Variations on ID3 for Text-to-Speech Conversion

Hermann Hild

June 21, 1989

Contents

1 Introduction

Rules for the pronunciation of English words have been of interest for linguists for a long time. With the development of hardware devices for voice synthesis, text-to-speech conversion has found a wide field of applications and is employed in many situations where machine generated "uncanned" speech is desirable.

A talking system must somehow generate English text which is then converted into a symbolic representation of the pronunciation of the text; finally a voice synthesizer has to translate the symbolic pronunciation to (actual) acoustic speech.

We are only interested in the mapping from English text to a phonetic representation. This is not a trivial task, the English language has many pronunciation rules and probably even more exceptions. In order to find the correct pronunciation for a given word, commercial systems like the DECtalk machine use large dictionaries as lookup tables and hand-crafted pronunciation rules to handle words not stored in the dictionary.

On the other hand, the machine learning approach is to *automatically* extract rules from a learning set. In the ideal case, the learning set is only a small subset of all word-pronunciation pairs and the extracted pronunciation rules are general enough to correctly predict the pronunciation of most words in the language.

In 1986 T. Sejnowski and C.R. Rosenberg presented a Neural Network that could learn to pronounce English text. In this so called NETtalk experiment the back-propagation algorithm was used to train a network with one hidden layer. While the achieved results have outperformed other inductive learning methods, a drawback is that the back-propagation learning algorithm is extremely expensive.

As a response to NETtalk, Stanfill and Waltz presented a memory based reasoning approach (MBRtalk). They used a nearest neighbor classification with a sophisticated metric of similarity in order to find representative examples in the data base.

The objective of this research was to apply another well known learning algorithm – ID3 – to the pronunciation task. ID3 constructs a decision tree to explain the data given in the learning set. ID3 is biased to prefer simple explanations, i.e. small decision trees. The features selected as tests in the decisions trees determine the complexity of a tree; ID3 employs a heuristic to find "good" features and thus a simple tree.

A first goal was to implement ID3 and compare the results with those achieved by the back-propagation algorithm. To allow a meaningful comparison, the data set and the encoding of the learning task were kept as close as possible to the conditions in Sejnowski's NETtalk experiment.

Further objectives were to achieve an improved performance by experimenting with different encodings of the problem such as learning groups of letters (instead of each letter individually) or learning separate trees for different letters. Except for the variation of the parameter "chi-square test" and the combined ID3/Nearest Neighbor approach, the ID3 algorithm itself was always used in its standard form by the "new" algorithms. The new approaches employed different encodings of the problem (e.g. separate trees for different letters) or a more sophisticated postprocessing of ID3's outcome (e.g. "best guess strategies").

The impact of size and selection of the training set were also examined. Many of the

performed experiments could improve the performance and helped to gain a better understanding of the problem.

It turns out that one of the difficulties of this problem is the grain size of learning. On the one hand, the task has to be decomposed into simple learning functions in order to achieve a good performance in predicting unseen examples. On the other hand, when the results of the relatively well performing simple functions are aggregated, the overall performance suffers since it relies on the correctness of every single low level function.

Organization of this paper. The second chapter introduces the data set, the learning task and its encoding in greater detail. Chapter Three briefly describes Sejnowski's NETtalk experiment. In Chapter Four the simple standard Nearest Neighbor algorithm is applied to the NETtalk data and the more advanced MBRtalk experiment of Stanfill and Waltz is introduced.

Most of the work done in the scope of this research project is summarized in Chapter Five. After a short introduction to the ID3 algorithm, the various experiments and their results are described and discussed.

In Chapter Six ID3 was applied to random boolean concepts. The purpose of this exercise was to validate some of the algorithms and to compare their results on two different tasks.

The appendix contains additional statistical information about the data in the dictionary, the decision trees and the performance evaluations as well as other results.

2 Description of the Data and the Learning Task

The learning task is to convert text to speech. The **text** consists of English words from a dictionary. The **speech** is represented as a string of phonemes and stresses, which can be pronounced by a hardware device, such as the DECtalk machine.

The following paragraphs give a description of the function we want to learn (the mapping from English words to phonemes and stresses) and of the data source — the NETtalk dictionary — used to train and test the various learning algorithms. The encoding described below is the same — or as close as possible to — the encoding used by Sejnowski for the NETtalk experiment.

2.1 The Nettalk Dictionary

A dictionary of 20,008 English words and their pronunciation was made available to us by T.Sejnowski.

The format of the dictionary can be seen by this excerpt of the first ten words of the data:

```
aardvark   a-rdvark     1<<<>2<<     0
aback      xb@k-        0>1<<        0
abacus     @bxkxs       1<0>0<       0
abaft      xb@ft        0>1<<        0
```

```
abalone     @bxloni       2<0>1>0       0
abandon     xb@ndxn       0>1<>0<       0
abase       xbes-         0>1<<         0
abash       xb@S-         0>1<<         0
abate       xbet-         0>1<<         0
abatis      @bxti-        1<0>2<        2
```

The words in the dictionary contain four fields

1. A character string representing the English word.
2. A string of "phonemes".
3. A string of "stresses".
4. An integer $\in \{0, 1, 2\}$
   98.8 % of the words are marked '0', which stands for "regular word" 0.4% fall into category '1', the irregular words and
   0.8% are marked '2' for "foreign words".

Let

$\mathcal{A} = \{$ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z _ - .$\}$

$\mathcal{P} = \{$ a b c d e f g h i k l m n o p r s t u v w x y z
A C D E G I J K L M N O Q R S T U W X Y Z @ ! # * ^ + - _ . $\}$

$\mathcal{S} = \{$ 0 1 2 > < - $\}$

$\mathcal{A}$ contains 29 characters to represent the English text. '-' and '.' are only needed for the continuous speech [1] experiments in NETtalk [Sejnowski87]; they do not occur in the Nettalk dictionary but were kept in $\mathcal{A}$ to maintain compatible data formats.

$\mathcal{P}$ is the set of phonemes. The list of phonemes in appendix A of [Sejnowski87] is slightly inconsistent with the data in the dictionary we obtained from Sejnowski. Appendix A in [Sejnowski87] lists the phonemes $\mathcal{P} \cup \{\text{'|'}\} \setminus \{\text{'+'}\}$. The dictionary makes no use of 'Q', '|', '.'and '_'. The latter two are needed for continuous speech only.

$\mathcal{S}$ is the set of stresses. '1', '2' and '0' indicate the first vowel in a nucleus of a syllable receiving primary, secondary or no stress, respectively. The arrows '<' and '>' indicate a syllable boundary, '-' stands for a word boundary and is not used in our case, since all words are treated individually.

The dictionary was partitioned into a **training set** and a **test set**. The training set consists of the 1000 [2] most common English words as reported in [Kuchera67] [3] and is

[1] Sejnowski and Rosenberg [Sejnowski87] use both a dictionary and continuous speech for their NETtalk experiment.

[2] For an unfortunate reason, the word "I" was excluded, leaving only 999 words in the training set.

[3] Only 808 of the first 1000 most common words in [Kuchera67] can be found in the NETtalk dictionary. The gap was filled by taking more words from [Kuchera67] and the fact that some of the words in the NETtalk Dictionary appear twice (as the same English word with two different pronunciations).

used by the algorithms for learning purposes. During training the algorithm can see both the English word and its pronunciation. The remaining 19003 words from the NETtalk Dictionary form the test set, [4] which is used for performance evaluation. During testing, only the English word can be seen, the algorithm has to come up with its own answer, which is compared with the pronunciation as given in the dictionary.

2.2 Decomposing the Dictionary Entries into Learning Examples.

Theoretically one has to understand the structure of the English text in order to determine the correct pronunciation of a word. In the sentence "I have read that many people can't read." the pronunciation of the word "read" depends on its grammatical context. However, this is a difficult problem in the area of natural language understanding; we neglect it and look at each word independently. The next section describes how the task of learning an individual word is decomposed in two steps: First the word is broken down into text-windows, so that every letter can be learned individually. Second, the text-window and the associated phoneme and stress are represented in a binary format and each phoneme and stress bit is learned individually. This encoding is adopted from Sejnowski's NETtalk experiment.

Breaking the words into learning examples. The learning task is to map an English word to a correct string of phonemes and stresses. Sejnowski's NETtalk [Sejnowski87], MBRtalk, and all algorithms described in this paper (with the exception of the Multi-level-ID3) are learning each word letter by letter. Of course the pronunciation of a letter is highly context dependent, therefore for each letter to learn, a window is used in which the letter to learn is surrounded by its three pre- and succeeding neighbors. If not enough neighbors are present, the character '_' is used instead.
The learning function on letter level. The function we want to learn is the mapping from a 7-letter window to a phoneme/stress pair:

$$f : \mathcal{A}^7 \rightarrow \mathcal{P} \times \mathcal{S}$$

Example: The center letter '`a`' in the window "`_aback_`" maps to the phoneme '`@`' and the stress '`1`':

$$f(\texttt{"\_aback\_"}) = \texttt{`@1'}$$

To convert a dictionary entry into learning examples, the following procedure is applied: The English word (a k character string "$l_1 \ldots l_k$", $l_i \in \mathcal{A}$), the corresponding phonemes ("$p_1 \ldots p_k$", $p_i \in \mathcal{P}$) and stresses ("$s_1 \ldots s_k$", $s_i \in \mathcal{S}$) are broken down into k windows $w_1, \ldots w_k$, where $w_i = (\text{"}l_{i-3} \ldots l_i \ldots l_{i+3}\text{"} \quad p_i \;\; s_i)$ and l_j is substituted by '_' if $j \notin \{1 \ldots k\}$.

Example: The dictionary entry

[4] 5 of the 19008 words had an inconsistent (e.g. different number of phonemes and stresses) representation and were excluded from the test set.[Bakiri89]

```
("aback"  "xb@k-" "0>1<<")
```

is converted to

```
("___abac"  x  0)
("__aback"  b  >)
("_aback_"  @  1)
("aback__"  k  <)
("back___"  -  <)
```

Each window is used as a learning example. The 1000 word training set consists of 5521 learning examples (i.e. letters), the 19003 word test set has 141392 learning examples.

Binary Representation of a Learning Example. All learning algorithms described in this paper are using binary features and binary classes. The appendix contains a table with the exact representations of the letters, phonemes and stresses.

Converting the letters. The feature vector (the 7-letter window) of a learning example is converted into a binary feature vector by transforming each letter into a binary representation. Exactly the same representation as in the NETtalk experiment [Sejnowski87] is used: Every letter of the 7-letter window is encoded as a 29-bit vector. The i-th letter in the Alphabet $\mathcal{A}$ is represented by a bit-vector $< b_1..b_i..b_{29} >$ in which all but the i-th bit are set to 0. By concatenating the seven 29-bit-vectors we obtain a 203-bit-vector which represents the 7-letter window.

Converting the phonemes. [Sejnowski87] represents the phonemes in terms of 21 articulatory features. These features describe physical or physiological properties of the phoneme (or the way it is articulated) such as "vowel height" or "position in the mouth". In the average three or four of the 21 features apply to a particular feature, so the bit-vectors representing the phonemes contain mainly zeros. However, the appendix in [Sejnowski87] lists 22 different articulatory features. [5] In the absence of a consistent 21 bit encoding, all experiments in this paper use a 22 bit representation for the phonemes.

Converting the stresses. The stresses are encoded by a 5-bit vector [6] . The first two bits represent the syllable boundaries, the third and fourth bit indicate primary, secondary or tertiary emphasis, and the fifth bit stands for word boundaries, a feature which is only needed for continuous speech. In the case of the dictionary data it is always set to 0 and therefore irrelevant and easy to learn.

Example: The binary representation of
`("___abac" x 0)` is

```
( <00000000000000000000000000010 00000000000000000000000000010 00000000000000000000000000010
```

[5] There is a feature "voiced" and "unvoiced". One could interpret the latter one as the negation of the first, but then it is unclear what the value of "voiced" should be when none of the two is listed.

[6] Implementation remark: For an unfortunate historical reason, the stresses are actually represented by a 6-bit-vector, where the first bit is always set to 0. Since this bit does not affect the results, it it ignored and not mentioned further.

```
 10000000000000000000000000000 01000000000000000000000000000 10000000000000000000000000000
 00100000000000000000000000000>
<00000100000000000010000>
<000010>  )
```

The learning function on the bit level. Having a binary representation for the phonemes and the stresses, the learning function

$$f : \mathcal{A}^7 \rightarrow \mathcal{P} \times \mathcal{S}$$

can be decomposed into a set of learning functions where every function has to deal with binary values only:

$$f_{p_1}, ..f_{p_{22}} : \{0,1\}^{203} \rightarrow \{0,1\}$$
$$f_{s_1}, ..f_{s_5} : \{0,1\}^{203} \rightarrow \{0,1\}$$

Given a 203-bit input vector, each of the functions $f_{p_1}, ..f_{p_{22}}$ learns one of the phoneme bits and $f_{s_1}, ..f_{s_5}$ learn the stress bits.

Figure 1 gives an overview of the NETtalk data and their decomposition.

2.3 Aggregating the Results

After we learned each bit of the phoneme/stress pair individually, we face two problems: 1) The found bit-vectors do not necessarily represent a legal phoneme or stress vector, a "best match" has to be found. Several strategies are discussed in Chapter Five. 2) Even if we achieve a high performance on the correctness for each single bit, there is still a good chance that one of the bits is wrong; consequently the entire phoneme (or stress) will be wrong. If we assume a uniform performance of 97% for each of the 27 bits, [7] we can only expect 0.97^{27} = 43.9% performance on the letter level. For an average word of length 5 or 6 the situation becomes even worse, expected performance decreases to $0.439^5 = 1.6\%$.

2.4 Performance Evaluation

The evaluation of any learning algorithm in this paper follows the same scheme: The test data, a subset of the words in the dictionary are presented to the algorithm such that only the English word can be seen. The phonemes and stresses found by the algorithm are compared to the correct answers as given in the dictionary and a statistic of the following format is maintained:

[7] this is a good approximation of the real situation.

Figure 1: The NETtalk data. Every letter is learned individually in the context of a surrounding text-window.

ID3 , no CHI-SQUARE		WORDS	LETTERS	(PHON/STRESS)		BITS	ID3-TREES NODES	DEPTH
best guess(3)	TRAIN:	91.0	98.1	98.8	98.8	99.8	165.3	23.1
best guess(3)	TEST:	12.9	67.2	79.9	78.0	96.2	165.3	23.1
best guess(3)	ALL:	16.8	68.4	80.6	78.8	96.3	165.3	23.1

EXPLANATIONS:

`ID3, no CHI-SQUARE, best guess(3)` describe the learning method.

Unless otherwise indicated, the **data** on which an algoritm is evaluated is taken from one of the three sources:

`TRAIN` The training set with the 1000 most common English words.

`TEST` The test set with the remaining 19003 words which are not in the training set.

`ALL` The entire dictionary.

Correctness is measured on the word-, letter-, phoneme-, stress- and bit-level and is defined as:

`BITS` The average correctness of all phoneme and stress bits.

`STRESS` A stress is correct if all corresponding stress bits are correct.

`PHON` A phoneme is correct when all phoneme bits are correct.

