NORTHEASTERN UNIVERSITY
GRADUATE SCHOOL OF COMPUTER SCIENCE
M.S. THESIS APPROVAL FORM

THESIS TITLE: Applying Unsupervised Grammar Induction to OCR Error Correction

AUTHOR: Samuel R. Scarano

M.S. Thesis Approved as an Elective towards the Master of Science Degree in Computer Science.

Thesis Advisor  

Date

Thesis Reader

Date

Thesis Reader

Date

Thesis Reader

Date

GRADUATE SCHOOL APPROVAL:

Director, Graduate School

Date

COPY RECEIVED IN GRADUATE SCHOOL OFFICE:

Recipient’s Signature

Date

Distribution: Once completed, this form should be scanned and attached to the front of the electronic Dissertation document (page 1). An electronic version of the document can then be uploaded to the Northeastern University-UMI website.
Applying Unsupervised Grammar Induction to Improve OCR Error Correction

Samuel R. Scarano
Advisor: David A. Smith
December 17, 2014

Abstract

This thesis presents a system for correcting errors from optical character recognition (OCR) software. As a noisy-channel error correction system, it uses a language model to provide a prior over the true text. For this purpose, we introduce a lexicalized version of Klein and Manning’s Dependency Model with Valence, a grammar that is trained without structure annotation. The novel language model provides error correction performance that is slightly better than a 4-gram baseline on a corpus of historical English text. When interpolated with the 4-gram model, a relative reduction in word error rate of 32.1% is achieved, which is 2.5% more than the 4-gram model alone. However, the improvement is primarily attributable not to the modeling of dependency structure, but rather to the modeling of word classes, which is included therein. We determined this by achieving a similar improvement while constraining the model to a uniformly right-branching structure.
# Contents

1 Introduction ................................................. 5  
2 Background .................................................. 6  
3 Related Work .................................................. 7  
  3.1 Unsupervised Induction of Dependency Grammars ................. 7  
  3.2 Structured Language Modeling .................................. 9  
  3.3 OCR Error Correction .......................................... 10  
4 Models ......................................................... 11  
  4.1 Grammar Induction Using the DMV ............................. 11  
  4.2 Lexicalizing the DMV ......................................... 12  
  4.3 Contrastive Estimation ....................................... 14  
5 Implementation ............................................... 15  
  5.1 Word Clustering ........................................... 15  
  5.2 The Channel Model .......................................... 18  
  5.3 OCR Error Correction ........................................ 18  
6 Evaluation .................................................... 21  
  6.1 Parsing Accuracy ........................................... 21  
  6.2 The Corpus ................................................ 22  
  6.3 OCR Correction Experiments ................................. 24  
  6.4 The Right-branching ILDMV Baseline ......................... 25  
  6.5 Hyperparameters ........................................... 25  
7 Discussion .................................................... 28  
  7.1 Results ................................................... 28  
  7.2 Future Work ............................................... 31
1 Introduction

Optical character recognition (OCR) systems convert images of printed text into character-encoded electronic text. OCR systems are not perfect, and introduce errors at a rate that depends on several factors, including the printing quality, the image scan quality, the match of the character image model to the printing, and the match of the OCR system’s language model (if any) to the document. The goal of automatic OCR error correction is to detect and correct the errors in the OCR system output (without introducing new errors).

While most error-correction systems rely on an n-gram language model to provide a prior distribution over strings, we tested a structured language model, both in isolation, and combined with an n-gram model. While structured language models (SLMs) have been used in error-correction problems such as automatic speech recognition (e.g., Chelba and Jelinek, 2000), this is, to our knowledge, the first use for OCR error correction.

More interestingly, this may be the first use of unsupervised training of a structured language model for an error correction task. The parsers used by structured models are generally trained using supervision (annotated parse trees). The field of unsupervised grammar induction is still young. Even parsers at or near the state-of-the-art achieve parsing accuracy of only 55.7% (on WSJ section 23) (Blunsom and Cohn, 2010).

But we note that the parse accuracy score only captures the ability of the parser to predict the annotation provided by linguists, and may not be a good predictor of its value as a language model. Structure annotation involves a number of decisions that appear arbitrary (such as the Penn TreeBank’s flat internal noun phrase structures). In fact, the most common error made by the unsupervised parser of Klein and Manning (2004) is the choice of determiners as the heads of noun phrases—a decision which, the authors note, some linguists actually support (Abney, 1987).

Our structured language model is based on Klein and Manning’s, which they call the Dependency Model with Valence (DMV). This is a fairly simple model of headword–argument dependency that can be trained without structure annotation. Although it was designed to operate on part-of-speech tags (either annotated or automatically assigned), we adapted it to perform fully lexical parsing. We refer to our model as an interpolated lexical DMV, or ILDMV.

In addition to the standard expectation maximization (EM) training of the DMV, we explore a training method called contrastive estimation (Smith and Eisner, 2005). We test a novel variation of contrastive estimation (a neighborhood function which we call Error), which results in better parsing accuracy on text with punctuation than the previously published neighborhoods.

The ILDMV enables our OCR error correction system to achieve a slightly better reduction in error rate than an n-gram model. This compares favorably with supervised SLMs, which tend to do less well than a good n-gram model. The best-performing variation of the system is contrastively trained using the Error neighborhood. When interpolated with the n-gram model, the drop in error rate improves; on our test set, the interpolated language model achieves an error rate of 3.52%, compared to the n-gram model’s 3.66%.

We believed that this finding provided evidence that the modeling of dependency struc-
ture was helpful in correcting errors in text. However, shortly before finishing this report, we performed an experiment that suggests that most of the benefit of the SLM is due to the incorporation of induced word clusters into the model. The same gains can be obtained by constraining the model to generate dependents in a strictly right-branching structure.

Our OCR correction experiments use a corpus derived from the Women Writers Project for training and testing. It is composed of books spanning the 16th century through the 19th century. The historical nature of the works and the low OCR error rate (5.2%) presented special challenges. The details about the corpus and how it is used are described in section 6.2.

2 Background

Like previous error correction systems for both OCR and speech, ours is based on a generative “noisy channel” model. This means that the sequence of words \( W \) in the original text is modeled by one probability distribution, \( p(W) \), and that the process of generating the observed OCR output \( O \) is a noisy channel, modeled by the conditional distribution \( p(O|W) \). The system therefore tries to find the sequence \( \hat{W} \) that maximizes \( p(\hat{W}|O) \), which is given by Bayes’ rule:

\[
p(W|O) = \frac{p(W)p(O|W)}{p(O)}
\]

A noisy-channel correction system outputs the string that is most likely to be true:

\[
\hat{W} = \arg \max_{W'} p(W')p(O|W)
\]

\( (O \) is constant, so \( p(O) \) drops out. \( p(W) \) (the language model) and \( p(O|W) \) (the channel model) are estimated separately.

By far the most common type of language model is the n-gram model. N-gram models have no latent variables, and can therefore be easily trained on a large unannotated corpus.

Syntactic, or structured, language models attempt to predict the underlying syntactic structure of sentences to aid in estimating their probabilities. We will be primarily concerned with dependency parsing. This is distinct from, though very closely related to phrase-structure, or constituency parsing.

A dependency tree has words as nodes. Each word also has one edge pointing to it from its head. To simplify analysis, the sentence is considered to have an extra word, called the ROOT, which we consider to be at the special token position 0 in each sentence. An example dependency tree is shown in figure 1.

The dependency parsing models we’ll discuss are restricted to projective trees; this means that the set of nodes reachable from node \( i \) (its projection) is an uninterrupted span of the sentence. To put it another way, arcs in a projective tree may not crisscross.
3 Related Work

3.1 Unsupervised Induction of Dependency Grammars

Carroll and Charniak (1992) attempted to induce dependency grammars using only the word strings, not structure annotation. For these experiments, the input data were sample sentences generated from a toy grammar. Their basic method was to randomly initialize the grammar model, re-estimate the parameters of the model using the inside-outside algorithm (Baker, 1979), and repeat the procedure until convergence. This iterative procedure is an instance of expectation maximization (EM) (Dempster et al., 1977).

The procedure failed to learn a grammar that was even close to correct. In a second experiment, the search was guided by grammar constraints. This achieved better results, but the grammar constraints encode prior linguistic knowledge that may be impractical to specify for real languages.

This work highlighted a key difficulty that continues to vex unsupervised grammar induction: the objective function (the likelihood of the data under the model) is riddled with local maxima. In fact, 300 different random initializations of the learning procedure yielded 300 different (bad) grammars.

