Predicting Learner Levels for Online Exercises of Hebrew

Markus Dickinson, Sandra Kübler, Anthony Meyer
Indiana University
Bloomington, IN, USA
{md7,skuebler,antmeyer}@indiana.edu

Abstract

We develop a system for predicting the level of language learners, using only a small amount of targeted language data. In particular, we focus on learners of Hebrew and predict level based on restricted placement exam exercises. As with many language teaching situations, a major problem is data sparsity, which we account for in our feature selection, learning algorithm, and in the setup. Specifically, we define a two-phase classification process, isolating individual errors and linguistic constructions which are then aggregated into a second phase; such a two-step process allows for easy integration of other exercises and features in the future. The aggregation of information also allows us to smooth over sparse features.

1 Introduction and Motivation

Several strands of research in intelligent computer-assisted language learning (ICALL) focus on determining learner ability (Attali and Burstein, 2006; Yannakoudakis et al., 2011). One of the tasks, detecting errors in a range of languages and for a range of types of errors, is becoming an increasingly popular topic (Rozovskaya and Roth, 2011; Tetreault and Chodorow, 2008); see, for example, the recent HOO (Helping Our Own) Challenge for Automated Writing Assistance (Dale and Kilgarriff, 2011). Only rarely has there been work on detecting errors in more morphologically-complex languages (Dickinson et al., 2011).

In our work, we extend the task to predicting the learner's level based on the errors, focusing on Hebrew. Our system is targeted to be used in a university setting where incoming students need to be placed into the appropriate language level—i.e., the appropriate course—based on their proficiency in the language. Such a level prediction system for Hebrew faces a number of challenges: 1) unclear correspondence between errors and levels, 2) missing NLP resources, and, most critically, 3) data sparsity.

Placing learners into levels is generally done by a human, based on a written exam (e.g. (Fulcher, 1997)). To model the decision process automatically, we need to understand how the types of errors, as well as their frequencies, correspond to learner levels. There is only little work investigating this correspondence formally (see (Hawkins and Filipović, 2010; Alexopoulou et al., 2010) for discussion) and only on error-annotated English learner corpora. For this reason, we follow a data-driven approach to learn the correspondence between errors and levels, based on exercises from written placement exams. Although the exact exercises will vary across languages and language programs, the methodology is widely applicable, as developing a small set of exercises requires minimal effort—effort already largely expended for paper exams. Currently, we focus on an exercise in which the learner has to order a set of words into a grammatical sentence. Our goal is to move towards freer language production and to analyze language proficiency through more variables, but, in the interest of practicality, we start in a more restricted way.

For lesser-resourced languages, there is generally little data and few NLP resources available. For Hebrew, for example, we must create our own pool of

learner data, and while NLP tools and resources exist (Goldberg and Elhadad, 2011; Yona and Wintner, 2008; Itai and Wintner, 2008), they are not adapted for dealing with potentially ill-formed learner productions. For this reason, we are performing linguistic analysis on the gold standard answers to obtain optimal linguistic analyses. Then, the system aligns the learner answer to the gold standard answer and determines the types of deviations.

Since Hebrew is a less commonly taught language (LCTL), we have few placement exams from which to learn correspondences. Compounding the data sparsity problem is that each piece of data is complex: if a learner produces an erroneous answer, there are potentially a number of ways to analyze it (cf. e.g. (**?**)). An error could feature, for instance, a letter inserted in an irregular verb stem, or between two nouns; any of these properties may be relevant to describing the error (cf. how errors are described in different taxonomies, e.g. (Díaz-Negrillo and Fernández-Domínguez, 2006; Boyd, 2010)). Specific error types are unlikely to recur, making sparsity even more of a concern. We thus need to develop methods which generalize well, finding the most useful aspects of the data.

Our system is an online system to be used at the Hebrew Language Program at our university. The system is intended to semi-automatically place incoming students into the appropriate Hebrew course, i.e., level. As with many exams, the main purpose is to "reduce the number of students who attend an oral interview" (Fulcher, 1997).

2 The Data

Exercise type We focus on a scrambled sentence exercise, in which learners are given a set of well-formed words and must put them into the correct order. For example, given (1), they must produce one of the correct choices in (2). This gives them the opportunity to practice skills in syntactic ordering.[1]

(1) *barC beph dibrw hybrit ieral la tmid*

(2) a. *la tmid dibrw beph*
not always spoke in-the-language
hybrit barC ieral .
the-Hebrew in-land-of Israel .
'They did not always speak in the Hebrew language in the land of Israel.'

b. *barC ieral la dibrw tmid beph hybrit .*

c. *la tmid dibrw barC ieral beph hybrit .*

(3) *barC ieral* ***la tmid*** *dibrw beph* ***hybriM*** *.*

Although the lexical choice is restricted—in that learners are to select from a set of words—learners must write the words. Thus, in addition to syntactic errors, there is possible variation in word form, as in (3), where *hybrit* is misspelled as *hybriM*.