`LETTER` A letter is correct when both the phoneme and the stress is correct.

`WORD` A word is correct when all its letters are correct.

`NODES` The average number of nodes contained in all phoneme and stress ID3-trees.

`DEPTH` The average maximum depth of all phoneme and stress ID3-trees.

Much more data was actually collected for each evaluation. See the appendix for full statistics on a evaluation run.

The correctness on bit-level is somewhat misleading. Only 3 to 5 bits in the 22-bit phoneme vector are set to '1', therefore guessing constantly '0' will already achieve a correctness higher than 85 %. The detailed statistic in the appendix additionally measures bit-correctness by counting the bits which are '0' but should be '1' and vice verse.

Figure 2: Architecture used in the NETtalk experiment.

3 Neural Nets (NETtalk)

Sejnowski and Rosenberg were the first (to my knowledge) to apply a connectionist learning method to the Text-to-Speech conversion problem. In fact, the data and its format as introduced in Chapter Two is due to [Sejnowski87]. This chapter briefly describes the results achieved by [Sejnowski87] with his NETtalk experiment. The neural net simulation was recently repeated by [Mooney88] and [Dietterich89].

Sejnowski's NETtalk. The neural net used in the classic NETtalk experiment had three layers, an input, hidden and output layer. The input layer is built from 203 processing elements, one for each bit in the input vector (i.e. the binary representation of a 7-letter word). The hidden layer has 120 processing elements and the output layer 26, one for each bit of the phoneme and stress vector. The net is fully connected, thus requiring 27809 weights.[8] Figure 2 depicts the architecture of the net.

The so called back-propagation algorithm is used to train the neural net. An input vector is fed to the net and the actual outcome is compared with the desired outcome. Then the weights of the connections are changed in a way to minimize the error, i.e. the difference

[8] There are $203+120+26 = 329$ processing elements and $203*120 = 24360$ connections between input and hidden layer and $120*26 = 3120$ connections between the hidden and the output layer. With one additional weight per node for the "bias term" this sums up to a total of $329 + 24360 + 3120 = 27809$ weights.

between the actual and the desired outcome. The training process is extremely expensive, since it requires many (up to 50) passes through the training set. All 27809 weights have to be updated for every single example in every single pass.

Unfortunately, [Sejnowski87] is not very precise with the definition of the achieved performance figures. An performance of 98% on the training data and 77% on the entire dictionary is reported. The text suggests "performance on the letter level", but similar experiments with the same algorithm [Mooney88] [Dietterich89] achieve lower results which suggests that the results should be interpreted as "performance on the phoneme level".

NEURAL NET(BACK-PROPAGATION)		WORDS	LETTERS	(PHON/STRESS)		BITS
SEJNONWSKI	TRAIN:	-	98.0		-	-
	ALL:	-	77.0		-	-
DIETTERICH	TRAIN:	75.4	94.6	96.8	97.3	99.6
	TEST:	12.1	67.1	77.6	79.6	96.1
MOONEY	TEST:	-	62.0	-	-	-

[Mooney88] uses only a 808 word training set, the intersection of the 1000 most common words reported in [Kuchera67] and the NETtalk dictionary.

The neural nets have one clear advantage over the ID3 algorithm: They learn all phoneme and stress bits in parallel. Therefore the outcome of a particular bit can be influenced by the outcomes of other bits. In order to explore the effects of this potential mutual influences, [Dietterich89] is currently experimenting with a back-propagation algorithm which learns each bit independently (like ID3).

4 Nearest Neighbor Classification and MBRtalk

The Nearest Neighbor Classification uses different principles than inductive learning methods such as ID3. No learning in the sense of generalizing the data in the training set by extracting regularities (rules etc.) takes place. Instead the training examples are simply stored as (hopefully) representative memories from the past. Generalization occurs at run time when a new object is classified by directly referring to the memory and selecting the "most similar" example to classify the new object.

The first part of this chapter describes the application of the straightforward "standard" Nearest Neighbor Classification algorithm to the Nettalk data. The standard algorithm and two variations were implemented not in expectation of a high performance but rather for curiosity and to obtain some bench-marks.

Soon after Sejnowski presented his NETtalk experiment, Stanfill and Waltz [Stanfill86] responded with a successful nearest neighbor approach that used a more sophisticated metric of similarity. This experiment [MBRtalk] is briefly described in the second part of this chapter.

4.1 Simple NN Classification, the Shrink and Growth NN Classification

A simple Nearest Neighbor (NN) version and two variants suggested by [Kibler87] were implemented. All three version have the same principles. During training, all or a subset of the training examples are stored in the Nearest Neighbor Database, called NN-DB for further reference. In order to determine the class of a new example x, the example y in the NN-DB which is "closest" to x has to be found. The class of y is assigned to the new example x. The idea behind the "shrink" and "growth" method is to reduce the number of training examples stored in the NN-DB. Here is a brief description of the algorithms:

Simple NN Store *all* training examples in the NN-DB.

Shrink NN Store all training examples in the NN-DB. Then repeat to remove an example x if the NN-DB $\backslash\{x\}$ classifies x correctly. The order in which we look at the examples will affect the outcome.

Growth NN Start with an empty NN-DB and insert an example x from the training set only if the examples which are already in the NN-DB can not classify x correctly. Again the order of the presentation of the examples will affect the outcome.

Training is very cheap, testing relatively expensive since the usually rather large data base has to be searched in order to find the most similar example(s). The metric for similarity is the Hamming distance. Fortunately not all phoneme and stress bits have to be learned separately, since the entire phoneme and stress vector is defined once the best matching example in the NN-DB is found. Due to the special binary representation of the characters in the 7-letter windows (exactly one bit is set for each character), the Hamming distance of two feature vectors corresponds to the number of different characters in the 7-letter window. There is no need to store the windows as bit-vectors, we can compute the Hamming distance on character-level, thereby gaining a significant improvement in efficiency.

However, using this simple NN method (with the Hamming distance as distance metric) has some major deficiencies, for example:

- there is no notion of similarity between different letters. In terms of the pronunciation, an '`N`' should be closer to an '`M`' than to a '`Z`'.
- No consideration is given to the position of the similarities in the two examples to be compared. Let's assume we want to classify the feature vector `"__too__"`, i.e. we want to find phoneme and stress for the first 'o' in "too". Both `"__the__"` and `"__tools"` have the same Hamming distance (2) to `"__too__"` , but it is obvious that `"__the__"` would be a bad choice.

Here are the results for the three versions of the Nearest Neighbor Classification:

		WORDS	LETTERS	(PHON/STRESS)		BITS
simple NN (4822)	TRAIN:	91.9	98.3	99.1	98.8	99.8
	TEST:	3.3	53.1	61.1	74.0	92.6
SHRINK NN (2423)	TRAIN:	49.9	87.6	89.6	94.4	98.2
	TEST:	2.4	48.8	57.1	71.2	91.8
GROWTH NN (2570)	TRAIN:	56.3	89.9	91.7	95.3	98.5
	TEST:	2.6	50.6	58.8	71.9	92.1

The numbers in parenthesis indicate the size of the set which was stored as the data-base of the exemplars. Shrinking or growing the NN-DB has a similar effect as pruning decision trees with the CHI-SQUARE cutoff: the size of the concept representation can be reduced to half. The price is a small decrease in performance.

4.2 MBRtalk

A very successful attempt to find a more meaningful metric of similarity was made with the MBRtalk (Memory Based Reasoning) algorithm in [STANFILL86]. The simple Hamming distance metric was modified by using two additional chunks of information :

- how useful is a particular letter (at a particular position in the window) to predict phoneme and stress. In other words, how much can this letter constrain the possible outcomes.
- how similar are two letters in terms of predicting phoneme and stress.

Stanfill and Waltz report quite impressive performance figures in their MBRtalk experiment:

MBRtalk	WORDS	LETTERS	(PHON/STRESS)		BITS
TEST on 100 words:	43.0	86.0	-	-	-

However, these numbers are not very suitable to compare with the ID3 or Neural Net results:

- MBRtalk uses 4438 words for training but only 100 words for testing. (versus 1000 for training and 19000 for testing in NETtalk and ID3)
- While ID3 and NETtalk use only a seven letter window (the three preceding and succeeding letters), MBRtalk utilizes the preceding four and the succeeding ten (!) letters.

Of the many advantages listed in [STANFILL86], the following seem to be the important ones which the Neural Nets and ID3 don't have:

- The system "knows" how good a guess was; it can even reject a classification and tell the user that an example is not at all similar to any of the exemplars stored in the data-base.
- A relatively straightforward explanation of the reasoning can be provided.
- Due to the data processing in ID3 or the Neural Net, information might be lost. In the memory based system, all information is stored and potentially accessible.

A disadvantage of the nearest neighbor algorithms is that they are much more expensive in terms of time and space. MBRtalk was implemented on a connection machine, the three simple nearest neighbor versions were by at least one order of magnitude slower than the ID3 algorithm (for classification).

5 Variations on Decision Trees (ID3talk)

ID3 is a learning algorithm that has been successfully applied to many learning task in the past decade. For the experiments described in this Chapter, the ID3 algorithm was implemented in its standard form with the information-driven criterion for selecting important features in the decision tree and (optionally) the CHI-SQUARE test to rule out irrelevant features.

The ID3 algorithm was applied to the NETtalk data and experiments of the following nature were performed:

- Application of ID3 to the NETtalk data under conditions as similar as possible to those in the NETtalk experiment.
- Variation of a parameter of the ID3 algorithm: the CHI-SQUARE test.
- Modification of the postprocessing of the phoneme and stress vectors found by ID3. Since the vectors found by ID3 do not necessarily represent a legal phoneme or stress, a "best guess" has to be made.
- Experimenting with new encodings for the problem, such as
  - Learning the stresses from the phonemes.
  - Learning separate trees for different letters.
  - Learning groups of letters instead of each letter individually.
  - Using a more compact binary representation for the 7-letter text-window.
- Using a combined ID3/Nearest Neighbor approach.
- Postprocessing the decision trees found by ID3. (Quinlan's tree-to-rules conversion).
- Examination of the impact of size and selection of the training set.

5.1 Introduction to ID3

The ID3 Algorithm. ID3 is a learning algorithm of the TDIDT (Top-Down Induction of Decision Trees) family [Quinlan86]. ID3 constructs a decision tree to classify a given set of learning examples. A learning example is a pair (x, c), where x is a feature vector $f_1 \dots f_n$ and c is the outcome associated with x. In the general case, the features f_i and the outcome c can be multivalued. All ID3-algorithms described in this paper use only binary feature vectors and binary classes, therefore only the binary case is presented in this introduction. The generalization is straightforward, the interested reader is referred to [Quinlan86].

Sketch of the ID3 algorithm:

INPUT: A set of m training examples $C = \{ex_1 \dots ex_m\}$. ex_i is a pair (x_i, c_i), where $x_i \in \{0,1\}^n$ is a n-dimensional binary feature vector and $c_i \in \{+,-\}$ is the binary class associated with x_i.

OUTPUT: A decision tree to classify the examples in C.

The decision tree is formed by the following recursive procedure:

- if all training examples have the same class, i.e. $\exists c \in \{+,-\} : \forall i \in \{1 \dots m\} : c_i = c$, then we are done: the tree is a single leaf of class c.
- if not, we chose a feature f_i to partition the set C into two sets C_0 and C_1 such that C_0 contains all examples in which feature $f_i = 0$ and C_1 the remaining ones, i.e. for $k = 0, 1 : C_k = \{(x, c) \in C \mid x = f_1..f_i..f_n \wedge f_i = k\}$. The feature f_i is stored as a test in the current node, C_0 and C_1 are passed to the left and right son of the node and the process is repeated recursively for each of the two subsets passed to the two subtrees. Sometimes there is no good feature to select, although C contains examples of both classes. [9] In this case, the *majority-rule* is applied: a leaf is formed and labeled with the class of the more numerous examples.

Figure 3 shows how a decision tree is built from a simple training set.

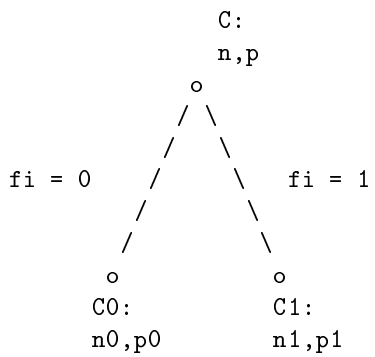
A Decision tree can be built from any set of training examples, the objective is to keep the tree as small as possible, since we prefer the simplest explanation (i.e. the smallest tree) for the training set.

The choice of the features determines the size of the tree. Choosing irrelevant features as the tests in the nodes of the tree will cause the tree to branch more or less randomly. Testing all possible trees in order to find the smallest one is too expensive ("combinatorial explosion"), ID3 uses a heuristic strategy to find important features. Lets assume we are at some node at the tree and have to select which feature we want to use as test in this node.

[9] This can happen due to noise (i.e. inconsistencies) in the data or because it was decided that none of the features is relevant enough to justify a partition

Figure 3: An example of a simple decision tree. The frames show how the tree grows when the training examples are partitioned.

The effect of choosing feature f_i to partition C is depicted below:



C = set of training examples
n, p = the number of negative and positive examples in C
f_i = selected feature
C_0,C_1 = the sets of examples partitioned by f_i.
n_0, p_0 = the number of negative and positive examples in C_0
n_1, p_1 = the number of negative and positive examples in C_1

If all examples in (say) C_0 are positive ($n_0 = 0$) or all are negative ($p_0 = 0$), we have the ideal case. If the distribution is half and half ($p_0 = n_0$), we can make no guess at all about what class we should assign to the leave containing C_0. This suggests to use the entropy function

$$entr(n,p) = -\frac{n}{n+p}\log_2\frac{n}{n+p} \quad - \quad \frac{p}{n+p}\log_2\frac{p}{n+p}$$

as a measure for the uncertainty of a guess based on a partition with p positive and n negative examples.

In order to take account of the different number of examples in the sets C_0 and C_1, the uncertainty associated with the feature f_i is the weighted average of the entropies of C_0 and C_1:

$$unc(f_i) = \frac{n_0+p_0}{n+p}entr(n_0,p_0) \; + \; \frac{n_1+p_1}{n+p}entr(n_1,p_1)$$

After computing this uncertainty value for every feature, we chose the feature f_j with the lowest uncertainty value $unc(f_j)$.

ID3 and the NETtalk Data. Since the ID3 algorithm described above can only learn to discriminate between binary valued classes, we have to build a decision tree for each single bit in the phoneme and stress vector. With 22 phoneme bits and 5 stress bits there are 27 decision trees to learn in order to to find a phoneme/stress pair for a given letter (7-letter window). It should be noted that ID3 is not limited to learn binary concepts. [Bakiri89] ran the NETtalk data with a ID3 version that processes multi-valued categories and achieved results comparable to the one presented in the following Chapters.

5.2 Impact of the "Best Guess" Strategy

Finding the Best Match. This section describes the process of finding a legal phoneme/stress. Two improvements to the standard method are suggested.

In the testing phase, the 7-letter window we want to classify is presented to the system as a bit-vector $bv \in \{0,1\}^{203}$. The system answers with two bit-vectors $pv \in \{0,1\}^{22}$ and $sv \in \{0,1\}^{5}$. pv / sv represent the phoneme/stress pair associated with the input vector bv.

