Speech Technology Project: Comparing models of L2 vowel perception using simulation and visualization

Universiteit van Amsterdam

Klara Weiand, 0529478
Jelle Kastelein, 0026549

13th September 2006

Contents

1 Introduction

Second language acquisition (L2A) is a field of linguistic research in which learning plays a vital role (indeed, it *is* learning). The sensible step thus is to look into other fields of research that have experience in dealing with similar types of learning problems. With this in mind, we present here a step towards a combined effort on this subject, between the sciences of Phonetics and Artificial Intelligence.

In this paper, we will investigate the workings of several supervised machine learning algorithms from a phonetic point of view. We will show the value of using such techniques to model the dynamics of the vowel space during L2A tasks, and provide some initial insights into how a model of an unexperienced subject can be built that bases categorizations in a non-native language on a mapping from the configuration of the native vowel space to the non-native one. Furthermore, we will compare several different learning algorithms with respect to an empirical experiment carried out on human subjects[1].

We will start by giving the necessary background information on the empirical experiment that is to be modelled. We will then proceed by introducing various visualization techniques that will be helpful for data analysis. Finally, the larger part of the article will be devoted to the learning algorithms, the simulations, and their results.

2 Experiments and previous work

The empirical data used here was gathered in an experiment where native Dutch speakers and learners of Dutch with Spanish or Portuguese as their native language were confronted with 339 synthetic vowels that differed in their F1 and F2 frequencies and durations. There were 14 different F1 values, 10 different F2 values and three different durations, 0.1, 0.14142 and 0.2 seconds.

Participants were asked to label the vowels according to their L1 orthographic labels. The non-native Dutch participants additionally classified the vowels using Dutch orthographic labels in a separate experiment. A final set of experiments required participants to use both L1 and L2 labels for categorization.

The data were summarized in tabled and the most frequent responses were plotted with regards to their position in the vowel space. Some preliminary results were found using these techniques, e.g. differences in Dutch vowel boundaries between native Dutch speakers and beginning learners of Dutch, that showed an influence from the native language and decreased with increasing proficiency. An ANOVA run on a dataset for Spanish subjects categorizing stimuli with Dutch vowels, over 4 levels of increasing proficiency, revealed this effect to be highly significant, $F(3) = 78.804$, $p < .0001$.

Another observed effect from the participant's first language is orthographic in nature, and can be attributed to the fact that participants used Dutch orthographic representations to label the vowels, but employed grapheme-phoneme correspondences from their native language.

2.1 Vowels

The data uses a total of 26 vowel labels for 14 different vowels. In what follows, the vowels will be referred to with labels that are not necessarily identical to the orthographic representation or their IPA symbols.
In the experiment, there were 12 Dutch, 7 Portuguese and 5 Spanish vowels which, together with their corresponding IPA symbols, are displayed in table 1. Additionally, in the bilingual data from the experiment where participants could chose between vowels from their native language and Dutch, the Spanish and Portuguese vowels were prefixed with "S" and "P" respectively in order to represent the participant's choice of language. The 26 vowel labels used thus are: "A", "a", "E", "e", "I", "i", "O", "o", "u", "Y", "y", "2", "Sa", "Se", "Si", "So", "Su", "Pa", "Pct", "Pe", "Pef", "Pi", "Po", "Pu", "ct", "ef".
The prefixing of vowels to indicate the language the participant associated with the stimuli opens an important question in modeling the results that will be discussed in section 5.2.1.

[1]For the remainder of this article, we will use the term 'empirical data' to refer to the data derived from human subjects, and the term 'simulated data' to refer to the data derived from the models produced by the learning algorithms.

Language	Vowels											
Dutch	“A”	“a”	“E”	“e”	“I”	“i”	“O”	“o”	“u”	“Y”	“y”	“2”
	ɑ	aː	ɛ	eː	ɪ	i	ɔ	oː	u	ʏ	y	øː
Spanish	“a”	“e”	“i”	“o”	“u”							
	a	eː	i	oː	u							
Portuguese	“a”	“ct”	“e”	“ef”	“i”	“o”	“u”					
	a	ɔ	eː	ɛ	i	oː	u					

Table 1: Vowel representations used for the different languages.

3 Visualizations

In order to increase visual clarity for the L2A data, both empirical and simulated, several visualization techniques were implemented and tested using the python PyX plotting package[2]. We will discuss the workings and advantages of each of the styles below.

One of the challenges of creating the visualizations is that we are faced with multiple vowel guesses per point in the vowel space (one guess for each participant). Furthermore, we may wish to overlay the 3 different durations in one 2D plot, or plot a simulation together with a corresponding empirical experiment for ease of comparison. The techniques presented here are specifically targeted to visualizing this multiplicity of data.

3.1 Symbolic Plot

The first technique used was to plot each of the symbols found at a particular point in the vowel space. Since we use a discretized vowel space for plotting, and since we have multiple vowel guesses per point, we give each symbol a small random offset in the horizontal and vertical planes, to ensure that they do not overlap. The size of each symbol is controlled by its frequency in the data. Categories with a very small relative frequency (10% in the experiments below) are regarded as noise, and are not drawn.

To make the different regions occupied by distinct vowels more easily distinguishable, the vowels are each supplied with their own color. Alternatively, we can choose to output the guesses in a different color for each language, which is useful in visualizations for bilingual experiments, where a subject can provide guesses with both Dutch vowels, or those of her native language. Examples of these types of plots are shown in figures 1 and 2. The first uses the vowel labels found in the data, whereas the second plots the data in abstract symbols, which show fewer overlap (but lack the informational content of having the corresponding vowel label present).

[2]http://pyx.sourceforge.net/

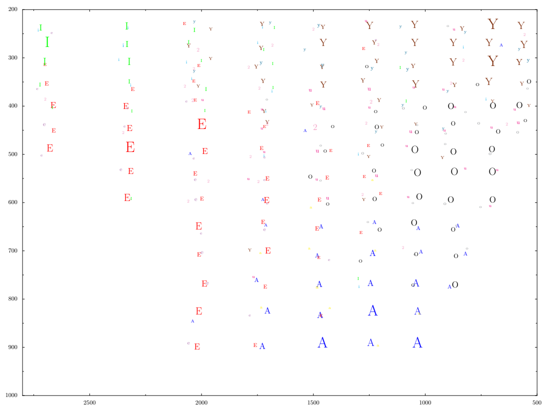

Figure 1: Human detail symbolic text plot for Brazilian subjects categorizing stimuli with Dutch vowels, colored by vowel. Symbol sizes represent frequency (All vowels shown, for Duration = 1).

3.2 Bordered Plot

Because the vowel boundaries are important for the purpose of this experiment, we may wish to indicate them explicitly. To this purpose, we check for each vowel, for each point in the step-wise vowel space, whether that point contains points with similar vowel categorizations directly to its left, right, top and bottom. If no similar vowel is found, a bor-

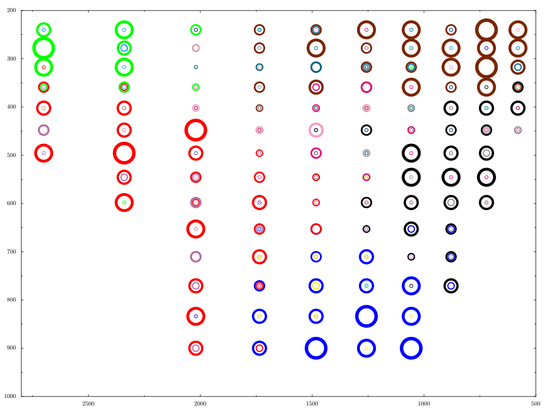

Figure 2: Human detail symbolic circle plot for Brazilian subjects categorizing stimuli with Portuguese vowels, colored by vowel. Circle sizes represent frequency (All vowels shown, for Duration = 1).

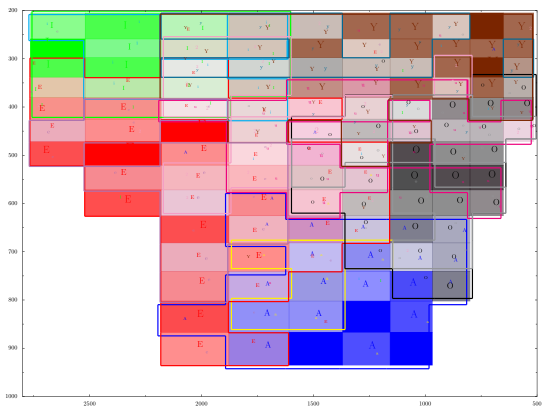

Figure 3: Human detail plot for Brazilian subjects categorizing stimuli with Portuguese vowels (All vowels shown, for Duration = 1).

der is drawn between them[3]. Regions that occupy less than two blocks are ignored in order to avoid a cluttered image.