This exercise was chosen because: a) it has been used on Hebrew placement exams for many years; and b) the amount of expected answers is constrained. Starting here also allows us to focus less on the NLP preprocessing and more on designing a machine learning set-up to analyze proficiency. It is important to note that the proficiency level is determined by experts looking at the whole exam, whereas we are currently predicting the proficiency level on the basis of a single exercise.

Placement exams The data for training and testing is pooled from previous placement exams at our university. Students who intend to take Hebrew have in past years been given written placement exams, covering a range of question types, including scrambled sentences. The learners are grouped into the first to the sixth semester, or they test out. We are using the following levels: the first four semesters (100, 150, 200, 250), and anything above (300+).

We use a small set of data—38 learners covering 128 sentences across 11 exercises—all the data that is available. While this is very small, it is indicative of the type of situation we expect for resource-poor languages, and it underscores the need to develop methods appropriate for data-scarce situations.

(Manual) annotation For each of the 11 unique exercises, we annotate an ordered list of correct answers, ranked from best to worst. Since Hebrew possesses free word order, there are between 1 and 10 correct answers per exercise, with an average of 3.4 gold standard answers. The sentences have between 8 and 15 words, with an average of 9.7 words per exercise. This annotation concerns only the gold standard answers. It requires minimal effort and needs to be performed only once per exercise.

[1] We follow the transliteration scheme of the Hebrew Treebank (Sima'an et al., 2001).

```
T09:      SURFACE qnw
          SEGMENTATION (VB-BR3V qnw)
          PRE_PARTICLES    -
          MAIN_WORD:
              INDICES    0,1,2,
              TAG    VB-BR3V
              BINYAN    PAAL
              INFL_PREFIX    -
              STEM    0,1,
              ROOT    0,1,h,
              INFL_SUFFIX    2,
          PRO_SUFFIX    -
```

Figure 1: An example annotated word for *qnw* ('bought'), token T09 in one particular exercise

To annotate, we note that all the correct answers share the same set of words, varying in word order and not in morphological properties. Thus, we store word orders separately from morphological annotation, annotating morphology once for all possible word orders. An example of morphological annotation is given in fig. 1 for the verb *qnw* ('bought'). Segmentation information is provided by referring to indices (e.g., STEM 0,1), while TAG and BINYAN provide morphosyntactic properties.

Since the annotation is on controlled, correct data, i.e., not potentially malformed learner data, we can explore automatically annotating exercises in the future, as we expect relatively high accuracy.

3 System overview

The overall system architecture is given in fig. 2; the individual modules are described below. In brief, we align a learner sentence with the gold standard; use three specialized classifiers to classify individual phenomena; and then combine the information from these classifiers into an overall classifier for the learner level. This means the classification is performed in two phases: the first phase looks at individual phenomena (i.e., errors and other properties); the second phase aggregates all phenomena of one learner over all exercises and makes a final decision.

4 Feature extraction

To categorize learners into levels, we first need to extract relevant information from each sentence. That is, we need to perform a linguistic and/or error analysis on each sentence, which can be used for classification (sec. 5). Although we extract features for classification, this analysis could also be used for other purposes, such as providing feedback.

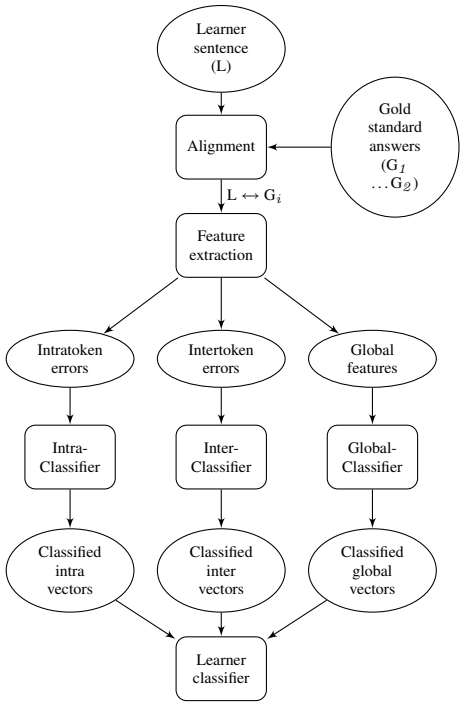

Figure 2: Overall system architecture (boxes = system components, circles = data)

4.1 Phenomena of interest

We extract features capturing *individual phenomena*. These can be at the level of individual words, bigrams of words, or anything up to a whole sentence; and they may represent errors or correctly-produced language. Importantly, at this stage, each phenomenon is treated uniquely and is not combined or aggregated until the second phase (see sec. 5).

