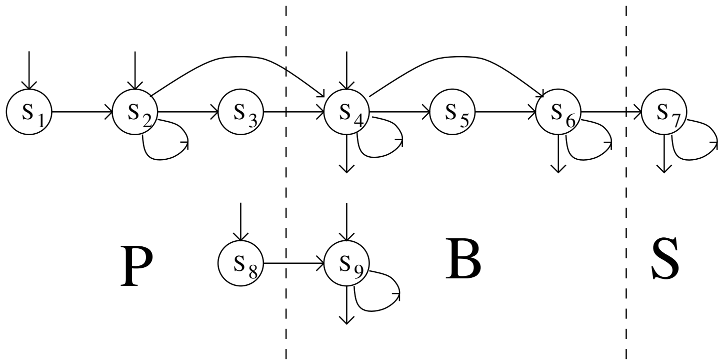


Figure 4.5: An adapted version of the initial HMM model for orthographic data, based on the Cairns and Feinstein model. The original model was presented in figure 2.18 of chapter 2. The model has been divided in three parts: a part P in which the prefix rules operate, a part B generated by the basic word hypotheses and a part S in which the suffix hypotheses work. The new states $s_8$ and $s_9$ are copies of $s_1$ and $s_2$ and take care of the production of words that do not contain vowels.

*[ʃmiŋk]*, *schminkt [ʃmiŋkt]* and *schwung [ʃwuŋ]*. The phonetic model performed worse on the negative test strings but its performance was better than the performance of the orthographic model on the negative strings. It was able to reject 517 of 600 strings (86.2%).[8]

## 3.3 Adding extra linguistic constraints

In the previous chapters we have used the phonetic model by (Cairns and Feinstein 1982) as a model for possible innate linguistic knowledge. We will use this model in our ILP experiments as well. One problem we have to solve is the conversion of the Cairns and Feinstein model to the ILP rule structure. For this purpose we will use our the modified bigram HMM initialization model shown in figure 2.18 of chapter 2. This model consists of a set of states with a limited number of links and with restrictions on the characters that can be produced by each state. We will restructure this model and derive some usable constraints from it.

We want to divide the model in three parts: one part in which only prefix hypotheses operate, one part in which only suffix hypotheses work and one part that is generated by basic word hypotheses. This division is shown in figure 4.5: part P is the part for the prefix hypotheses, part B is for the basic word hypotheses and part S

[8]When we take into account that 26 of the negative strings are reasonable (see section 4.4 in chapter 2) then the phonetic model rejects 514 of 574 negative strings (89.5%).

is for the suffix hypotheses. Each character production by states in the parts P and S corresponds to a set of prefix or suffix hypotheses. The states $s_5$ and $s_6$ have been put in the basic word hypothesis part because $s_6$ is able to produce vowels and we want to produce all vowels in the basic word hypotheses.

The division of the model caused one problem. The original model contained an exit link from state $s_2$. This link would make it impossible to include state $s_2$ in the prefix hypothesis part. States with exit links must be part of either the basic word hypothesis part or the suffix hypothesis part. State $s_2$ in figure 2.18 is on its own capable of producing a word like *t* which is present in the orthographic data. However, in our ILP rule models the use of a basic word hypothesis is obligatory in the production of a valid word. Having an exit link in state $s_2$ would allow word productions that do not include the use of a basic word hypothesis.

We have solved this problem by splitting the model in two parallel finite state automata (figure 4.5). The states $s_1$ and $s_2$ and their links have been copied to two new states $s_8$ and $s_9$. These new states will take care of the production of words that were produced by using the exit link from state $s_2$. This made it possible to remove this exit link from state $s_2$. Since $s_9$ has been put in the basic word hypothesis part, the words produced by $s_8$ and $s_9$ will also require the application of a basic word hypothesis. The larger automaton will produce all words that used the exit links from states other than $s_2$. Therefore there is no necessity for links from $s_9$ to $s_3$ or $s_4$.

Dividing the model in two parts has solved the problem we had with state $s_2$. We can put the states $s_1$, $s_2$, $s_3$ and $s_8$ in the prefix hypothesis part because all exit links from this group of states go to states in the basic word hypothesis part. That part will include states $s_4$, $s_5$, $s_6$ and $s_9$. The suffix part contains only one state: $s_7$.

The finite state automaton of figure 4.5 is equivalent to the finite state automaton of figure 2.18 in chapter 2. However, for learning purposes there are differences between the two. Each character production in the P and the S parts can be modeled with one prefix or one suffix hypothesis. But we cannot produce every single character in the B part with one basic word hypothesis because basic words consists of character sequences and we do not have the opportunity for combining basic word hypotheses in the ILP rule model. Therefore we will use one basic word for modeling the production of a character sequence by a group of cells. This may cause problems when there are internal parts which repeat themselves an arbitrary number of times. The behavior invoked by the self-links from the states $s_4$, $s_6$ and $s_9$ cannot be modeled with the basic word hypotheses. The ILP learning process cannot generate models with extendible basic word hypotheses and this means that the generalization capabilities of the resulting rule models will be weaker than those of the previously used HMMs.