Paskin (2001) performed similar experiments on actual English text (30 million words of newswire). The model factors the parse into its graphical structure $G$ and the set of resulting dependency relations. Each dependency relation has a head word $h$, a dependent, or argument, $a$, and a direction $d$. The generative story starts by generating the root token, then generates each dependent independently, and recurses. $G$ is assumed to be uniformly distributed. So the estimate of a dependency structure $D$ is given by

$$p(D) = p(G) \prod_{(a,h,d) \in G} p(x_a|x_h,d)$$

where $x_i$ is the word at position $i$. $p(G)$ is constant for each sentence. The parameters of $p(x_a|x_h,d)$ are estimated using EM. The resulting model is evaluated by parsing the structure-annotated WSJ portion of the Penn TreeBank (Marcus et al., 1993). The result is an unimpressive 39.7% recall in predicting undirected dependency relations (that is, forgiving errors in direction if
the predicted dependency has the correct end-points).\footnote{Paskin claims that this is better than a baseline that selects dependencies randomly (31.3%). But curiously, in the comparison given by Klein and Manning (2004), this is below the random baseline, which is stated to be 41.7%. The discrepancy could be due to a number of possible differences in how the text was processed. In any case, it is certainly lower than a baseline that chooses adjacent words as dependencies (i.e., right- or left-branching structures).}

The Dependency Model with Valence (DMV) is similar, but it parses the sequence of part-of-speech (POS) tags, instead of the words themselves (Klein and Manning, 2004). It also introduces learning of parameters that govern $p(G)$. Instead of simply generating all dependents independently, the DMV uses the estimated distribution $p_{\text{stop}}$ to decide whether to continue generating dependents; this distribution is conditioned on whether dependents have previously been generated in that direction (valence). Furthermore, the DMV introduces a “harmonic” initializer that is not random, nor uniform, but rather biased in favor of short-distance dependencies.

The DMV (trained with EM and inside-outside re-estimation) was the first dependency grammar model that achieved better performance (43.2% accuracy) in predicting directed dependencies than a left-branching baseline (33.6%) on the WSJ10 corpus. Note that the left-branching baseline out-performs a random guesser (30.1% directed accuracy), and, for that matter, a right-branching baseline.

The DMV is the inspiration of much subsequent work in unsupervised grammar induction, including the lexicalization we introduce for language modeling. It is described further in section 4.1.

Smith and Eisner (2005) adapted the DMV to use a training procedure they call contrastive estimation (CE), a novel alternative to EM. While EM relies only on positive evidence (grammatical sentences), CE seeks to learn from implicit negative evidence. A “neighborhood” function $N(x)$ is used to generate a set of sentences (specified by a lattice) that are similar to, but different from, $x$. The sentences in this neighborhood are usually (though not always) ungrammatical, and may form a more targeted, and more helpful, body of negative evidence than the entire set of sentences not seen in training (which is, in effect, what EM draws probability mass away from).

The best-performing neighborhood function they evaluated was $\text{Trans1}$, which is the set of sentences in which one adjacent pair of tokens in the input sentence is transposed (e.g., $\langle \text{NNP VBD DT NN} \rangle \in \text{Trans1}(\langle \text{NNP VBD NN DT} \rangle)$). With a uniform initializer and no regularization, the average test-on-train and test-on-test accuracy reported were 48.7% and 48.8%.\footnote{10-fold cross-validation was used over the WSJ10 corpus to arrive at these scores.} This is much better than the 35.2% (for both test-on-train and test-on-test) reported for the EM-trained DMV.\footnote{The discrepancy with the accuracy reported by Klein and Manning (2004) is attributed to a different set of head rules used to convert the constituency annotation to dependency parses. It is also noteworthy that the reported standard deviation among cross-validation folds for the DMV’s test-on-train accuracy is 6.6 percentage points.}

The $\text{Length}$ neighborhood function was also evaluated. $\text{Length}$ is simply the set of all sentences of the same length as the input sentence. This is not as computationally unwieldy as it may sound, because the vocabulary size $V$ is only the number of POS tags, and the set of sentences of a length $n$ is represented as a lattice with only $nV$ arcs. This neighborhood performs best with the harmonic initializer. Without regularization, its average 10-fold cross-validated accuracy is 42.1% on the training set, and 42.3% on the test set. Slightly higher scores were
achieved by adding a Gaussian prior.

An alternative model to the DMV is Extended Valence Grammar (EVG) (Headden et al., 2009). The EVG differs from the DMV by conditioning the dependent tag choice on the valence frame (adjacent vs. non-adjacent) in addition to the head tag and direction. (By contrast, the DMV uses the valence frame only to condition the stop/continue decisions.) To mitigate the extra sparsity, this distribution is smoothed to one that does not condition on the valence frame, using linear interpolation. A lexicalized version, the L-EVG, is also presented. It further conditions the dependent tag on the head word, smoothing to the unlexicalized EVG. Using a more sophisticated training procedure than EM, the L-EVG achieves a directed accuracy of 68.8% on WSJ10, section 23. This is 3.8 percentage points better than the unlexicalized EVG (65.0%), which illustrates the importance of going beyond POS tags and including lexical information.

3.2 Structured Language Modeling

Although we’re not aware of any previous attempts to use structured language modeling for OCR error correction, structured language models (SLMs) have been applied previously to the problem of automatic speech recognition (ASR). ASR is a closely related problem to OCR error correction; in fact, it is solved using the same noisy channel approach, except that the “channel” is the speech signal, as interpreted by an acoustic model. Therefore, the role of the language model in the ASR task is the same: it provides a prior distribution over word sequences.

Chelba and Jelinek (2000) describe an early but successful application of a structured language model to ASR. This approach uses a shift-reduce parser with a model that conditions each word’s probability on the full history of dominating head words and tags. In practice, that history length is limited by pruning parameters in A* search. The Switchboard corpus was used for training and testing. This SLM requires a supervised training procedure, and therefore bootstraps from the subset of Switchboard with structure annotation from the Penn TreeBank. The total training set size is 2.3 million words. When the resulting SLM is applied to the ASR task, the word error rate (WER) is worse than the trigram baseline by an absolute margin of 1.1%. However, a combined model that interpolates the SLM with the trigram model yields a 1.0% improvement over the baseline.

A more modern variation of this is presented by Rastrow et al. (2012). This work also applies a dependency-based SLM for ASR, using a shift-reduce parser. But it introduces an improved smoothing scheme for state history. Hierarchical Jelinek-Mercer interpolation is used, on histories that include both words and tags. Their best-performing variation drops the word, then the tag, in separate interpolation steps, before moving on to the next state. Another key innovation is the use of a relative-entropy-based pruning scheme to decrease the size of the models, which makes large training corpora more practical. Their model is trained on the Broadcast News corpus (EARS BN03), with 42 million words of training, and 45,000 words for testing. The parser is trained on the BN treebank and the WSJ portion of the Penn Treebank. The interpolated model (SLM + 4-gram) provides a 0.6% reduction in WER in the ASR evaluation, versus the 4-gram

---

4 Accuracy for all sentences of section 23 is not reported.
baseline. The SLM is not tested in isolation in the ASR task, but it is tested for corpus perplexity. Its test-set perplexity is 0.6% above the 4-gram baseline; the interpolated model’s is 10% below baseline.

Collins et al. (2005) also applies an SLM to ASR, but takes a different tack. This approach uses the constituency parser of Collins (1999) to add syntactic features to a discriminative model. They used the 1000-best output of a baseline system with a 6-gram (generative) language model as the input to the discriminative re-ranker (a perceptron). The use of discriminative modeling using n-gram features reduced WER by 0.9%; more importantly for the present discussion, the addition of syntactic features reduced WER by a further 0.3%. The syntactic feature templates include production rule instances, and tuples that include words, POS tags, and non-terminal types; each such tuple is populated from particular structural patterns. The parser was trained on the Penn Switchboard treebank; the LM parameters were estimated from Switchboard sections totaling 3.3 million words.

3.3 OCR Error Correction

Among the first uses of an n-gram LM within the noisy channel framework for the problem of correcting OCR errors was presented by Tong and Evans (1996). It uses a bigram LM. Their other important innovation is the use of EM to iteratively improve the character-level error model. It starts with a model that predicts a character error with probability $\alpha$ (a specified parameter), and divides the remaining $1 - \alpha$ mass uniformly among the possible errors. A new error model is estimated from the output of the first correction pass. The next correction pass uses the new model, and so on. This procedure is effective in improving accuracy, and overcomes the need for a training corpus of OCR aligned to correct transcription.

90% of a 33-million-word Ziff-Davis news wire corpus was used for training. Of the remaining 10%, 70 pages (55,699 words) were scanned and converted to text using OCR software. The OCR word error rate was 22.9%, but words containing numbers were excluded, resulting in a test set with an OCR WER of 14.7%. The correction process reduces this to 5.87%. It is not clear to us whether the LM includes character case, or whether the evaluation penalizes case errors.

Kolak et al. (2003) describes a state-machine-based OCR correction system based on a sophisticated but cleanly factored model of OCR. Unlike previous systems, it models and corrects segmentation errors, both where the OCR system erroneously divides a word into two words, and merges two adjacent words into one. Another advantageous feature is a separate case model to predict whether words from the (caseless) LM manifest as lower-case, upper-case, or “leading” case (e.g., Case).