In addition to the borders, the area inside each block can be filled and colored with respect to their assigned vowel categorizations. By defining the transparency for the coloring of a block as 1.0 (maximum transparency), minus the square of its relative frequency, each vowel in a block contributes to its region's color with a proportionality relative to its frequency at that block's midpoint. Thus, the more intense a color is, the less variance there is among subjects in assigning its category.

Finally, we can overlay the symbol plots on top of the border plots, to provide additional information. A typical example of the combination of filled region plots with symbols overlayed is shown in figure 3.

3.3 Ellipsis Plot

A technique commonly used for plotting vowel space regions, is to calculate the average and standard deviation for the F1 and F2 values separately for each vowel, and then plotting an ellipsis with the average as its centroid, and the standard deviation in either direction as its radii[4]. A typical example of an ellipsis plot is figure 4.

A huge advantage of the ellipsis plots is the clarity of such a plot; it is much easier to inspect than either of the former techniques, and one can plot much more data without losing the most important visual cues in cluttered plots. However, by plotting only centroids and standard deviations, we lose a great deal of information. Therefore, as we will see below, it is useful to make visual comparisons on both the ellipsis and region + symbol plots.

4 Simulations

This next section will concentrate on simulating vowel regions within the vowel space and their dynamics during second language acquisition. For those who have little experience with the various algorithms used, we will give a brief description below.

4.1 Classifiers

Very broadly speaking, in the artificial intelligence field of machine learning, an (admittedly rough) division can be made between supervised and unsupervised learning algorithms. In supervised learn-

[3]The blocks are defined by the midpoints between F1 and F2 steps. Since borders may sometimes overlap, they receive a tiny random offset in either plane.

[4]Often, the deviations are calculated on the principal components of the F1, F2 space, resulting in an ellipsis which is tilted in the direction of maximum covariance.

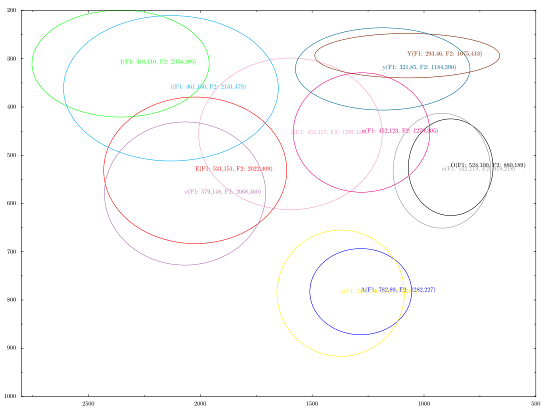

Figure 4: Human ellipsis plot for vowel classification regions for Brazilian subjects categorizing stimuli with Portuguese vowels (All vowels shown, for Duration = 2). Midpoint indicates average F1 and F2 values, size in F1 and F2 directions indicates standard deviation.

ing[5], the class labels (i.e., the vowel category for a stimulus) are provided together with the data (i.e., the F1, F2 and duration values) during training. The algorithm then makes a *generalization* about how the data and classes interact with one another – that is, how to select the best class label for a new instance that we wish to classify, given the available information. Since the class labels are available during training, the learning procedure is generally easier, and the outcome of the algorithms is relatively easy to evaluate. We know what the ideal classification would be, and are able to test that against the observed classifier output. An unfortunate consequence is that, in general, a supervised classification algorithm can only output the classes which it has observed. This is relevant for this experiment. If we train on Spanish data, containing five vowels, and then test it on Dutch data (which contains the exact same F1, F2 and duration values, but a different set of vowels), the output can only consist of Spanish vowel classifications, and so there will be no correct classifications whatsoever on any of the instances labeled with Dutch orthographic labels that are not in the set of Spanish labels.

In contrast, unsupervised learning algorithms usually make use of some form of clustering. They find groups within the available data by determining some measure of similarity between them, on the basis of their attribute values. In this case, we have no predetermined class labels, and thus such a method is quite hard to evaluate.

Finally, there are several algorithms which combine these two approaches in a semi-supervised setting. Usually, they start with a small set of labeled instances, as in the supervised setting, and then bootstrap on these instances to identify clusters.

We concentrate here on three instances of the supervised case[6], for two reasons. First, as mentioned earlier, the output is easier to evaluate. Second, in general, classification is an easier task than clustering, and so we can generally expect the classification results to be superior to those that we find in an unsupervised setting, so that this may give a better indication of the viability of using further AI techniques to model L2 acquisition. Finally, L2A is usually seen as a supervized learning task in which the learner obtains feedback from his environment and adjusts his actions accordingly.

First, we will discuss an instance of a memory-based learning algorithm, called the Nearest Neighbor (NN) classifier, which memorizes all of its training data and then classifies new instances according to that memory. We will then proceed by describing the Naive Bayes (NB) classifier, a probabilistic algorithm that stores only the frequency distributions of attributes over classes. A good introduction to both algorithms is given in [8]. We will close with the Gradual Learning Algorithm (GLA) used in Stochastic Optimality Theory (OT), which makes almost no use of memory, but simply stores the relative importance of a set of *constraints*. Note that there is a trend here from making full use of memory without any form of abstraction, to storing only very abstract representations.

4.1.1 Nearest Neighbor

One standing theory on language learning is the Memory Based Learning (MBL) paradigm. Basi-

[5]When referring to supervised learning in this paper, we will be refering to classification, rather than regression and value prediction tasks.

[6]Optimality Theory is a special case described here, and falls outside the usual range of machine learning techniques, as it is very specifically designed and optimized for language learning. Although Stochastic OT is not necessarily fully supervised, we concentrate here on the supervised part of the algorithm.

cally, this theory views learning as being instance-based. That is, rather than creating abstract categories based on examples, we store these examples explicitly, and compare them to any new instances that are to be categorized.

There is a particular range of machine learning algorithm which is analogous to this idea, referred to as instance-based (or case-based) learning methods. The standard example of such an algorithm, which corresponds directly to (and is based on) MBL theory, is the Nearest Neighbor classification algorithm. This is a so-called lazy learner, meaning that it does not require complicated learning procedures, but rather performs the necessary calculations at time of classification of new instances[7]. The basic idea underlying this algorithm is very simple.

First, in order to make use of such methods, we should be able to express the difference between two instances in terms of a numeric value. This means that each instance must be expressible as occupying a location in a highly dimensional Euclidean space. The intuition is that two instances which are very far apart in this space will be highly dissimilar, while more similar instances will be grouped closer together.

To learn, one simply stores each the examples observed by the learner with the corresponding label. This then gives us an instance space populated by the observations that the learner has stored up until the current point. To classify a new instance, one calculates the Euclidean distance to each example in the instance space, and looks at the closest neighboring point. The label of this point is then assigned to the new instance.

To illustrate, let us suppose that we have the three instances in table 2. If we are faced with a new instance, with values 300, 1600 and 1 for F1, F2 and duration respectively, and we want to assign the most likely label to this instance, we would proceed in the following manner.

First, we may normalize the coordinates for the examples on each axis, by:

[7]Several optimizations can be made however, in which decision rules or classification regions can be found based on the memory space of the lazy learner. A thorough analysis of such memory efficient MBL techniques is given in [7]. As most of these techniques should have minimal impact on classification performance, we will not discuss them further.

Vowel	F1	F2	Duration
E	200	2200	1
O	450	800	2
A	900	1000	2

Table 2: A vowel classification toy problem.

Vowel	F1	F2	Duration
E	0.0	1.0	0.0
O	0.3571	0.0	1.0
A	1.0	0.1429	1.0
?	0.1428	0.5714	0.0

Table 3: Vowel classification normalized distances.

$$a_i(x_j) = \frac{v_i(x_j) - min(v_i)}{max(v_i) - min(v_i)}$$

This gives us the normalized instance space in table 3 (with the new instance indicated by a "?"). If we do not normalize in this way, the dimension along which the instances are spread within the largest range would implicitly be given an larger weight.

We then proceed by calculating the distance to each of the neighboring points, by:

$$d(?, x_j) = \sqrt{\sum_{r=1}^{n}(a_r(x_j) - a_r(x_j))^2}$$

which gives us table 4. Since the point with label E is the closest neighbor, we assign the label E to the new instance.

Note that the NN algorithm performs a form of "learning by smoothing", in which it produces piecewise linear decision boundaries. The space between each of the instances is automatically smoothed over to accommodate for unseen instances. This is a significant problem for an algorithm like the Naive Bayes classifier, which needs to make assumptions on the prior distribution underlying the data before it can classify on the basis

Vowel	Distance
E	**0.4518**
O	1.4586
A	1.6537

Table 4: Vowel classification toy problem distances.

of unseen values for continuous attributes.

