A Scalable Probabilistic Classifier for Language Modeling

Joel Lang
Institute for Language, Cognition and Computation
School of Informatics, University of Edinburgh
10 Crichton Street, Edinburgh EH8 9AB, UK
J.Lang-3@sms.ed.ac.uk

Abstract

We present a novel probabilistic classifier, which scales well to problems that involve a large number of classes and require training on large datasets. A prominent example of such a problem is language modeling. Our classifier is based on the assumption that each feature is associated with a predictive strength, which quantifies how well the feature can predict the class by itself. The predictions of individual features can then be combined according to their predictive strength, resulting in a model, whose parameters can be reliably and efficiently estimated. We show that a generative language model based on our classifier consistently matches modified Kneser-Ney smoothing and can outperform it if sufficiently rich features are incorporated.

1 Introduction

A Language Model (LM) is an important component within many natural language applications including speech recognition and machine translation. The task of a generative LM is to assign a probability $p(w)$ to a sequence of words $w = w_1 \dots w_L$. It is common to factorize this probability as

$$p(w) = \prod_{i=1}^{L} p(w_i | w_{i-N+1} \dots w_{i-1}) \qquad (1)$$

Thus, the central problem that arises from this formulation consists of estimating the probability $p(w_i | w_{i-N+1} \dots w_{i-1})$. This can be viewed as a classification problem in which the target word W_i corresponds to the class that must be predicted, based on features extracted from the conditioning context, e.g. a word occurring in the context.

This paper describes a novel approach for modeling such conditional probabilities. We propose a classifier which is based on the assumption that each feature has a predictive strength, quantifying how well the feature can predict the class (target word) by itself. Then the predictions made by individual features can be combined into a mixture model, in which the prediction of each feature is weighted according to its predictive strength. This reflects the fact that certain features (e.g. certain context words) are much more predictive than others but the predictive strength for a particular feature often doesn't vary much across classes and can thus be assumed constant. The main advantage of our model is that it is straightforward to incorporate rich features without sacrificing scalability or reliability of parameter estimation. In addition, it is simple to implement and no feature selection is required. Section 3 shows that a generative[1] LM built with our classifier is competitive to modified Kneser-Ney smoothing and can outperform it if sufficiently rich features are incorporated.

The classification-based approach to language modeling was introduced by Rosenfeld (1996) who proposed an optimized variant of the maximum-entropy classifier (Berger et al., 1996) for the task. Unfortunately, data sparsity resulting from the large number of classes makes it difficult to obtain reliable parameter estimates, even on large datasets and the high computational costs make it difficult train models on large datasets in the first place[2]. Scal-

[1]While the classifier itself is discriminative, i.e. conditioning on the contextual features, the resulting LM is generative. See Roark et al. (2007) for work on discriminative LMs.

[2]For example, using a vocabulary of 20000 words Rosenfeld (1994) trained his model on up to 40M words, however employing heavy feature pruning and indicating that "the computational load, was quite severe for a system this size".

ability is however very important, since moving to larger datasets is often the simplest way to obtain a better model. Similarly, neural probabilistic LMs (Bengio et al., 2003) don't scale very well to large datasets. Even the more scalable variant proposed by Mnih and Hinton (2008) is trained on a dataset consisting of only 14M words, also using a vocabulary of around 20000 words. Van den Bosch (2005) proposes a decision-tree classifier which has been applied to training datasets with more than 100M words. However, his model is non-probabilistic and thus a standard comparison with probabilistic models in terms of perplexity isn't possible.

N-Gram models (Goodman, 2001) obtain estimates for $p(w_i|w_{i-N+1} \dots w_{i-1})$ using counts of N-Grams. Because directly using the maximum-likelihood estimate would result in poor predictions, smoothing techniques are applied. A modified interpolated form of Kneser-Ney smoothing (Kneser and Ney, 1995) was shown to consistently outperform a variety of other smoothing techniques (Chen and Goodman, 1999) and currently constitutes a state-of-the-art[3] generative LM.

2 Model

We are concerned with estimating a probability distribution $p(Y|x)$ over a categorical class variable Y with range $\mathcal{Y}$, conditional on a feature vector $x = (x_1, \dots, x_M)$, containing the feature values x_i of M features. While generalizations are conceivable, we will restrict the features X_k to be binary, i.e. $x_k \in \{0, 1\}$. For language modeling the class variable Y corresponds to the target word W_i which is to be predicted and thus ranges over all possible words of some vocabulary. The binary input features x are extracted from the conditioning context $w_{i-N+1} \dots w_{i-1}$. The specific features we use for language modeling are given in Section 3.