This system requires an aligned OCR–transcription corpus to train its character-level error model. It uses an n-gram LM, with a closed-vocabulary assumption; for evaluation, the words that appear only in the test set are included in the model as unseen vocabulary. The evaluation is performed on an OCR scan of the Christian Bible in French.5 The text was divided into a training set for the LM and other sub-models (90%) and a test set (500 verses from the remaining

---

5 Using an English OCR system; the authors are primarily interested in cross-linguistic OCR.
The evaluation forgives character case differences. Their most basic model (unigram, with no case model or re-segmentation) reduces the WER from 18.31% (the original OCR) to 7.11%. Switching to a trigram model brings the WER to 7.06%. Adding an enhanced case model and allowing re-segmentation brings the WER down to 6.75%.

4 Models

4.1 Grammar Induction Using the DMV

The Dependency Model with Valence is a head-outward generative model of dependency structure.

The DMV only knows about part-of-speech tags, not the words themselves. The use of tags instead of words obviously mitigates the sparsity of the data. It also prevents the DMV from falling into a trap that plagued Paskin’s model, namely, that words that co-occurred frequently because of a semantic relationship tended to be incorrectly assigned a syntactic relationship.

Most experiments on the DMV (and the variations that are derived from it) exploited the part-of-speech annotation provided with the WSJ portion of the Penn TreeBank. This thread of research has also explored the use of automatically induced word clusters as substitutes for annotated POS tags. The use of induced clusters with the DMV was studied most thoroughly by Spitkovsky et al. (2011).

Also note that the WSJ corpus used for the original DMV experiments includes only sentences up to 10 (non-punctuation) tokens long, and that punctuation is removed. This subset of the WSJ corpus is referred to as WSJ10. Removal of un-pronounced punctuation has also been standard practice in subsequent derivative work.

The generative story is as follows. Each word (starting at the pseudo-token root) generates its non-stop arguments on the left side first, from right to left. Then it generates a stop. Then it generates its non-stop arguments on the right side, from left to right, followed by stop. Each stopping decision depends only on the head word’s tag (whether part-of-speech tag or induced word class), the direction (left or right), and adjacency (whether this would be the first argument to be generated in this direction). Each time the model decides not to stop, it must generate an argument tag; this distribution depends on the head word’s tag and the direction (not adjacency).

For example, the probability of the subtree headed by pays in figure 1 is given by:

\[
\text{Pr} \left( \begin{array}{c}
\text{VBZ} \\
\text{pays}
\end{array} \right)
\cdot \text{Pr} \left( \begin{array}{c}
\text{DT} \\
\text{the}
\end{array} \right)
\cdot \text{Pr} \left( \begin{array}{c}
\text{NN} \\
\text{piper}
\end{array} \right)
\cdot \text{Pr} \left( \begin{array}{c}
\text{STOP|VBZ, LEFT, TRUE}
\cdot \text{Pr} \left( \begin{array}{c}
\text{STOP|VBZ, RIGHT, TRUE}
\cdot \text{Pr} \left( \begin{array}{c}
\text{NN|VBZ, RIGHT}
\cdot \text{Pr} \left( \begin{array}{c}
\text{DT} \\
\text{the}
\end{array} \right)
\cdot \text{Pr} \left( \begin{array}{c}
\text{NN} \\
\text{piper}
\end{array} \right)
\cdot \text{Pr} \left( \begin{array}{c}
\text{STOP|VBZ, RIGHT, FALSE}
\end{array} \right)
\end{array} \right)
\end{array} \right)
\end{array} \right)
\]
Formally, let the tree be defined by the function \( \tau(h,d) \), which gives the set of dependents of word \( h \) in direction \( d \). Define \( t_i \) as the tag at position \( i \), and let \( \text{adj}_i(a) \) be true iff word \( i \) is the nearest dependent of its head. The probability of a dependency tree headed by \( h \) is given by

\[
P_{DMV}(\tau,h) = \prod_{d \in \{\text{left},\text{right}\}} \left( \prod_{a \in \tau(h,d)} p_{\text{stop}}(\neg\text{stop}|t_h,d,\text{adj}_\tau(a)) \cdot p_{\text{tag}}(t_a|t_h,d) \cdot P_{DMV}(\tau,a) \right) \cdot p_{\text{stop}}(\text{stop}|t_h,d,[\tau(h,d) = \emptyset])
\]

(4)

The EM procedure locally searches the parameters of \( p_{\text{stop}} \) and \( p_{\text{tag}} \) to maximize the following objective function:

\[
\prod_x \sum_{\tau \in T(x)} P_{DMV}(\tau,0)
\]

(5)

which is the likelihood of the data, marginalized over possible parses (given by \( T(x) \)). Recall that 0 is the index of the root pseudo-token.

Parsing is performed using the Eisner-Satta algorithm for split-head grammars (Eisner and Satta, 1999). This populates a chart that compactly stores the conditional probabilities of all possible parse fragments for the sentence. The inside-outside algorithm (Baker, 1979) is then used to assign an expected count to each stop and attachment event. \( p_{\text{stop}} \) and \( p_{\text{tag}} \) are re-estimated from those expected counts and used in the next EM iteration.

When a single parse is needed for a string (such as when testing the parser’s accuracy against ground truth), the parser performs Viterbi search for the parse that maximizes \( P_{DMV}(\tau,0) \).

The initial \( p_{\text{stop}} \) and \( p_{\text{tag}} \) models are estimated using what Klein and Manning call the “harmonic” initializer. For the first round of EM, all words are considered equally likely to be the root (the immediate dependent of root). Below the root, all dependencies are considered possible, but with a probability inversely proportional to distance. This initializer represents the intuition that nearby words are much more likely to form dependency relationships, without introducing any other bias. It is noted that when a uniform initializer is used, the models have below-random parsing accuracy.

We trained our DMV models using only sentences of up to 10 tokens. Including longer sentences did not improve performance. This is consistent with the findings of others. For example, Spitkovsky et al. (2010) found that parsing accuracy on WSJ training sets with longer sentences plateaus shortly after the maximum sentence length exceeds 10, and decreases after 20.

EM convergence was detected as a relative improvement in likelihood of less than \( 10^{-5} \). No smoothing was used for the training procedure, but we did apply additive smoothing to \( p_{\text{tag}} \) while testing and while training the ILDMV (see next section).

4.2 Lexicalizing the DMV

One trivial way to lexicalize the DMV is to simply go one step further and generate the words from their tags. We refer to this as the LDMV. To express this concisely, we first replace \( p_{\text{tag}} \) with
an abstract model, \( Q \):

\[
P(\tau, h) = \prod_{d \in \{\text{left, right}\}} \left( \prod_{a \in \tau(h,d)} p_{\text{stop}}(\neg \text{stop}|t_h,d,\text{dep}_\tau(a)) \cdot Q(a|h,d) \cdot P(\tau,a) \right) 
\cdot p_{\text{stop}}(\text{stop}|t_h,d,\tau(h,d) = \emptyset)
\] (6)

The original DMV uses the following as \( Q \):

\[
Q_{\text{DMV}}(a|h,d) = p_{\text{tag}}(t_a|t_h,d)
\] (7)

The LDMV lexicalization substitutes the following:

\[
Q_{\text{LDMV}}(a|h,d) = p_{\text{tag}}(t_a|t_h,d) p_{\text{lex}}(x_a|t_a)
\] (8)

where \( p_{\text{lex}} \) is the distribution of words conditioned on their tags. The maximum likelihood estimate of \( p_{\text{lex}} \) is trivially extracted from the corpus.

One important property of the LDMV is that its parsing behavior is the same as the DMV, because \( p_{\text{lex}}(x_a|t_a) \) does not vary among possible parses.

The LDMV is only slightly more useful as a language model, because it only allows context to inform correction decisions at arm’s length. In fact, a choice between candidate words with the same tag can be made solely on the basis of unigram probability.