While features can be based on individual phenomena of any type, we base our extracted features largely upon learner errors. Errors have been shown to have a significant impact on predicting learner level (Yannakoudakis et al., 2011). To detect errors, we align the learner sentence with a gold standard and extract the features. Although we focus on errors, we model some correct language (sec. 4.3.3).

4.2 Token alignment

With a listing of correct answers, we align the learner sentence to the answer which matches best:

We iterate over the correct answers and align learner tokens with correct tokens, based on the *cost* of mapping one to the other. The aligned sentence is the one with the lowest overall cost. The cost between a source token w_s and target token w_t accounts for:

1. Levenshtein distance between w_s & w_t (Lev)
2. similiarity between w_s & w_t (longest common subsequence ($LCSq$) & substring ($LCSt$))
3. displacement between w_s & w_t ($Displ$)

This method is reminiscent of alignment approaches in paraphrasing (e.g. (Grigonytè et al., 2010)), but note that our problem is more restricted in that we have the same number of words, and in most cases identical words. We use different distance and similarity metrics, to ensure robustness across different kinds of errors. The third metric is the least important, as learners can shift tokens far from their original slot, and thus it is given a low weight. The only reason to use it at all is to distinguish cases where more than one target word is a strong possibility, favoring the closer one.

The formula for the cost between source and target words w_s and w_t is given in (4), where the distance metrics are averaged and normalized by the length of the target word w_t. This length is also used to convert the similarity measures into distances, as in (5). We non-exhaustively tested different weight distributions on about half the data, and our final set is given in (6), where slightly less weight is given for the longest common substring and only a minor amount for the displacement score.

(4) $cost(w_s, w_t) = \theta_1 Displ(w_s, w_t) + \frac{\theta_2 Lev(w_s,w_t)+\theta_3 d_{LCSq}(w_s,w_t)+\theta_4 d_{LCSt}(w_s,w_t)}{3\times len(w_t)}$

(5) $d_{LCS}(w_s, w_t) = len(w_t) - LCS(w_s, w_t)$

(6) $\theta_1 = 0.05; \theta_2 = 1.0; \theta_3 = 1.0; \theta_4 = 0.7$

In calculating Levenshtein distance, we hand-created a small table of weights for insertions, deletions, and substitutions, to reflect likely modifications in Hebrew. All weights can be tweaked in the future, but we have observed good results thus far.

The total alignment is the one which minimizes the total cost (7). A is an alignment between the learner sentence s and a given correct sentence t. Alignments to NULL have a cost of 0.6, so that words with high costs can instead align to nothing.

(7) $align = \arg\min_A \sum_{(w_s,w_t)\in A} cost(w_s, w_t)$

4.3 Extracted features

We extract three different types of features; as these have different feature sets, we correspondingly have three different classifiers, as detailed in sec. 5.1. They are followed by a fourth classifier that tallies up the results of these three classifiers.

4.3.1 Intra-token features

Based on the token alignments, it is straightforward to calculate differences within the tokens and thus to determine values for many features (e.g., a deleted letter in a prefix). We calculate such *intra-token* feature vectors for each word-internal error.

For instance, consider the learner attempt (8b) for the target in (8a). We find in the learner answer two intra-token errors: one in *hmtnwt* (cf. *hmtnh*), where the fem.pl. suffix *-wt* has been substituted for the fem.sg. ending *-h*, and another in *hnw* (cf. *qnw*), where *h* has been substituted for *q*. These two errors yield the feature vectors presented as example cases in table 1.

(8) a. *haM hN eilmw hrbh ksP*
Q they.FEM paid much money
bebil hmtnh$_1$ ehN qnw$_2$?
for the-gift which-they.FEM bought ?
'Did they pay much money for the gift that they bought?'

b. *haM hN eilmw hrbh ksP bebil* ***hmtnwt***$_1$ *ehN* ***hnw***$_2$ *?*

Features 1 and 11 in table 1 are the POS tags of the morphemes *preceding* and *following* the erroneous morpheme, respectively. The POS tag of the morpheme containing the error is given by feature 2, and its person, gender, and number by feature 3. The remaining features describe the error itself (f. 6–8), as well as its word-internal context, i.e., both its left (f. 4–5) and right (f. 9–10) contexts.