Of course not all 2^{22} (2^5) potential answers for a phoneme (stress) bit-vector represent one of the 54 legal phoneme (or 6 stress) vectors.

Let pv be the bit-vector found by the algorithm and $\mathcal{LPV}$ the set of the 54 legal phonemes vectors. To find a legal phoneme (stress) for an arbitrary phoneme (stress) bit-vector, one of the following three "best guess" strategies can be applied:

(0) Do nothing. pv is considered wrong if $pv \notin \mathcal{LPV}$

(1) Find a legal phoneme vector $lpv \in \mathcal{LPV}$ with the smallest Hamming or Euclidean distance[10] to pv.

(2) As before. Additionally, if there is more than one candidate for the smallest Hamming distance, we choose the more likely one. In order to know the likelyhood of a particular candidate, the following statistic is maintained. For every letter in the training set, we count how often the letter is mapped to each phoneme. The relative frequency of the pair (L, p_i) is interpreted as the probability that the letter L maps to the phoneme p_i.

(3) To find the best guess, we consider only the phonemes to which a particular letter is potentially mapped. Again the more likely phoneme is used in the case of a tie.

The same scheme is applied to find the closest stress. Instead of finding the closest phoneme and stress independently, one could treat the concatenated phoneme/stress vector as an entity and try to find the closest legal phoneme/stress pair. This results in (potentially) different answers.

In the remainder of the paper, the 4 strategies mentioned above will be referred to as "best guess(0)" .. "best guess(3)".

The following results show the effect of the different "best guess" strategies. ID3 without CHI-Square cutoff (see next section) was applied:

							ID3-TREES	
ID3, no CHI-Square		WORDS	LETTERS	(PHON/STRESS)		BITS	NODES	DEPTH
best guess (0)	TEST:	6.0	57.0	69.9	73.3	96.0	165.3	23.1
best guess (1)	TEST:	9.1	60.8	75.4	74.4	95.8	165.3	23.1
best guess (2)	TEST:	11.9	66.1	78.4	78.0	96.2	165.3	23.1
best guess (3)	TEST:	12.9	67.2	79.9	78.0	96.2	165.3	23.1
best guess (3b)	TEST:	13.1	67.2	79.9	78.0	96.2	165.3	23.1

[10] The Euclidean and Hamming distance is identical in the case of bit-vectors.

In the last line (3b) the information about the frequencies of the letter-phoneme/stress mappings were not collected from the 1000 word training set but instead from the entire dictionary. Of course this is "unfair", because we are not supposed to extract information from the test set. However, the bigger part of the errors is made one step earlier: about 47% of all letters are mapped to a phoneme vector which represents a legal phoneme but not the correct one. The corresponding number for stresses is even higher: 79% of all letters are mapped to legal but wrong stress vectors. This means that less than a fifth of the wrong letters can be potentially corrected by using a clever best guess strategy. [11]

Since the neural net experiments do not employ any sophisticated "best guess" strategies, their results have to be compared with the second line of the above table. The back-propagation algorithm outperforms ID3 by several percent. It seems that especially the stresses can be learned much better by the neural net.

5.3 The Chi-Square Test

[Quinlan86] suggests the CHI-SQUARE test as a measure for the irrelevance of a feature to the class of the examples in C. As before, let C be a set with n negative and p positive training examples. The feature f_i divides C into the two subsets C_0 (with n_0 negative and p_0 positive examples) and C_1 (with n_1 negative and p_1 positive examples). If f_i is irrelevant to the class of an examples in C, we expect the two subsets C_0 and C_1 to have the same ratio of positive and negative examples as C, i.e.

$$\frac{p_i'}{n_i + p_i} = \frac{p}{n+p} \quad and \quad \frac{n_i'}{n_i + p_i} = \frac{n}{n+p} \quad for\ i = 0, 1$$

where p_0', n_0', p_1', n_1' are the expected values for p_0, n_0, p_1, n_1, respectively.
The CHI-SQUARE statistic

$$\sum_{i=0}^{1} \frac{(p_i - p_i')^2}{p_i'} + \frac{(n_i - n_i')^2}{n_i'}$$

can be used to test the hypothesis that p_0', n_0', p_1', n_1' are stochastically dependent on p_0, n_0, p_1, n_1, which in turn means that the feature f_i is irrelevant to the class of the examples in C.

To incorporate the CHI-SQUARE test into the ID3 algorithm, we do not consider a feature f_j unless the hypothesis that f_j is irrelevant can be rejected with a high confidence level. [12]

[11] Remember that both phoneme and stress have to be correct to make a correct letter.

[12] An efficient way to do this is to use the chi-square test as a filter. First the feature with the lowest uncertainty value is considered. If the feature passes the CHi-Square test, we can select it, otherwise the feature with the second-lowest uncertainity value is tested and so on. In most of the cases the first feature will pass the chi-square test.

All ID3-chi-square versions described in this paper use either a 90% or a 99% confidence level. With the restriction of the CHI-Square test, the following situation may occur: There are no more eligible features but the set of examples at the corresponding leaf in the tree still contains instances of both classes. In this case the **majority rule** is applied: The more numerous class is selected.

In the results below three ID3 versions were used: 1) No CHI-SQUARE, 2) CHI-SQUARE using a 90% confidence level and 3) CHI-SQUARE with a 99% confidence level. The "best guess" strategy is "closest phoneme/stress without preference or constraints (method 1)".

							ID3-TREES	
ID3, Best Guess(1)		WORDS	LETTERS	(PHON/STRESS)	BITS	NODES	DEPTH	
no CHI-SQUARE	TEST:	9.1	60.8	75.4 74.4	95.8	165.3	23.1	
CHI-SQUARE (90%)	TEST:	9.1	59.8	74.4 73.9	95.7	129.2	18.7	
CHI-SQUARE (99%)	TEST:	8.6	59.0	74.0 73.6	95.6	75.5	14.4	

The CHI-SQUARE tests prunes the tree size significantly. At the same time, the performance decreases slightly. However, it is interesting to see that almost the same performance can be achieved with trees which are less than half as big as the trees generated by ID3 without CHI-SQUARE cutoff.

The effect of cutting off leaves of the trees becomes more obvious if we look at the performance on the training data:

							ID3-TREES	
ID3, Best Guess(1)		WORDS	LETTERS	(PHON/STRESS)	BITS	NODES	DEPTH	
no CHI-SQUARE	TRAIN:	89.8	97.5	98.4 98.6	99.8	165.3	23.1	
CHI-SQUARE (90%)	TRAIN:	65.5	91.7	95.0 95.8	99.3	129.2	18.7	
CHI-SQUARE (99%)	TRAIN:	33.9	79.4	86.2 90.4	98.1	75.5	14.4	

The CHI-SQUARE was designed to deal with noisy data [Quinlan86]. Although the dictionary contains inconsistent data, it is questionable whether one should view these inconsistencies as noise. The inconsistencies in the dictionary result either from too small window sizes or because there are exceptions in the pronunciations. Noise on the other hand usually affects *all* data by *randomly* corrupting the examples.

5.4 Learning the Stresses from the Phonemes

Tom Dietterich suggested learning the stresses from the phonemes instead of from the letter. During the training phase, the 5 trees which learn the 5 stress bits are not presented the 7-letter window but instead the binary representation of the corresponding 7 phonemes is used as input.

Two phases are involved during the classification of a k-letter word: First all k phonemes have to be classified. In the second phase a 7-phoneme window is shifted through the phoneme string in the same fashion as the 7-letter windows are used. The binary representation of a 7-character phoneme window is used as input to the 5 stress trees, thus the stress decision trees learn the mapping form phonemes to stresses.

							ID3-TREES	
ID3, no CHI-SQUARE		WORDS	LETTERS	(PHON/STRESS)		BITS	LEAVES	DEPTH
Stress from phonemes	TRAIN:	92.1	98.0	98.8	98.5	99.8	154.9	22.5
	TEST:	13.8	67.7	79.9	76.3	96.1	154.9	22.5

Observations:

- Although the performance on the stress level degraded by 1.7 %, the letter level gained 0.5% and the word level almost 1% in performance.
- The trees which learned the stresses from the phonemes are smaller than the trees that map the text directly to the stresses.

5.5 Learning Separate Trees for Each Letter

Since there seems to be no relationship between the pronunciation of (for example) an 'a' and 'b', there is no reason to force the mapping functions of all letters into the same decision tree. Instead we could try to learn each letter separately.

For this experiment, the set $\mathcal{C}$ of training examples was partitioned into the 26 classes $\mathcal{C}_A, ..\mathcal{C}_Z$ such that class $\mathcal{C}_{<L>}$ contains only examples which have the letter $< L >$ in the center of their 7-letter window. For each of this 26 classes, a separate set of decision trees was learned, with the following results:

							ID3-TREES	
Single Letter ID3		WORDS	LETTERS	(PHON/STRESS)		BITS	LEAVES	DEPTH
no CHI-SQ, best guess(3)	TEST:	12.5	66.7	80.1	77.4	96.1	-	-

Observations:

- The phonemes were learned more accurately.
- The overall performance decreased. This is due to the bad results on the stresses. The bad performance on the stresses is not unexpected since the relation between a particular letter and a stress is weaker than the relation between the letter and a phoneme. By partitioning the training examples every tree can only see a fraction of the examples it saw before.

- The 26 sets of "single-letter-trees" need only 75 % of the space of the standard tree set. An explanation could be following: Lets assume feature f_a was found to be the most important feature to determine a phoneme-bit for the letter A and feature f_b is the most important one for the letter B. In the big tree, these two features have to compete for a top position in the tree. If (say) f_a wins and goes to the top position, feature f_b will (most likely) appear somewhere down in either branch of the tree. If separate trees are used, both features can go to the root of the tree, neither one has to appear twice.

Since the stresses suffer from the partition and the phonemes are better than before, the logical consequence is not to use the single letter trees for the stresses but instead to learn them from the phonemes. The results are shown below:

Single Letter ID3							ID3-TREES	
no CHI-SQ, best guess(3)	WORDS	LETTERS	(PHON/STRESS)		BITS	LEAVES	DEPTH
stresses from phonemes	TEST:	13.7	67.8	80.1	76.3	96.1	-	-

Although the stress performance decreases again [13] ,the letter and word level performance increases by more than 1 %.

A closer look on the performance of each single letter shows that some letters improve their performance while others get worse.(see table below) Obviously the reason for getting worse is that the corresponding trees can no longer see the whole training set but only the subset defined by the partition.

To make more training examples available to the trees that performed worse, all those partitions which caused a deteriorated performance where grouped together. 0.5% performance over the results shown above could be gained:

Single Letter ID3							ID3-TREES	
no CHI-SQ, best guess(3)	WORDS	LETTERS	(PHON/STRESS)		BITS	LEAVES	DEPTH
stresses from phonemes, some letters grouped.	TEST:	14.3	68.2	80.5	76.5	96.2	-	-

Here is a detailed report on the letter performance. Row 1 shows the results achieved by the traditional ID3 algorithm (no CHI-SQUARE, best guess(3)). Row 2 is after learning each letter separately. In Row 3, all the "loser" letters C,D,F,I,K,M,Q,S,U,W,Y,Z are grouped together and are learned with a single tree (per bit).

Correctly classified LETTERS:

char	all	-	.	A	B	C	D	E	F	G
1: %	67.2	100.0	100.0	39.7	81.8	79.1	84.5	57.4	90.0	63.0

[13] They showed the same interesting behavior (decreasing stress by increasing overall performance) in the original letter to phoneme mapping experiment

```
2: %      67.8  100.0  100.0   43.4   84.3   77.2   84.3   58.4   88.9   66.1
3: %      68.2  100.0  100.0   43.2   84.1   79.6   84.7   58.4   89.7   66.0


char         H      I      J      K      L      M      N      O      P      Q
-------------------------------------------------------------------------------
1: %      78.4   49.2   90.4   75.4   78.7   83.9   84.8   38.7   87.6   79.3
2: %      80.5   48.4   90.7   73.5   79.0   83.2   85.2   41.7   88.0   78.1
3: %      80.5   49.6   91.1   73.0   79.8   83.2   85.4   41.7   87.9   78.4

char         R      S      T      U      V      W      X      Y      Z      _
-------------------------------------------------------------------------------
1: %      81.8   75.5   81.0   43.3   74.3   82.8   83.6   74.6   75.5  100.0
2: %      82.0   73.5   81.9   41.6   78.3   76.3   83.6   73.8   74.1  100.0
3: %      82.5   74.7   81.8   42.6   78.5   74.1   83.6   75.5   74.1  100.0


Letters which didn't improve:

C,D,F,I,K,M,Q,S,U,W,Y,Z
```

Note that both phoneme and stress have to be correct for a correct letter.

5.6 Multi-Level ID3

Learning each letter individually seems not a very natural way to solve the problem. [14]

If people look at the word `example`, it seems unlikely that they first try to figure out how to pronounce the `e` and then, after they made their decision, take care of the character `x`. Instead one recognizes `ex` as a block of letters (a prefix in this case) which belong together and should be treated as an entity. Other examples are suffixes like `ing` or `ed`, or letter combinations like `st`, `ion`, `ous`, `ity` and many more. The appendix contains statistics on frequently appearing letter-blocks of length 2, 3, 4, and 5.

A Strategy for Learning Groups of Letters. In order to implement the idea of learning frequently appearing letter blocks on a higher level, the following strategy was implemented:

(1) Collecting the data. Select the n (e.g. n = 100 or n = 200) most frequently appearing k-letter blocks (for k = 2 .. 5) from the 1000 word training set. Alternatively one can select the blocks from the entire dictionary. [15] All possible phoneme/stress strings to which a particular block is mapped are also recorded.

(2) Learning on a higher level. A separate set of decision trees is learned for each level k (i.e. for blocks of size 2, 3, 4, and 5.) Only windows containing one of the "promising" blocks of length k (the frequently appearing blocks are called "promising") are

[14] Ghulum Bakiri brought this idea to my attention. His initial thought was that one should not look at a single letter if this letter appears in a group of letters belonging together.

[15] Of course this violates the rule not to extract information from the test data. On the other hand, if one views the entire dictionary as the (complete) domain of the problem, one could declare this information as domain-knowledge.

presented to the decision trees on level k. A block with k letters has $k \times (22 + 5)$ phoneme and stress bits, therefore $k \times 27$ decision trees have to be learned on level k. Like the single letters, the windows for the blocks contain the block and the three adjacent letters on each side. Thus a window on level k has the size $k + 6$.

(3) Classification. Every word to classify is first examined for a "promising" block. If there is one, the block is classified using the decision trees on the corresponding level. Every part of the word that contains no promising blocks of length k is classified on the next lower level $k - 1$. The classification starts at the highest level ($k = 5$), the last level ($k = 1$) is just the regular one-letter classification.