A more sophisticated lexicalization combines the LDMV with a fully lexical bigram model of dependency. This is the Interpolated Lexical DMV, or ILDMV. It is specified by equation 6 with following \( Q \) model:

\[
Q_{\text{ILDMV}}(a|h,d) = (1 - \beta)Q_{\text{LDMV}}(a|h,d) + \beta(\gamma_{h,d}p_{\text{word}}(x_a|x_h,d) + (1 - \gamma_{h,d})p_{\text{word}}(x_a|d))
\] (9)

The interpolation weight \( \beta \) is set as a hyperparameter. \( \gamma_{h,d} \) is a Witten–Bell smoothing weight (Bell et al., 1990).\(^6\)

The final expression for the ILDMV is:

\[
P_{\text{ILDMV}}(\tau,h) = \prod_{d \in \{\text{left, right}\}} \left( \prod_{a \in \tau(h,d)} p_{\text{stop}}(\neg \text{stop}|t_h,d,\text{dep}_\tau(a)) \cdot Q_{\text{ILDMV}}(a|h,d) \cdot P(\tau,a) \right) 
\cdot p_{\text{stop}}(\text{stop}|t_h,d,\tau(h,d) = \emptyset)
\] (10)

We will also refer to the joint probability of a string \( x \) and parse \( \tau \) as follows:

\[
p_{\text{ILDMV}}(x,\tau) = P_{\text{ILDMV}}(\tau,0)
\] (11)

\(^6\)We also tried smoothing \( p_{\text{word}} \) directly to \( Q_{\text{LDMV}} \) (by setting \( \beta \) by Witten–Bell, with \( \gamma_{h,d} = 1 \)), but this resulted in inferior performance.
The ILDMV is not trained the same way as the DMV. Instead, a vanilla DMV is trained first using the EM procedure described in section 4.1. Then that model is used to parse the training set. \( p_{\text{word}} \) is estimated from the single best parse of each sentence. Additive smoothing (with \( \alpha = 1 \)) is applied to \( p_{\text{tag}} \) for parsing. We also tried estimating the LDMV using inside-outside expected counts instead of the best parse, but this resulted in worse OCR error rates.\(^7\)

### 4.3 Contrastive Estimation

It is well known that EM’s maximum likelihood objective does not approximate parse accuracy. Even when initialized with correct dependencies, EM pushes the model away from this optimal state. This “oracle” training experiment was performed, for example, by Spitkovsky et al. (2010), who found that EM training on WSJ10 took away about 15 percentage points.

Contrastive estimation (CE) is a generalization of EM which seeks to optimize a different objective. While EM’s objective is globally conditioned, CE allows the objective to be conditioned in a purposeful way. A “neighborhood” function \( N \) maps each positive training example to a set of negative examples.

The purpose of training is to optimize the following log-likelihood objective function

\[
L_N(\theta) = \sum_x \log p(x|N(x);\theta)
\]

where \( \theta \) is the vector of model parameters, and \( p(x;\theta) \) is marginalized over all possible parses of \( x \). The parameters correspond to the events of the parsing model (e.g., the DMV), but in a log-linear setting instead of a generative setting. So

\[
p(x|x \in N(x);\theta) = \frac{\sum_{\tau \in \mathcal{T}(x)} \exp (\theta \cdot f(x,\tau))}{\sum_{x' \in \mathcal{N}(x),\tau \in \mathcal{T}(x')} \exp (\theta \cdot f(x',\tau))}
\]

where \( f(x,\tau) \) is a feature vector function which counts the number of instances of each of the model’s event types that occur in \((x,\tau)\). Training proceeds by hill-climbing \( L_N \) using L-BFGS (Liu and Nocedal, 1989). This requires the gradient:

\[
\frac{\partial L_N}{\partial \theta_i} = \sum_x E_{\theta}[f_i|x] - E_{\theta}[f_i|N(x)]
\]

Both expectations are estimated using inside-outside. The first is the expected count of event \( f_i \) in the parse of \( x \); the second is the expected count of \( f_i \) in the parse of the lattice \( N(x) \).

The \textsc{Trans1} and \textsc{Length} neighborhood functions, which were discussed in section 3.1 resulted in comparatively good parsing performance. We introduce a novel neighborhood called \textsc{Error}. \textsc{Error}(\( x \)) contains \( x \), and adds a randomly labeled alternative arc (the “error”) for each arc in \( x \). For example, \( \langle \text{NNP VBD CC NN} \rangle \) may be in \textsc{Error}(\( \langle \text{NNP VBD NN DT} \rangle \)). The motivation for \textsc{Error} is that the resulting objective function is a measure of the model’s resilience

\(^7\)It also resulted in very, very large models, diluted by low-probability bigrams that were “observed” in very unlikely parses.
to token-by-token substitutions, which is typically what results from OCR errors. It can also
be thought of as a small random subset of \( \text{LENGTH}(x) \)—which can be trained on much more
efficiently, due to the smaller size of the lattice.

Lexicalization of CE models is similar to the EM-trained model, but involves an extra step.
Because the CE tag model is log-linear and not globally conditioned, it cannot be directly inter-
polated with the lexical model. Instead, one more pass is made over the parser training (length
\( \leq 10 \)) to re-estimate a globally conditioned model (using inside-outside). The resulting tag model
is interpolated with the lexical model (following equation 9).

5 Implementation

5.1 Word Clustering

Once again, the DMV model parses sequences of tags, not word tokens. Klein and Manning
(2004) used the Penn TreeBank, and therefore were able to use the annotated part-of-speech
tags associated with each terminal. Of course, text typically does not come with part-of-speech
annotation, so Klein and Manning also experiment with training and parsing using automatically
induced word clusters as stand-ins for annotated POS tags. They used the linear distributional
word clustering algorithm presented by Schütze (1995) to assign words to clusters.\(^8\) Spitkovsky
et al. (2011) found that using clusters found by the Brown word clustering algorithm (Brown
et al., 1992) can result in parsing performance on a par with the Schütze clusters if the right
number-of-clusters parameter is specified.

The Brown algorithm finds a clustering \( C \) that maps words to clusters. It defines the quality
of a clustering as the likelihood of the data under a bigram model of word clusters, with an extra
step to generate words from their clusters. That is:

\[
\mathcal{L}_C(w_1, \ldots, w_n) = \prod_{i=1}^{n} p_{\text{cluster}}(C(w_i)|C(w_{i-1}))p_{\text{lex}}(w_i|C(w_i))
\]

(15)

where \( C(w) \) is the cluster of word \( w \). The algorithm begins by assigning each word to its own
cluster. It then iteratively performs greedy merges of cluster pairs which are chosen to maximize
\( \mathcal{L}_C \) of the training data. It stops after \( V - c \) cluster merge operations, where \( V \) is the number
of words, and \( c \) is the desired number of clusters. Although the sequence of merges can be
used to build a hierarchical clustering, we use only the cluster identity, not its position in such a
hierarchy; that is, \( C \) defines a partition over words.

We used the implementation of the Brown clustering algorithm presented in Liang (2005). In
both the parser accuracy tests and OCR correction tests, we used the parser training data as the
clustering training as well.

The cluster map is caseless. That is, the training set is downcased before training, and at
runtime, each token is downcased before querying for the associated cluster.

\(^8\) In fact, Schütze gives three clustering algorithms, including two that cluster words paired with information about
their contexts. Only the algorithm that clusters bare words (by their context distributions) is used in this work.
The DMV, and subsequent efforts that sprung from it, stripped the text of punctuation marks and parsed only pronounced tokens. However, the OCR correction application requires the processing of all tokens. Therefore, the word clusters include all non-word symbols in addition to words.

In order to enable the parser to handle out-of-vocabulary (OOV) words at test-time, we created a special cluster for them. To ensure that the parser observed this OOV cluster during training in contexts similar to where it would likely show up at test time (and with about the same frequency), we mapped all words that occur only once in the training set to the OOV cluster.

Processing word clusters instead of words slashes the sparsity of the data from which the grammar inducer has to learn. But for this work, it has a second benefit. Our corpus is drawn from a wide range of dates, and has a wide range of spelling variation (as will be discussed in section 6.2. The clustering algorithm appears, from casual observation, to do a good job of assigning variant spellings of the same word to the same cluster. This can be expected to help the parser make more effective use of training data from differing historical periods.

<table>
<thead>
<tr>
<th>Example Cluster 1</th>
<th>Example Cluster 2</th>
<th>Example Cluster 3</th>
<th>Example Cluster 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>lord</td>
<td>thing</td>
<td>on</td>
<td>into (7121)</td>
</tr>
<tr>
<td>king</td>
<td>thynge</td>
<td>upon</td>
<td>profitheth (8)</td>
</tr>
<tr>
<td>author</td>
<td>thyng</td>
<td>about</td>
<td>strikes (4)</td>
</tr>
<tr>
<td>duke</td>
<td>forte</td>
<td>over</td>
<td>decides (4)</td>
</tr>
<tr>
<td>pope</td>
<td>thinge</td>
<td>within</td>
<td>wirh (3)</td>
</tr>
<tr>
<td>empress</td>
<td>precaution</td>
<td>vpon</td>
<td>elevates (3)</td>
</tr>
<tr>
<td>lorde</td>
<td>refource</td>
<td>above</td>
<td>drue (2)</td>
</tr>
<tr>
<td>queene</td>
<td>joynet</td>
<td>near</td>
<td>uate (2)</td>
</tr>
<tr>
<td>emperor</td>
<td>jot</td>
<td>ouer</td>
<td>enrage (2)</td>
</tr>
<tr>
<td>captain</td>
<td>encroachment</td>
<td>behind</td>
<td>enlargeth (2)</td>
</tr>
</tbody>
</table>

Figure 2: The most common 10 words of a few example clusters. The last includes the corpus frequency of each word.