Now that we have changed the Cairns and Feinstein initialization model to the structure that we are using in the chapter, we can attempt to derive usable constraints from this model. The Cairns and Feinstein model imposes constraints on the characters that can be generated by a particular state. We have defined these constraints explicitly for our orthographic data in the B matrix in figure 2.17 of chapter 2: the vowels *a*, *e*, *i*, *o*, *u* and the quote character ' can only be generated by the states $s_4$ and $s_6$, the ambiguous vowel/consonant *y* can be generated by any state and all other characters are consonants which can be generated by any state except $s_4$. The new states $s_8$ and $s_9$ are consonant states: they can generate any character except the six characters *a*, *e*, *i*, *o*, *u* and *'* (in this chapter we will regard the quote character as a vowel).

When we inspect the model with these character production constraints in mind we can make two interesting observations. First, the prefix hypothesis states cannot produce the characters *a*, *e*, *i*, *o*, *u* and *'*. We will call these characters PURE VOWELS. Since the characters produced by these states are put before a word by a prefix hypotheses, this means that prefix hypotheses cannot add a pure vowel prefix to a word. Second, the suffix hypothesis state cannot produce a pure vowel. A character produced by this state is a character appended to a word by a suffix hypothesis. This means that suffix hypotheses cannot append a pure vowel to a word. We can summarize these two observations in the following two rules:

> PREFIX HYPOTHESIS CONSTRAINT
> In a prefix hypothesis PH(I,S) the character I that is appended to a word cannot be a pure vowel.
>
> SUFFIX HYPOTHESIS CONSTRAINT
> In a suffix hypothesis SH(P,F) the character F that is appended to a word cannot be a pure vowel.

It is not possible to derive a similar constraint for the basic word hypotheses because these can contain both vowels and consonants. The derivation presented here applies only to orthographic data. In a similar fashion one can take the initial phonetic model from figure 2.22 in chapter 2, generate an adapted model like presented in figure 4.5 and derive similar constraints for prefix and suffix hypotheses.[9] Our phonetic data contains 18 vowels.

We have repeated our ILP experiments for deriving rule-based models for our orthographic and our phonetic data by using the prefix and the suffix hypothesis constraints presented in this section. Apart from these extra constraints the experiment setups were the same as described in the previous section The resulting models were submitted to our standard test sets of 600 correct words and 600 incorrect strings. The results of these tests can be found in figure 4.6.

The models needed approximately the same number of training rounds to stabilize as in our previous experiments. Because of the constraints on the format of the prefix and the suffix hypotheses, the initialized models contain fewer prefix hypotheses and fewer suffix hypotheses. This means that fewer words have been divided in smaller parts and as a result of that the models contain more basic word hypotheses. The size differences between these models and the previous ones are largest for the orthographic data.

[9]An additional constraint can be derived for phonetic data: the basic word hypotheses cannot consist of a mixture of vowels and consonants. We did not use this constraint because we expected it would cause practical problems in the learning phase.

| data type | rounds | number of hypotheses basic word | prefix | suffix | accepted positive strings | rejected negative strings |
|---|---|---|---|---|---|---|
| orthographic | 4 | 128 | 166 | 178 | 586 (97.7%) | 564 (94.0%) |
| phonetic | 2 | 64 | 324 | 207 | 593 (98.8%) | 565 (94.2%) |

Figure 4.6: The performance of the ILP algorithm with the prefix and the suffix hypothesis constraints and the models generated by this algorithm. The ILP algorithm converts the training strings in models that contain fewer rules than the previous models (472 and 595 compared with approximately 700). The models perform well on the positive test data (best rejection rate 2.3%) and a little worse on the negative test data (rejection rates of about 6%).

The added constraints during learning make the ILP process generate better models. The orthographic model performs worse in accepting positive test data (97.7% compared with the earlier 98.8%) but remarkedly better in rejecting negative data (94.0% versus 63.2%). The phonetic model performs exactly as well in accepting positive test data (98.8%) and a lot better in rejecting negative data (94.2% versus 86.2%).[10]

## 3.4 Discussion

The orthographic model derived by the ILP algorithm without constraints for orthographic data performs quite poorly in rejecting negative test strings. It has become too weak. The fact that this model consist of a set of rules gives us the opportunity to inspect it and find out why exactly it is making errors. This is an advantage of rule-based models over statistical and connectionist models. In the experiments described in the previous section we did not have the opportunity to correct models by changing their internal structure.

The orthographic model accepts the incorrect consonant string *kwrpn*. The model can use only one set of rules for proving this string: the four prefix rules PH(*k*,*w*), PH(*w*,*r*), PH(*r*,*p*) and PH(*p*,*n*) and the basic word rule BWH(*n*). The first two prefix rules are correct as we can see from the two correct Dutch words *kwarts* and *wrakst*. The third prefix rule is wrong[11] and the fourth one is a rare one for Dutch but correct. The basic word hypothesis is strange. Like in English it is possible to say that *n is the 14th character in the alphabet*. However with consonants as basic words we cannot

[10]When we take into account that 26 of the negative strings are reasonable (see section 4.4 in chapter 2) then the phonetic model rejects 562 of 574 negative strings (97.9%).