There are many variations on this basic idea. A first addition is to include the votes of multiple neighbors. The label with the highest number of votes is then assigned to the instance at hand. This variant is often called k-Nearest Neighbors, where k stands for the number of points to be considered in the voting. A natural extension of this approach is to include all points in the training set. In this case, note that the vote that each point casts for a certain vowel must be weighted by some measure based on its distance to the new instance, otherwise the label with the most points in the training set always wins, regardless of the distance of the new instance to the center of gravity of the region that this vowel occupies. Usually, this weighting measure is taken to be the square of the distance.

Nearest Neighbor methods have several properties which may be relevant to the task at hand. First of all, the NN algorithm in its simplest form is very sensitive to noise and outliers in the training data. The vowel space categorization produced by human beings typically contains a lot of noise, which is why we used a weighted-k-neighbors NN scheme, which is more robust to such noise.

Second, the algorithm in itself cannot output categorizations in a probabilistic fashion, and so the same label will be assigned to the same points every time. This is a severe shortcoming, since one of the traits that we have to account for in modeling human performance is the abundance of overlapping vowel regions. To accommodate for this, we also performed an experiment where we used the relative number of votes (weighted by their normalized squared distance to the new instance) as a probability distribution over possible vowel labels, and chose a label according to this distribution. Note that we can do this, since these weighted frequencies sum to 1.0, but that it is a very rough measure.

4.1.2 Naive Bayes

Another widely used learning algorithm, which is based on the Bayesian approach in probability theory, is the Naive Bayes classifier. This method has been very successfully used in a wide range of classification problems, and has been the topic of much research.

The Classifier attempts to maximize the likelihood of the (training) data, given the model, such that, given attribute values $\langle a_1, a_2, ..., a_n \rangle$, the class c_{NB} assigned by the NB classifier is

$$c_{NB} = argmax_{c_j \in C} P(c_j) \prod_i P(a_i|cj)$$

with $P(c_j)$ the a priori probability that the instance belongs to class c_j and $P(a_i|c_j)$ the conditional probability of observing attribute value a_i in an instance of class c_j.

Basically, Naive Bayes classifiers do relative frequency estimation under an assumption of independence between variables. The independence assumption, although not always justified, makes the computation feasible and fast. In our case, we will assume that F1, F2 and duration values are independent (even though they may be correlated).

During training, the classifier is again presented with labeled training instances. The actual learning consists of estimating the parameters $P(c_j)$ and $P(a_i|c_j)$.

One shortcoming of NB classifiers is that they are not primarily aimed at dealing with non-discrete attributes. Possible solutions include (but are not limited to) either discretizing the training and test data (if the range of possible values is limited, and known in advance), or estimating the mean and variance parameters of an attribute given each class, and using these as parameters in a Gaussian distribution. Note that in the latter case, the algorithm must make an assumption about the distribution that generated the data. Neither solution is really satisfying, but since the training data was already discretized in the form of step wise duration, F1 and F2 values, we simply used these values.

Obviously, a statistical method like this one lends itself quite naturally to produce probabilistic output. We therefore based our classification of new instances not just on the most likely class label, but on the probability distribution over all vowels.

4.1.3 Stochastic OT and GLA

The Gradual Learning Algorithm (GLA) [1] is an error-driven learning algorithm used with Stochastic Optimality Theory grammars that has successfully been used in a number of comparable experiments [6][5][2].
Stochastic OT is an extension to Optimality Theory [9], a linguistic model originally applied to phonology but also used in many other areas of lin-

guistics. The main idea of OT is that the *optimal* candidate is selected from a set of generated candidates by means of a hierarchy of soft constraints. The optimal candidate is the one that incurs the least serious violations of the constraints. Since the constraints are ranked, "least serious" means that the optimal candidate is the one that has the least violations for the highest ranked constraint where the candidates differ in their number of violations. The ranking is absolute, so a candidate with a violation of a higher ranked constraint will always be discarded in favor of any other candidate that does not violate that constraint. A supervised learning of the best possible constraint ranking can then be implemented as an adaptation of the order of the constraints according to the known yields, that is, the output in the training data, of an input.

As an illustration, consider the example in figure 5. The top left cell specifies the input[8], that is, the stimulus, a vowel with an F1 step value of 1, an F2 value of 1 and a duration of 1.
The other cells in the first row list the –negatively worded [2]– constraints: The highest-ranked constraint says that an input with an F1 value of 1 should not be classified as an /i/, the second one says that if the F2 value is 1, the stimulus should not be classified as a /u/ and so on.
The candidates, /i/ and /u/, are represented in the second and third row of the first column. The remaining cells in these rows specify violations (marked with a "*") or non-violations for the corresponding constraints: /i/ violates the first constraint which says that the given input values should not yield /i/ as an output. Since /u/ does not violate this constraint, the violation is critical in determining the optimal candidate and is additionally marked with a "!". The remaining constraints and violations thus play no role in finding the optimal candidate and the cells are greyed and the optimal candidate, /u/, is indicated with a symbol (☞).

Stochastic OT differs from classical OT in that constraint rankings are not discrete and ordinal but rather arranged on a continuous scale, which means that the distance between constraints is not fixed, and can thus be learned from training data. Stochastic OT was chosen here, as it is not only based on OT which was conceived as a linguistic model, but also has been previously successfully used in similar experiments.

In addition to shifting as a result of training, the ranking value varies due to a noise component that is added during evaluation and that makes the selection of the optimal candidate non-deterministic. This is because the order of rankings relatively close (depending on the amount of noise) in ranking value can temporarily switch ranking orders as the result of the added jitter, yielding a different total ranking used for the computation of the optimal candidate. In the example above, /u/ always wins because the evaluation is the same every time: /i/ violates the highest-ranked constraint and /u/ does not, thus /u/ is the optimal candidate.
In Stochastic OT, the evaluation and the selection works as in classical OT in principle. However, the fact that there is a value specifying the distance between the constraints and that the noise provides a probabilistic component means that, depending on how close the ranking values of the first two constraints are and how high the noise value is, the two constraints can switch their order during evaluation, meaning that some of the time /i/ will be the winning candidate.

The gradual learning algorithm (GLA) takes a set of constraints and input-output pairs augmented with frequency information and consequently adjusts the ranking values of the constraints in such a way that the number of errors (i.e. cases where the generated optimal candidate for an input does not match the output) is minimized.

The amount by which a constraint's ranking value is adjusted is called plasticity. While a large plasticity value (i.e. the constraints are moved in big steps) makes for faster learning, the steps made may be too big, preventing convergence.

Boersma and Hayes [3] propose starting with a higher plasticity, which is then decreased during consequent learning, thus combining the advantages of high and low plasticity values, while at the same time being cognitively plausible with regard to findings on the speed of language acquisition in humans[9]. We adopted this strategy for our simu-

[8]The values are given as steps here and not in Hertz or seconds, due to space constraints.

[9]Neither of the algorithms as outlined above is capable of explicitly decreasing their rate of learning. Instead, the learning rate changes implicitly as a result of the data already learned. To see how this works, imagine feeding the

[F1=240Hz], [F2= 580Hz], [Dur=0.1s]	[F1=240Hz] is not /i/	[F2=580Hz] is not /u/	[F2=580Hz] is not /i/	[F1=240Hz] is not /u/	[Dur=0.1s] is not /u/	[Dur=0.1s] is not /i/
☞/u/		*		*	*	
/i/	*!		*			*

Figure 5: An example OT tableau.

lations.

The constraints we used were created according to the scheme of negatively worded constraints proposed by Escudero and Boersma [5] as used in the example above. That is, for each of the values of our three features, F1, F2 and length and each of the vowel labels, a constraint of the form "*value* is not *vowel*" was created, yielding a total of 702 constraints[10].
Each input pair then was a triple of an F1, F2 and length value, and the candidates were vowel labels. The input-output pairs and their frequencies were retrieved from the empirical data. For each possible input, all vowels used in the experiment formed the candidate set.

5 Experiments

Of the algorithms that were used for the simulations below, we implemented the Naive Bayes and Nearest Neighbor algorithms[11] in python from scratch, and used the Praat program[12] [4] for the OT-GLA simulations. The grammar and constraint sets for the OT simulations were self-created specifically for this experiment.

NN algorithm a single example. It will now classify every instance as belonging to the same class as the single training instance. When we add a second example with a different class label, the instance space will be altered radically. After adding a large number of examples, however, each new training instance will have only a minor impact on the memory space, and so the rate of learning will be low. A similar effect occurs for the NB classifier.

[10] $14 \times 26 + 10 \times 26 + 3 \times 26$, 26 being the number of all vowel labels.

[11] A good experimental environment for the Naive Bayesian and Nearest Neighbor algorithms, as well as a phlethora of other machine learning techniques can be found in the WEKA machine learning package (http://www.weka.org). However, this package did not provide some of the output features that we required, and so we decided to re-implement the algorithms ourselves.