The context features refer to individual character slots, which may or may not be occupied by actual characters. For example, since the error in *hmtn**wt*** is word-final, its two right-context slots are empty, hence the '#' symbol for both features 9 and 10.

The feature values for these character slots are generally not literal characters, but rather abstract labels representing various categories, most of which

	Features	***h**nw*	*hmtn**wt***
1.	Preceding POS	PRP	H
2.	Current POS	VB	NN
3.	Per.Gen.Num.	3cp	-fs
4.	Left Context (2)	#	R2
5.	Left Context (1)	#	R3
6.	Source String	h	wt
7.	Target String	q	h
8.	Anomaly	h→q	wt→h
9.	Right Context (1)	R2	#
10.	Right Context (2)	INFL-SFX	#
11.	Following POS	yyQM	REL

Table 1: Intra-token feature categories

	Features	**Seq. 1**	**Seq. 2**
1.	Anomaly	TRNS	SHFT
2.	Sequence Label	RB↔VP	PP
3.	Head Per.Gen.Num.	3cp	---
4.	Head POS.(Binyan)	VB.PIEL	IN
5.	Sequence-Initial POS	VB	IN
6.	Sequence-Final POS	RB	JJ
7.	Left POS (Learner)	RB	NNP
8.	Right POS (Learner)	IN	RB
9.	Left POS (Target)	RB	RB
10.	Right POS (Target)	IN	yyDOT
11.	Sequence Length	2	2
12.	Normalized Error Cost	0.625	0.250
13.	Sent-Level@Rank	200@2	200@2

Table 2: Inter-token feature categories

are morphological in nature. In *hmtn**wt***, for example, the two left-context characters *t* and *n* are the second and third radicals of the root, hence the feature values R2 and R3, respectively.

4.3.2 Inter-token features

The inter-token features encode anomalies whose scope is not confined to a particular token. Such anomalies include token displacements and missing tokens. We again use the Levenshtein algorithm to detect inter-token anomalies, but we disable the substitution operation here so that we can link up corresponding deletions and insertions to yield "shifts."

For example, suppose the target is A B C D, and the learner has D A B C. Without substitutions, the minimal cost edit sequence is to delete D from the beginning of the learner's input and insert D at the end. Merging the two operations results in a D shift.

The learner sentence in (9b) shows two inter-token anomalies with respect to the target in (9a). First, the learner has transposed the two tokens in sequence 1, namely the verb *dibrw* ('speak-PAST') and the adverb *tmid* ('always'). Second, sequence 2 (the PP *beph hybrit*, 'in the Hebrew language') has been shifted from its position in the target sentence.

(9) a. *barC ieral la* ***dibrw***$_1$
in-land-of Israel not speak-PAST
tmid$_1$ ***beph***$_2$ ***hybrit***$_2$.
always in-the-language the-Hebrew .

b. *barC ieral* ***beph***$_2$ ***hybrit***$_2$ *la* ***tmid***$_1$ ***dibrw***$_1$.

Table 2 presents the inter-token feature vectors for the two anomalies in (9b). After *Anomaly*, the next three features provide approximations of phrasal properties, e.g., the phrasal category and head, based on a few syntactically-motivated heuristics. *Sequence Label* identifies the lexical or phrasal category of the shifted token/token-sequence (e.g., PP). Note that sequence labels for transpositions are special cases consisting of two category labels separated by an arrow. *Head Per.Gen.Num* and *Head POS.(Binyan)* represent the morphosyntactic properties of the sequence's (approximate) head word, namely its person, gender, and number, and its POS tag. If the head is a verb, the POS tag is followed by the verb's *binyan* (i.e., verb class), as in VB.PIEL.

The cost feature, *Normalized Error Cost*, is computed as follows: for missing, extra, and transposed sequences, the cost is simply the sequence length divided by the sentence length. For shifts, the sequence length and the shift distance are summed and then divided by the sentence length. *Sent-Level@Rank* indicates both the difficulty level of the exercise and the word-order rank of target sentence to which the learner sentence has been matched.

4.3.3 Global features

In addition to errors, we also look at *global features* capturing global trends in a sentence, in order to integrate information about the learner's overall performance on a sentence. For example, we note the percentage of target POS bigrams present in the learner sentence (*POS recall*). Table 3 presents the global features. The two example feature vectors are those for the sentences (8b) and (9b) above.