[11]Incorrect hypotheses are caused by the presence of single consonants in the training data. For example, in combination with the word *in* the single consonant word *n* would cause the incorrect prefix hypothesis PH(i,n) to be derived.

expect to be able to create a good model.

Unfortunately all single characters are present in the orthographic data: 21 in the training corpus and 5 in the test corpus. We had expected the final basic word hypothesis set to include the six Dutch vowels and a few rare vowel sequences which could not be made smaller by the learning algorithm. This was indeed the case but apart from that the basic word hypotheses contained 17 single consonants. Most of them were present in the training data as complete words. We supposed that the latter fact caused the consonants to appear in the basic hypothesis set. In order to test this we have removed the single consonant words from the training words and performed an extra training session. However all consonants returned as basic words in this extra orthographic model.

We have inspected the training data to find out which words did not fit in the simplest word model we could think of. This model assumes that every word contains a vowel with possibly adjacent vowels and an arbitrary number of consonants in front or behind the vowels. The following word types did not fit in this model:

1. Single character consonant words (16 items). All characters except for *a*, *e*, *i*, *o*, *u*, *y* and *'* have been regarded as consonants. Examples of words in this class are *n* and *t*.

2. Words with two or more vowel groups (111 items). Nearly all these words were loan words. Examples of this class are *toque* en *leagues*

3. Multiple character words without vowels (3 items). These were the three interjections which were present in our training corpus: *st*, *sst* and *pst*.

When we remove all three data types from the training corpus and apply the ILP learning algorithm we obtain an orthographic model with only 19 basic word hypotheses, 188 prefix rules and 198 suffix rules. This model accepts 97.6% of the complete training data, accepts 97.0% of the positive test strings and rejects 96.7% of the negative test data. This is an improvement with respect to our orthographic experiments without extra constraints. However we are looking for models which accept all training data and thus this model is unacceptable.

The problems in the phonetic model that was generated without using constraints are not that obvious. The model contains five single consonant basic word hypotheses, 55 prefix hypotheses which append a vowel and 51 suffix hypotheses which add a vowel to a word. The data contains the two interjections st and pst, one single consonant word *s* but no words with more than one vowel group. When we remove these three words from the training data and run the ILP algorithm one more time then we obtain a model which accepts 99.9% of the training data, accepts 98.8% of the positive test data and rejects 94.7% of the negative test data. The model contains 324 prefix rules, 207 suffix rules and 62 base rules. Performance and model size are almost exactly the same as for the phonetic model with linguistic initialization.

We may conclude that application of ILP to our learning task leads to a reasonable performance. The models that are generated by the ILP algorithm without the constraints derived from the model of Cairns and Feinstein perform worse than the models generated while using these constraints. The problems of the earlier models can be explained by a few problematic words. Application of ILP with the extra linguistic constraints is more robust with respect to these words and thus generates better models.

# 4 Alternative rule-based models

In the previous section we have described how ILP can be used for deriving rule-based orthographic and phonetic models. In this section we will modify the internal structure of these models. First we will create more elaborate models, that is models which have a richer internal structure. We will show how these models can be build by using ILP and perform some learning experiments with them. After that we will decrease the size of the models by making the rules process character sets rather than single characters.

## 4.1 Extending the model

In the previous section we have used the Cairns and Feinstein model for initializing a rule-based model. We have seen that our ILP models do not have an internal structure that is as rich as the Cairns and Feinstein model. The latter model divides the word production process in seven different stages which correspond with the seven states in the initial model. Our rule-based models only contain three different stages: one for prefix hypotheses, one for suffix hypotheses and one for basic word hypotheses. We have mentioned in section 3.3 that this restriction has a negative influence on the generalization capabilities of the models.

We want to test a more elaborate rule-based model to see if it performs better than our current three-state model. We will aim for a similar structure as the Cairns and Feinstein model with seven states but we want to keep our concepts of prefix hypotheses, suffix hypotheses and basic word hypotheses. Therefore we have defined the following extended hypotheses concepts:

> EXTENDED BASIC WORD HYPOTHESIS
> An extended basic word hypothesis BWH($w_1 ... w_n$,s$_i$) defines that the string $w_1 ... w_n$ can be produced in state s$_i$.
> Rule notation: producible($w_1 ... w_n$,s$_i$)←BWH($w_1 ... w_n$,s$_i$).
> EXTENDED SUFFIX HYPOTHESIS
> An extended suffix hypothesis SH($w_{n-1}$,$w_n$,s$_i$) defines that when a string $w_1 ... w_{n-1}$ can be produced in a predecessor state of state $s_i$ then the string $w_1 ... w_{n-1} w_n$ can be produced in state $s_i$.
> Rule notation: producible($w_1 ... w_{n-1} w_n$,s$_i$)←SH($w_{n-1}$,$w_n$,s$_i$),
> producible($w_1 ... w_{n-1}$,$s_j$),
> predecessor(s$_j$,s$_i$).