[12] http://www.praat.org/

In all of the experiments below, we only used emperical data from subjects performing in their native language (i.e.: Spanish subjects categorizing stimuli according to Spanish labels) as training and test data for the learning algorithms. By training the algorithm on Spanish data and testing it on the empirical native Dutch monolingual dataset, we can test the performance of a simulated Spanish speaker on a Dutch vowel categorization task. The reason for the use of purely monolingual data is twofold. First, this data is relatively clear-cut (as obviously all subjects have a high command of their native language and tend to agree more in their judgments) and is considered to constitute a measure of "correctness". Second, if we were to simply train a model on a dataset containing classifications of speakers with increasing levels of proficiency classifying stimuli with vowel labels in a foreign language (which is already the data that we want to model), the model itself provides little or no explanation as to how the vowel space for these non-native speakers came to look the way it does.

Each of the stimuli in the test set is presented a number of times, and the classifier output is compared to the label in the test data each time. In the case of having non-deterministic output, this results in a distribution of vowel categorizations for each stimulus.

Since the regions for different vowels have soft borders, where multiple categorizations appear to be possible even for native speakers, the data itself is quite noisy. This is even more obvious for non-native speakers, who express more uncertainty in their categorizations, and whose data therefore display even softer region borders. Consequently, we do not expect (or aim for) the best performance in terms of the percentage of correct guesses, but rather concentrate on a more abstract comparison based on spread and point of gravity of the different vowel regions. The percentage correct gives only a small indication of this.

5.1 Monolingual experiments

The native speaker simulations (e.g.: A simulated Spanish speaker categorizing stimuli as Spanish vowels) were performed to find a reference level for the performance of a beginning or naive learner of Dutch. The native language performance is much more noise-free, and so we will expect the learners to do quite well on this data. In contrast, the L2 experiments will be much harder to mimic. This experiment also provides a testing ground for finding appropriate settings for each algorithm's parameters (such as the most appropriate k in k-NN).

5.2 L2 acquisition modeling experiments

A first experiment that is of interest is to see how simulated speakers of one language perform when classifying stimuli according to vowel categories of a language they have no experience with. In the experiments below, the foreign language is always Dutch, and the native language will be either Spanish or Portuguese.

During the experiments performed on human subjects, the participants were told which vowels were available for use in categorization, but our virtual learner does not have this information. If a classifier has never seen any instance of a certain vowel, it will assign no probability mass of the classification to the class denoted by that vowel. Since this means that the classifiers will be unable to generalize to vowels that were not observed in their training set, a way must be provided to do so. If we assume that a learner has no significant experience with Dutch whatsoever, we can not just feed Dutch examples as part of the training set. Furthermore, smoothing by distributing a small part of the probability mass equally over the unobserved classes will result in completely random guesses, which is not realistic in the sense that non-Dutch speakers will have some intuition about what a certain Dutch vowel will sound like, perhaps based on orthographic cues or generalization from their own native language. Thus, we assume a *mapping* of the participant's native vowel space onto the Dutch one, which will be pseudo-random (up to a certain degree), depending on similarities between the two languages.

Since the learning algorithm has no knowledge about orthography and language similarity, we created such a mapping by hand, based on observations made on how human participants performed on the same task. Each mapping consisted of a list of possible Dutch vowel guesses with a certain frequency for each native vowel separately. The mapping was then applied to the output of the classifier, where a random element of the list of possible mappings was chosen.

5.2.1 Increasing proficiency levels

To simulate an increasing exposure to Dutch language as the L2 acquisition process progresses, we added increasing samples of Dutch training data to the existing native data. This means that the learner can find either a known Dutch vowel as its chosen category, or a known native vowel. If a native vowel is chosen, it is mapped onto a Dutch one using the mappings described above. This way, the learner continues his mapping even at higher proficiency levels, but the number of guesses (mappings) that need to be made decreases as the number of categorizations based on Dutch experience increases. We expect the learner's vowel space configuration to show increasing similarity to the native Dutch vowel space as the size of the Dutch sample increases. Another approach to modeling L2 acquisition with increasing proficiency when starting from a Spanish (or Portuguese) vowel space distribution, is to have the output vowels in the training data not prefixed with "S" (or "P"), i.e. not to code specifically for the language.
The learner then already has representations of five of the vowels of Dutch and, instead of learning a completely different set of vowels, only has to adjust the perceptive field for the vowels that occur in both his native language and Dutch.

The difference between the two approaches – introducing vowels with language-specific prefixes and without – translated to human processing would be that in one case initially the learner implicitly or explicitly assumes that the vowels that occur in both languages have the same properties with regards to their location and distribution in the vowel space. In the other case, vowels in the new language are generally assumed to be distinct

Classifier	Dutch	Spanish	Portuguese
3-NN	49.9%	74.4%	60.5%
NB	**61.8%**	**82.3%**	**70.1%**
OT	55.4%	78.2%	69.7%

Table 5: Monolingual classification performance for each language per classifier.

from the vowels in the native language, although the mapping function between the native language and Dutch might mean that both approaches yield similar results and be functionally equivalent..
We plan to investigate this difference in future experiments, especially in connection with testing performance in the native language after the learner has acquired Dutch, which we assume to require several vowelspaces to be held by the learners.

6 Results

In this section, we will discuss the results of the experiments as described above. First, a baseline performance test on monolingual data (that is, simulated participants categorizing in their "native" language), will be briefly discussed, after which the performance of each classifier is investigated individually.

6.1 Monolingual Experiments

For the evaluation of results of the simulations we will concentrate on the corner vowels /i/, /u/ and /a/ (for duration of step size 1 (0.1s)), with notes on other vowel regions where it is deemed appropriate.

The results for the monolingual data (in percentage of categorized vowels with a matching test set label) are shown in table 5. These results are for a simulation of a native Spanish person categorizing stimuli with Spanish vowels. Since the result is supposed to be noisy, there will be a large number of false negatives, and so these results should only be taken as a lower bound, and an indicative measure of whether the classifiers are able to learn anything at all. As is seen from the table, the Naive Bayes classifier scores highest on all three monolingual language experiments. However, since we train and test on the same data for this experiment, it may simply have overfitted the data[13]. What is more important is that the results suggest that the data contain certain learnable patterns. It is also important to note that the accuracy of each of the classifiers decreases as more classes (more vowels) are available for a language. Spanish, containing only 5 vowels, is easiest to learn. Portuguese, with 7 vowels, is a bit harder, while Dutch, containing a grand total of 12 vowels is very hard to learn.

An ANOVA run on the three types of simulations (k-NN, NB and OT, with empirical data as reference value) reveals no significant difference between the empirical data and any of the simulations, p = .724, p = .781 and p = .978 for k-NN, NB and OT respectively (note that the difference for OT is the least significant by far).

In the following paragraphs we will describe the results of an initial cross-language experiment, in which we test using the mapping from Spanish to Dutch data as a viable way to recreate the human subject data for inexperienced Spanish learners of Dutch, with respect to each of the classifiers.

To see how this approach can be justified, the border plots below show plots for Spanish (human) subjects classifying stimuli with Dutch vowels (in figure 6), and Spanish (human) subjects classifying stimuli with Spanish vowels (in figure 7). The different colored regions represent classification regions in which a certain vowel was chosen. As is clear from figure 7, the Spanish vowel space is divided into 5 clear-cut regions, and there is little variance in the choices made (as is indicated by the intensity of the colors). Compare the regions in the plot of figure 6 to those in the plot in figure 7 (without paying attention to the vowel labels). There appears to be a clear correspondence between the dominant regions for the distinct vowels in the different vowel spaces. An ANOVA with the experiment type as factor reveals no significant difference between the empirical data and the simulations, although the significance level is not as low as for the Spanish monolingual simulations; p > 0.05 for all simulations.

[13]Overfitting is a known problem for many machine learning algorithms, where an algorithm has adapted nearly perfectly to its training data, but is unable to adequately generalize beyond this data, resulting in poor classification performance on new and unseen instances.

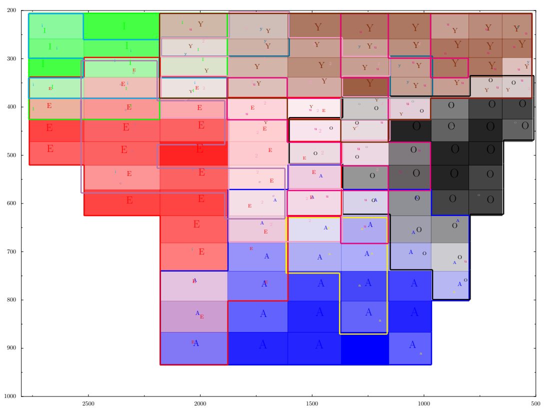

Figure 6: Human detail plot for Spanish subjects categorizing stimuli with Dutch vowels with low proficiency (All vowels shown, for Duration = 1).