The next table shows the results after implementing the above mentioned strategy. Best guess strategy (3) and no CHI-SQUARE cutoff were used. Additional decision trees were constructed for the 100 most frequently appearing 2,3,4 and 5-letter blocks.

best guess(3), no CHI-SQ	WORDS	LETTERS	(PHON/STRESS)		BITS	ID3-TREES LEAVES	DEPTH
No multilevels TEST:	12.9	67.2	79.9	78.0	96.2	165.3	23.1
a) Blocks from TRAINING set:	14.7	68.6	80.0	78.3	96.2	165.3	23.1
b) Blocks from ALL words :	14.9	69.9	81.3	79.1	96.4	165.3	23.1

An Effective Simplification. Building the relatively complex structure of four additional layers of decision trees was actually not necessary. A similar but much simpler method achieved even slightly better results. As before, we start at the highest level and look for promising blocks. Whenever such a k-letter block is found, we do not use the specially trained trees on level k, instead the "regular" one-letter classification is used. The possible outcome for the phoneme/stress string of this block is constrained to the outcomes that were observed in the training data.

Example: the possible phoneme/stress strings found for the 3-letter block `ION` in the training data are `ION: -xn0<< .81 -xn0>> .11 yxn0<< .08`

The numbers indicate the frequency of the mappings. If the block `ION` appears in a word to classify, the algorithm might come up with the wrong phoneme/stress string `Axn1<<`. Since only one of the three above outcomes is allowed, the one closest to `Axn1<<` is selected.

Aside: if we look at the possible phoneme/stress strings in the entire dictionary, we get the following list (which shows that there are quite a lot of things we do not see in the training data).

```
ION:   -xn0<< .90   -xn0>> .04   yxn0<< .03   ixn00< .01
       -xn0<>   Axn10<   ian01<   ixn00>   Axn20>   Axn20<
       yxn0<>  Axn10>  -xn>>>  Aan12<  yxn0>>  Aan01<  x-n0<<
```

The next table shows the results when the blocks are only used to constrain possible outcomes (no multi-level trees). The last two rows show the effect of choosing the most frequently appearing 200 k-letter blocks. [16]

[16] 200 blocks are only selected for 2- and 3-letter blocks. Bigger blocks are no longer very common: (0.12%)

	WORDS	LETTERS	(PHON/STRESS)		BITS	ID3-TREES LEAVES	DEPTH
100 Blocks from TRAINING set:	14.8	68.3	79.8	78.0	96.1	165.3	23.1
200 Blocks from TRAINING set:	15.2	68.3	79.8	77.8	96.1	165.3	23.1
100 Blocks from ALL words :	15.6	69.8	81.2	79.0	96.4	165.3	23.1
200 Blocks from ALL words :	16.3	70.5	81.6	79.4	96.5	165.3	23.1

The method for constraining the possible outcomes used the standard letter-by-letter classification to obtain a phoneme/stress string for a "promising" block. This string was then mapped to a "legal" block string. Each single phoneme/stress vector was mapped to the best legal phoneme/stress *before* the constraints on the block-level were applied. This is not optimal since some information is lost when the bit-vectors are mapped to the legal phoneme/stresses.

In another modification the phoneme/stress bit-vectors found for the corresponding block are presented in their original binary form to the routine which finds the closest legal block. One percent performance on the letter level is gained:

	WORDS	LETTERS	(PHON/STRESS)		BITS	ID3-TREES LEAVES	DEPTH
100 Blocks from TRAINING set:	15.3	69.2	80.3	78.7	96.2	165.3	23.1
200 Blocks from TRAINING set:	15.8	69.3	80.3	78.6	96.2	165.3	23.1
100 Blocks from ALL words :	17.1	70.9	81.9	79.8	96.5	165.3	23.1
200 Blocks from ALL words :	18.2	71.8	82.4	80.3	96.6	165.3	23.1

A reason why it was not helpful to *learn* the "promising blocks" with the multi-level trees might be that the number of examples decreases with increasing level. While the standard ID3 algorithm sees all training examples, only the small fraction of those examples which contain a promising block are presented to the higher level trees.

The improvements are only marginal. The reason is probably that the most common blocks are classified correctly anyway. The following observation supports this assumption: If we classify all the frequently appearing blocks which are mapped to the same phoneme/stress string with a high probability ($\geq 90\%$) (with respect to the entire dictionary) by just selecting this most likely phoneme/stress, the performance decreases. (*Example*: In the case of `ION` we would always answer with `-xn0<<`, since `ION` is mapped to `-xn0<<` in 90% of all cases). This means that these blocks are already classified on a correctness level higher than 90%.

Further improvement might be possible by using a technique that allows blocks to overlap.

5.7 A Combined ID3/Nearest-Neighbor Approach

Nearest Neighbor Classification stores all training examples as representative episodes. ID3 discriminates all examples by its features. The idea behind a combined approach is to use

only some features, preferably relevant or important ones, to discriminate between examples. Objects which can not be discriminated by those features are stored in the form of an exception list in the leaves of the pruned decision tree. If the class of a new example can't be determined by just looking at its features (because descending in the decision tree did not lead to a class but to an exception list), nearest neighbor classification is applied to the exceptions list.

Strategy. To build decision trees with the described characteristics, the CHI-SQUARE test was used to reject irrelevant features for discrimination. Whenever at a particular leaf a set of examples (with instances of both classes) could not be partitioned further because the selection of any features was rules out, the examples were stored in an exception list. [17]

The Hamming distance is used to find a most similar example in the exception lists. Often there are many examples having the same smallest HD, in this case the class of the more numerous (equally similar) examples is used.

Alternative methods such as rejecting features with a high uncertainty value could have been used to reduce the number of features in favor for more exception lists.

For this experiment, no-, 90%- and 99%- CHI-SQUARE cutoff were used to build trees which store exception lists in their leaves. Even when no CHI-SQUARE is applied, there can be cases in which a leaf contains examples with instances of class negative and positive (inconsistencies).

In the regular ID3 tree building procedure, when no exception lists are used and the selection of further features in ruled out by the CHI-SQUARE test, the training examples at that particular leaf are generalized by the majority rule. However, when the training examples at this particular leaf are stored, all the information can be retrieved. Therefore the ID3/NN approach shows the same performance on the training data regardless of how much the tree was pruned:

best guess (3)			WORDS	LETTERS	(PHON/STRESS)		BITS	ID3-TREES LEAVES	DEPTH
ID3-NN	no CHI-SQ	TRAIN:	91.0	98.1	98.8	98.8	99.8	165.3	23.1
ID3-NN	CHI-SQ (90%)	TRAIN:	91.0	98.1	98.8	98.8	99.8	129.2	18.7
ID3-NN	CHI-SQ (99%)	TRAIN:	91.0	98.1	98.8	98.8	99.8	75.5	14.4
ID3	no CHI-SQ	TRAIN:	91.0	98.1	98.8	98.8	99.8	165.3	23.1
ID3	CHI-SQ (90%)	TRAIN:	74.6	94.2	97.1	96.5	99.5	129.2	18.7
ID3	CHI-SQ (99%)	TRAIN:	39.4	83.1	89.2	91.9	98.3	75.5	14.4

Unfortunately, in terms of classifying new examples, the exception lists at the leaves of the pruned tree performed slightly worse than the majority rule did when the regular method was used.

[17] Implementation remark: for space efficiency, the 203-bit-vectors were kept in an array and only indices to the array were stored in exception lists.

Only in the case where no CHI-SQUARE was applied and the exception lists were only used for unresolvable inconsistencies, a 0.1% performance gain for the stresses and the word level could be observed:

best guess (3)			WORDS	LETTERS	(PHON/STRESS)		BITS	ID3-TREES LEAVES	DEPTH
ID3-NN	no CHI-SQ	TEST:	13.0	67.2	79.9	78.1	96.2	165.3	23.1
ID3-NN	CHI-SQ (90%)	TEST:	12.7	66.6	79.4	77.6	96.1	129.2	18.7
ID3-NN	CHI-SQ (99%)	TEST:	10.1	63.7	77.5	76.2	95.8	75.5	14.4
ID3	no CHI-SQ	TEST:	12.9	67.2	79.9	78.0	96.2	165.3	23.1
ID3	CHI-SQ (90%)	TEST:	13.1	66.9	79.6	77.8	96.2	129.2	18.7
ID3	CHI-SQ (99%)	TEST:	11.7	65.8	78.7	77.4	96.1	75.5	14.4

With the strong CHI-SQUARE cutoff, the exception lists became relatively large and therefore classification was more expensive by approximately one order of magnitude.

5.8 Converting Decision Trees to Rules

[Quinlan87] reports a method to convert decision tress into a set of rules. The method makes use of the data from which the decision tree was generated and tries to generalize the rules by eliminating tests in individual rules or entire rules from the set of rules. In [Quinlan87] this method is successfully applied to reduce error rate and size of the knowledge representation of several problems. Quinlan's algorithm was implemented and applied to the decision trees found by ID3 for the NETtalk data. No performance enhancement could be achieved for the NETtalk data. However, the method worked well for another learning task, random boolean concepts (see Chapter Six).

The procedure has three steps:

(1) Convert the given decision tree to a set of rules. Every path from the root of the tree to a leaf forms one rule, with the conjunction of the tests in the nodes as the left-hand-side (*lhs*) of the rule and the class at the leave as the right-hand-side (*rhs*) or the rule.

(2) Examine each of the rules individually; check whether there are tests in the *lhs* a rule that can be removed without affecting the performance of the rule (with regard to the training set)

(3) After the rules are shrunk as individuals, try to shrink the rule set as a whole. A rule R can be removed from the set if the set without R do not perform worse with regard to the training set.

As before, let $C = \{ex_1, \ldots ex_m\}$, the set of m training examples; $ex_i = (x_i, c_i)$, where $x_i \in \{0,1\}^n$ is a n-dimensional binary feature vector and $c_i \in \{+,-\}$ is the binary class associated with x_i.

$RSET = \{R_1, \ldots R_k\}$, the rules extracted from the given decision tree. A rule $R \in RSET$ has the form $(t_1 \wedge \ldots t_i \ldots \wedge t_p \rightarrow class)$, where t_i is a test from a node in the decision tree, e.g. $t_i \equiv f_{155} = 0$

The algorithm in more detail:

(1) Extracting the Rules from the Tree. Straightforward as described above.

(2) Shrinking each Rule Individually. Let $R = (t_1 \wedge \ldots t_i \ldots \wedge t_p \rightarrow c)$ be any rule from the rule set. To determine whether a test t_i can be removed from the rule body, we compute the following contingency table:

Examples satisfying	class c	not class c
$t_1 \wedge \ldots t_i \ldots t_p$	sc	$s\bar{c}$
$t_1 \wedge \ldots \neg t_i \ldots t_p$	$\bar{s}c$	$\bar{s}\bar{c}$

sc is the number of examples that satisfy the left-hand-side (*lhs*) of rule R and are classified correctly by R. $s\bar{c}$ is the number of incorrectly classified examples satisfying the *lhs* of R. Unless the decision tree was trained on inconsistent examples, $s\bar{c}$ will always be zero.

$\bar{s}c$ and $\bar{s}\bar{c}$ are the corresponding numbers for those examples which satisfy the *lhs* of R except test t_i.

Adding the results from the first and second row has the effect of ignoring the impact of test t_i

Examples which satisfy	class c	not class c
$t_1 \wedge \ldots t_{i-1} \wedge t_{i+1} \ldots t_p$	$sc + \bar{s}c$	$s\bar{c} + \bar{s}\bar{c}$

A **certainty-factor** CF can be defined for the rule R by simply computing the ratio of the examples that satisfy the rule-body of R and are correctly classified by R to the total number of examples that satisfy the rule-body of R, i.e.

$$CF(sc, s\bar{c}) = \frac{sc}{sc + s\bar{c}}$$

Quinlan suggests computing the certainty factor as

$$CF(sc, s\bar{c}) = \frac{sc - \frac{1}{2}}{sc + s\bar{c}} \quad (*)$$

since the first way "can be rather optimistic, especially when the numbers are small".

With the above, we can measure the effect of removing the test t_i. $CF(sc, s\bar{c}) = \frac{sc}{sc+s\bar{c}}$ is the certainty-factor for the rule R including the test t_i and $CF(sc + \bar{s}c, s\bar{c} + \bar{s}\bar{c})$ is the certainty-factor after removing test t_i. If $CF(sc, s\bar{c}) \leq CF(sc + \bar{s}c, s\bar{c} + \bar{s}\bar{c})$ then test t_i is a candidate for removal, because removing t_i will improve the CF of R.

[Quinlan87] calls the mysterious constant $\frac{1}{2}$ in $(*)$ "Yates' correction for continuity". The correction does underestimate the CF of the rule, especially for small numbers. However, if the numbers are small, there are cases for which I believe that the correction is a bad idea. Consider the following example: There is only one object in the training set that satisfies the *lhs* of R and it is classified correctly. Say there are 6 examples that satisfy the premise of R if we negate the test t_i. R classifies half of them right and the other half wrong. This corresponds to the following contingency table:

$sc = 1$	$s\bar{c} = 0$
$\bar{s}c = 3$	$\bar{s}\bar{c} = 3$

Then $CF(sc, s\bar{c}) = \frac{1-0.5}{1} = 0.5 \quad \leq \quad CF(sc + \bar{s}c, s\bar{c} + \bar{s}\bar{c}) = \frac{4-0.5}{7} = 0.5$
and therefore the test t_i will be removed, which does not seem to be adequate. Since I was suspicious about this correction factor, I also ran a version of the algorithm which didn't make use of it.
Here is the final procedure for shrinking rules individually:

For each rule $R \in RSET$ do:
repeat
Try to find irrelevant test in the lhs of R by computing
the contingency tables for each test and remove the test which
leads to the best improvement of the CF of R, if there is such a test.
until no more irrelevant tests can be found.

For every test t_i to remove, the contingency-tables for all tests in the rule have to be (re)computed . [18]

(3) Shrinking the Set of Rules.
This last step tries to find rules which can be removed without harm. See [Quinlan87] for more details.

repeat
For each rule $R \in RSET$ do:
Compute the performance of $RSET \setminus \{R\}$ on the training data.
If $RSET \setminus \{R\}$ performs better than $RSET$,
then R is a candidate for removal.
Remove the "best" candidate, if there is one.
until no more rules to remove can be found.

[18] Fortunately we do not have to scan the whole training set C for every test t_i, one pass is sufficient. An example either satisfies all tests t_j or it violates exactly one test or it violates more than one test. We are only interested in the first two cases and can keep a statistic for all tests in parallel in a single pass.

In order to verify the algorithm I additionally ran it on a set of 25 boolean concepts (k-DNF). It worked perfectly on the k-DNF, but it didn't for the NETtalk data. Actually, the second step (shrinking the rule set as a whole) was too expensive for the NETtalk data, it would have consumed several CPU days and was therefore omitted. For the k-DNF data the second step was feasible. It could improve the results slightly, but the first step was more effective.
Here are the unsuccessful results for the NETtalk data:

Rules from ID3 (CHI-SQ 90%)							ID3-TREES	
best guess(3)		WORDS	LETTERS	(PHON/STRESS)		BITS	LEAVES	DEPTH
CONT-FACT 0.5	TEST:	6.7	63.0	74.3	77.4	95.4	0.0	0.0
CONT-FACT 0.0	TEST:	11.6	66.5	79.1	77.9	96.1	0.0	0.0
CONT-FACT 0.5	TRAIN:	32.4	79.6	84.4	91.5	97.6	0.0	0.0
CONT-FACT 0.0	TRAIN:	63.3	91.1	94.7	95.6	99.2	0.0	0.0

5.9 Learning Curves

In this section the impact of the size of the training set is measured. Subsets of 25, 50, 100, 200, 400, and 800 words were generated by randomly selecting the corresponding number of words from the 1000 word training set. Additionally sets of size 2000, 3000 and 4400 [19] were selected randomly from the entire dictionary.