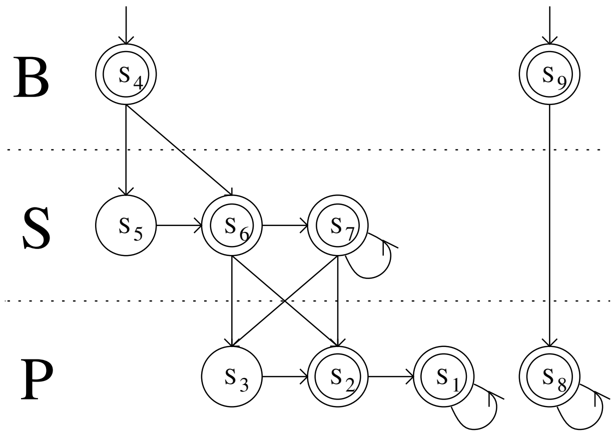


Figure 4.7: An adapted version of the initial HMM model for phonetic data, based on the Cairns and Feinstein model. The original model was presented in figure 2.22 of chapter 2. The model has been divided in three parts: a part P in which the extended prefix hypotheses operate, a part S in which the extended suffix hypotheses work and a part B generated by the extended basic word hypotheses. States $s_8$ and $s_9$ are copies of $s_1$ and $s_2$ that take care of the production of words that do not contain vowels. All states except state $s_3$ and state $s_5$ are final states. The connections from $s_4$ to $s_3$ and $s_2$ have not been included in this diagram.

> EXTENDED PREFIX HYPOTHESIS
> An extended prefix hypothesis PH($w_1$,$w_2$,$s_i$) defines that when a string $w_2...w_n$ can be produced in a predecessor state of state $s_i$ then the string $w_1w_2...w_n$ can be produced in state $s_i$.
> Rule notation: producible($w_1w_2...w_n$,$s_i$)←PH($w_1$,$w_2$,$s_i$),
> producible($w_2...w_n$,$s_j$),
> predecessor($s_j$,$s_i$).

Furthermore, final state hypotheses are necessary for defining which states can be final states since only a few states will be allowed to act as a final state. Final state hypothesis FS($s_i$) defines that processing may end in state $s_i$. There is no equivalent definition necessary for the concept of initial state because these will implicitly be defined by the basic word hypotheses. Nothing can be produced before a basic word and processing a word can only start with processing a basic word. Therefore any state present in an extended basic word hypothesis will automatically be an initial state.

Figure 4.7 shows an initial model that uses extended hypotheses. Like the model shown in figure 4.5 it has been divided in three parts for the three types of hypotheses. Note that the processing order has been changed slightly. The model will start processing basic words, continue with working on suffix characters and finally deal with the prefix characters. This unusual processing order was caused by our wish to obtain a model that was closely related to the three hypothesis types. The more intuitive processing order prefix characters - basic word - suffix characters was not possible because basic words impose restrictions on prefix characters. In a string generation model processing must start with the basic word.

Example: suppose that we want the model to produce the word *bAst* and we know that the hypotheses BWH(A,$s_4$), PH(b,A,$s_2$), SH(A,s,$s_6$) and SH(s,t,$s_7$) are available. The model would start in state $s_4$ and produce *A*. After this it would continue with state $s_6$ (*As*) and state $s_7$ (*Ast*). Processing would end in state $s_2$ (*bAst*). The word is valid since processing has ended in a final state and the complete word has been produced.

## 4.2 Deriving extended hypotheses

Deriving extended hypotheses is more difficult than deriving standard hypotheses. For example, we cannot use the suffix inference rule of section 2.3 for deriving a suffix hypothesis from basic word hypotheses BWH(clans) and BWH(lans) because we need to define the state which produces the *c*. The basic words do not provide a clue to which state would be correct.

We will design a method for deriving the rules based on the assumption that we have some general initial model to start with. The task of the ILP learning process will be to fill in the details of the model. The initial model will be a quadruple consisting of a set of states, a set of predecessor state definitions, a set of final state definitions and a set of character classes that can be produced by the states. An example of such a model is shown in figure 4.7. The states and the links have been shown in the figure. The characters that can be produced by the states are vowels for state $s_4$ and consonants for all other states (see also the B matrix in figure 2.21 in chapter 2).

These constraints on the production of characters leave some processing freedom. For example, the suffix characters of our example word *bAst* can be produced by two different state combinations. The *s* can be produced by state $s_5$ and the *t* by $s_6$. Alternatively the *s* can be produced by state $s_6$ and the *t* by $s_7$. In the Cairns and Feinstein model for Dutch the second option is the only one that is permitted. In order to enable the ILP algorithm to find this solution we need to supply it with extra information. An example of such extra information could be dividing the consonants in subclasses and putting more restrictions on the characters that the states can produce. However, we do not want to make an extra partition in the character sets because we have not done something like that in the experiments with HMMs and SRNs. Using a finer division here would provide the ILP algorithm information the other two algorithms did not have and this would make a performance comparison unfair.