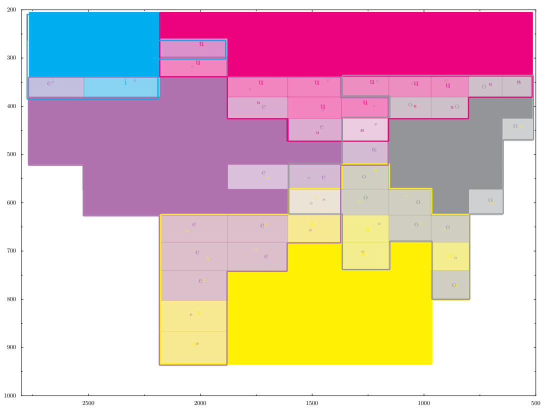

Figure 7: Human detail plot for Spanish subjects categorizing stimuli with Spanish vowels (All vowels shown, for Duration = 1).

A mapping from one onto the other would explain why some of the vowel categories in the cross-language categorization seem to be much more prominent than others; they correspond to a mapping from a consistent category from the subjects' native language. The simulation shows that such a mapping is indeed possible.

Note that this simulation, although dealing with cross-language categorization, really only depends on the learner's ability to learn Spanish, and on the hand tailored mapping.

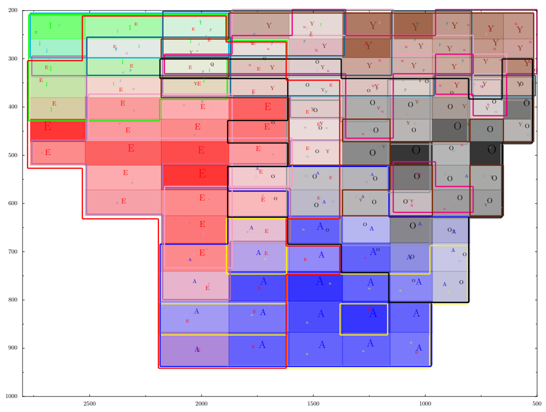

Figure 8: 3-Nearest Neighbor detail plot for a simulated Spanish subject with low Dutch proficiency categorizing stimuli with Dutch vowels (All vowels shown, for Duration = 1).

6.1.1 Vowel Space Mapping

Figure 8 shows a detailed colored region plot of the 3-Nearest Neighbor simulation of a Spanish person making categorizations based on Dutch vowels without any experience. Similar plots for the Naive Bayes classifier and the Optimality Theory outputs are shown in figures 9 and 10 respectively. The result was found by mapping the classifier's Spanish vowel output onto a Dutch vowel, using a hand tailored, semi-random mapping. The mapping was done as given in table 6.

Although each of the plots bears quite a bit of resemblance to the real data, shown in figure 6, sceptics will note that the match is not perfect. The main vowel regions are clearly there in each of the simulations, but there is more uncertainty in the mapped simulations, and the guesses are more erratic, as is seen by the lack of consistent color gradients in the plot, and the irregularity of the borders. Where the classifiers often have multiple smaller regions for a less common vowel, and gaps in the larger regions, the human subjects appear to be more consistent, increasing their confidence on which vowel to choose when categorizing stimuli that are closer to the centroid of a region of a vowel in their native vowel space. It is believed that we may be better able to simulate this for the NB classifier if the stimulus values are used as continuous attributes (using a normal distribution), rather

Vow	A (ɑ)	a (aː)	E (ɛ)	e (eː)	I (ɪ)	i (i)	O (ɔ)	o (oː)	2 (øː)	u (u)	Y (ʏ)	y (y)
Sa (a)	$\frac{14}{17}$	$\frac{2}{17}$	$\frac{1}{17}$	0	0	0	0	0	0	0	0	0
Se (ɛ)	0	0	$\frac{8}{12}$	$\frac{2}{12}$	0	$\frac{1}{12}$	0	0	$\frac{1}{12}$	0	0	0
Si (i)	0	0	0	$\frac{1}{12}$	$\frac{8}{12}$	$\frac{2}{12}$	0	0	0	0	0	$\frac{1}{12}$
So (ɔ)	0	0	0	0	0	0	$\frac{11}{16}$	$\frac{3}{16}$	0	$\frac{1}{16}$	$\frac{1}{16}$	0
Su (u)	0	0	0	0	0	0	0	0	$\frac{1}{14}$	$\frac{2}{14}$	$\frac{9}{14}$	$\frac{2}{14}$

Table 6: Distribution used for Spanish to Dutch vowel mappings based on empirical data.

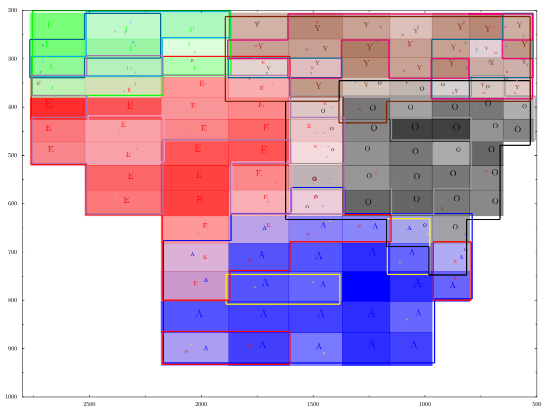

Figure 9: Naive Bayes detail plot for a simulated Spanish subject with low Dutch proficiency categorizing stimuli with Dutch vowels (All vowels shown, for Duration = 1).

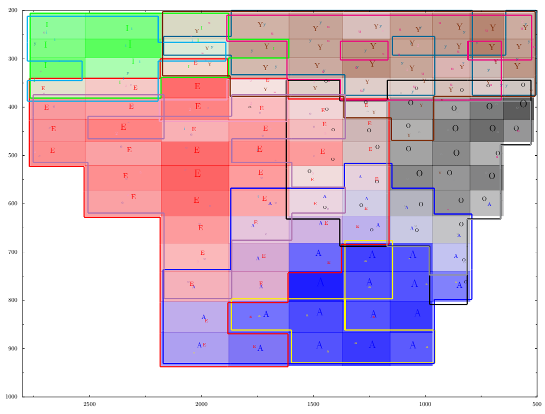

Figure 10: Optimality Theory detail plot for a simulated Spanish subject with low Dutch proficiency categorizing stimuli with Dutch vowels (All vowels shown, for Duration = 1).

than discrete steps.

Nevertheless, the plots seem to suggest that it may well be possible to explain the layout of the vowel regions by taking a Spanish vowel space, and projecting Dutch vowels onto it.

To make the differences and similarities more clear, figure 11 shows an ellipsis plot for the three corner vowels, /i/, /a/, and /u/ for both the simulations and the empirical data. Recall that the size of an ellipsis is determined by the variance components of the corresponding vowel in the F1 and F2 directions, with the midpoint determined by the F1 and F2 averages for that vowel. Comparing the ellipsis plots, we can see that for all the corner vowels, the match to the human data (shown in green) is quite good. The inconsistency with respect to the human categorization are in part due to the perceptual interference caused by the short stimulus duration. Our mappings do not yet provide for different behaviors for different durations.

Note that the NN classifier, which had the lowest matching score in the trial run described earlier, also shows the most inconsistencies with the human data. This is to be expected, as the mappings to Dutch vowels are based on the Spanish categorization. The NB performs somewhat better, but still appears to be a bit noisy. The OT simulation appears to be the most consistent.

6.2 Cross-language Experiments

Next, it will be interesting to look at the changes that occur within the simulated vowel space, as we increase the amount of Dutch 'experience' that a learner has. The corresponding empirical (human) data, for high proficiency Spanish speakers categorizing stimuli using Dutch vowels, is shown in figure 12. Note that, although we would expect the uncer-

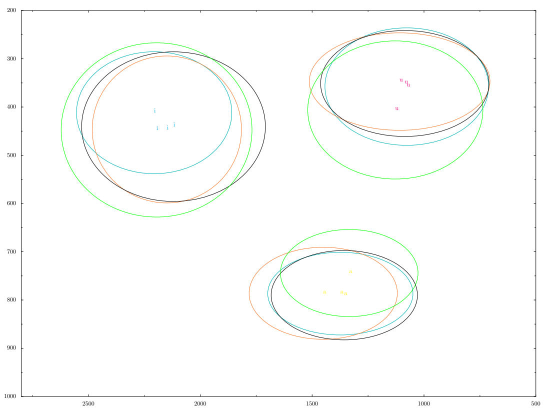

Figure 11: Ellipsis plot for simulated Spanish subjects for all three simulations with low Dutch proficiency categorizing stimuli with Dutch corner vowels, with overlaid corresponding human experiment (NN: Orange, NB: Blue, OT: black, Human: Green, Corner vowels shown, for Duration = 1).