We assume sparse features, such that typically only a small number of the binary features take value 1. These features are referred to as the active features and predictions are based on them. We introduce a bias feature which is active for every instance, in order to ensure that the set of active features is non-empty for each instance. Individually, each active feature X_k is predictive of the class variable and predicts the class through a categorical distribution[4] distribution, which we denote as $p(Y|x_k)$. Since instances typically have several active features the question is how to combine the individual predictions of these features into an overall prediction. To this end we make the assumption that each feature X_k has a certain predictive strength $\theta_k \in \mathbb{R}$, where larger values indicate that the feature is more likely to predict correctly. The individual predictions can then be combined into a mixture model, which weights individual predictions according to their predictive strength:

$$p(Y|x, \theta) = \sum_{k \in \mathcal{A}(x)} v_k(x) p(Y|x_k) \tag{2}$$

where

$$v_k(x) = \frac{e^{\theta_k}}{\sum_{k \in \mathcal{A}(x)} e^{\theta_k}} \tag{3}$$

Here $\mathcal{A}(x)$ denotes the index-set of active features for instance (y, x). Note that since the set of active features varies across instances, so do the mixing proportions $v_k(x)$ and thus this is not a conventional mixture model, but rather a *variable* one. We will therefore refer to our model as the variable mixture model (VMM). In particular, our model differs from linear or log-linear interpolation models (Klakow, 1998), which combine a typically small number of components that are *common across instances*.

In order to compare our model to the maximum-entropy classifier and other (generalized) linear models, it is beneficial to rewrite Equation 2 as

$$p(Y = y|x, \beta) = \frac{1}{Q(x)} \sum_{k=1}^{M} \sum_{j=1}^{|\mathcal{Y}|} \phi_{j,k}(y, x) \beta_{j,k} \tag{4}$$

$$= \frac{1}{Q(x)} \beta^\top \phi(y, x) \tag{5}$$

where $\phi_{j,k}(y, x)$ is a sufficient statistics indicating whether feature X_k is active and class $y = y_j$ and

$$\beta_{j,k} = e^{\theta_k + \log p(y_j|x_k)} \tag{6}$$

$$Q(x) = \sum_{k \in \mathcal{A}(x)} e^{\theta_k} \tag{7}$$

Table 1 shows the main differences between the VMM, the maximum-entropy classifier and the perceptron (Collins, 2002).

[3]The model of Wood et al. (2009) has somewhat higher performance, however, again due to high computational costs the model has only been trained on training sets of at most 14M words.

[4]commonly referred to as a multinomial distribution

VMM	Maximum Entropy	Perceptron
$p(y\|x,\beta) = \frac{1}{Q(x)}\beta^\top \phi(y,x)$	$p(y\|x,\beta) = \frac{1}{Q(x)}e^{\beta^\top \phi(y,x)}$	$score(y\|x,\beta) = \beta^\top \phi(y,x)$
$Q(x) = \sum_{k \in \mathcal{A}(x)} e^{\theta_k}$	$Q(x) = \sum_{j=1}^{\|\mathcal{Y}\|} e^{\beta^\top \phi(y_j,x)}$	

Table 1: A comparison between the VMM, the maximum-entropy classifier and the perceptron. Like the perceptron and in contrast to the maximum-entropy classifier, the VMM directly uses a predictor $\beta^\top \phi(y,x)$. For the VMM the sufficient statistics $\phi(y,x)$ correspond to binary indicator variables and the parameters β are constrained according to Equation 6. This results in a partition function $Q(x)$ which can be efficiently computed, in contrast to the partition function of the maximum-entropy classifier, which requires a summation over all classes.

2.1 Parameter Estimation

The VMM has two types of parameters:

1. the categorical parameters $\alpha_{j,k} = p(y_j|x_k)$ which determine the likelihood of class y_j in presence of feature X_k;
2. the parameters θ_k quantifying the predictive strength of each feature X_k.