We have chosen to accept both suffix generation possibilities. This means that for generating the word *bAst* we would derive the following set of hypotheses:

BWH(A,$s_4$)
PH(b,A,$s_2$)

SH(A,s,$s_5$)
SH(s,t,$s_6$)
SH(A,s,$s_6$)
SH(s,t,$s_7$)

The derivation process of the extended hypotheses will contain the following steps:

1. Take some initial model of states, predecessor state definitions, final state definitions and producible character classes. The set of hypotheses starts empty.

2. Take a word from the training data and derive all hypotheses which can be used for explaining the word with the initial model. Add these hypotheses to the set of hypotheses

3. Repeat step 2 until all words in the training data have been processed.

4. The result of this ILP process is the initial model combined with the set of hypotheses.

In this derivation process we will use modified versions of the background knowledge rules we have defined in section 2.2 for prefix hypotheses, suffix hypotheses and basic words. We also need final state definitions, predecessor state definitions and a validity definition. Furthermore we will apply an extended hypothesis inference rule which will replace the three inference rules we have defined in section 2.3.

EXTENDED BACKGROUND KNOWLEDGE BASIC WORD RULE
The existence of a basic word hypothesis BWH($w_1...w_n$,$s_i$) implies that string $w_1...w_n$ can be produced by state $s_i$.
Rule notation: producible($w_1...w_n$,$s_i$)← BWH($w_1...w_n$,$s_i$)

EXTENDED BACKGROUND KNOWLEDGE SUFFIX RULE
Suppose there exists a suffix hypothesis SH($w_{n-1}$,$w_n$,$s_i$).
In that case the fact that string $w_1...w_{n-1}$ can be produced in a predecessor of state $s_i$ implies that $w_1...w_{n-1}w_i$ can be produced in $s_i$.
Rule notation: producible($w_1...w_{n-1}w_n$,$s_i$)← SH($w_{n-1}$,$w_n$,$s_i$),
producible($w_1...w_{n-1}$,$s_j$),
predecessor($s_j$,$s_i$)

EXTENDED BACKGROUND KNOWLEDGE PREFIX RULE
Suppose there exists a prefix hypothesis PH($w_1$,$w_2$,$s_i$).
In that case the fact that $w_2...w_n$ can be produced in a predecessor of state $s_i$ implies that $w_1w_2...w_n$ can be produced in state $s_i$.[12]
Rule notation: producible($w_1w_2...w_n$,$s_i$)← PH($w_1$,$w_2$,$s_i$),
producible($w_2...w_n$,$s_j$),
predecessor($s_j$,$s_i$)

[12]Because of the processing order shown in figure 4.7 prefix character states can have predecessor states.

FINAL STATE DEFINITION
The existence of a final state definition finalState($s_i$) implies that state $s_i$ is a final state and vice versa.
PREDECESSOR STATE DEFINITION
The existence of a predecessor definition predecessor($s_i$,$s_j$) implies that state $s_i$ is a predecessor of state $s_j$ and vice versa.
VALIDITY DEFINITION
A string is valid according to a model consisting of a set of states, a set of predecessor state definitions, a set of final state definitions, a set of character classes produced by the states and a set of hypotheses if and only if the string can be produced in one of the final states of the model.
EXTENDED HYPOTHESIS INFERENCE RULE
Any ground prefix hypothesis, suffix hypothesis and basic word hypothesis that can be used for proving that a string in the training data is valid according to the initial model should be derived.

The extended hypothesis inference rule derives sets of hypotheses when they can be used for producing a string in the model. It is difficult to capture this in the rule notation and therefore no rules have been included in the definition of this inference rule.

## 4.3 Experiments with the extended model

We have used ILP for deriving models for our orthographic data and our phonetic data. For each data type we have performed two experiments: one that started from a random model and one that started from a linguistic model derived from the Cairns and Feinstein model. The linguistic model for phonetic data can be found in figure 4.7. The linguistic model for orthographic data is similar to the model shown in figure 4.5 but this model has the same processing order as the phonetic model: first prefix hypothesis states, then suffix hypotheses states and finally the basic word hypothesis states. In the initial orthographic model the states $s_4$, $s_5$ and $s_6$ have been combined into one state which is capable of producing multicharacter strings.

Just like in the previous experiments we wanted to compare initialized extended models with non-initialized extended models in order to be able to measure the influence of the initialization. Constructing the initial models for the two non-initialized experiments was a non-trivial task. We wanted the models to contain some random initialization like the random initialization models we have used for HMMs and SRNs. However, our extended rule-based models do not contain numeric values which we can initialize with arbitrary values. We have decided to use open models as starting models. Open models are models without the restrictions imposed on the linguistically initialized model. In these models the prefix hypotheses, suffix hypotheses and basic word hypotheses may use any state, states may produce any character and all states are connected with each other.

| data type | initialization | number of hypotheses basic word | prefix | suffix | accepted positive strings | rejected negative strings |
|---|---|---|---|---|---|---|
| orthographic | random | 27 | 376 | 376 | 595 (99.2%) | 360 (60.0%) |
| phonetic | random | 41 | 577 | 577 | 595 (99.2%) | 408 (68.0%) |
| orthographic | linguistic | 116 | 273 | 190 | 587 (97.8%) | 586 (97.7%) |
| phonetic | linguistic | 20 | 528 | 450 | 595 (99.2%) | 567 (94.5%) |

Figure 4.8: The performance of the ILP algorithm for the extended models described in this section. The models with a random initialization perform worse than the standard models when it comes to rejecting negative data (average rejection score 64% compared with 75%). The extended initialized models perform slightly better than the standard initialized models (compare with figure 4.5).