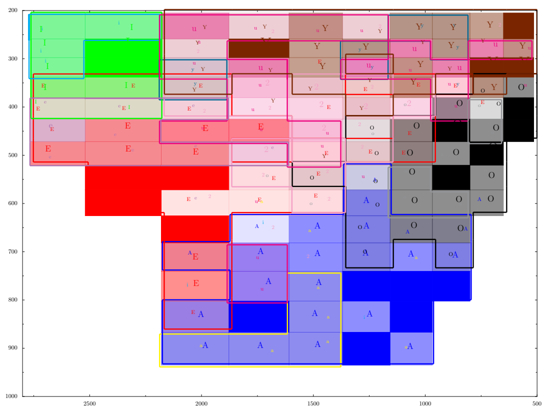

Figure 12: Human detail plot for Spanish subjects categorizing stimuli with Dutch vowels with high proficiency (All vowels shown, for Duration = 1).

tainty for a proficient learner to be much lower, the plot seems to suggest otherwise. Some vowels are very easily identified, as seen by the intensity of the colors, while other stimulus regions are categorized with many different vowel labels (as indicated by an almost white region). The correspondence between the vowel space configuration of Spanish natives using Dutch vowels, and that of native Dutch speakers also seems to be quite far apart still.

Upon investigation, the average amount of time that the Spanish native participants had been speaking Dutch, measured over all participants for which this information was available was 11.0% of their lifetime, while none of the participants had been speaking Dutch for a fraction of time higher than 17.9% of their age. Figures 14, 15 and 16 show simulated vowel spaces for learners (k-NN, NB and OT respectively) that were trained on 100% of the Spanish data, combined with 30% Dutch data. As is seen from the graphs, the correspondence to the Spanish-to-Dutch mapped space is still relatively high for the NN and NB learners. The NN again shows a lot of classification uncertainty. This may be partially due to the low k values used (3), resulting in very local decision boundaries.

The OT simulation learns fastest, and has reached a near-Dutch configuration after adding only 30% Dutch training data. In figure 20, the corner vowels for each of the simulations are shown after 30% Dutch exposure, together with the empirical native Dutch, and empirical high-proficiency Spanish-Dutch ones. Note that the OT simulation data corresponds almost exactly to the Dutch data at this point for both the /i/ and /u/ vowels. The other learners have also distanced themselves from the Spanish configurations, and appear to fit the Dutch vowel space configuration more closely than was the case for the low-proficiency simulation[14], although traces of Spanish influences are still clearly visible.

In contrast, each of the simulations trained with 100% of the Spanish monolingual data and 60% of the Dutch data, shown in figures 18, 17 and 19, shows a much higher correspondence to the Dutch native perceived vowel space, shown in figure 13.

This can be seen as a *unanimous* prediction that speakers with an abundance of Dutch experience will have a perceived vowel space category configuration much closer to the Dutch one than their less experienced counterparts.

6.3 Discussion

Although so far the OT simulations seem to do best, the nature of the mappings used may well cloud our judgement. At the moment, a mapping

[14]Note that, although the learners seem to have picked up more Dutch than the human participants, the amount of Dutch training for the learners is actually more than 10% higher.

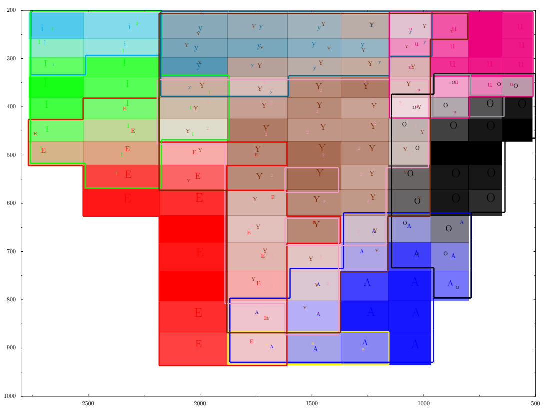
Figure 13: Human detail plot for Dutch subjects categorizing stimuli with Dutch vowels (All vowels shown, for Duration = 1).

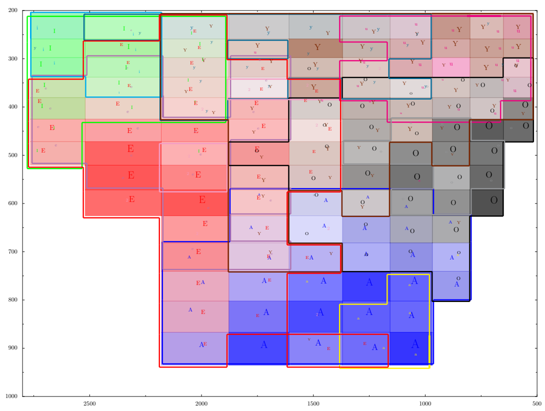
Figure 14: k-NN detail plot for simulated Spanish subjects categorizing stimuli with Dutch vowels, after 30% exposure to Dutch training data (All vowels shown, for Duration = 1).

is chosen *after* an algorithm has selected a native vowel. Once the Spanish categorization is chosen, Spanish to Dutch mappings for other potential candidates can no longer compete, and so the borders between mappings are rather solid. This may also offer a partial explanation for the lack of centralization in the NB and NN simulations. However, since the OT simulation can provide better results under the same circumstances, this may be sufficient grounds to rule in favor of this algorithm as the best candidate algorithm so far.

7 Conclusions

In this paper, we have presented a trinity of both visualization techniques and algorithms, in order to carry out a number of initial simulations of the vowel space during the process of second language acquisition. In addition, we provided empirical evidence that performance in stimulus categorization by non-native low proficiency speakers of Dutch can be achieved by a simple mapping made from their native vowel space, without further subdivisions or adjustments of the regions in that space.

Although the verdict is still out on what the best model will be in the end, the most promising of the three algorithms is by and large the Optimality Theoretic simulation using the Gradual Learning Algorithm.

As it is, we find that all of our models, when trained on a large portion of Dutch data, shift their vowel space configuration from a mapped Spanish, towards a Dutch native space. This can be seen as a prediction; as a (human) subject is exposed to increasing quantities of non-native input, the native vowel space will undergo a transformation in which the region borders are aligned with the new input. We have shown that all three learning algorithms make this prediction, although they differ in their details.

8 Future Work

First and foremost, it should be noted that none of the algorithms as presented here are capable of learning without labeled information[15]. It would be interesting, then, to see if similar achievements can be made using unsupervised learning methods.

Secondly, in light of the predicted shift in vowel space configuration, it would also be interesting to investigate the vowel perception of real Spanish natives that have spent more than half their lives in the Netherlands, and compare this data to the predictions made by these models.

Furthermore, the mapping made from Spanish to Dutch seems to be backed up by the empirical data presented here. If there is a mapping from the native language to Dutch, then the reverse may also be true for long time speakers of Dutch who

[15]The OT simulation is capable of making a separate clustering step, but so far, this has not been explored.

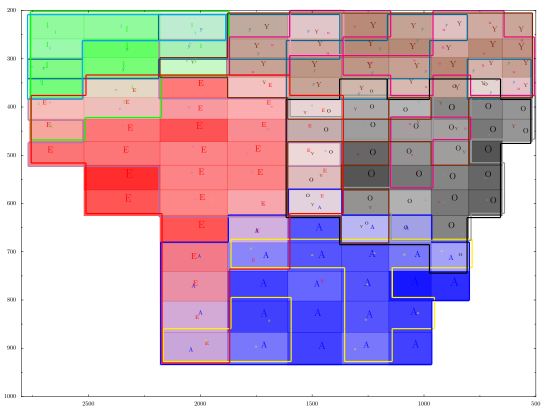

Figure 15: NB detail plot for simulated Spanish subjects categorizing stimuli with Dutch vowels, after 30% exposure to Dutch training data (All vowels shown, for Duration = 1).

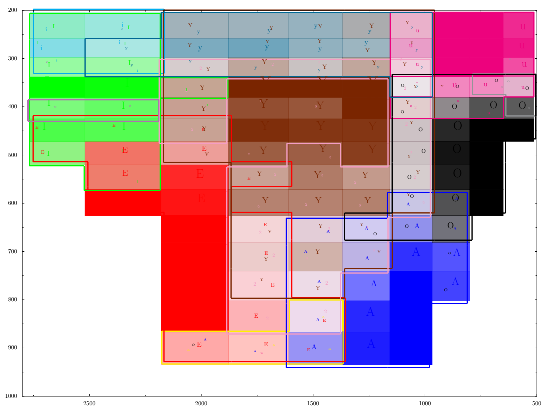

Figure 16: OT detail plot for simulated Spanish subjects categorizing stimuli with Dutch vowels, after 30% exposure to Dutch training data (All vowels shown, for Duration = 1).

have Spanish as their primary language. If this is indeed the case, we may find that, when having such subjects classifying with Spanish vowels, they perform a mapping from Dutch to Spanish. The models above can be used to predict the effects of such a mapping.

Finally, it may be more prudent to apply the mappings to the possible classifications *before* a final choice is made, rather than after the classifier has chosen a Spanish vowel, and make the final mapping choice based on the resulting distribution. This will give the mappings more opportunity to compete, and may result in more realistic distributions.