The two types of parameters are estimated from a training dataset, consisting of instances $(y^{(h)}, x^{(h)})$. Parameter estimation proceeds in two separate stages, resulting in a simple and efficient procedure. In a first stage, the categorical parameters are computed independently for each feature, as the maximum likelihood estimates, smoothed using absolute discounting (Chen and Rosenfeld, 2000):

$$\alpha_{j,k} = p(y_j|x_k) = \frac{c'_{j,k}}{c_k}$$

where $c'_{j,k}$ is the smoothed count of how many times Y takes value y_j when X_k is active, and c_k is the count of how many times X_k is active. The smoothed count is computed as

$$c'_{j,k} = \begin{cases} c_{j,k} - D & \text{if } c_{j,k} > 0 \\ \frac{D \cdot NZ_k}{Z_k} & \text{if } c_{j,k} = 0 \end{cases}$$

where $c_{j,k}$ is the raw count for class y_j and feature X_k, NZ_k is the number of classes for which the raw count is non-zero, and Z_k is the number of classes for which the raw count is zero. D is the discount constant chosen in $[0,1]$. The smoothing thus subtracts D from each non-zero count and redistributes the so-obtained mass evenly amongst all zero counts. If all counts are non-zero no mass is redistributed.

Once the categorical parameters have been computed, we proceed by estimating the predictive strengths $\theta = (\theta_1, \ldots, \theta_M)$. We can do so by conducting a search for the parameter vector θ^* which maximizes the log-likelihood of the training data:

$$\begin{aligned} \theta^* &= \arg\max_\theta ll(\theta) \\ &= \arg\max_\theta \sum_h \log p(y^{(h)}|x^{(h)}, \theta) \end{aligned}$$

While any standard optimization method could be applied, we use stochastic gradient ascent (SGA, Bottou (2004)) as this results in a particularly convenient and efficient procedure that requires only one iteration over the data (see Section 3). SGA is an online optimization method which iteratively computes the gradient ∇ for each instance and takes a step of size η in the direction of that gradient:

$$\theta^{(t+1)} \leftarrow \theta^{(t)} + \eta \nabla \tag{8}$$

The gradient $\nabla = (\frac{\partial ll^{(h)}}{\partial \theta_1}, \ldots, \frac{\partial ll^{(h)}}{\partial \theta_M})$ computed for SGA contains the first-order derivatives of the data log-likelihood of a particular instance with respect to the θ-parameters which are given by

$$\frac{\partial}{\partial \theta_k} \log p(y|x,\theta) = \frac{v_k(x)}{p(y|x,\theta)} \left[p(y|x_k) - p(y|x,\theta) \right] \tag{9}$$

The resulting parameter-update Equation 8 has the following intuitive interpretation. If the prediction of a particular active feature X_k is higher than the current overall prediction, the term in square brackets in Equation 9 becomes positive and thus the predictive strength θ_k for that feature is increased and conversely for the case where the prediction is below the overall prediction. The magnitude of the

Type	Extracted Features

Standard N-Grams (BA,SR,LR)	`* * *` (bias)
	`Mr Thompson said`
	`* Thompson said`
	`* * said`

Skip N-Grams (SR,LR)	`Mr * said`
	`Mr Thompson *`
	`Mr * *`
	`* Thompson *`

Unigram Bag Features (SR,LR)	`Mr`
	`Thompson`
	`said`

Long-Range Unigram Bag Features (LR)	`Yesterday`
	`at`
	`the`
	`press`
	`conference`

Table 2: Feature types and examples for a model of order N=4 and for the context `Yesterday at the press conference Mr Thompson said`. For each feature type we write in parentheses the feature sets which include that type of feature. The wildcard symbol $\star$ is used as a placeholder for arbitrary regular words. The bias feature, which is active for each instance is written as $\star\ \star\ \star$. In standard N-Gram models the bias feature corresponds to the unigram distribution.

update depends on how much overall and feature prediction differ and on the scaling factor $\frac{v_k(x)}{p(y|x,\theta)}$.

In order to improve generalization, we estimate the categorical parameters based on the counts from all instances, except the one whose gradient is being computed for the online update (leave-one-out). In other words, we subtract the counts for a particular instance before computing the update (Equation 8) and add them back when the update has been executed. In total, training only requires two passes over the data, as opposed to a single pass (plus smoothing) required by N-Gram models.

3 Experiments

All experiments were conducted using the SRI Language Modeling Toolkit (SRILM, Stolcke (2002)), i.e. we implemented[5] the VMM within SRILM and compared to default N-Gram models supplied with SRILM. The experiments were run on a 64-bit, 2.2 GHz dual-core machine with 8GB RAM.