In our ILP experiments we want to derive any hypothesis that can be used for explaining a string. In a fully connected open model there will be many different possible explanations for words. All the processing paths found by the ILP algorithm will be equivalent. It does not matter if a path contains the state sequence $s_1$-$s_2$-$s_3$-$s_4$ or $s_1$-$s_1$-$s_1$-$s_1$ because all states will be linked to the same states and produce the same characters and after training and there will be a duplicate of every hypothesis for every state. For that reason an open model with one state will have the same explanatory power as an open model with more states. We want our initial model to be as simple as possible and therefore we have limited the number of states in the open initial models to one.

The 5577 orthographic training words and the 5084 phonetic words have been supplied to the ILP algorithm. From this data the algorithm derived extended hypotheses defined in section 4.1 by using as background knowledge three background hypothesis rules, three definitions and the extended hypothesis inference rule which were defined in section 4.2. One training round was sufficient in each experiment because all possible hypotheses for one word could be derived by considering the word independently of other words or other hypotheses. The results of the experiments can be found in figure 4.8.

The models which were built by starting from an open initialized model performed poorly with respect to rejecting negative strings. The extended orthographic model rejected only 360 negative test strings (60.0%). This performance is even worse than that of the standard non-initialized orthographic model (63.2%). The phonetic model performs slightly better by rejecting 408 negative strings (68.0%).[13] However this is a lot worse than our standard phonetic model which rejected 517 negative test strings (86.2%).

[13] When we remove the 26 plausible strings from the negative phonetic data (see section 4.4 of chapter 2) then the extended random model has rejected 405 of 574 strings (70.6%).

The poor performance of these models can be explained by the fact that the current learning experiments are lacking an important implicit constraint that was present in the previous experiments. In our earlier experiments we have only derived a prefix hypothesis when it could be used for explaining one existing word with another one. In these experiments we derive any prefix hypothesis that can be used for explaining a word regardless of the other words. The same is true for suffix hypotheses. When we use an open model as initial model then this derivation strategy will result in models that accept all possible substrings of the training words. For example, the word *bak* will not only lead to the correct derivation of PH(b,a,$s_1$), BWH(a,$s_1$) and SH(a,k,$s_1$) but also to the incorrect cluster PH(b,a,$s_1$), PH(a,k,$s_1$) and BWH(k,$s_1$) and the incorrect cluster BWH(b,$s_1$), SH(b,a,$s_1$) and SH(a,k,$s_1$). Thus each single token will be a basic word and there will be many prefix hypotheses and many suffix hypotheses. The resulting models are too weak. They accept too many strings.

The models that were built starting from the linguistically initialized model perform much better when it comes to rejecting strings of the negative test data. The orthographic model rejects 586 strings (97.7%) while the phonetic model rejects 567 strings (94.5%). The latter model performs as well as the previous initialized phonetic model (94.2%)[14] and the orthographic model achieves a higher score than before (94.0%, see figure 4.6). The current models perform slightly better when it comes to accepting positive test data. The orthographic model accepts 587 correct words (97.8%) compared with the earlier 586 words (97.7%). The phonetic model accepts 595 words (99.2%) while the earlier figure was 593 words (98.8%).

In this section we have applied the ILP algorithm to our learning problems with as a goal deriving more elaborate target models. However, the linguistically initialized models after training achieve approximately the same performance as the earlier models. Extending the models does not seem to have achieved much.

## 4.4 Compressing the models

The models built by ILP while starting from a linguistically initialized model perform reasonably well. However, the models are quite large. The number of rules per model varies from 472 for the standard orthographic model (figure 4.6) to 998 for the extended phonetic model (figure 4.8). We would like to decrease the size of these models because smaller models are easier to understand and easier to work with.

One of the main reasons for the models being this big is that they contain separate rules for every character. For example, the orthographic models contain seven separate rules for defining that the seven vowel characters can appear in a basic word hypothesis on their own. We would like to encode information like this in one rule.