References

[1] Paul Boersma. *Functional Phonology. Formalizing the interactions between articulatory and perceptual drives.* PhD thesis, Universiteit van Amsterdam, 1998.

[2] Paul Boersma and Paola Escudero. Learning to perceive a smaller l2 vowel inventory: an optimality theory account. Rutgers Optimality Archive, 2004.

[3] Paul Boersma and Bruce Hayes. Empirical tests of the gradual learning algorithm. *Linguistic Inquiry*, 32:45–86, 2001.

[4] Paul Boersma and David Weenink. Praat: doing phonetics by computer (version 4.4.24) [computer program], 2006.

[5] Paola Escudero and Paul Boersma. Modelling the perceptual development of phonological contrasts with optimality theory and the gradual learning algorithm. In *Penn Working Papers in Linguistics 8.1: Proceedings of the 25th Penn Linguistics Colloquium*, pages 71–85, 2003.

[6] Paola Escudero and Paul Boersma. Bridging the gap between l2 speech perception research and phonological theory. *Studies in Second Language Acquisition*, 26(4):551–585, 2004.

[7] Jhy-Han Lin and Jeffrey Scott Vitter. A theory for memory-based learning. Technical Report Technical report DUKE–TR–1993–29, 1993.

[8] Thomas M. Mitchell. *Machine Learning.* 1997.

[9] Alan Prince and Paul Smolensky. Optimality theory: Constraint interaction in generative grammar. Technical report, Rutgers Center for Cognitive Science, 1993.

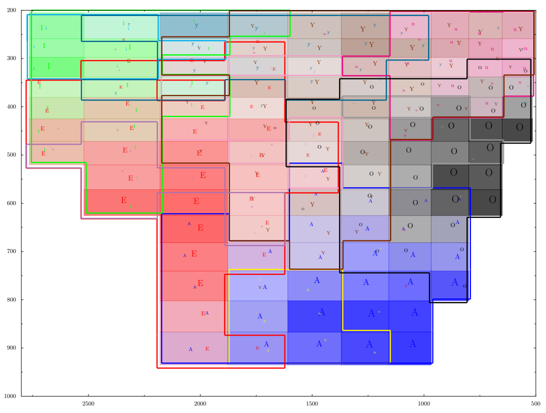

Figure 17: *k*-NN detail plot for simulated Spanish subjects categorizing stimuli with Dutch vowels, after 60% exposure to Dutch training data (All vowels shown, for Duration = 1).

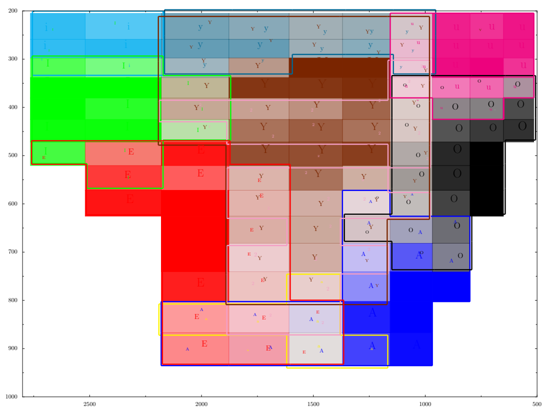

Figure 19: OT detail plot for simulated Spanish subjects categorizing stimuli with Dutch vowels, after 60% exposure to Dutch training data (All vowels shown, for Duration = 1).

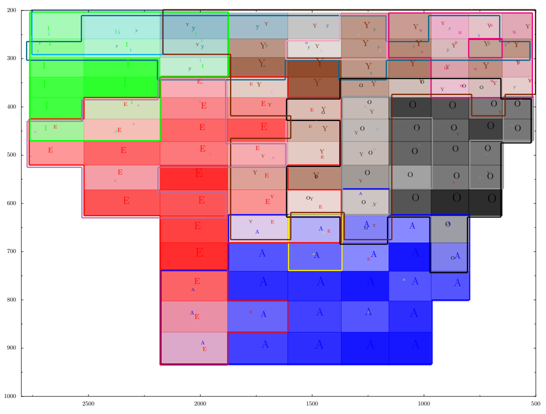

Figure 18: NB detail plot for simulated Spanish subjects categorizing stimuli with Dutch vowels, after 60% exposure to Dutch training data (All vowels shown, for Duration = 1).

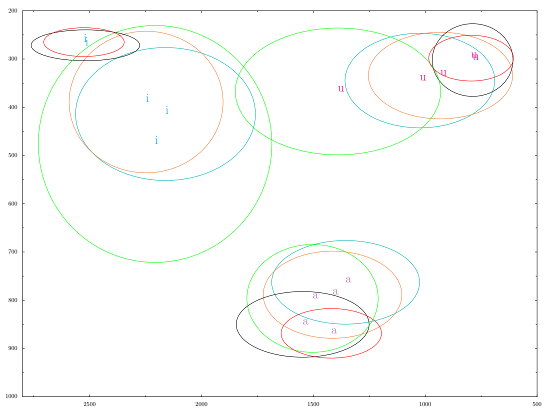

Figure 20: Ellipsis plot for simulated Spanish subjects for all three simulations with high Dutch proficiency, categorizing stimuli with Dutch corner vowels, with overlaid corresponding human Spanish experiment and human native Dutch experiment (NN: Orange, NB: Blue, OT: black, Human native Spanish: Green, Human native Dutch: Red, Corner vowels shown, for Duration = 1).

A Appendix: Software Documentation

A.1 Plotting

extraSymbols.py This file defines the drawing methods for several extra symbols (mostly IPA symbols) so that they can be used for plotting. This was done because the number of symbols in PyX is limited. We recommend using text-based symbols, however, as they look nicer.

vowels.py This file can output several conversion tables for the vowel data. `get_coordinates()` outputs translation tables for F1, F2 and duration steps into their continuous real values. `get_vowels()` outputs a dictionary with properties used to plot the different vowels, such as vowel and language color, and corresponding plotting symbols.

vdict.py This file contains a dictionary wrapper class that can be used to assign internal variables to the dictionary.

plotData.py This script uses the Python graphics package Pyx[16] to produce the visualizations discussied in section 3. It also makes use of Psyco[17], a Python module that compiles bits of the Python code to C++ binary code to speed up the execution.

The function `plotData(filename, outputpath, lookuptable, f1s, f2s, lengths, symcolors, bounds="grid", fill_regions, cutoff, removenoise, removelow, per_vowel=False, include_vowels, include_lengths)` is the plotting function for the border and symbol plots. It takes `filename` as input file, and outputs a PDF plot to `outputpath`. It uses `lookuptable`, `f1s`, `f2s` and `lengths`, output by `vowels.py` as vowel property and coordinate translation tables respectively. The three attributes `cutoff`, `removenoise` and `removelow` represent different noise removal cutoff values. It is recommended to leave them at their defaults. The function includes only vowels and lengths specified in `include_vowels` and `include_lengths`. `symcolors` specifies the choice for type of symbol and symbol coloring to use. The different symbol styles are:

- "text" uses text symbols, colored by language.
- "coloredtext" uses text symbols, colored by vowel.
- "symbols" uses PyX symbols, a different symbol for each vowel (e.g. IPA), colored by language.
- "coloredsymbols" uses PyX symbols, a different symbol for each vowel (e.g. IPA), colored by vowel.
- "coloredsymlanguage" uses PyX symbols, a different symbol for each language (e.g. circles, squares), colored by language.
- "coloredsymvowel" uses PyX symbols, a different symbol for each language (e.g. circles, squares), colored by vowel.

The argument `bounds="grid"` means that the function will use the function `findBoundaries()` to find and draw vowel regions. `fill_regions` specifies whether or not the regions inside these boundaries should be filled.

The bash plotting function with which to plot all boundary or symbolic data in a directory is `plotAllDataInDir(datadirs, outputpath, symcolors, boundaries="grid", fill_regions, character_cutoff, removenoise, removelow, per_vowel, include_vowels, include_lengths)`. It takes a list of directories (`datadirs`) and creates a series of plots for each data file found in that directory with the specified features.

The function `plotEllipsisForFiles(files, plotdir, lookuptable, include_vowels, length)` plots ellipsis plots for all files in `files` in one plot (output to directory `plotdir`) according to the properties output for the vowels by `vowels.py`, in `lookuptable`. It will output only the vowels and lengths in the lists `include_vowels` and `length` respectively. It uses the function `computeAverages(data)` to compute the necessary statistics (average, variance, covariance, correlation) for F1, F2 and duration values per vowel.

[16] http://pyx.sourceforge.net/
[17] http://psyco.sourceforge.net/

To plot (separate) ellipsis plots for all data files in a list of directories, use `plotEllipsisAllDataInDir(datadirs, plotdir, lookuptable, include_vowels, include_lengths)`.