	Features	**Ex. (8b)**	**Ex. (9b)**
1.	POS Bigram Recall	2.000	1.273
2.	LCSeq Ratio	2.000	1.250
3.	LCStr Ratio	1.200	0.500
4.	Relaxed LCStr Ratio	2.000	0.500
5.	Intra-token Error Count	1.500	0.000
6.	Inter-token Error Count	0.000	1.500
7.	Intra-token Net Cost	1.875	0.000
8.	Norm. Aggregate Displ.	0.000	0.422
9.	Sentence Level	200	200

Table 3: Global feature categories

Except for feature 9 (*Sentence Level*), every feature in table 3 is multiplied by a weight derived from the sentence level. These weights serve either to penalize or compensate for a sentence's difficulty, depending on the feature type. Because features 1–4 are "positive" measures, they are multiplied by a factor proportional to the sentence level, namely $l = 1 \ldots 4$, whose values correspond directly to the levels 150–300+, respectively. Features 5–8, in contrast, are "negative" measures, so they are multiplied by a factor *inversely* proportional to l, namely $\frac{5-l}{4}$.

Among the positive features, *LCSeq* looks for the longest common subsequence between the learner sentence and the target, while *LCStr Ratio* and *Relaxed LCStr Ratio* both look for longest common substrings. However, *Relaxed LCStr Ratio* allows for token-internal anomalies (as long as the token itself is present) while *LCStr Ratio* does not.

As for the negative features, the two *Error Count* features simply tally up the errors of each type present in the sentence. The *Intra-token Net Cost* sums over the token-internal Levenshtein distances between corresponding learner and target tokens. *Normalized Aggregate Displacement* is the sum of insertions and deletions carried out during inter-token alignment, normalized by sentence length.

5 Two-phase classification

To combine the features for individual phenomena, we run a two-phase classifier. In the first phase, we classify each feature vector for each phenomenon into a level. In the second phase, we aggregate over this output to classify the overall learner level.

We use two-phase classification in order to: 1) modularize each individual phenomenon, meaning that new phenomena are more easily incorporated into future models; 2) better capture sparsely-represented phenomena, by aggregating over them; and 3) easily integrate other exercise types simply by having more specialized phase 1 classifiers and by then integrating the results into phase 2.

One potential drawback of two-phase classification is that of not having gold standard annotation of phase 1 levels or even knowing for sure whether individual phenomena can be classified into consistent and useful categories. That is, even if a 200-level learner makes an error, that error is not necessarily a 200-level *error*. We discuss this next.

5.1 Classifying individual phenomena

With our three sets of features (sec. 4), we set up three classifiers. Depending upon the type, the appropriate classifier is used to categorize each phase 1 vector. For classification, every phase 1 vector is assigned a single learner level. However, this assumes that each error indicates a unique level, which is not always true. A substitution of *i* for *w*, for example, may largely be made by 250-level (intermediate) learners, but also by learners of other levels.

One approach is to thus view each phenomenon as mapping to a set of levels, and for a new vector, classification predicts the *set* of appropriate levels, and their likelihood. Another approach to overcome the fact that each uniquely-classified phenomenon can be indicative of many levels is to rely on phase 2 to aggregate over different phenomena. The advantage of the first approach is that it makes no assumptions about individual phenomena being indicative of a single level, but the disadvantage is that one may start to add confusion for phase 2 by including less relevant levels, especially when using little training data. The second approach counteracts this confusion by selecting the most prototypical level for an individual phenomenon (cf. criterial features in (Hawkins and Buttery, 2010)), giving less noise to phase 2. We may lose important non-best level information, but as we show in sec. 6, with a range of classifications from phase 1, the second phase can learn the proper learner level.

In either case, from the perspective of training, an individual phenomenon can be seen, in terms of level, as the set of learners who produced such a phenomenon. We thus approximate the level of each

Feature type	Feature type
1. 100-level classes	7. Intra-token error sum
2. 150-level classes	8. Inter-token error sum
3. 200-level classes	9. Sentences attempted
4. 250-level classes	10. 250-level attempts
5. 300-level classes	11. 300-level attempts
6. Composite error	

Table 4: Feature categories for learner level prediction

phenomenon by using the level of the learner from the gold standard training data. This allows us not to make a theoretical classification of phenomena (as opposed to taxonomically labeling phenomena).

5.2 Predicting learner level

We aggregate the information from phase 1 classification to classify overall learner levels. We assume that the set of all individual phenomena and their quantities (e.g., proportion of phenomena classified as 200-level in phase 1) characterize a learner's level (Hawkins and Buttery, 2010). The feature types are given in table 4. Features 1–6 are discussed in sec. 6.1; features 7–8 are (normalized) sums; and the rest record the number of sentences attempted, broken down by intended level of the sentence. Lower-level attempts are not included, as they are the same values for nearly all students. When we incorporate other exercise types in the future, additional features can be added—and the current features modified—to fold in information from those exercise types.