Every training set was learned with three different ID3 versions: ID3 with no CHI-SQUARE and ID3 with CHI-SQUARE using a 90% and 99% confidence level. The trees built from the subsets of the 1000 word training set were tested on the 19000 word test set. The big trees learned from the 2000, 3000 and 4400 word training set were tested on the entire dictionary. Using the entire dictionary instead of the test set leads of course to different results because some of the words were already seen during training. Therefore corrected numbers were computed to neutralize this effect. [20] The best guess strategy was switched at 1000 examples.

Figure 4 contains the tables of all results.

Figure 5 shows the learning curves for letters, phonemes and stresses up to training sets of 1000 words. Observe how the CHI-SQUARE test hurts the performance.

Figure 6 depicts the learning curve for letters on training sets up to 4400 words. The best guess strategy was switched at 1000 words. The lower graph shows how the number of leaves in the trees is reduced by the CHI-SQUARE test. Clearly the number of leaves grows

[19] The number 4400 was selected because [Stanfill86] used the same number of training examples for the MBRtalk experiment

[20] Since the set used for testing is a superset of the training set the following adjustment was computed:

$$c_{corrected} = \frac{c_{test} \times S_{test} - c_{train} \times S_{train}}{S_{test} - S_{train}}$$

c_{test}, c_{train} is the correctness on the test and training set and S_{test}, S_{train} is the size of the test and training set.

linear with the size of the training set. This indicates that ID3 has a nice run time behavior (for this data set), since the run time is linear to the number of nodes in the tree.

Figure 7 shows the learning curves for phonemes and stresses for the large training sets.

5.10 Impact of the Selection of the Training Data

Is it important or even adequate to choose the 1000 most frequent English words as the training data? How much is the performance dependent on the selection of the training examples.

To answer this questions, ten different sets of 1000 words each were randomly selected from the entire dictionary and used as the training set in the same way the 1000 most common word set was used. The performance of the learned trees was tested on the entire dictionary. No CHI-SQUARE cutoff was applied, the best guess strategy was (3), the letter/phoneme and letter/stress mappings to constrain the best guess were selected from the entire dictionary.

The table shows the results for the "classic" most common word training set and the ten randomly chosen 1000 word sets:

ID3, no CHI-SQUARE, best guess(3)		WORDS	LETTERS	(PHON/STRESS)		BITS	ID3-TREES NODES	DEPTH
1000 most common words	ALL:	17.0	68.4	80.6	78.8	96.3	165.3	23.1
1000 random words 1	ALL:	17.8	71.5	83.4	80.5	96.8	251.8	27.8
1000 random words 2	ALL:	17.6	71.5	83.8	80.0	96.8	250.6	27.6
1000 random words 3	ALL:	18.7	72.2	84.0	80.5	96.9	249.9	27.0
1000 random words 4	ALL:	19.5	72.2	84.2	80.6	96.9	239.0	26.9
1000 random words 5	ALL:	18.9	72.1	84.0	80.6	96.9	241.6	25.4
1000 random words 6	ALL:	16.4	71.5	83.4	80.2	96.8	247.7	27.6
1000 random words 7	ALL:	18.6	71.8	84.0	80.5	96.9	250.4	27.7
1000 random words 8	ALL:	18.2	72.3	84.2	80.9	96.9	246.8	27.3
1000 random words 9	ALL:	18.4	72.0	83.9	80.7	96.9	249.7	27.9
1000 random words 10	ALL:	19.4	72.3	84.1	81.0	96.9	245.7	28.7

It is interesting to note that *all* of the 10 randomly chosen training sets produced better trees than the most common 1000 word set. However, one reason for the better performance of the randomly chosen sets are that they contain more training data, since the average word length of a word in the entire dictionary is 7.34 letters while the most common 1000 words have only 5.53 letters in the average. Therefore a randomly chosen 1000 word set is expected to contain about 7340 letters, 1700 more than the most common 1000 word training set. While the better performance can be explained by the fact that the random sets contain more examples as the 1000 most common word training set, the size of the trees is overproportional.

5.11 Other Variations

Changing the binary representation of the 7-letter window. Every character in the 7-letter window is represented by 29 bits. Do we really need this huge 203-bit input vectors, or can we use a more concise representation? For this experiment, the minimum number

ID3, no CHI-SQUARE		WORDS	LETTERS	(PHON/STRESS)		BITS	ID3-TREES LEAVES	DEPTH
best guess(1)	25 TEST:	0.6	29.0	43.7	58.5	90.8	10.9	6.1
best guess(1)	50 TEST:	1.3	35.3	52.4	61.3	92.3	16.4	7.5
best guess(1)	100 TEST:	2.5	45.1	62.3	63.6	93.5	28.3	9.5
best guess(1)	200 TEST:	4.0	52.5	67.0	71.0	94.6	46.6	11.9
best guess(1)	400 TEST:	6.3	55.8	70.8	72.2	95.1	81.4	16.4
best guess(1)	800 TEST:	8.4	58.3	73.7	72.1	95.5	138.1	19.7
best guess(1)	1000 TEST:	9.1	60.8	75.4	74.4	95.8	165.3	23.1
best guess(2)	1000 TEST:	11.9	66.1	78.4	78.0	96.2	165.3	23.1
best guess(2)	2000 corr:	17.7	73.0	84.6	81.5	97.1	436.5	34.2
best guess(2)	3000 corr:	20.5	75.0	86.0	82.3	97.3	612.6	35.5
best guess(2)	4400 corr:	22.7	76.9	87.2	83.6	97.6	834.5	38.7

ID3, CHI-SQUARE (90%)		WORDS	LETTERS	(PHON/STRESS)		BITS	ID3-TREES LEAVES	DEPTH
best guess(1)	25 TEST:	0.4	26.8	39.1	59.4	90.7	9.1	5.6
best guess(1)	50 TEST:	1.3	35.1	51.9	61.3	92.0	14.2	6.8
best guess(1)	100 TEST:	1.8	42.8	61.5	63.0	93.4	23.7	8.7
best guess(1)	200 TEST:	3.8	51.3	65.4	70.2	94.5	39.6	10.7
best guess(1)	400 TEST:	5.8	54.7	69.7	71.9	95.0	67.2	15.0
best guess(1)	800 TEST:	7.7	57.5	73.0	71.9	95.5	112.4	17.7
best guess(1)	1000 TEST:	9.1	59.8	74.4	73.9	95.7	129.2	18.7
best guess(2)	1000 TEST:	12.1	65.6	77.9	77.8	96.1	129.2	18.7
best guess(2)	2000 corr:	17.7	72.7	84.3	81.8	96.7	325.1	26.9
best guess(2)	3000 corr:	19.9	74.6	85.9	82.3	97.4	443.5	30.6
best guess(2)	4400 corr:	22.8	76.7	87.2	83.6	97.5	532.5	31.2

ID3, CHI-SQUARE (99%)		WORDS	LETTERS	(PHON/STRESS)		BITS	ID3-TREES LEAVES	DEPTH
best guess(1)	25 TEST:	0.3	24.9	40.4	55.4	90.4	6.7	4.9
best guess(1)	50 TEST:	0.9	30.7	50.9	58.2	92.0	9.7	6.0
best guess(1)	100 TEST:	0.9	38.7	57.5	61.7	93.0	16.5	7.9
best guess(1)	200 TEST:	2.9	49.2	62.6	70.7	94.1	24.9	9.5
best guess(1)	400 TEST:	3.9	53.4	67.2	71.8	94.9	40.4	11.5
best guess(1)	800 TEST:	6.9	56.0	71.7	71.6	95.3	65.6	13.2
best guess(1)	1000 TEST:	8.6	59.0	74.0	73.6	95.6	75.5	14.4
best guess(2)	1000 TEST:	11.1	64.8	77.4	77.4	96.1	75.5	14.4
best guess(2)	2000 corr:	16.5	71.3	82.8	81.6	97.0	160.7	19.6
best guess(2)	3000 corr:	18.9	73.7	85.1	82.1	97.2	231.3	23.2
best guess(2)	4400 corr:	21.4	75.7	86.3	83.5	96.6	301.0	23.2

Figure 4: Tables for different CHI-SQUARE versions on training sets of increasing size.

Figure 5: Learning Curves for letters, phonemes and stresses.

Figure 6: Learning curve for letters (big training sets); the tree sizes for different CHI-SQUARE versions and increasing number of training examples.

Figure 7: Learning curves for phonemes and stresses (big training sets).

$ceiling(\log_2(29)) = 5$ of bits necessary to represent the 29 characters was used to represent the 7-letter window as a $7 \times 5 = 35$ bit-vector. However, the denser representation does not work very well:

ID3, no CHI-SQUARE		WORDS	LETTERS	(PHON/STRESS)	BITS	ID3-TREES NODES	DEPTH
5-bit representation	TRAIN	89.0	97.3	98.2	98.6	99.8	246.0	14.1
	TEST :	1.3	40.8	51.5	63.5	94.0	246.0	14.1

Note that the trees became almost twice as big as before. The problem with this approach is that ID3 has to test up to five features in order to discriminate one letter from another one, while there is exactly one bit per letter in the 29-bit-per-letter encoding.

5.12 Learning and Data Compression

Learning can be viewed as data compression. By finding regularities in the given training examples we can obtain a more concise representation of the knowledge.

Information content of the dictionary. The plain NETtalk dictionary contains 20002 English words and a total of 146913 characters, phonemes and stresses. Since there are 29 different characters, 54 different phonemes and 6 different stresses, the information content of a character, phoneme and stress is 5,6,and 3 bits respectively. [21] Therefore the total information stored in the dictionary is $146914 * (5 + 6 + 3) = 2056796$ bits.

Information content of the decision trees. A node or leaf of a decision tree is either a feature (there are 203 features) or a class (+,-). Using (e.g.) postfix notation, we need $2n \times 8$ bits to encode a decision tree with n leaves and $n - 1$ nodes. The decision trees constructed by ID3 without CHI-SQUARE have 165 leaves in the average, therefore all 27 decision trees have an information content of $27 \times 2 \times 165 \times 8 = 71280$ bits. The trees built with a CHI-SQUARE cutoff of 99% have 76 leaves in the average which corresponds to 31616 bits total information content.

Information content of the neural net. The neural net described in Chapter Three has 27809 connections, each represented by a real number. While it was very interesting to see that the full precision of the real numbers are needed to ensure proper learning [Dietterich89], the precision of the weights in the trained net are much more fault tolerant. Experiments (testing the performance when the weights are corrupted with noise) suggest that the precision actually needed can be guaranteed with 4 bits. [Sejnowski89]. Thus the information stored in the neural net can be estimated as roughly $27809 \times 4 = 111236$ bits.

Conclusions. The learning methods described above can learn nearly 70% of the letters in the dictionary correctly, but they need less than 4% ($\frac{71280}{2056796}$ in the case of ID3) of the information stored in the dictionary to represent their knowledge.

[21] Actually the information content is only $log_2(29 * 54 * 6) = 13.2$ bits. If we exploit the different probabilities of the characters (Huffman coding), we could reduce the number further, but this is already data compression.

6 Boolean Concepts (k-DNF)

Since both the CHI-SQUARE version of ID3 and the Tree-to-Rules conversion algorithm did not perform as well as expected on the NETtalk data, I ran both versions on another learning task, random boolean k-DNF concepts.

The Boolean Concepts. For the following experiments 25 different randomly generated k-DNFs [22] over 30 boolean variables were used. There were 5 groups with increasing complexity of the randomly chosen k-DNFs. The parameters for the k-DNFS were chosen randomly within the following boundaries:
Number of variables = 30
$t_{min} \leq$ number of disjuncts in k-DNF $\leq t_{max}$
$k_{min} \leq$ number of variables in disjuncts $\leq k_{max}$

	k_{min}	k_{max}	t_{min}	t_{max}
group 1:	1	5	3	6
group 2:	2	8	6	8
group 3:	3	10	10	14
group 4:	3	12	20	22
group 5:	4	15	30	40

For each single group, 5 k-DNFs were generated. The first 3 k-DNFs are different, the last 2 are the same as the 3rd, but the feature vectors of the training and test data of the 4th and the 5th were corrupted with 5% and 20% noise, respectively.

Here is an example of a k-DNF of complexity group 4 (12-DNF):

```
( OR
     (AND (NOT 2) 3 (NOT 5) 12 15 (NOT 18) 22 (NOT 26) 27 (NOT 29))
     (AND 10 (NOT 12) (NOT 14) (NOT 29))
     (AND 1 (NOT 2) (NOT 3) 13 (NOT 15) (NOT 17) (NOT 24) 25 (NOT 27) (NOT 29))
     (AND (NOT 0) (NOT 1) 5 (NOT 9) (NOT 19) (NOT 20) (NOT 23) (NOT 28))
     (AND 4 11 (NOT 16) 18 20 21 (NOT 22) 28 29)
     (AND (NOT 6) 7 (NOT 20) (NOT 24) (NOT 27))
     (AND (NOT 1) (NOT 15) 19 (NOT 24) 27)
     (AND 0 2 (NOT 12))
     (AND (NOT 0) 5 (NOT 10) (NOT 12) 16 (NOT 18) 20 (NOT 25))
     (AND 1 3 (NOT 5) (NOT 11) 13 15 23 26 27 28)
     (AND (NOT 2) 8 10 (NOT 19) 20 23)
     (AND (NOT 0) (NOT 9) 11 17 (NOT 19) (NOT 21))
     (AND (NOT 6) (NOT 7) 8 (NOT 9) (NOT 12) (NOT 13) 17 (NOT 19) 21 28)
     (AND (NOT 0) 3 4 (NOT 7) (NOT 10) (NOT 20) 26 28)
     (AND (NOT 9) 23 26)
     (AND (NOT 8) 9 (NOT 12) (NOT 13) 16 (NOT 21) (NOT 25) (NOT 26) (NOT 29))
     (AND (NOT 0) (NOT 1) (NOT 3) 4 5 (NOT 7) (NOT 11) 14 26)
     (AND 14 (NOT 15) 18 19 (NOT 21) (NOT 24) 28)
     (AND (NOT 2) (NOT 5) (NOT 20) (NOT 21) 23)
     (AND (NOT 1) 6 11 12 15)
     (AND (NOT 1) (NOT 3) (NOT 4) (NOT 5) 6 (NOT 15) (NOT 18) 25 (NOT 27))
  )
 21 terms       (min average max) = (4  8.095238095238095  11)
```

[22] A **k-DNF** if a boolean function f with the following properties: 1) f is in Disjunctive Normal Form (DNF) 2) Every disjunct of f has at most k boolean variables.