Example cluster 1 in figure 2 contains words that are clearly syntactically related and, for whatever it’s worth to a parser, semantically related. Example cluster 2 also contains nouns, but the semantic theme is less clear. It also illustrates the ability of the clustering algorithm to group variant spellings of the same word, with 4 variations of thing.

Example cluster 4 is representative of a pattern that several clusters follow: a single comparatively common word nearly has a cluster to itself, being grouped only with rare words that unfortunately appear to be largely unrelated. The punctuation clusters tend to follow this pattern. The division between these near-singleton clusters and higher-entropy clusters appears to loosely correspond to the distinction between closed word classes (e.g., prepositions) and open word classes (e.g., nouns).

---

9 Although jot is more familiar as a verb in modern English, it occurs only as a noun in this corpus.
As example cluster 4 illustrates, rare words are not clustered as well as common words.

```
" after but from into men of said there well
" all by god is more on fame they were
, am came good it moft one fay thefe what
- an can great know mr only felf thing when
— (U+2014) and come had let much or she this where
— (U+2015) another d hath life my other so thou which
( any dear have like nature out fome thoe who
) are did he little neither own fuch thus whom
, as do her lord never part take time will
: at done him love no parts than to with
; be each his made nor power that told without
. been every how make not put the two world
? being firft i man nothing rather their up would
a body for if many now reafon them very yet
able both found in may o s then was you
```

Figure 3: The most common word of each of the 150 automatically induced word clusters.

Figure 3 gives a broader view of a clustering at $c = 150$ by showing the most common word in each cluster. We used 150 clusters for the parsing experiments and most of the OCR correction experiments reported. Among 67, 100, 150, and 200, $c = 150$ resulted in the best error correction performance on the dev set using the EM-trained ILDMV. This is not surprising, given the experiments of (Spitkovsky et al., 2011) and (Headden III et al., 2008), who both maximized parsing performance with 100–200 clusters.

Starting from intuitions about POS tags, $c = 150$ may strike the reader as being too large, possibly forcing the algorithm to invent distinctions between words that seem to belong to the same syntactic class. Intuitively, we might want to group, for example, the quantifiers all, each, every, and some (or at least some pair within that set); yet no pair of these words is assigned to the same cluster. We have an informal conjecture about why the optimal number of clusters is larger than intuitions about part-of-speech tags might suggest. While a perfect clustering would make all valuable word-class distinctions without making any unnecessary distinctions, the available clusterings are far from perfect; so we have to pay for higher recall among the valuable distinctions by increasing the number of clusters, which increases the number of useless distinctions. The cost of introducing those useless distinctions is an increase in the sparsity of the data. But we believe that for a bigram model like the DMV, the marginal cost associated with data sparsity at a vocabulary size as small as 150 is low.
5.2 The Channel Model

The channel model \( p(O|W) \) in equation 2 is estimated entirely on the character sequence. It therefore estimates \( p(O|C) \), where \( C \) is the sequence of characters corresponding to and completely determined by \( W \). The model decomposes into the sequence of operations that results in the most probable alignment between \( C \) and \( O \), and assigns the product of the probabilities associated with those operations (they are assumed to be independent).

Let \( O_{i+} \) be the subsequence of \( O \) from position \( i \) to the end. The model is specified as follows:

\[
p(O_{i+}|C_{j+}) = \max \left\{ p(O_i|C_j) \cdot p(O_{i+1+}|C_{j+1+}), p(O_i|\epsilon) \cdot p(O_{i+1+}|C_{j+}), p(\epsilon|C_j) \cdot p(O_{i+}|C_{j+1+}) \right\}
\]

with \( p(O_{O+1+}|C_{C+1}) = 1 \) to break the recursion. The most probable alignment is found using the standard dynamic program for aligning two sequences (Wagner and Fischer, 1974).

The parameters that must be estimated are \( p(o|c) \) (substitutions, which include the case of correct recognition, \( o = c \)), \( p(o|\epsilon) \) (insertions), and \( p(\epsilon|c) \) (deletions), for all \( o \) and \( c \) in their respective alphabets. The training is a 488,000-token subset of the corpus aligned with OCR (the corpus is described in section 6.2). This portion was determined, using heuristics, to be poetry or plays, not prose. (The aligned prose was saved for the OCR correction test set.)

The initial alignment is ambiguous, because different sequences of substitutions, insertions, and deletions can result in the same sequence \( O \). For example, the character sequence \( in \) can be transformed into \( h \) with either a deletion followed by a substitution or a substitution followed by a deletion. We would like the model to learn to resolve this ambiguity in the most likely way (that \( n \) is more likely than \( i \) to transform into \( h \), in this example). We therefore use an EM procedure to train the channel model.\(^\text{10}\) It is seeded with a uniform model that assigns a certain probability (0.9) to replications (substitutions in which the OCR character is the same as the true character), and distributes the remaining probability mass evenly among (unlike) substitutions, insertions, and deletions. It then uses this naive model with the forward-backward algorithm on each line of the channel model training corpus to get expected counts for each particular substitution, insertion, and deletion. The total of these counts is then used to estimate a better model. The new model is used for the next iteration, and this is continued until convergence. The convergence criterion of a relative change in likelihood of less than \( 10^{-5} \) is achieved after 9 iterations. No smoothing was used during this training procedure.

5.3 OCR Error Correction

There are three variants of our OCR error correction system, corresponding to three methods of language modeling: n-gram, lexicalized DMV, and an interpolation between those. Furthermore, 3 lexicalized DMV models are used, each with a different training regime: EM, and CE with two neighborhood functions (ERROR and TRANS1).\(^\text{11}\) Each version follows the same processing steps

\(^{10}\)This is similar to the procedure described by Brill and Moore (2000), except that \( N \) is restricted to 1.

\(^{11}\)LENGTH was deemed too computationally expensive.
before and after using its characteristic language model to help select a correction hypothesis. Those core processing steps are broadly similar to other noisy-channel implementations. This section describes them and walks through salient design decisions and implementation details.

The correction system processes each OCR sentence independently. For each sentence, a lattice of correction candidates is generated. The lattice is a weighted finite-state machine whose nodes correspond to positions between tokens. Each arc’s emission string constitutes a hypothesis about what might be the correct text for the corresponding span. Arc weights are given by:

$$\log p_{\text{channel}}(O|\hat{C})$$

where $\hat{C}$ is the hypothesis string. Paths through the lattice represent possible correction hypotheses for the sentence, and the sum of the arc weights is its channel-conditional probability in log space. All lattice processing for this work uses the log semiring.

The lattice is populated with the OCR text verbatim, plus a set of alternatives. Alternatives for single tokens are added for each pair of adjacent nodes. Arcs for 2-token spans containing punctuation, and 3-token spans with punctuation in the middle, are populated to support correction of letters that are mis-read by the OCR system as punctuation marks.\(^{12}\)

For each one-token span (and two- and three-token spans with punctuation), a set of alternatives is generated by using an inexact dictionary data structure which efficiently finds words that have a small edit distance from a query string. This dictionary is queried on the OCR text corresponding to each span, plus, if the span contains a hyphen character, the de-hyphenated version of that string. The addition of de-hyphenated spans to the lattice is intended to boost the efficacy of the language models, which are trained on likewise de-hyphenated text, in re-scoring the lattice.

The dictionary data structure is based on an information retrieval technique that utilizes vector space representations of strings (as described, for example, by Salton, 1989), and is modeled on the corresponding module of the OCR correction system presented by Tong and Evans (1996). All dictionary words and query strings are divided into character trigrams (plus bigrams for words of up to 4 characters), with special ‘start’ and ‘end’ characters surrounding each string. Words in the dictionary are associated with each of their constituent trigrams (plus bigrams for short words) to form an inverted index that maps character n-grams to words.

When the inexact dictionary is queried with an OCR string, it breaks it up into character trigrams/bigrams to query the inverted index. The inverted index provides a list of up to 1000 words that contain at least one of the character n-grams, ranked by how many character n-grams match. This list is then re-ranked according to the channel model’s estimate of the probability that the candidate correction string is read by the OCR system as the OCR string. The top $N$ candidate strings are added to the lattice. For our experiments, we set $N = 10$. Higher values of $N$ have minimal impact on correction performance, but slow down the correction system.

\(^{12}\)We have also tried populating the lattice with single-word alternatives for each two-token span (which facilitates some degree of correction for word segmentation errors in the OCR). However, this did not consistently improve correction performance (and came at great cost to computational performance), so we stopped doing it. Our final correction system therefore preserves the word boundaries found by the OCR system, except surrounding punctuation.
Once the lattice is built, it is passed on to one of the language model modules, which finds the best path through the lattice under the models.