An example To take an example, one of our (300+) learners attempts four sentences, giving four sets of global features, and makes four errors, for a total of eight phase 1 individual phenomena. One phenomenon is automatically classified as 100-level, one as 150, four as 200, one as 250, and one as 300+. Taking the 1-best phase 1 output (see section 6.3), the phase 2 vector in this case is as in (10a), corresponding directly to the features in table 4.

(10) a. 0.25, 0.25, 1.00, 0.25, 0.25, 2.00, 0.50, 0.50, 4, 1, 0

b. 0.25, 0.00, 1.00, 0.25, 0.00, 1.625, 0.00, 0.50, 4, 1, 0

In training, we find a 300+-level learner with a very similar vector, namely that of (10b). Depending upon the exact experimental set-up (e.g., $k_2 = 1$, see section 6.3), then, this vector helps the system to correctly classify our learner as 300+.

6 Evaluation

6.1 Details of the experiments

We use TiMBL (Daelemans et al., 2010; Daelemans et al., 1999), a memory-based learner (MBL), for both phases. We use TiMBL because MBL has been shown to work well with small data sets (Banko and Brill, 2001); allows for the use of both text-based and numeric features; and does not suffer from a fragmented class space. We mostly use the default settings of TiMBL—the IB1 learning algorithm and overlap comparison metric between instances—and experiment with different values of k.

For prediction of phenomenon level (phase 1) and learner level (phase 2), the system is trained on data from placement exams previously collected in a Hebrew language program, as described in sec. 2. With only 38 learners, we use leave-one-out testing, training on the data from the 37 other learners in order to run a model on each learner's sentences. All of phase 1 is completed (i.e., automatically analyzed) before training the phase 2 models. As a baseline, we use the majority class (level 150); choosing this for all learners gives an accuracy of 34.2% (13/38).[2]

Phase 1 probability distributions Because TiMBL retrieves all neighbor with the k nearest distances rather than the k nearest neighbors, we can use the number of neighbors in phase 1 to adjust the values of, e.g., *150-level classes*. For example, the output from phase 1 for two different vectors might be as in (11). Both have a distribution of $\frac{2}{3}$ 150-level and $\frac{1}{3}$ 200-level; however, in one case, this is based on 6 neighbors, whereas for the other, there are 12 neighbors within the nearest distance.

(11) a) 150:4, 200:2 b) 150:8, 200:4

With more data, we may have more confidence in the prediction of the second case. The *classes* features (f_x) of table 4 are thus calculated as in (12), multiplying counts of each class ($c(x)$) by their probabilities ($p(x)$).

[2]We are aware that the baseline is not very strong, but the only alternative would be to use a classifier since we observed no direct correlation between level and number of errors.

k	Intra	Inter	Global	Overall
1	28.1%	38.6%	34.4%	34.7%
3	34.2%	44.6%	44.6%	**41.9%**
5	34.2%	37.1%	36.7%	36.3%

Table 5: Phase 1 accuracies

		Phase 1			
		1-best	$k_1 = 1$	$k_1 = 3$	$k_1 = 5$
Phase 2	Max	42.1	47.4	57.9	42.1
	$k_2 = 1$	**60.5**	57.9	36.8	39.5
	$k_2 = 3$	42.1	44.7	44.7	42.1
	$k_2 = 5$	39.5	42.1	44.7	**60.5**

Table 6: Phase 2 accuracies for different phase 1 settings

		System					
	1-best	100	150	200	250	300+	Acc.
Gold	100	6	1				6/7
	150	2	7	3		1	7/13
	200		2	7	1	1	7/11
	250			1		1	0/2
	300+			1	1	3	3/5

	k=5	100	150	200	250	300+	Acc.
Gold	100	5	2				5/7
	150	2	9	2			9/13
	200		2	9			9/11
	250			2			0/2
	300+		1	4			0/5

Table 7: Classification confusion matrices

(12) $f_x = \sum_{phase1} c(x)p(x)$

The *Composite error* feature combines all *classes* features into one score, inversely weighing them by level, so that more low-level errors give a high value.

6.2 Predicting phenomena levels

We first evaluate phase 1 accuracy, as in table 5. Using $k = 3$ gives the best phase 1 result, 41.9%. We evaluate with respect to the single-best class, i.e., the level of the learner of interest. Accuracy is the percentage of correctly-classified instances out of all instances. We assume an instance is classified correctly if its class corresponds to the learner level.