ID3 and the CHI-SQUARE TEST. Three different ID3 - versions were used:

1. NO CHI-SQUARE
2. CHI-SQUARE (90%)
3. CHI-SQUARE (99%)

Each version was run on 4 different training and test sets: [23]

	training	test
1.	5000	15000
2.	10000	30000
3.	15000	25000
4.	20000	20000

See the following page for the results of the $3 \times 4 = 12$ runs.

Observations:

- With very few exceptions, CHI-SQUARE cutoff generally improves the performance. Especially on the noisy data there is a significant improvement.
- (As expected,) training on more examples increases performance. While the performance of the less complex concepts is already relatively saturated with 5000 training examples, the more complex concepts gain up to 8% in performance with the increasing number of training examples.
- There are $2^{30} = 10^9$ different vectors. ID3 could see only 0.001 % of all possible examples.

[23] The selection of the high numbers of training and test examples is unfortunate, because the curves are already highly saturated with so many examples. The original idea behind this numbers was to use data sets of similar size to a set of 1000, 2000, 3000, and 4000 word from the dictionary.

```
ID3 on random k-DNFs
====================
```

	NO CHI-SQUARE				CHI-SQUARE (90%)				CHI-SQUARE (99%)			
	5	10	15	20	5	10	15	20	5	10	15	20
1:	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
2:	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
3:	99.9	100.0	100.0	100.0	99.9	100.0	100.0	100.0	99.7	100.0	100.0	99.9
4:	87.5	88.0	88.0	88.4	92.8	92.9	93.1	93.6	94.9	95.2	95.4	95.3
5:	70.6	71.1	71.0	70.9	76.9	76.8	78.1	78.8	82.7	83.0	83.3	83.5
6:	98.4	99.1	99.5	99.7	98.8	99.4	99.6	99.7	98.6	99.5	99.6	99.8
7:	98.3	99.5	99.8	99.8	98.5	99.5	99.8	99.9	98.8	99.6	99.8	99.9
8:	97.9	98.6	99.0	99.2	98.2	99.0	99.3	99.4	99.0	99.3	99.4	99.6
9:	87.4	89.1	89.4	89.2	91.5	93.7	93.9	94.4	94.4	95.1	95.2	95.8
10:	74.1	75.1	75.4	75.6	80.4	80.8	81.7	81.7	84.4	86.0	86.4	86.5
11:	91.4	93.2	94.1	94.6	92.6	94.6	95.3	95.5	94.1	95.8	96.1	96.4
12:	94.2	95.1	96.1	96.7	95.3	96.1	96.9	97.7	95.9	97.0	97.4	98.0
13:	88.6	91.1	92.1	92.9	90.0	92.8	93.8	94.3	90.6	94.0	94.8	94.9
14:	79.4	81.2	82.2	82.6	83.2	86.2	87.3	87.8	86.2	88.7	89.9	90.6
15:	64.0	65.7	66.0	66.1	69.6	69.6	71.4	72.3	75.6	76.7	78.1	78.0
16:	89.0	90.2	90.1	92.3	90.0	92.0	91.8	93.9	91.6	93.0	93.4	94.6
17:	83.6	85.6	88.3	88.9	84.9	87.5	89.6	90.4	86.9	89.4	91.0	91.4
18:	87.0	89.1	91.3	93.2	88.0	89.8	92.2	94.2	89.1	90.0	92.5	93.8
19:	77.5	80.0	80.4	82.0	80.7	83.1	84.8	86.0	83.5	85.4	86.9	87.7
20:	65.5	66.0	66.0	66.4	68.4	69.5	71.0	71.1	73.5	75.0	75.8	75.8
21:	75.3	79.8	81.3	83.3	78.1	81.0	82.6	85.1	78.9	82.7	83.9	85.6
22:	74.6	79.6	80.8	82.2	76.4	81.1	82.4	84.2	78.1	82.8	84.2	86.1
23:	78.2	82.7	84.6	85.9	81.1	85.2	86.5	87.6	82.3	86.8	87.9	88.8
24:	72.5	74.5	76.1	75.5	75.1	78.6	80.0	79.9	78.8	82.4	83.6	83.6
25:	64.0	64.4	64.4	64.9	67.9	68.5	68.4	69.4	73.5	73.8	75.0	74.7

```
(Recall that there is 5% feature noise in 4,9,..24  and 20% feature noise
  in 5,10,..25)
```

Converting Decision Trees to Rules. The same 5 groups of kDNF as above are used for this experiment. The "tree-to-rules-conversion" is applied to the decision trees which were learned from the 25 k-DNFs. ID3 with CHI-SQUARE(99%) [24] was used to build the trees.

The table below show the results off three runs:

1. `The ID3-tree (CHI-SQUARE 99%), trained on 5000 vectors`
2. `The set of Rules from the ID3-tree. Each Rule is shrunk individually.`
3. `The set of Rules, additionally the the rules are shrunk as a set.`

[24] The high CHI-SQUARE cutoff was necessary in order to keep the trees (i.e. the number of rules) small, since shrinking the rules is very expensive.

The number of rules is the same in (1) and (2) but reduced in (3). The number of tests (average for all rules) is reduced from (1) to (2).

	TRAIN			TEST			NUMBER OF RULES		NUMBER Of TESTS	
	1	2	3	1	2	3	1,2	3	1	2
1:	100.0	100.0	100.0	100.0	100.0	100.0	9	9	4.3	2.6
2:	100.0	100.0	100.0	100.0	100.0	100.0	15	14	4.3	2.5
3:	99.7	100.0	100.0	99.7	100.0	99.9	57	41	7.0	3.8
4:	94.7	95.2	95.2	94.9	95.5	95.4	49	36	6.4	4.6
5:	82.9	82.9	83.1	82.7	83.3	83.3	47	28	6.1	4.1
6:	98.7	99.9	99.9	98.6	99.9	99.7	59	40	7.1	4.4
7:	99.4	100.0	100.0	98.8	99.5	99.8	56	34	7.3	4.0
8:	99.0	99.2	99.2	99.0	99.2	99.2	56	36	6.9	4.0
9:	94.5	94.6	94.6	94.4	94.5	94.5	44	28	6.3	4.5
10:	86.4	85.8	86.2	84.4	84.1	84.3	44	34	6.3	4.7
11:	94.9	98.8	98.9	94.1	98.3	98.3	90	60	7.3	5.2
12:	96.5	98.3	98.3	95.9	97.9	98.1	83	58	7.2	5.3
13:	93.5	97.0	97.4	90.6	95.3	96.7	122	71	7.8	5.3
14:	88.2	89.9	90.0	86.2	88.2	88.9	97	57	7.1	5.5
15:	78.1	77.2	78.1	75.6	75.6	76.2	57	39	6.5	4.8
16:	93.4	97.0	97.3	91.6	95.7	96.3	107	73	7.5	5.6
17:	90.0	93.6	93.6	86.9	91.5	92.0	112	75	7.5	5.7
18:	92.4	94.8	95.5	89.1	93.2	93.6	122	74	7.8	5.6
19:	87.1	87.5	89.3	83.5	85.2	86.7	105	66	7.5	5.6
20:	77.8	78.0	78.3	73.5	75.1	75.3	67	48	6.5	4.8
21:	82.9	87.7	89.0	78.9	84.9	86.3	128	82	7.5	5.6
22:	82.2	87.3	87.7	78.1	85.9	86.3	101	73	7.1	5.5
23:	84.0	93.7	93.8	82.3	92.3	92.6	98	47	7.2	5.7
24:	80.0	84.0	84.1	78.8	83.3	84.0	84	44	6.8	5.2
25:	74.9	75.2	75.2	73.5	73.8	73.8	46	26	5.8	4.2

Observations:

- With one exception in the noisy kDNF-data, shrinking the individual rules generally improved the performance on the test data (roughly) by 3 to 5%. (The improvements on the less complex kDNFs are not so good, since their high performance percentage is already saturated)
- Shrinking the rule set achieves another small gain in performance.
- Note how the average number of tests in a rule is reduced from step 1 to step 2 and the number of rules is reduced from step 2 to 3.

7 Conclusions

Summary of results. After implementing the plain ID3 algorithm, I experimented with various modifications. Two improvements of the process of finding a legal phoneme/stress could be made by using statistical information about the letter to phoneme/stress-mapping in the training set.

Adding the CHI-SQUARE test to the ID3 algorithm was successful in terms of reducing the size of the trees, but could not enhance the performance. However, on the random k-DNF concepts the CHI-SQUARE test was very effective.

Learning the stresses from the phonemes instead of from the English text showed the interesting characteristic that the overall performance improved, although the stresses got worse.

When a separate set of trees was learned for each letter, some letters improved and others didn't, while the stress performance suffered generally. By staying with separate trees only for the "winner letters" and learning the stresses from the phonemes allowed for further gains in performance.

In the "multi-level-ID3" experiment the goal was to learn on a higher level in some cases. Common letter combinations such as `er` or `ion` were extracted from the data and treated as an entity during the learning phase. Trying to learn on this level was not as successful as expected, probably because the number of training examples became too small. In fact, it turned out that using the common letter-blocks just to constrain the outcome found with the standard letter-by-letter classification was even more successful.

A combined ID3/NN-approach kept subsets of the training examples as exceptions lists in some of the leaves of the tree. This method was not as successful as the regular way of generalizing such leaves with the majority rule.

The postprocessing of rules extracted from the decision trees as suggested by [Quinlan87] enhanced performance for the boolean k-DNF concepts but was not successful on the NETtalk data.

The selection of examples for the training set seems not to have a big influence on the performance. Trees trained on ten randomly selected 1000 word training sets show only minor deviations in their performance. However, all ten randomly selected training sets result in much bigger trees then the most common thousand word training set.

Here is a summary of the achieved improvements:

							ID3-TREES	
		WORDS	LETTERS	(PHON/STRESS)		BITS	LEAVES	DEPTH
(0) Neural Net	TEST:	12.1	67.1	77.6	79.6	96.1	-	-
(1) ID3, plain	TEST:	9.1	60.8	75.4	74.4	95.8	165.3	23.1
(2) ID3, b.g.(3)	TEST:	12.9	67.2	79.9	78.0	96.2	165.3	23.1
(3) ID3	TEST:	14.3	68.2	80.5	76.5	96.2	-	-
(4) ID3	TEST:	15.8	69.3	80.3	78.6	96.2	165.3	23.1

(0) Neural Net, back-propagation [Dietterich89].
(1) Plain ID3, with the straightforward "best guess" strategy.
(2) ID3, with an improved "best guess" strategy.

```
(3) Combined method of learning some letters individually and
    classifying the stresses from the phonemes
(4) Learning on a higher level with common letter blocks
```

The neural net (0) outperforms the comparable ID3 result (1). None of the ID3 versions can learn the stresses as well as the back-propagation algorithm. It would be interesting to see how much performance one could gain if the methods used to improve ID3 were applied to the neural net.

One reason for the better performance of the neural nets might be fact that they learn all bits simultaneously.

Grain-size of Learning. The task of learning a word is decomposed in two steps: First each letter of the word is learned individually. Furthermore, the task of learning the pronunciation of a letter is broken down into learning 27 properties of the phoneme and the stress. This low level representation was chosen to break the complexity of the high level learning task. The mapping from a 203-bit-vector to a boolean value is easier to handle than the mapping from a English word to its phoneme and stress string. The problem with this approach is the aggregation of the results. Although each of the low level functions could be learned with about 97% correctness, less than 15% correctness remains when it comes to the word level.

On the other hand, if we try to learn on a higher level, we face the problem that the algorithm can not see as many examples for a particular concept as before. When we tried to learn blocks instead of single letters, there were only a few examples for each block versus the entire dictionary for each letter. In the extreme case, when we try to learn entire words at a time, there is only one example for each different mapping.

Further improvements. ID3, the neural net (back-propagation) and the nearest neighbor classification are general purpose learning algorithms and can be applied to any learning problem. One way to improve the performance of any of these algorithms might be to make use of the specific domain knowledge of the pronunciation task.

The general algorithms allow any phoneme/stress - string as outcome. The solution space could be reduced by using constraints such as

- There has to be at least one vowel(-sound) in each phoneme string.
- There are many sequences of phonemes which are illegal or even "unpronounceable".
- Many stress combinations are illegal, e.g. any subsequence '><' is undefined. No stress string can start with '<' or end with '>'. This may be a better way to find a stress string: First find all letters which are emphasized (indicated by the stresses 1, 2, 0 and then determine the syllable boundaries.

8 Appendix

8.1 About the Implementation

All algorithms were implemented in Common LISP. The code ran on SUN 4 workstations with 32 MByte main memory. Unix and the X-Window system provided an efficient environment for developing and debugging the code.

Computing the 27 decision trees for the 5521 letters in the training set consumed about 50 CPU minutes. The 141392 letters in the test set could be classified in approximately 35 CPU minutes.

8.2 Inconsistencies in the NETtalk Dictionary

Even the perfect learning algorithm could not achieve a 100% performance on the Nettalk DATA, because there are inconsistencies in the dictionary. One reason for inconsistencies is the size of the text windows used. To know how to pronounce the `"th"` in the word `"thought"`, a 7-letter window `"___thou"` is not enough, we need a look-ahead up to the last letter to tell the difference between `"thought"` and (e.g.) `"though"`. However, there are words that are inherently inconsistent because they are listed twice as the same word with a different pronunciation. The following paragraph describes the inconsistencies on bit-, letter- and word-level.

Since all phoneme and stress bits are learned separately, we have a separate set of training examples for each phoneme and stress bit. A learning example ex on the bit-level has the format $ex = x.c$, where $x \in \{0,1\}^{203}$ and $c \in \{0,1\}$.

Let S be the set of all learning examples $ex_k = x_k.c_k$ for a particular phoneme or stress bit. We say that the example ex_k is inconsistent on the bit-level if there is another l.e. $ex_j = x_j.c_j \in S\,(ex_j \neq ex_k)$ such that $x_k \neq x_j$ but $c_k = c_j$.

For the training and the test set we get the following average rate of bit-level inconsistency for phoneme and stress bits:

		PHONEME-BIT		STRESS-BIT	
average	total	incons.	%	incons.	%
TRAIN:	5521	7.9	0.14	13.2	0.24
TEST :	141392	264.0	0.19	779.8	0.55

A learning example on the letter-level has the format $ex_k = (w_k p_k s_k)$ where w_k is a 7 letter window, p_k is a phoneme s_k is a stress

An example is defined to be inconsistent on letter-level, if there are two l.e. $ex_k = (w_k p_k s_k)$ and $ex_j = (w_j p_j s_j)$ which have the same 7 letter window ($w_k = w_j$) but map to a different phoneme or stress $p_k \neq p_j$ or $s_k \neq s_j$

the average rate of letter-level inconsistency is:

LETTERS	total	incons.	%
TEST :	141392	4239	3.0
TRAIN:	5521	90	1.6

An entire word is considered inconsistent, if one of its letters are inconsistent. The inconsistency rate on word level is

WORDS	total	incons.	%
TEST :	19003	3358	17.7
TRAIN:	999	66	6.5

The most obvious case of inconsistency occurs when one word is represented twice in the dictionary (with two different phonetic or stress strings). There are 35 double words in the training set and 206 in the whole dictionary (i.e. 171 in the test set).