The bash plot function used to plot the data for the simulations (combined ellipsis and border plots) is `plotAllSimulationsData(simulationsdir, outputdatapath, outputplotpath, lookup, symcolors="text", boundaries="grid", fill_regions, character_cutoff, removenoise, removelow, per_vowel, include_vowels, include_lengths)`. It rewrites the output of the simulations to the format that is used in the plotfunctions (and outputs the data to `outputdatapath`), and plots the data to the directory `outputplotpath`. `lookup` is the vowel properties table output by `vowels.py`. `symcolors` is the type of (symbolic and/or border) plot used. `boundaries` specifies whether or not boundaries should be drawn. `fill_regions` specifies whether or not regions should be filled in the border plots. `per_vowel=True` outputs a separate plot for each vowel. `include_vowels` and `include_lengths` specify which lengths and vowels should be plotted. The function creates all border plots for the simulations, as well as several combined ellipsis plots for different models with the same language.

Aside from the functions producing the plots, the `plotData.py` file also contains several data reformatting functions. `toComparativeStatisticsFormat( filenames, experiment_type, outputfilename)` rewrites the data in all of the files in `filenames` to a single outputfile, adding a column with the text in `experiment_type` (which should be specified for each file in `filenames`), so that different experiments can be compared in SPSS. If no experiment type is specified for a file, the filename is used.

A.2 Modeling

textToARFF.py This file contains methods to convert the (column-wise) experimental data to (row-wise) ARFF formatted data, to be used as input to WEKA or the python classifiers described below. The function `allVowelToARFF(path, arffdatadir)` will convert all files in directory `path` to ARFF files, and write them to separate files in directory `arffdatadir`. The output format contains the following columns.

F1 (NUMERIC)
F2 (NUMERIC)
LENGTH (NUMERIC)
F1N (NOMINAL)
F2N (NOMINAL)
LENGTHN (NOMINAL)
F1st (NOMINAL)
F2st (NOMINAL)
LENGTHst (NOMINAL)
VOWEL (NOMINAL)

and outputs one line for every vowel in the empirical data (so, one line for every vowel times it's frequency).

A.2.1 Nearest Neighbour & Naive Bayes

The Naive Bayes and Nearest Neighbor classifiers were re-implemented from scratch in python. The software package consists of 6 files. They use ARFF formatted (row-wise) input.

fileWrappers.py This file contains a number of I/O methods, the most important of which is the function `loadARFFDataFile(filename)`, which reads ARFF formatted data from a file and outputs a list of lists (one list of elements per line in the file). IMPORTANT: The training and test-sets should be reloaded this way before each new experiment, even if the same sets are used.

NaiveBayesClassifier.py This file contains the Naive Bayes classifier class, including testing and training functions. First, one must create a new classifier instance, by stating "`classifier = NaiveBayesClassifier()`". The function `runAllVowelSimulations()` can then be used to run a number of simulations automatically.

Alternatively, one can specify a new experiment by loading a new trainingset by the use of the function `loadARFFDataFile(trainingsetfilename)`, which is specified in file `fileWrappers.py`. One can use *n*-fold (hold out) cross validation on this data by using the function `NaiveBayesClassifier.crossValidate( database, n_fold, class_index, ignore, continuous_attr, cont_type='gauss', k=3, output_file, probability_matching,` `vowel_translation)`. The most important attributes here are `database` (the trainingset), `ignore` (which specifies which features – i.e.: column numbers – should NOT be used for training and testing), `probability_matching` (which is either `True` or `False`) and `vowel_translation` (which provides the mapping of native to second language vowels, e.g.: mapping from Spanish to Dutch). If the class label is not the last element of each row, the index must be specified. The other attributes should not be altered, as they represent experimental procedures that have not been thoroughly tested.

Similarly, one can also run the validation on a separate testset, using `NaiveBayesClassifier.validateOnSeparateTestSet(database, testset, class_index, ignore, continuous_attr, cont_type='gauss', k, output_to_file, probability_matching, vowel_translation)`.

kNearestNeighborClassifier.py This file contains the Nearest Neighbor classifier methods. One can run the simulations directly by using the function `runAllVowelSimulations()`. Additionally, one can run a new simulation by creating a model using "`classifier = kNearestNeighbor(trainingset, class_label_index, ignore)`". Again, the trainingset must be loaded using `loadARFFDataFile(trainingsetfilename)` first.

Validation is currently only possible on a separate testset (no explicit method implemented for cross-validation yet), using the function `kNearestNeighbor.test( testset, class_label_index, ignore, k, weigh_square_distance, probability_matching, output_file, vowel_translation)`. Here, `k` is the number of neighbors used (-1 if all), `weigh_squared_distance` is a boolean indicating whether or not the neighbor vote should be weighted by it's distance to the point to be classified (always true in the case of considering all neighbors), and the other settings are similar to those in the NaiveBayesClassifier test method. Testing is currently very slow for this classifier, so be warned.

translateVowels.py This file contains the functions for mapping vowels from one language to the other. The function `translatevowel(from_language, to_language, vowel)` is used to translate a vowel from one language (ES, BR or LA at the moment) to another (only Dutch (D) is implemented). The matching is done by choosing a random element from a list of possible translations (separate for each vowel in the source language). If no match is found, the vowel is returned without being translated.

trainSet.py This file can generate an artificial normally distributed data set with one variable for two classes. Used as a test function for the Naive Bayes classifier.

priorityQueue.py This file implements a priority queue interface.

A.2.2 OT

The software used for creating, training and testing the OT learner consists of four different scripts. `GenerateGrammar.py` creates a new OT grammar, `GenerateData.py` converts the human data into training files and `training.praat` trains and tests the grammar. `training.praat` also calls `makeStatistics.py` which converts the testing data into tables analogous in format to the representation used for the human data.

GenerateGrammar.py The Python script does not take any input and as an output generates an untrained OT grammar `phonGrammar.OTGrammar`. A list of the vowels and stimuli is retrieved from `vowels.py` and the file `translationtable` respectively. The script then generates the constraints, one for every combination of F1, F2 and duration value and every vowel. All constraints are initiated with a value of 100. Then for every stimulus a table that indicates which candidate (every vowel is a possible candidate for every input) violates which constraints is generated. The resulting grammar is written to a file and can consequently be opened with Praat.

GenerateData.py This Python script is called without arguments and as an output creates training files for the OT learner in the folder training. It loops through the result tables in a directory specified in the code and converts each into a list of input-output pairs where the input is the stimulus and the output is a vowel. Each input-output pair is augmented with a relative frequency corresponding to the percentage that the output was given as a response to the input.

Thus, for every input the relative frequency values of the pairs it appears in add up to 1. The function `readVowelDataFromTabbedFile` takes a filename and reads it into a variable which is then returned. `MakeDatafile` takes a filename and for each stimulus-vowel pair for which the count is not zero finds the relative frequency and writes the pair into the current training file.

training.praat This Praat script takes the name of a grammar, a training file, the number of chews. replications and testing samples per location as an input. Additionally "Translate" can be set to on or off. "Grammar": By default, this is set to "original" which means that the untrained grammar generated by `GenerateGrammar.py` is used an a initial grammar. Since all grammars are saved to grammars in the OT folder after training, a pre-trained grammar that resulted from a previous simulation can also be chosen. In this case the name of the grammar in question has to be given without the ".OTGrammar" extension. "Training": Here, the name of a training file from the training folder can be entered (again without the file extension, i.e. ".PairDistribution"). By default the Dutch training data containing all lengths is used. "Chews" specifies the number of times each input-output pair is presented to the grammar. By fefault it is set to 10. "Replications" gives the number of repetitions per plasticity. The plasticity is decreased from 1.0 to 0.1 in three steps during training, so the number of total replications is the number entered here times four. Testing samples per locations is by default set to 50. The number specifies the number of times that every stimulus is presented during testing and thus could be seen as being analogous to the number of participants in the experiments on humans. Finally, if "Translate" is checked, this means that the testing data will be mapped to Dutch using the mappings discussed in the text. For obvious reasons this should only be checked when the training set is not Dutch. The script trains a grammar according to the parameters given in the input and then outputs the testing results into the folder tmp, one file per stimulus. It then calls the Python script `makeStatistics.py`.

makeStatistics.py This script takes the number of a grammar and a training file and a boolean indicating whether the testing results should be mapped to Dutch as an input. It reads the testing files in "tmp" and sums the data into a table identical in format with that of the human data, that is, the columns represent vowels and the rows stimuli and the cells give the number of times that the vowel was given as a response to the stimulus. If mapping is enabled, the responses are transformed using `translateVowels` in the Classifiers package.

A.3 Prognosis

In the near future, we wish to reorganize the package in a cleaner manner. For instance, at the moment, some of the scripts (`plotData.py` in particular), contain many different functions designed for a variety of tasks. These functions should be organized more cleanly if more users wish to make use of them. We also want to add more and better comments to the code, and create a better documentation. Finally, if time permits it, we would like to create a more user-friendly interface for both the plotting and simulation software.