The simplest of the three language models is the n-gram model. We used a 4-gram model with interpolated Kneser-Ney smoothing, as modified by Chen and Goodman (1999). This serves as a performance baseline, but is also a component of the interpolated system (see below). We use the SRI language modeling toolkit, SRILM, to estimate this model, and to select the most likely word sequences using the Viterbi algorithm.

The second of our three OCR correction variants uses an ILDMV as a language model. The candidate lattice is parsed by our lattice parser, an Eisner-Satta chart parser. As in a string parser, each chart entry is weighted with its probability under the ILDMV in log space. Unlike a string parser, the terminals are arcs in the lattice, and their weights (the channel probabilities) are included as parsing decisions. The parser’s search maximizes the joint probability of the lattice path and the parse:

$$\hat{x} = \arg \max_x \left[ p_{\text{channel}}(O|x) \max_{\tau \in T(x)} p_{\text{ILDMV}}(x, \tau) \right]$$

(18)

It would be preferable to find the path that maximizes the marginal probability of the string over all possible parses, but that is an NP-hard problem (Sima’an, 1996).

The third of our OCR correction variants interpolates between the n-gram model and the ILDMV. Our implementation uses “n-best lists”. The n-gram model outputs its top $n$ correction candidates, along with both their channel model probabilities and language model probabilities. Then each of those candidates is parsed and scored by the ILDMV parser.

The use of n-best lists is inefficient, and introduces pruning that is ignorant of one of the two models. Fortunately, however, n-best re-ranking caused very little loss in error correction performance for other researchers (Roark et al., 2004). It also has the advantage of allowing us to efficiently marginalize over parses:

$$p_{\text{ILDMV-marg}}(x) = \sum_{\tau \in T(x)} p_{\text{ILDMV}}(x, \tau)$$

(19)

The n-gram and ILDMV probabilities are not suitable for direct interpolation, however, because the ILDMV probabilities include a model of structure in addition to word choices, and are therefore generally of a lower order of magnitude. To put the models on a level playing field, we renormalize both probabilities by dividing by the sum for all candidates in the n-best list. In effect, we use probabilities conditioned on the assumption that one of the entries in the n-gram model’s n-best output is correct. The final score of each sentence is

$$p_{\text{interp}}(x) = \lambda \frac{p_{\text{ILDMV-marg}}(x)}{\sum_{x' \in X'} p_{\text{ILDMV-marg}}(x')} + (1 - \lambda) \frac{p_{\text{n-gram}}(x)}{\sum_{x' \in X'} p_{\text{n-gram}}(x')}$$

(20)

where $X'$ is the set of candidates in the n-best output of the n-gram model (after ranking by $p_{\text{channel}}(x)p_{\text{n-gram}}(x)$), and $\lambda$ is a model interpolation hyperparameter.

A discussion of the hyperparameters of each of the models is in section 6.5.


6 Evaluation

6.1 Parsing Accuracy

Applying the DMV to the OCR correction task requires a number of changes to the model and the experimental conditions. For example, while work in this area traditionally discards unpronounced punctuation tokens, we must include them. In order to explore the effects these changes have on parse quality, we did a number of experiments that measured parsing accuracy using the WSJ portion of the Penn TreeBank for both training and testing.

Some work on unsupervised grammar induction, including that of Klein and Manning (2004), uses the entire WSJ portion of the Penn TreeBank for both training and testing. However, for correcting OCR we need the parser to be able to generalize to text outside the training set. So we used sections 2 through 21 as a training set, and sections 23 as a test set. We trained only on sentences of length up to 10 tokens, including punctuation (4233 sentences). For some experiments, punctuation is excluded, to more closely match the conditions of previous research. We call these training subsets TRAIN10 and TRAIN10W, respectively.

We tested parsing performance on six test sets: TRAIN10, TEST10 (section 23, up to 10 tokens), TEST∞ (all sentences), and the unpunctuated ‘W’ versions of each of those.

We used the Penn2Malt utility to convert the Penn TreeBank’s constituency trees into dependency trees. This uses heuristics to assign heads for each constituent, and creates a dependency from each non-head (modifier) to the head.

Our accuracy scores are calculated as the percentage of directed dependency arcs the system guesses correctly (including the arc from the Root pseudo-token). The “baseline” is a hypothetical parser that outputs a completely left-branching parse. That is, each token is the head of its left neighbor, and the last token is the root. On the Penn2Malt-binarized WSJ TreeBank, this is superior to both a right-branching and random baseline.

To ensure that there are no zeros in \( p_{\text{tag}} \) in equation 4, we estimate it with Laplace (add-1) smoothing. This is done at test-time; no smoothing was used during the training procedures.

The results are shown graphically in figure 4.

We separately examined the effects of including punctuation and using automatically induced clusters in lieu of annotated part-of-speech tags. While previous research has used large corpora for inducing clusters, we only use the parser training set (but without the sentence length limit). As figure 4 shows, parsing accuracy suffered substantially for many conditions when punctuation was added back in. Surprisingly, however, EM’s parsing of TEST∞ slightly improved. It is also notable that parsing with punctuation results in much less of a gap between TEST10 and TEST∞, suggesting that punctuation is useful for parsing longer sentences. What is perhaps most surprising is that the replacement of part-of-speech tags with Brown clusters (at \( c = 150 \)) improved TEST∞ accuracy for EM, CE/LENGTH, and CE/ERROR.

These numbers show striking differences between the CE neighborhood functions. One surprise is that LENGTH is the equal of TRANS1 on both TRAIN10W and TEST10W; Smith and Eisner, by contrast, reported that TRANS1 was superior. We do not know why this is, but we can point
out that the biggest difference that we’re aware of between the two evaluations is that because TEST10W is constructed by filtering by length before removing punctuation, it contains fewer, shorter sentences. It should also be pointed out that unlike Smith and Eisner, we are not averaging over 10 cross-validation folds, so our figures include more noise.

It is also noteworthy that the ERROR neighborhood performs better than LENGTH across the board. One very striking fact is that TRANS1 performs consistently below-baseline on punctuated text.

Finally, we note that EM provides the best parsing accuracy on TEST∞ with Brown word clusters, at 36.5%, or 9.9% above baseline. CE / ERROR is second-best, at 33.8%.

6.2 The Corpus

The training and testing of the OCR correction system required the use of both high-quality transcription of printed works and OCR output. The transcription is from the Women Writer’s Online corpus (http://wwp.northeastern.edu/wwo), which has high-quality transcription of 349 works by women authors spanning a range of publication years from 1526 to 1850.

Something should be said about the diversity of the WWO corpus. Figure 5 shows the distribution of publication years for the prose works that were used as either language model training or test data. The English language changed markedly during this span. The latter works look very much like modern English, but the earliest works look like this sentence from Sermons of Barnardine Ochine of Sena, Translator’s Preface, written by Ann Bacon in 1548:

DEATH (GOOD READER) as scripture declareth, and our dayly experience practeth, to all mankynde is a thing most ceretaine and sure.

It clearly uses different orthographic conventions, and even different glyphs and diacritics. The long S (ȝ), in particular, fell out of use gradually during the 18th century. Therefore, most works
in the corpus use the long $S$, but a great many do not.

![Figure 5: Number of prose works in WWO corpus by decade of publication](image)

The WWO corpus is stored in an XML format according to Text Encoding Initiative (TEI) guidelines (http://www.tei-c.org). Substantial processing was required to transform this representation into text suitable for training. The rest of this section will discuss that processing in detail.

First, it was filtered. For example, page numbers (which are included in the transcription with a particular XML attribute) are discarded because they degrade the quality of the data as training for language models. Footnotes were also filtered out, for an entirely different reason. The OCR text with which some of the transcription would eventually be aligned includes footnotes where they occur visually, which alters the text order vis-à-vis the transcription. Instead of attempting to cope with the out-of-order nature of footnotes in the alignment algorithm, we simply exclude them.

The transcription was also transformed in certain small ways, sometimes to improve its quality as a sample of natural language, and sometimes to match the OCR better. TEI replaces quotation marks with markup indicating that a span of text was spoken; these were re-inserted. In documents that would be used for language model training, hyphens that break words across lines were detected and removed, to glue broken-up words back together. All the text is tokenized by inserting token boundaries at whitespace, and both before and after all punctuation characters.

The transcription was divided into four sets. By far the largest set (4.2 million tokens) consisted of those documents for which we do not have matching OCR text to align with, and which were identified as prose works. The identification of prose (as opposed to poetry and plays) was accomplished by applying heuristics that exploit the fact that certain TEI tags occur much more frequently in verse and performance works. This set was used as training for word clustering and language models.