Example:

```
that    |D-@t|  >>1<
that    |D-xt|  >>0<
```

Summarized percentages of inconsistencies:

Inconsistencies	double WORDS	WORD	LETTER	BIT(PHONEME/STRESS)	
TEST :	0.8	17.7	3.0	0.19	0.55
TRAIN:	3.5	6.5	1.6	0.14	0.24

8.3 Top Features of the Decision Tress

At each node in the tree, ID3 heuristically tries to find the "best" feature in order to build a compact decision tree. Therefore the feature at the root of the tree is the one considered to be most important. In the case of the NETtalk data a feature stands for a particular letter at a particular position in the 7-letter window. The table below shows the letters and their window positions selected by ID3 for first three levels of the decision trees. As we would have expected, the center letter of the window is selected as the top feature for all except two of the stress trees.

Decision Tree	Features of topmost three levels
PHON-DT 0	$(T_0\ (C_0\ S_0\ C_{-1})\ (H_1\ T_{-1}\ E_2))$
PHON-DT 1	$(N_0\ (S_0\ T_0\ S_{-1})\ (G_1\ N_{-1}\ E_2))$
PHON-DT 2	$(L_0\ (A_0\ E_0\ E_{-1})\ (\#_2\ L_{-1}\ \#_1))$
PHON-DT 3	$(O_0\ (H_0\ U_0\ \#_{-1})\ (N_1\ \#_1\ L_2))$
PHON-DT 4	$(M_0\ (P_0\ I_0\ P_{-1})\ (M_{-1}\ S_{-1}$ -))
PHON-DT 5	$(R_0\ (O_0\ A_0\ I_{-1})\ (E_{-1}\ \#_{-2}\ \#_1))$
PHON-DT 6	$(C_0\ (R_0\ O_0\ E_{-1})\ (E_1\ I_1$ -))
PHON-DT 7	$(X_0\ (J_0\ G_0$ +) +)
PHON-DT 8	$(S_0\ (F_0\ V_0\ F_{-1})\ (S_{-1}\ L_1$ -))
PHON-DT 9	$(W_0\ (H_0\ Q_{-1}\ \#_{-1})\ (\#_{-1}\ Y_2\ O_2))$
PHON-DT 10	$(R_0\ (L_0\ R_{-3}\ L_{-1})\ (R_{-1}\ A_{-3}$ -))
PHON-DT 11	$(N_0\ (M_0$ - $M_{-1})\ (N_{-1}$ + -))
PHON-DT 12	$(T_0\ (D_0\ P_0\ D_{-1})\ (H_1\ O_2$ -))
PHON-DT 13	$(O_0\ (Y_0\ A_0\ A_{-1})\ (N_1\ R_1\ \#_2))$
PHON-DT 14	$(R_0\ (N_0\ L_0\ N_{-1})\ (R_{-1}\ A_{-3}$ -))
PHON-DT 15	$(I_0\ (E_0\ Y_0\ \#_{-3})\ (N_2\ E_2\ D_{-1}))$
PHON-DT 16	$(A_0\ (O_0\ U_0\ R_1)\ (\#_{-3}\ P_{-1}\ E_{-1}))$
PHON-DT 17	$(O_0\ (A_0\ E_0\ E_{-1})\ (I_{-1}\ O_{-1}\ R_1))$
PHON-DT 18	$(E_0\ (H_0\ \#_{-2}\ \#_{-1})\ (\#_1\ R_1\ \#_{-3}))$
PHON-DT 19	-
PHON-DT 20	-
PHON-DT 21	$(E_0\ (H_0\ \#_{-2}\ \#_{-1})\ (\#_1\ R_1\ \#_{-3}))$
STRESS-DT 1	$(\#_2\ (\#_{-1}\ O_{-1}$ -) $(\#_{-2}\ R_1\ A_{-1}))$
STRESS-DT 2	$(\#_{-1}\ (\#_2\ E_0\ Y_1)\ (A_0\ N_1$ -))
STRESS-DT 3	$(O_0\ (A_0\ I_0\ E_{-1})\ (I_{-1}\ O_{-1}\ \#_1))$
STRESS-DT 4	$(I_0\ (A_0\ E_0\ E_{-1})\ (A_{-1}\ E_{-1}$ -))
STRESS-DT 5	-

Features of the three topmost levels of the Decision trees.

The parentheses represent the tree-structure of the highest three levels in the decision trees. The letters stand for the actual letter selected as the feature (the test) at the particular node in the tree. To enhance readability, the character '_' (beyond word boundary) is represented by a '#'. The subscripts indicate the position of the letter in the 7-letter window.

Example: $(A_0\ (X_{-1}\ ..)\ (C_3\ ..))$ means that the root feature of the tree is the test for a letter A in the center position of the 7-letter window. The two tests on the second level of the tree check for a letter X one position to the left of the center and for a C at the rightmost position.

8.4 Saturation Points for the Letter to Phoneme Mapping

How much do we know about the 20000 words in the dictionary when we look at a subset of 1000 words? Is it as good as looking at 500 words, or as 200? How many data do we have to see before we know all the phonemes to which a letter can be mapped? To answer this kind of questions, the entire dictionary was randomly partitioned into 16 sets of 1250 words each. Then the number of phonemes to which a letter is mapped was counted for the words in the first set, in the first and second set and so on. This is like increasing our horizon by 1250 words in each measurement. A **saturation point** was computed for each letter. For example, after looking at 5000 words, the letter `E` reached the point where it was mapped to 13 different phonemes. This is the maximum number of phonemes to which `E` is mapped. The graph below shows the saturation point for each letter. The curve indicates the average number of different phonemes to which all letters are mapped to after looking at the corresponding number of examples. Of course the outcome of this results are highly dependent on the chosen subsets and the order in which we look at them.

- - - - picture - - - -

8.5 An Example of a Decision Tree

The decision tree for phoneme bit 0, learned with ID3 and CHI-SQUARE cutoff (90%). All nodes on the same tree level have the same horizontal indentation. `+`, `-` show the class of a leave. A number behind the class indicates that the majority rule was applied, e.g. `- (62)` means that class negative was assigned to a leave containing 62 examples with no unique class. This is a rather small tree with 62 leaves, in the non CHI-SQUARE version trees grow as big as 394 leaves.

```
----------------------------------------
---          ID3-Tree                  ---
 (106=0)
    (89=0)
       (105=0)
          (102=0)
             (92=0)
                (94=0)
                   (97=0)
                      (103=0)
                         (110=0)
                            (123=0)  -
                            (123=1)
                               (13=0)  -
                               (13=1)  +
                         (110=1)
                            (193=0)
                               (157=0)  +
                               (157=1)  -
                            (193=1)  -
                      (103=1)  +
                   (97=1)
                      (60=0)
                         (196=0)  +
                         (196=1)  -
                      (60=1)  -
                (94=1)
                   (85=0)
                      (80=0)
                         (166=0)
                            (160=0)
                               (58=0)  -
                               (58=1)  +
                            (160=1)  +
                         (166=1)  +
                      (80=1)
                         (130=0)  -
                         (130=1)  +
                   (85=1)
                      (165=0)  +
                      (165=1)
                         (192=0)  -
                         (192=1)  +
             (92=1)
                (63=0)
                   (72=0)  +
                   (72=1)
                      (143=0)  +
                      (143=1)  -
                (63=1)  -
          (102=1)
             (73=0)
                (134=0)  +
                (134=1)
                   (75=0)  +
                   (75=1)  -
             (73=1)  -
       (105=1)
```

```
          (85=0)
             (76=0)
                (135=0)
                   (134=0)
                      (71=0)
                         (123=0)
                            (179=0)
                               (131=0)
                                  (187=0)
                                     (46=0)
                                        (0=0)
                                           (82=0)
                                              (62=0)
                                                 (8=0)  -(62)
                                                 (8=1)  +
                                              (62=1)
                                                 (53=0)  -
                                                 (53=1)  +
                                           (82=1)  -
                                        (0=1)  +
                                     (46=1)
                                        (191=0)  -
                                        (191=1)  +
                                  (187=1)  -
                               (131=1)  +
                            (179=1)  +
                         (123=1)  +
                      (71=1)  +
                   (134=1)  +
                (135=1)  +
             (76=1)  -
          (85=1)  +
    (89=1)
       (60=0)
          (76=0)
          (76=1)  -(4)
       (60=1)
          (193=0)  -
          (193=1)  +
 (106=1)
    (123=0)
       (77=0)
          (152=0)  +
          (152=1)  -
       (77=1)  -
    (123=1)
       (149=0)
          (27=0)
             (12=0)  +
             (12=1)  +(3)
          (27=1)
             (192=0)
                (66=0)
                   (193=0)
                      (194=0)
                         (145=0)  +
                         (145=1)  -(2)
                      (194=1)  -(3)
                   (193=1)  -
                (66=1)  -
             (192=1)  -
       (149=1)
          (188=0)  -(23)
          (188=1)  +

Nodes      : 62
Max. Depth : 17
----------------------------------------
```

8.6 The Mapping from Letters to Phonemes and Stresses

Letter	Phonemes to which a letter is mapped
-	
.	
A	@ .30 x .24 e .19 - .13 a .07 c .04 E .02 I .01 o W y i
B	b .96 - .04
C	k .71 s .16 C .07 - .03 S .03 g z t x
D	d .94 - .04 J .01 t D
E	- .48 E .20 x .12 i .10 I .08 e .01 Y A u a o y U
F	f .91 - .09 v
G	g .51 J .27 - .20 Z f G o k
H	- .73 h .27 E w
I	I .42 x .20 - .16 A .12 i .09 y Y @ a
J	J .98 h Z - y i
K	k .66 - .34
L	l .70 L .21 - .09
M	m .93 - .04 M .03 n G
N	n .87 G .07 N .04 - .02
O	x .31 o .22 a .19 - .08 c .07 W .04 u .04 O .02 U .01 ^ .01 + I w A y e R @ Y
P	p .88 f .07 - .04
Q	k .97 - .03
R	r .72 R .26 - .02 Y
S	s .73 z .11 - .07 S .07 Z .01 C
T	t .80 S .08 T .04 - .04 C .02 D .01 Z
U	- .31 ^ .21 Y .18 x .14 u .07 w .05 U .03 y I E o A R O
V	v 1.00 f -
W	w .53 - .38 * .09
X	X .84 # .10 K .04 z .03
Y	i .65 - .11 A .11 I .08 y .03 x .02 Y
Z	z .87 - .08 s .03 ! .02 Z
_	

Statistic on the letter-phoneme mapping
The numbers indicate the relative frequency of a mapping. No numbers are given if the relative frequency is below 1 %.
Example: In 97% of all cases, the letter **Q** is mapped to the phoneme k and in 3% of the cases **Q** is mapped to the phoneme -.

The picture below is a graphical representation of the letter to phonemes mapping. A vertical bar at position (L, p_i) indicates that there is a mapping from letter L to phoneme p_i. The likelyhood (frequency) of a particular mapping is expressed by the thickness of the vertical bars. The fat bars in the diagonal show that most of the letters have a "preferred" phoneme. It is also interesting that all letters except '**X**' can be silent ('**-**').

- picture -

Letter	Stress to which a letter is mapped
-	
.	
A	1 .43 0 .34 2 .14 < .07 >
B	> .81 < .19
C	> .61 < .39 0
D	< .52 > .48
E	0 .36 < .29 1 .26 2 .07 > .02
F	> .77 < .23
G	> .54 < .46 0
H	> .72 < .28 0
I	0 .54 1 .29 2 .11 < .05 >
J	> .94 < .06 0
K	< .62 > .38
L	> .55 < .45 0
M	> .55 < .41 0 .03
N	< .81 > .19 0
O	1 .40 0 .31 < .14 2 .14 >
P	> .75 < .25
Q	> .80 < .20
R	< .52 > .48 0
S	< .52 > .48 1 2
T	< .53 > .47 0 2 1
U	1 .41 0 .25 < .21 2 .10 > .03
V	> .62 < .38
W	> .71 < .26 1 .02 0 .01 2
X	< .97 > .03
Y	0 .71 < .10 1 .10 2 .06 > .03
Z	< .55 > .45
_	

Statistic on the letter-stress mapping

8.7 Frequently Appearing Letter Blocks

no	2	%	3	%	4	%	5	%
1	ER	2.06	ION	0.99	TION	1.02	ATION	0.96
2	ON	1.81	TIO	0.85	ATIO	0.75	ILITY	0.16
3	AT	1.81	ATI	0.83	ABLE	0.28	INTER	0.14
4	TI	1.79	ATE	0.66	LITY	0.21	RATIO	0.14
5	IN	1.76	ENT	0.57	MENT	0.20	TATIO	0.14
6	TE	1.58	TER	0.52	TIVE	0.17	CATIO	0.14
7	AN	1.41	OUS	0.40	RATI	0.17	BILIT	0.13
8	RE	1.41	BLE	0.40	RATE	0.16	ATIVE	0.12
9	LE	1.36	ITY	0.40	IOUS	0.16	CTION	0.11
10	EN	1.35	CON	0.37	INTE	0.15	NATIO	0.11
11	AL	1.31	TIC	0.34	NTER	0.15	ITION	0.11
12	AR	1.27	ING	0.34	STIC	0.14	UNDER	0.11
13	RA	1.26	RAT	0.34	ATOR	0.14	ISTIC	0.10
14	ST	1.20	ANT	0.32	ILIT	0.14	ICATI	0.10
15	NT	1.11	IST	0.31	ANCE	0.14	ALITY	0.10
16	OR	1.10	TRA	0.28	NDER	0.14	IZATI	0.09
17	RI	1.08	TOR	0.28	STER	0.14	ZATIO	0.09
18	IC	1.08	INE	0.28	ICAT	0.13	LATIO	0.09
19	IT	1.06	PER	0.28	TATI	0.13	TABLE	0.09
20	IO	1.05	VER	0.28	ICAL	0.12	GRAPH	0.08
21	LI	1.01	NCE	0.27	BILI	0.12	ABILI	0.08
22	IS	0.97	ICA	0.26	ENCE	0.12	TRANS	0.08
23	CO	0.93	LIT	0.25	OLOG	0.12	ERATE	0.08
24	RO	0.85	ABL	0.25	CATI	0.11	ATORY	0.08
25	DE	0.83	PRO	0.25	SION	0.11	OLOGY	0.08
26	LA	0.83	STI	0.25	OVER	0.11	IFICA	0.07
27	CA	0.79	IVE	0.25	LATE	0.11	MENTA	0.07
28	TA	0.77	ALI	0.24	ENTI	0.10	RABLE	0.07
29	NE	0.72	STR	0.23	THER	0.10	FICAT	0.07
30	OU	0.72	INT	0.23	STRA	0.10	OGRAP	0.07
31	VE	0.71	PRE	0.22	CONT	0.10	OLOGI	0.06
32	MA	0.70	NTE	0.22	LATI	0.10	ALIZE	0.06
33	ES	0.69	EST	0.22	TORY	0.10	INATE	0.06
34	TR	0.69	MEN	0.22	ERAT	0.10	MINAT	0.06
35	US	0.66	RES	0.22	IGHT	0.10	RIOUS	0.06
36	SE	0.66	LAT	0.22	MATI	0.10	STRAT	0.06
37	CE	0.63	TRI	0.21	COMP	0.10	CIOUS	0.05
38	IL	0.62	CAT	0.21	ATIV	0.10	ICATE	0.05
39	ET	0.62	ESS	0.21	ISTI	0.09	MATIC	0.05
40	DI	0.60	MAN	0.21	CONS	0.09	SSION	0.05