Accuracy is rather low, at 41.9%. However, we must bear in mind that we cannot expect 100% accuracy, given that individual phenomena do not clearly belong to a single level. Intra-token classification is lowest, likely due to greater issues of sparsity: random typos are unlikely to occur more than once.

6.3 Predicting learner level

For the second phase, we use different settings for phase 1 instances. The results are shown in table 6. The overall best results are reached using single-best classification for phase 1 and $k = 1$ for phase 2, giving an accuracy of 60.5%. Note that the best result does not use the best performing setting for phase 1 but rather the one with the lowest performance for phase 1. This shows clearly that optimizing the two phases individually is not feasible. We obtain the same accuracy using $k = 5$ for both phases.

Since we are interested in how these two settings differ, we extract confusion matrices for them; they are shown in table 7. The matrices show that the inherent smoothing via the k nearest neighbors leads to a good performance for lower levels, to the exclusion of levels higher than 200. The higher levels are also the least frequent: the $k_1 = 5/k_2 = 5$ case shows a bias towards the overall distribution of levels, whereas the 1-best/$k_2 = 1$ setting is more likely to guess neighboring classes. In order to understand the results better, we also calculated weighted Fleiss' kappa for all the results in table 6. Based on kappa, the best result is based on the setting $k_1 = 1/k_2 = 1$ (0.647), followed by 1-best/$k_2 = 1$ (0.639). The kappa for $k_1 = 5/k_2 = 5$ is significantly lower (0.503).

We are also interested in whether we need such a complex system: phase 1 can outputs a distribution of senses ($k_1 = n$), or we can use the single best class as input to phase 2 (*1-best*). In a different vein, phase 2 is a machine learner ($k_2 = n$) trained on phase 1 classified data, but could be simplified to take the maximum phase 1 class (*Max*). The results in table 6 show that using the single-best result from phase 1 in combination with $k_2 = 1$ provides the best results, indicating that phase 2 can properly aggregate over individual phenomena (see sec. 5.1). However, for all other phase 2 settings, adding the distribution over phase 1 results increases accuracy. Using the maximum class rather than the machine learner in phase 2 generally works best in combination with more nearest neighbors in phase 1, provid-

ing a type of smoothing. However, using the maximum has an overall detrimental effect.

While the results may not be robust enough to deploy, they are high, given that this is only *one* type of exercise, and we have used a very small set of training data. When performing the error analysis, we found one student who had attempted only half of the sentences—generally a sign of a low level—who was put into level 300. We assume this student performed better on other exercises in the exam. Given this picture, it is not surprising that our system consistently groups this student into a lower level.

6.4 Ablation studies

We are particularly interested in how the different phases interact, 1) because one major way to expand the system is to add different exercises and incorporate them into the second phase, and 2) because the results in table 6 show a strong interdependence between phases. We thus performed a set of experiments to gauge the effect of different types of features. By running ablation studies—i.e., removing one or more sets of features (cf. e.g. (Yannakoudakis et al., 2011))–we can determine their relative importance and usefulness. We run phase 2 ($k = 1$) using different combinations of phase 1 classifiers (1-best) as input. The results are presented in table 8.

Intra	Inter	Global	Acc.
Y	Y	Y	**60.5%**
Y	Y	N	47.4%
Y	N	N	42.1%
N	Y	Y	42.1%
N	Y	N	42.1%
Y	N	Y	36.8%
N	N	Y	34.2%

Table 8: Ablation studies, evaluating on phase 2 accuracy

Perhaps unsurprisingly, the combination of all feature types results in the highest results of 60.5%. Also, using only one type of features results in the lowest performance, with the global features being the least informative set, on par with the baseline of 34.2%. If we use only two feature sets, removing the global features results in the least deterioration. Since these features do not directly model errors but rather global sentence trends, this is to be expected. However, leaving out inter-token features results in the second-lowest results (36.8%), thus showing that this set is extremely important—again not surprising given that we are working with an exercise designed to test word order skills.

7 Summary and Outlook

We have developed a system for predicting the level of Hebrew language learners, using only a small amount of targeted language data. We have predicted level based on a single placement exam exercise, finding a surprising degree of accuracy despite missing much of the information normally used on such exams. We accounted for the problem of data sparsity by breaking the problem into a two-phase classification and through our choice of learning algorithm. The classification process isolates individual errors and linguistic constructions which are then aggregated into a second phase; such a two-step process allows for easy integration of other exercises and features in the future. The aggregation of information allows us to smooth over sparse features.