The language model training was split into sentences using basic heuristics. Because the mean sentence length is so long on this historical corpus (47 tokens), the sentence breaker additionally breaks sentences at semicolon. This reduces the mean sentence length to 30 tokens.
The portion of the transcription for which we have matching OCR text was divided into three sets: a channel model training set, a dev set, and a test set. The OCR text was retrieved from the Internet Archive (http://archive.org), which stores scanned images and corresponding OCR text for a number of books whose copyright has expired. The OCR software used was ABBYY FineReader 8.0. The works consisting primarily of poetry and plays were used for the channel model training, and the rest (prose) were divided randomly between the dev and test sets. A forced alignment was performed on the corresponding OCR text for works in all three of these subsets. For some works, several independently scanned OCR versions are available. For the channel model training, we used alignments with each OCR version available; but for the dev and test sets, we used only one OCR version of each work and discarded the rest. The channel model training set contains 490,000 tokens (counting multiple OCR alignments of the same work separately).

The dev and test sets were split into sentences, using the same sentence-breaking rules that were applied to the language model training (which, again, consider the semicolon to be a sentence-terminating character). Because the correct transcription is not available when applying an OCR error correction system in practice, the sentence-breaking is performed on the OCR text. Then the alignment is used to locate each corresponding position in the transcription, resulting in sequences of aligned transcription/OCR sentences. It is important to note that this data, having been processed in a manner which could potentially be improved upon, should be considered part of the correction system pipeline, not part of the aligned corpus proper. It should also be noted that applying imperfect heuristics to imperfect OCR results in lower-quality sentence boundaries than the language model training.

Because the time complexity of parsing using Eisner-Satta is cubic in sentence length, we found it necessary to discard long sentences. We exclude sentences longer than 50 tokens from all language model training and test data. The resulting dev set contains a total of 13,117 sentences and 327,957 tokens. The test set contains 13,780 sentences and 303,798 tokens. For much of the development and tuning, even shorter sentence-length cut-offs were used, to speed up turn-around time for experiments.

6.3 OCR Correction Experiments

We focused on word error rate (WER) as a performance metric. We define the number of word errors as the edit-distance between the transcription and the system output, using word substitution, deletion, and insertion as the edit operations, each with a cost of 1. The word error rate is the number of word errors, divided by the number of tokens in the test transcription.\(^\text{13}\)

Our WER measure actually forgives certain differences. First, the transcription and output strings are “de-hyphenated” before being compared (the same restoration of words broken across lines that is applied to the language model training). Otherwise, our correction system would be

\(^\text{13}\)We initially measured character error rate also, but found that performance tuning would sometimes benefit one measure at the expense of the other. We decided to deal with this by devoting our attention to a single measure, and selected WER.
Figure 6: Summary of dev and test sets. Word error rate (WER) is as defined in section 6.3. The high OCR error rate of ‘reflectionsuponm00aste’ is due to the mis-recognition of long s (usually as f).

unfairly penalized when it restored such words. Second, it introduces a few character equivalence classes. Lowercase s and long s (f) are considered the same. Likewise, the non-ASCII variants of quotations marks and apostrophes are each considered equivalent to the ASCII.

To gauge the success of each version of our system, we compare WER to that of the 4-gram baseline described in section 5.3. An additional baseline is described in section 6.4.

6.4 The Right-branching ILDMV Baseline

A favorable comparison to the 4-gram model indicates that an LM is more useful, but it doesn’t tell us why. So we implemented an additional baseline that is intended to shed light on why the ILDMV improves upon the n-gram model. It is a version of the ILDMV that artificially constrains the parse to a right-branching structure. That is, the first word is the root, and each non-final word has its right-hand neighbor as its sole dependent. This can be thought of as an experimental control that differs from the ILDMV in only one regard: it allows no variation in syntactic structure.

The resulting LM interpolates between a word bigram model and the class-conditional word bigram model used to induce the word clusters (equation 15).

Training of the right-branching baseline is performed in two steps, like the ILDMV: first, the DMV sub-models ($p_{\text{stop}}$ and $p_{\text{tag}}$) are estimated; then $p_{\text{word}}$ is estimated from the “parsed” training corpus. The difference is that the lack of latent variables makes EM unnecessary; the sub-models are estimated from a single pass over the training.

6.5 Hyperparameters

All the models need to assign a probability to words that were not seen in the training set. Setting this value optimally is of particular importance for OCR error correction, because most
OCR errors result in out-of-vocabulary (OOV) words. The lower the estimated probability of an OOV word, the more likely such a word is to be replaced with an in-vocabulary correction candidate. But if the estimate is too low, then the system will introduce too many errors by altering correct OOV words (e.g., names). We noticed immediately that correction performance was sensitive to this parameter, because the best value depended on the OCR error rate. We therefore decided to set it as a hyperparameter, to be tuned empirically by optimizing WER on the dev set.

In the case of the n-gram LM, this hyperparameter simply sets the $p_{\text{OOV}}$ in the unigram model to which the n-gram model is ultimately smoothed. We do not bother to renormalize the model, as the effect on the total mass is so slight. Figure 7 shows the variation among the documents in the dev set. The document with the highest OCR error rate, ‘reflectionsuponm00aste,’ prefers a much lower value than the ones with lower error rates. The optimal value for the dev set as a whole is about $p_{\text{n-gram(OOV)}} = 10^{-7}$. This is the value we used on the test set.

The ILDMV, however, is less straightforward. OOV words are assigned probabilities twice: once by $p_{\text{lex}}$, and once by $p_{\text{word}}$. It would be nice to set both $p_{\text{lex}}(\text{OOV}|\text{oov})$ (where oov is the pseudo-cluster for unknown words) and $p_{\text{word}}(\text{OOV},d)$ to $p_{\text{n-gram(OOV)}}$, so that there’s really only one hyperparameter. However, these three values each affect the model in subtly different ways (due to the smoothing in equation 9 and the renormalization in equation 20), so we found it necessary to tune each separately. These were set to optimize WER of the ILDMV (without n-gram interpolation) using grid search, by factors of 10. Optimization on the dev set yielded
(a) $p_{\text{lex}}(\text{OOV})$ (with $p_{\text{word}}(\text{OOV})$ held at $10^{-8}$)

(b) $p_{\text{word}}(\text{OOV})$ (with $p_{\text{lex}}(\text{OOV})$ held at $10^{-6}$)

Figure 8: Sensitivity of ILDMV WER (on dev) to its OOV estimates. The blue bar represents the 4-gram baseline’s best score (3.746% at $p_{\text{n-gram}} = 10^{-7}$).

$p_{\text{lex}}(\text{OOV}) = 10^{-6}$, $p_{\text{word}}(\text{OOV}) = 10^{-8}$.

We also chose the number of word clusters experimentally, and independently for each of the 3 training regimes (EM, and CE with both the Error and Trans1 neighborhoods). For each trainer, we selected the number of word clusters that yielded the best dev set WER.

As figure 9 shows, there is significant variation among settings of the word cluster parameter, and it often does not appear to be systematic. CE / Error, for example, does very well at 100 and 200 clusters, but poorly at the intermediate setting of 150 clusters. We believe that much of this apparent extra noise is attributable to the sensitivity of DMV models to initial conditions, which is a symptom of searching a space with many local optima. For each trainer, the number of clusters with the lowest dev set error rate was selected.

```
<table>
<thead>
<tr>
<th></th>
<th>100</th>
<th>150</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>EM</td>
<td>3.74%</td>
<td>3.71%</td>
<td>3.74%</td>
</tr>
<tr>
<td>CE / ERROR</td>
<td>3.68%</td>
<td>3.82%</td>
<td>3.68%</td>
</tr>
<tr>
<td>CE / TRANS1</td>
<td>3.80%</td>
<td>3.65%</td>
<td>3.72%</td>
</tr>
</tbody>
</table>
```

Figure 9: WER of ILDMV training procedures at different word clusterings

Another important hyperparameter is $\beta$ in equation 9, the interpolation weight of $p_{\text{word}}$ in opposition to $p_{\text{tag}} \cdot p_{\text{lex}}$. This was tuned to 0.5 (see figure 10a). Finally, the top-level interpolation weight $\lambda$ (equation 20) was tuned to 0.2 (see figure 10b).
7 Discussion

7.1 Results

Figure 11 gives the test set word error rates after processing by the different versions of our OCR correction system. Unsupervised grammar induction shows more promise in language modeling tasks than the accuracy of the parses would lead one to expect. The ILDMV edges past the n-gram model in the OCR task. The improvement is statistically significant at $p < 0.01^{14}$ for both EM training and CE/ERROR training, but admittedly only a sliver, and not consistent across documents. Even the fact that the ILDMV does about as well as the n-gram model in the same task is remarkable, given that the ASR systems reviewed in section 3 only see an improvement when the structured model is interpolated with the n-gram model (see figure 12).^{15} It is further noteworthy that the ILDMV is fundamentally a bigram LM (all distributions are conditioned on at most one token’s worth of history), and appears to be getting at least as much information from those bigrams as the n-gram model gets from 4-grams. Comparing the EM (best-parse) variation to EM (marginalized) shows that marginalizing over all parses makes a negligible difference.