The number of rules can be decreased by dividing the characters into character classes. The character class boundaries could have been determined by the phonetic features of the characters. We do not want to use these features because we did not

[14]Without the 26 plausible strings from the negative phonetic data the extended initialized model would have rejected 564 of 574 strings (98.3%).

| | |
|---|---|
| 48.0 | a e |
| 46.0 | a e o |
| 44.0 | a e i o |
| 42.0 | a e i o u |
| 42.0 | l r |
| 41.0 | s t |
| 38.4 | c k l m n p r s t |

Figure 4.9: The result of the clustering process for orthographic data: the top 25% of the token groups that frequently appear in the same context. The scores indicate how often the tokens occurred in the same context: a score of 40.0 means that on average the tokens occurred together in 40 rules with the same context. In total there were 140 rule contexts and 26 clusters.

use them in our earlier experiments with statistical learning and connectionist learning. Instead we will use data-unspecific clustering algorithms for dividing tokens into token clusters (Finch 1993). Our modified rules will specify that a character class is possible in a character class context rather than specifying that a single character is possible in some context. Working with character classes will create the possibility to cover gaps that are present in the rule set and create models that generalize better.

We have applied a clustering algorithm to the rules we have obtained in the previous section. The clustering algorithm computed the frequencies of the occurrences of tokens in similar rule contexts. For example, the orthographic prefix rule PH(?,h,$s_1$) states which characters can be placed before an *h* in state $s_1$. According to our extended orthographic model, the question mark in the rule can be replaced by one of five possible tokens: *c*, *k*, *s*, *t* and *w*. We will assume that the fact that these characters can occur in the same context means that they are somehow related to each other. The clustering process will count the number of times that tokens occur in the same context and output lists of tokens which frequently appear in similar contexts.

Example: Among the extended prefix hypotheses for the orthographic data produced by our ILP algorithm with linguistic initialization we have the two hypotheses PH(v,l,$s_2$) and PH(v,r,$s_2$). They state that we can put a *v* before an *l* and an *r* in state $s_2$. So the *l* and the *r* occur in the same context: behind a *v* that is added in state $s_2$. This means that we can put the two characters in one cluster.

We cannot use all clusters that the algorithm produces. One cluster that is not very useful is the largest possible cluster which contains all tokens. We have chosen to work only with the top 25% of the clusters (a motivation for the size of the chosen cluster group will be given later). The clustering algorithm assigned scores to the clusters which indicate how close together the elements of the clusters are. These scores have been used to define which clusters are best.

The top 25% of the clusters for the orthographic data can be found in figure 4.9. The clustering process grouped the vowels together and created three probable con-

| | |
|---|---|
| 82.0 | l r |
| 75.0 | p t |
| 75.0 | n l r |
| 73.0 | k p t |
| 70.4 | y/ i: o: u: E A O I a: e: U |
| 69.7 | k p t x |
| 69.3 | i: o: u: E A O I |
| 68.0 | n l r m |
| 64.5 | k p t x n l r m |
| 60.0 | s k p t x n l r m |

Figure 4.10: The result of the clustering process for phonetic data: the top 25% of the token groups that frequently appear in the same context. The scores indicate how often the tokens occurred in the same context: a score of 60.0 means that on average the tokens occurred together in 60 rules with the same context. In total there were 220 rule contexts and 40 clusters.

sonant groups. The seven character groups presented here were used to decrease the number of rules for the orthographic data. All rules which contained 75% of more of the characters of a cluster were modified. The original characters were removed from the rules and replaced with a special token which represented the complete cluster.

Example: With a cluster that contains the characters *l* and *r* we can replace the two prefix hypotheses PH(v,l,$s_1$) and PH(v,r,$s_1$) by one hypothesis: PH(v,$cluster_{lr}$,$s_1$). Here $cluster_{lr}$ is the name for the cluster containing *l* and *r*. This replacement will only be made if there is no larger cluster available. For example, would it have been possible to have the characters of the cluster *cklmnprst* in place of the question mark in PH(v,?,$_1$) then we would have replaced the nine prefix hypotheses with PH(v,$cluster_{cklmnprst}$,$s_1$). The presence of 75% cluster tokens in a rule context is enough for replacement. This means that even if only seven characters of the cluster are possible in the example context, the seven hypotheses will be replaced. By putting the threshold at 75% rather than 100% we were able to deploy the clustering algorithm to cover gaps in the rule set that resulted from the training data.

We have applied the clustering algorithm to the models with extended hypotheses produced by the ILP algorithm that started with linguistic information. The derivation of these models was discussed in section 4.3 and the performance data for the models can be found in figure 4.8. We did not apply the clustering algorithm to the non-initialized models. These models performed poorly and modifying them by dividing the tokens in groups would make performance even worse. Clustering will implicitly add rules and thus make the models accept more strings. Non-initialized models already accepted to many negative test strings and we are not interested in models that accept even more strings.

The results of the clustering process can be found in figure 4.11. The number

| data type | initialization | number of hypotheses: basic word | prefix | suffix | accepted positive strings | rejected negative strings |
|---|---|---|---|---|---|---|
| orthographic | linguistic | 106 | 103 | 83 | 590 (98.3%) | 585 (97.5%) |
| phonetic | linguistic | 10 | 143 | 131 | 595 (99.2%) | 560 (93.3%) |

Figure 4.11: The performance of the ILP algorithm for the extended models with linguistic initialization after clustering. By grouping the characters in character classes the number of rules in the orthographic model has decreased with 50% and the number of rules in the phonetic model has decreased with 70%. The performance of the models is approximately the same as the original models presented in figure 4.7.

of rules in the orthographic model decreased by approximately 50% compared with the model presented in figure 4.8 from 579 to 292 hypotheses (excluding 5 cluster definitions).[15] Compared with the original model the model accepted three extra strings of the positive test data and one more string of the negative test data. Clustering had a greater influence on the phonetic model (the applied phonetic clusters can be found in figure 4.10). The number of hypotheses decreased by more than 70% from 998 to 284 (excluding 7 cluster definitions). This model accepted seven more negative test strings than the original model.[16] The size of accepted positive test data was the same.