In the immediate future, we are integrating other exercises, to improve the overall accuracy of level prediction (i.e., the second phase) and make automatic testing more valid (cf. e.g. (Fulcher, 1997)), while at the same time incorporating more linguistic processing for more complex input. For example, with question formation exercises, no closed set of correct answers exists, and one must use parse tree distance to delineate features. With multiple exercises, we have plans to test the system with incoming students to the Hebrew program at our university.

Acknowledgments

We would like to thank Ayelet Weiss for help throughout, as well as Chris Riley and Amber Smith. Part of this work was funded by the IU Jewish Studies Center, as well as by the Institute for Digital Arts and Humanities and the Data to Insight Center at IU. We also thank the three anonymous reviewers for their comments.

References

Dora Alexopoulou, Helen Yannakoudakis, and Ted Briscoe. 2010. From discriminative features to learner

grammars: a data driven approach to learner corpora. Talk given at *Second Language Research Forum*, University of Maryland, October 2010.

Yigal Attali and Jill Burstein. 2006. Automated essay scoring with e-rater v.2. *Journal of Technology, Learning, and Assessment*, 4(3), February.

Michele Banko and Eric Brill. 2001. Mitigating the paucity-of-data problem: Exploring the effect of training corpus size on classifier performance for natural language processing. In *Proceedings of HLT 2001, First International Conference on Human Language Technology Research*, pages 253–257, San Diego, CA.

Adriane Boyd. 2010. EAGLE: an error-annotated corpus of beginning learner German. In *Proceedings of LREC-10*, Valetta, Malta.

Walter Daelemans, Antal van den Bosch, and Jakub Zavrel. 1999. Forgetting exceptions is harmful in language learning. *Machine Learning*, 34:11–41.

Walter Daelemans, Jakub Zavrel, Ko van der Sloot, and Antal van den Bosch. 2010. Timbl: Tilburg memory based learner, version 6.3, reference guide. Technical report, ILK Research Group. Technical Report Series no. 10-01.

Robert Dale and Adam Kilgarriff. 2011. Helping our own: The HOO 2011 pilot shared task. In *Proceedings of the Generation Challenges Session at the 13th European Workshop on Natural Language Generation*, pages 242–249, Nancy, France, September.

Ana Díaz-Negrillo and Jesús Fernández-Domínguez. 2006. Error tagging systems for learner corpora. *Spanish Journal of Applied Linguistics (RESLA)*, 19:83–102.

Markus Dickinson and Joshua Herring. 2008. Developing online ICALL exercises for Russian. In *The 3rd Workshop on Innovative Use of NLP for Building Educational Applications (ACL08-NLP-Education)*, pages 1–9, Columbus, OH.

Markus Dickinson, Ross Israel, and Sun-Hee Lee. 2011. Developing methodology for Korean particle error detection. In *Proceedings of the Sixth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 81–86, Portland, OR, June.

Glenn Fulcher. 1997. An English language placement test: issues in reliability and validity. *Language Testing*, 14(2):113–138.

Yoav Goldberg and Michael Elhadad. 2011. Joint Hebrew segmentation and parsing using a PCFGLA lattice parser. In *Proceedings of ACL-HLT*, pages 704–709, Portland, OR, June.

Gintarè Grigonytè, João Paulo Cordeiro, Gaël Dias, Rumen Moraliyski, and Pavel Brazdil. 2010. Paraphrase alignment for synonym evidence discovery. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING)*, pages 403–411, Beijing, China.

John A. Hawkins and Paula Buttery. 2010. Criterial features in learner corpora: Theory and illustrations. *English Profile Journal*, 1(1):1–23.

John A. Hawkins and Luna Filipović. 2010. *Criterial Features in L2 English: Specifying the Reference Levels of the Common European Framework*. Cambridge University Press.

Alon Itai and Shuly Wintner. 2008. Language resources for Hebrew. *Language Resources and Evaluation*, 42(1):75–98.

Alla Rozovskaya and Dan Roth. 2011. Algorithm selection and model adaptation for ESL correction tasks. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 924–933, Portland, OR, June.

Khalil Sima'an, Alon Itai, Yoad Winter, Alon Altman, and N. Nativ. 2001. Building a tree-bank of Modern Hebrew text. *Traitment Automatique des Langues*, 42(2).

Joel Tetreault and Martin Chodorow. 2008. The ups and downs of preposition error detection in ESL writing. In *Proceedings of COLING-08*, Manchester.

Helen Yannakoudakis, Ted Briscoe, and Ben Medlock. 2011. A new dataset and method for automatically grading ESOL texts. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 180–189, Portland, OR, June.

Shlomo Yona and Shuly Wintner. 2008. A finite-state morphological grammar of Hebrew. *Natural Language Engineering*, 14(2):173–190.