The 40 most frequently appearing 2-, 3-, 4- and 5-letter blocks in the NETtalk dictionary

8.8 An Example of a Detailed Evaluation Statistic

The listing below is the full amount of information collected with every evaluation.

```
> (print-eval-data "data/c.test.stat.12" :full t)

############################################################

CLASSIFICATION - RESULTS:

Data from file: "~haya/expr/3/corpus.test"
DT   from file: "data/corpus.train.id.1"


COMMENTS:


Evaluation Data form file: ~haya/expr/3/corpus.test
ID3 Tree from file       : data/corpus.train.id.1
Data saved to file       : data/c.test.stat.12
EVALUATION: ID3
WITH closest P/S + MOST FREQ
Most-Freq-Phon from: inconsist/corpus.train.letmap
ID3  NO CHI-SQUARE


WORDS TOTAL:
0      : 18778    ( 98.8)
1      : 67       (  0.4)
2      : 158      (  0.8)
sum    : 19003    (100.0)


CORRECTLY CLASSIFIED WORDS:

0      : 2256     ( 11.9)
1      : 0        (  0.0)
2      : 2        (  0.0)
sum    : 2258     ( 11.9)


LETTERS TOTAL:
0      : 139738   ( 98.8)
1      : 526      (  0.4)
2      : 1128     (  0.8)
sum    : 141392   (100.0)


CORRECTLY CLASSIFIED LETTERS:

0      : 92650    ( 65.5)
1      : 268      (  0.2)
2      : 518      (  0.4)
sum    : 93436    ( 66.1)


CORRECTLY CLASSIFIED BITS:

0      : 3806954  ( 96.2)
1      : 0        (  0.0)
2      : 0        (  0.0)
sum    : 3806954  ( 96.2)


DECISION-TREES:
```

```
nodes    165.3
depth     23.1
```

Correctly classified PHONEME BITS:

	all	1	2	3	4	5	6	7	8	9
sum	3007432	139248	135128	131555	138356	134054	127578	137126	140316	140773
%	96.7	98.5	95.6	93.0	97.9	94.8	90.2	97.0	99.2	99.6
nodes	158.5	97	248	286	102	219	423	186	33	30
depth	24.3	41	36	30	22	35	35	17	10	10

	10	11	12	13	14	15	16	17	18	19
sum	140701	141294	141355	139521	130705	138796	134142	134732	128242	135513
%	99.5	99.9	100.0	98.7	92.4	98.2	94.9	95.3	90.7	95.8
nodes	36	16	5	53	301	125	241	188	394	251
depth	10	7	3	12	34	44	33	34	57	32

	20	21	22
sum	141392	141392	135513
%	100.0	100.0	95.8
nodes	1	1	251
depth	0	0	32

Correctly classified STRESS BITS:

	all	1	2	3	4	5	6
sum	799522	141392	128357	129707	126745	131929	141392
%	94.2	100.0	90.8	91.7	89.6	93.3	100.0
nodes	190.2	1	304	256	361	218	1
depth	19.0	0	24	35	34	21	0

PHON BIT is 0 and classified 0:

	all	1	2	3	4	5	6	7	8	9
'0'	2682135	109286	103339	117936	136465	105747	109123	126743	138577	127063
sum	2726416	110102	106103	122200	137572	109858	114824	128722	139131	127306

```
%         98.4   99.3   97.4   96.5   99.2   96.3   95.0   98.5   99.6   99.8

            10     11     12     13     14     15     16     17     18     19
-------------------------------------------------------------------------------
'0'     139018 123911 127564 113924 121544  95038 124241 130909 112179 118372
sum     139290 123944 127581 114577 125353  96177 127760 133479 118233 120710
%         99.8  100.0  100.0   99.4   97.0   98.8   97.2   98.1   94.9   98.1

            20     21     22
----------------------------
'0'     141392 141392 118372
sum     141392 141392 120710
%        100.0  100.0   98.1



PHON BIT is 1 and classified 1:

           all      1      2      3      4      5      6      7      8      9
-------------------------------------------------------------------------------
'1'     325297  29962  31789  13619   1891  28307  18455  10383   1739  13710
sum     384208  31290  35289  19192   3820  31534  26568  12670   2261  14086
%         84.7   95.8   90.1   71.0   49.5   89.8   69.5   81.9   76.9   97.3

            10     11     12     13     14     15     16     17     18     19
-------------------------------------------------------------------------------
'1'       1683  17383  13791  25597   9161  43758   9901   3823  16063  17141
sum       2102  17448  13811  26815  16039  45215  13632   7913  23159  20682
%         80.1   99.6   99.9   95.5   57.1   96.8   72.6   48.3   69.4   82.9

            20     21     22
----------------------------
'1'          0      0  17141
sum          0      0  20682
%        100.0  100.0   82.9



STRESS BITS is 0 and classified 0:

           all      1      2      3      4      5      6
-------------------------------------------------------
'0'     661951 141392  85024  93131 108880  92132 141392
sum     687572 141392  92762  97702 115633  98691 141392
%         96.3  100.0   91.7   95.3   94.2   93.4  100.0


STRESS BITS is 1 and classified 1:

           all      1      2      3      4      5      6
-------------------------------------------------------
'1'     137571      0  43333  36576  17865  39797      0
sum     160780      0  48630  43690  25759  42701      0
%         85.6  100.0   89.1   83.7   69.4   93.2  100.0



Correctly classified PHONEMES:

Phon       all      !      #      *      +      -      .      @      A      C
-------------------------------------------------------------------------------
```

sum	110806	0	17	87	0	17141	0	1566	447	526
all	141392	10	41	97	21	20682	0	3818	1750	678
%	78.4	0.0	41.5	89.7	0.0	82.9	100.0	41.0	25.5	77.6

Phon	D	E	G	I	J	K	L	M	N	O
sum	86	1945	557	4990	753	0	1240	19	105	44
all	114	3370	677	6798	1136	18	1614	141	341	163
%	75.4	57.7	82.3	73.4	66.3	0.0	76.8	13.5	30.8	27.0

Phon	Q	R	S	T	U	W	X	Y	Z	^
sum	0	2373	1285	302	60	255	188	488	15	796
all	0	2740	1607	434	282	423	378	1067	125	1252
%	100.0	86.6	80.0	69.6	21.3	60.3	49.7	45.7	12.0	63.6

Phon	_	a	b	c	d	e	f	g	h	i
sum	0	823	2790	562	3836	1764	2117	1031	653	2536
all	0	2746	2817	1149	3861	2619	2130	1569	863	4314
%	100.0	30.0	99.0	48.9	99.4	67.4	99.4	65.7	75.7	58.8

Phon	k	l	m	n	o	p	r	s	t	u
sum	5476	5351	4210	8243	934	4029	6776	5963	8408	249
all	5771	5491	4214	8438	2209	4039	7603	6604	8758	748
%	94.9	97.5	99.9	97.7	42.3	99.8	89.1	90.3	96.0	33.3

Phon	v	w	x	y	z
sum	1745	834	6176	95	920
all	1747	931	11476	211	1307
%	99.9	89.6	53.8	45.0	70.4

Correctly classified STRESS:

str.	all	-	0	1	2	<	>
sum	110346	0	16146	13408	883	43333	36576
all	141392	0	23313	19388	6371	48630	43690
%	78.0	100.0	69.3	69.2	13.9	89.1	83.7

Correctly classified LETTERS:

char	all	-	.	A	B	C	D	E	F	G
sum	93436	0	0	5007	2404	5184	3410	8824	1754	1663

all	141392	0	0	12711	2948	6641	4115	15463	1949	3028
%	66.1	100.0	100.0	39.4	81.5	78.1	82.9	57.1	90.0	54.9

char	H	I	J	K	L	M	N	O	P	Q
sum	2557	5864	244	818	6118	3773	8076	3667	3945	265
all	3263	12492	270	1094	7810	4528	9629	9700	4575	334
%	78.4	46.9	90.4	74.8	78.3	83.3	83.9	37.8	86.2	79.3

char	R	S	T	U	V	W	X	Y	Z	_
sum	8515	5636	8695	2271	1299	969	198	1914	366	0
all	10606	7492	10839	5431	1748	1173	451	2592	510	0
%	80.3	75.2	80.2	41.8	74.3	82.6	43.9	73.8	71.8	100.0

PHONEMES correct per LETTERS:

char	all	-	.	A	B	C	D	E	F	G
sum	110806	0	0	6882	2792	5888	3927	10695	1946	1916
all	141392	0	0	12711	2948	6641	4115	15463	1949	3028
%	78.4	100.0	100.0	54.1	94.7	88.7	95.4	69.2	99.8	63.3

char	H	I	J	K	L	M	N	O	P	Q
sum	2941	7811	263	1051	7265	4374	8956	5182	4469	325
all	3263	12492	270	1094	7810	4528	9629	9700	4575	334
%	90.1	62.5	97.4	96.1	93.0	96.6	93.0	53.4	97.7	97.3

char	R	S	T	U	V	W	X	Y	Z	_
sum	9214	6370	10135	2964	1745	1046	205	2017	427	0
all	10606	7492	10839	5431	1748	1173	451	2592	510	0
%	86.9	85.0	93.5	54.6	99.8	89.2	45.5	77.8	83.7	100.0

Letters which mapped directly into a legal but wrong phoneme:
14505 (47.42 %)

Letters which mapped directly into a legal but wrong stress:
24808 (79.91 %)

8.9 Binary Representation of Phonemes and Stresses

The following LISP code shows the binary encoding of phonemes and stresses.

```
  ;;; PHONEMES: The different bits represent the
  ;;  following properties of a phoneme.

  ;;; Alveolar = Central1  1
  ;;; Dental = Front2 2
  ;;; Glottal = Back2 3
  ;;; Labial = Front1 4
  ;;; Palatal = Central2 5
  ;;; Velar = Back1 6

  ;;; Affricative 7
  ;;; Fricative 8
  ;;; Glide 9
  ;;; Liquid 10
  ;;; Nasal 11
  ;;; Stop 12
  ;;; Tensed 13
  ;;; Voiced 14

  ;;; High 15
  ;;; Low 16
  ;;; Medium 17

  ;;; Elide 18
  ;;; FullStop 19
  ;;; Pause 20
  ;;; Silent 21

(setq *PHONEME-BOOLEAN-LIST*

        ;;; 01234567890123456789012
 '((#\a  #22*00000100000000100100000)
   (#\b  #22*00001000000001010000000)
   (#\c  #22*00000010000000000010000)
   (#\d  #22*01000000000001010000000)
   (#\e  #22*00100000000000100010000)
   (#\f  #22*10001000100000000000000)
   (#\g  #22*00000010000001010000000)
   (#\h  #22*10010000010000000000000)
   (#\i  #22*00001000000000101000000)
   (#\k  #22*10000010000001000000000)
   (#\l  #22*00100000000100010000000)
         ;;; 01234567890123456789012
   (#\m  #22*00001000000010010000000)
   (#\n  #22*01000000000010010000000)
   (#\o  #22*00010000000000100010000)
   (#\p  #22*10001000000001000000000)
   (#\r  #22*00000100001000100000000)
   (#\s  #22*11000000100000000000000)
```

```
   (#\t  #22*11000000000001000000000)
   (#\u  #22*00010000000000101000000)
   (#\v  #22*00001000100000010000000)
   (#\w  #22*00001000010000010000000)
   (#\x  #22*00000100000000000010000)
   (#\y  #22*00000100010000010000000)
   (#\z  #22*01000000100000010000000)
         ;;; 0123456789012345678901 2
   (#\A  #22*01100000000000100010000)
   (#\C  #22*10000101000000000000000)
   (#\D  #22*00100000100000010000000)
   (#\E  #22*00101000000000000010000)
   (#\G  #22*00000010000100100000000)
   (#\I  #22*00001000000000001000000)
   (#\J  #22*00000101000000010000000)   ; corrected 'J', Sejnowski's is wrong !
   (#\K  #22*10000111100000000000000)
   (#\L  #22*01000000001000100000000)
   (#\M  #22*00100000000100100000000)
   (#\N  #22*00000100000100100000000)
   (#\O  #22*01000100000001000100000)
   (#\Q  #22*00001011000010100000000)
   (#\R  #22*00000010001000100000000)
   (#\S  #22*10000100100000000000000)
   (#\T  #22*10100000100000000000000)
         ;;; 0123456789012345678901 2
   (#\U  #22*00000010000000001000000)
   (#\W  #22*00000110000001010100000)
   (#\X  #22*11100001000000000000000)
   (#\Y  #22*01101000000001010000000)
   (#\Z  #22*00000100100000100000000)
   (#\@  #22*00100000000000000100000)
   (#\!  #22*10101001000000000000000)
   (#\#  #22*00000111000000100000000)
   (#\*  #22*01001000010000100100000)
   (#\^  #22*01000000000000000100000)
   (#\+  #22*00000000000000000000000)
   (#\-  #22*00000000000000000001001)
   (#\_  #22*00000000000000000001010)
   (#\.  #22*00000000000000000000110)
))


(setq *STRESS-BOOLEAN-LIST*

 '((#\<  #6*010000)
   (#\>  #6*001000)
   (#\1  #6*000110)
   (#\2  #6*000100)
   (#\0  #6*000010)
   (#\-  #6*011001)))
```

References

[**Bakiri89**] Personal Communication with Ghulum Bakiri, Spring 1989

[**Dietterich89**] Personal Communication with T.G.Dietterich, Spring 1989

[**Kibler87**] Kibler, D. and David, W.A. (1987), *Learning Representative Exemplars of Concepts: An Initial Case Study*, Proceeding of the Fourth International Workshop on Machine Learning (24-30), Irvine, CA. Los Altos, CA: Morgan-Kaufmann

[**Kuchera67**] Kuchera, H. and Francis, W.N. (1967), *Computational Analyses of Modern-Day American English* , (Providence, Rhode-Island: Brown University Press, 1967)

[**Mooney88**] Mooney, R. , Shavlik, J. , Towell, G. , Gove, A. (1988), *An Experimental Comparison of Symbolic and Connectionist Learning Algorithms*, submitted (12/88) to the Eleventh International Joint Conference on Artificial Intelligence.

[**Sejnowski87**] Sejnowski, T.J. and Rosenberg, C.R. (1987), *Parallel Networks that Learn to Pronounce English Text*, Complex Systems Publications 1 (1987) 145-168

[**Stanfill86**] Stanfill, C. and Waltz, D. (1986), *Toward Memory-based Reasoning*, Communications of the ACM, Dec 1986, Vol 29, Number 12, 1213-1228

[**Quinlan86**] Quinlan, J.R. (1986), *Induction of Decision Trees*, Machine Learning, 1(1), 82-106

[**Quinlan87**] Quinlan, J.R. (1987), *Generating Production Rules From Decision Trees*, Proceedings of IJCAI-87, Los Altos: Morgan-Kaufman. 304-307.