The clustering process contains two parameters for which we have chosen rather arbitrary values. The first parameter is the percentage of used clusters. We chose to use the best 25% of the clusters. By increasing this percentage we would obtain more clusters and generate models with fewer hypotheses. However, the chance that the set of clusters includes nonsensical clusters will increase. Models which use nonsensical clusters might accept too many negative strings.

The second parameter is the acceptance threshold. Whenever 75% or more tokens of a cluster are allowed in a certain context then we will assume that all tokens of that cluster are allowed in that context. By increasing this value we can apply the clusters at more places and thus decrease the number of hypotheses. However, the extra generalizations that will be generated as a result of this might be wrong. We can also increase the acceptance threshold but that will result in fewer application possibilities and larger models.

We would have liked to decrease the phonotactic models with another 50% to models of approximately 150 hypotheses. However, we feel that modification the

[15] We have used seven clusters but we only needed five cluster rules because some clusters contained 75% of the characters of larger clusters. These subset clusters were automatically replaced by the larger clusters. They did not need to be defined because it was impossible for them to appear in the final models.

[16] When we remove the 26 plausible strings from the negative phonetic data then this model has rejected 558 of 574 strings (97.2%).

two clustering process parameters will decrease the performances of the models that will be generated. Therefore we have tried something else. We have added two extra clusters to the models: one containing the vowels as defined in the initial linguistic model and one containing the consonants of the same model. These two clusters were already part of our original linguistic initialization.

With the two extra clusters the orthographic model decreased to 241 hypotheses excluding 6 cluster definitions (-17%). Now the model accepted 594 positive test strings (99.0%) and rejected 580 negative strings (96.7%). The size of the new phonetic model was 220 hypotheses excluding 9 cluster definitions (-23%). It accepted 596 strings of the positive data (99.3%) and rejected 560 strings of the negative data (93.3%).[17] The performance of the phonetic model is almost the same as the performance of the previous phonetic model. The new orthographic model accepts more positive test strings but it also accepts more negative strings.

In this section we have used a clustering method for decreasing the size of the rule-based models we have obtained in the previous section. The clustering method was successful: the number of rules decreased at best with 58% for the orthographic model and 78% for the phonetic model. However, the number of rules in these models, 241 orthographic rules and 220 phonetic rules, is still quite large.

# 5 Concluding Remarks

We have started this chapter with an introduction to rule-based learning. We have explained that neither lazy learning nor decision trees are useful for our learning problem because we want to work with positive examples only. These two learning methods require both positive and negative learning input. We have chosen the learning method Inductive Logic Programming (ILP) for our experiments. An important constraint on ILP is that it should not contain information that was not available in our previous learning experiments. A violation of this constraint would make a comparison of the ILP results with the results we have obtained in the previous chapters unfair.

We have designed an ILP process which was capable of handling our orthographic and phonetic data. We performed two variants of the learning experiments: one that started without knowledge and one that was supplied with initial knowledge which was extracted from the syllable model by Cairns and Feinstein (Cairns and Feinstein 1982). The non-initialized process produced models which performed well in recognizing correct words but they performed poorly in rejecting negative test strings (see figure 4.4). The linguistically initialized process generated models that performed well in both cases (see figure 4.6).

After these experiments we have developed rule-based models with a richer internal structure. We wanted to know whether these extended models would be able to perform better than our previous rule-based models. However, we found only a small

[17] Again a removal of the 26 plausible strings from the negative phonetic data led to a rejection rate of 97.2% for this data set.

performance increase. The best-performing extended models, the initialized extended models, achieved results that were only a little better than the ones of the previous initialized models (see figure 4.8).

We have tried to decrease the size of the extended model in an attempt to obtain models which would be easier to work with for humans. By applying a clustering algorithm we were able to decrease the model size with more than 50% with almost no performance decrease (see figure 4.11). However with rule sets containing between 200 and 250 rules, these models are still quite large.

From the results of the experiments described in this chapter we may conclude that Inductive Logic Programming (ILP) is a good learning method for building monosyllabic phonotactic models. ILP with linguistic initialization generates models that perform much better than the models that were generated without initial knowledge. The results of section 4.3 show that the number of processing stages in the models (equal to the number of states) does not seem to have a large influence on their performance. There is room for optimization in the models generated by ILP since the number of rules that they contain could be decreased with more than 50% without a performance degradation. A clear advantage of the models produced by ILP is that their internal structure can be inspected and improved if necessary.