Disappointingly, however, we believe that the benefits of the ILDMV are not owed primarily to the modeling of structure, but rather to the modeling of word clusters. The eleventh-hour addition of the right-branching baseline (see section 6.4) was designed to tease the two effects apart: it differs from the n-gram model insofar as it includes the ILDMV’s bigram model of word clusters (from which words are generated); but unlike the ILDMV, it does not model latent

---

14 We performed a sign test over matched sentence pairs.

15 One possible explanation for bad performance of structured language modeling for ASR, which we did not explore, is that before the Switchboard TreeBank became available, structured LMs were trained on the WSJ, and then tested on the very different Switchboard corpus of spoken English. The mismatch between the training and test data for ASR may be greater than it is in our work, which deals only with written English.
Figure 11: OCR word error rates for correction systems using various LMs. Absolute improvement over n-gram baseline is shown in parentheses. The pure ILDMV models all select the string with the most likely parse (Viterbi). The best interpolated models marginalize string probabilities over all possible parses, but the EM model with best-parse correction is also shown for comparison.

Structure. Because the right-branching baseline performs about as well as the ILDMV, we can’t argue that a structured model has improved over a sequential model.

The right-branching baseline does slightly better when interpolated with the 4-gram baseline. This is not surprising, because even though both generate a sentence left-to-right, the 4-gram model has more state history and more advanced smoothing (Kneser-Ney, vs. Witten-Bell). Nevertheless, the three interpolated ILDMV models out-perform the interpolated right-branching model, if only by a hair’s width. We tentatively take this as evidence that the modeled structure information is more complementary to the 4-gram model’s information, and therefore a more helpful interpolation partner than the right-branching model.

Figure 12: Comparison of structured language models by WER in the reported noisy-channel task, with and without n-gram interpolation. Relative change in error rate from baseline (\( \text{WER}_{\text{n-gram}} \)) shown in parentheses. The two blank spaces correspond to experiments that were not performed.

It is disappointing, however, that interpolation with the n-gram model provides so little additional improvement. We speculate that this has to do with characteristics of how (IL)DMV parses differ from correct parses. Figure 13 shows the proportion of dependency arcs that are
directed toward an immediate neighbor. Notice that DMV parses have many more neighbor arcs (in one direction or the other) than the Penn TreeBank has in its annotation. Evidently, the DMV parses are closer to the locally branching structure implicit in an n-gram (specifically, bigram) model. This interesting fact may help explain why the ILDMV models perform similarly to the right-branching baseline, and also why they seem to have little additional information to impart to n-gram-interpolated models.

These statistics also underscore the fact that the parsers under consideration are poor approximators of linguist annotation. We do not have such annotation to allow us to measure parsing accuracy on the WWO corpus, but it is striking that these parse statistics differ so much from ground truth, especially in the CE-trained models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Right neighbor</th>
<th>Left neighbor</th>
<th>Long arcs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Truth</td>
<td>19.0%</td>
<td>26.9%</td>
<td>54.0%</td>
</tr>
<tr>
<td>EM</td>
<td>10.0%</td>
<td>61.2%</td>
<td>28.8%</td>
</tr>
<tr>
<td>CE / Error</td>
<td>62.9%</td>
<td>11.1%</td>
<td>26.0%</td>
</tr>
<tr>
<td>CE / Trans</td>
<td>62.8%</td>
<td>5.9%</td>
<td>31.2%</td>
</tr>
</tbody>
</table>

Figure 13: Percentage of arcs directed toward the right neighbor and left neighbor in ILDMV parses. “Long arcs” are the remainder (directed toward neither neighbor). “Truth” is over WSJ §23; the others are over a small sample of the dev set. Sentences up to 30 tokens were used, punctuation is included. The statistics for unlexicalized DMV parses are quite similar.

Unfortunately, there is neither a standard OCR test set nor a standard set of evaluation criteria. Combined with the great variety of OCR quality in the wild, this leaves us with no good way to directly compare the performance of correction systems from different authors. Nevertheless, the fact that the relative improvement in WER in our evaluations is only about half of what was achieved by other systems needs to be explained. One likely factor is that our training corpus of mixed historical works presents modeling difficulties. Although this is speculation, there are other factors which we can speak to with more clarity.

First, it should be noted that our initial OCR error rate (5.2%) is quite low in comparison. This is partly attributable to the fact that we filter out portions with high error rates (as a cheap way to reduce bad alignments due to mismatches between the printed material and the transcription). We also selected documents with good overall character alignment scores (> 90%) in order to filter out edition mismatches. And it is important that the ABBYY FineReader OCR software incorporates a frequency-weighted dictionary of modern English, probably applied as a unigram prior. The disadvantage of running on relatively good OCR is that we simply have less headroom to improve in.

Another possibly important difference is that Kolak et al. used OCR software for English on French text, which may have introduced a large number of systematic errors which were easy for the error model to learn and predict.

Tong and Evans also makes the problem easier in ways which we did not mimic. They discard OCR tokens that contain numbers and other non-letters. We, on the other hand, attempt to
Figure 14: OCR word error rates and relative reduction ($1 - \frac{\text{WER}_{\text{OCR}}}{\text{WER}_{\text{Reduction}}}$) for the present work and two related systems.

<table>
<thead>
<tr>
<th></th>
<th>OCR</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tong and Evans</td>
<td>14.72%</td>
<td>60.11%</td>
</tr>
<tr>
<td></td>
<td>5.87%</td>
<td></td>
</tr>
<tr>
<td>Kolak et al.</td>
<td>18.31%</td>
<td>61.17%</td>
</tr>
<tr>
<td></td>
<td>7.11%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7.06%</td>
<td>61.44%</td>
</tr>
<tr>
<td></td>
<td>6.75%</td>
<td>63.13%</td>
</tr>
<tr>
<td>Present</td>
<td>5.19%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4.29%</td>
<td>17.35%</td>
</tr>
<tr>
<td></td>
<td>3.66%</td>
<td>29.53%</td>
</tr>
<tr>
<td></td>
<td>3.55%</td>
<td>31.56%</td>
</tr>
<tr>
<td></td>
<td>3.53%</td>
<td>32.06%</td>
</tr>
</tbody>
</table>

restore the original OCR, including punctuation and numbers. Our system quixotically attempts to correct numbers in the OCR, like any other token; we concede that this is a deficiency.

Although Tong and Evans is not explicit on this point, we believe that both systems use de-cased LMs and caseless evaluations. The fact that our LM is case-agnostic (and trained and used on mixed-case text) is a deficiency of our system. Although we consciously decided to perform a case-aware evaluation, necessitating a case-aware LM, we could surely improve performance by handling case specially (as Kolak et al. did).

### 7.2 Future Work

We discovered that the task of correcting OCR text that has a low error rate to begin with is largely a matter of determining which out-of-vocabulary words are to be considered errors and corrected. The parameters that most directly inform this decision are the probabilities assigned to OOV words by the language models. This is why the performance of the system is so sensitive to those parameters.

We therefore expect that it would be valuable to estimate the prior probability of OOV word more accurately by using a character-level n-gram model. The observed token ‘That,’ for example, would be given a low prior, as $t$ is rarely followed by $b$, and would be more likely to be corrected to ‘That.’ The proper name ‘Thad,’ on the other hand, would get a comparatively high prior, even if it is not observed in the training.

Furthermore, there are practical tricks that might help mitigate this problem. We suggest considering OOV words that occur more than once in a document to have a much higher probability in the LM. A more practical system should also use heuristics to detect classes of tokens which are less likely to require correction (e.g., capitalized words), or should not be corrected at all (e.g., numbers).
We believe that a number of advances in unsupervised grammar induction which we did not have time to explore may provide better language modeling. Given that the ILDMV is effectively tied with the right-branching baseline, improvements in it may lead to improvements in correction performance.

Some of these advances improve the grammar model, and some improve the search procedure used to train the model. An example of an improved model is the Lexical Extended Valence Model (L-EVG) (Headden III et al., 2008) described in section 3.1. As a side-note, L-EVG is only partially lexicalized, conditioning on head words using a smoothing method that only uses commonly occurring head words to better predict dependent tags. This may lead to better parsing, but more thorough lexicalization is probably needed to serve as an adequate language model.

More importantly, our two-step training process, which learns from lexical features only after a tag-based parser has been trained, allows for a variety of features which has yet to be exploited. Previous work has shied away from features that harm grammar induction by exacerbating data sparsity. However, such features can be more safely introduced in the second training step, as we have done with word bigrams. For example, conditioning on a deeper head-chain history than the immediate head may yield a better LM.

The need for better search procedures is highlighted by the unexpected and irregular variation in performance we saw when trying various numbers of word clusters. This is evidence of great variation among the local optima in which EM and CE training tend to converge. Variational Bayes (Cohen et al., 2008) and Viterbi EM (Spitkovsky et al., 2010) have been shown to substantially improve the parsing accuracy of the DMV by searching the parameter space more effectively. These are likely to result in better language models as well.

References