Data The experiments were carried out on data from the Reuters Corpus Version 1 (Lewis et al., 2004), which was split into sentences, tokenized and converted to lower case, not removing punctuation. All our models were built with the same 30367-word vocabulary, which includes the sentence-end symbol and a special symbol for out-of-vocabulary words (UNK). The vocabulary was compiled by selecting all words which occur more than four times in the data of week 31, which was not otherwise used for training or testing. As development set we used the articles of week 50 (4.1M words) and as test set the articles of week 51 (3.8M words). For training we used datasets of four different sizes: D1 (week 1, 3.1M words), D2 (weeks 1-3, 10M words), D3 (weeks 1-10, 37M words) and D4 (weeks 1-30, 113M words).

Features We use three different feature sets in our experiments. The first feature set (*basic*, BA) consists of all features also used in standard N-Gram models, i.e. all subsequences up to a length $N-1$ immediately preceding the target word. The second feature set (*short-range*, SR) consists of all basic features as well as all skip N-Grams (Ney et al., 1994) that can be formed with the $N-1$ length context. Moreover, all words occurring in the context are included as bag features, i.e. as features which indicate the occurrence of a word but not the particular position. The third feature set (*long-range*, LR) is an extension of SR which also includes longer-distance features. Specifically, this feature set additionally includes all unigram bag features up to a distance $d = 9$. The feature types and examples of extracted features are given in Table 2.

Model Comparison We compared the VMM to modified Kneser-Ney (KN, see Section 1). The order of a VMM is defined through the length of the context from which the basic and short-range features are extracted. In particular, VM-BA of a certain order uses the same features as the N-Gram models of the same order and VM-SR uses the same conditioning context as the N-Gram models of the same order. VM-LR in addition contains longer-distance features, beyond the order of the corresponding N-Gram models. The order of the models was varied between $N = 2 \ldots 5$, however, for the larger two datasets D3 and D4 the order 5 models would not fit into the available RAM which is why for order 5 we can only report scores for D1 and D2. We could resort to pruning, but since this would have an effect on performance it would invalidate a direct comparison, which we want to avoid.

[5]The code can be downloaded from `http://code.google.com/p/variable-mixture-model`.

Model	N	D1 3.1M	D2 10M	D3 37M	D4 113M
KN	2	209.2	178.2	155.3	139.3
	3	164.9	127.7	98.9	78.1
	4	160.9	122.2	91.4	68.4
	5	164.5	124.6	–	–
VM-BA	2	217.9	209.8	162.8	144.7
	3	174.1	159.7	114.3	87.3
	4	164.9	147.7	102.7	78.2
	5	163.2	144.2	–	–
VM-SR	2	215.1	210.1	161.9	144.4
	3	180.1	137.3	112.7	84.6
	4	157.8	117.7	94.8	68.8
	5	147.8	109.7	–	–
VM-LR	2	207.5	170.8	147.4	128.2
	3	160.6	124.7	103.2	79.3
	4	146.7	112.1	**89.8**	**66.0**
	5	**141.4**	**107.1**	–	–

Table 3: The test set perplexities of the models for orders N=2..5 on training datasets D1-D4.

Model Parametrization We used the development set to determine the values for the absolute discounting parameter D (defined in Section 2.1) and the number of iterations for stochastic gradient ascent. This resulted in a value $D = 0.1$. Stochastic gradient yields best results with a single pass through all instances. More iterations result in overfitting, i.e. decrease training data log-likelihood but increase the log-likelihood on the development data. The step size was kept fixed at $\eta = 1.0$.

Results The results of our experiments are given in Table 3, which shows that for sufficiently high orders VM-SR matches KN on each dataset. As expected, the VMM's strength partly stems from the fact that compared to KN it makes better use of the information contained in the conditioning context, as indicated by the fact that VM-SR matches KN whereas VM-BA doesn't. At orders 4 and 5, VM-LR outperforms KN on all datasets, bringing improvements of around 10% for the two smaller training datasets D1 and D2. Comparing VM-BA and VM-SR at order 4 we see that the 7 additional features used by VM-SR for every instance significantly improve performance and the long-range features further improve performance. Thus richer feature sets consistently lead to higher model accuracy. Similarly, the performance of the VMM improves as one moves to higher orders, thereby increasing the amount of contextual information. For orders 2 and 3 VM-SR is inferior to KN, because the SR feature set at order 2 contains no additional features over KN and at order 3 it only contains one additional feature per instance. At order 4 VM-SR matches KN and, while KN gets worse at order 5, the VMM improves and outperforms KN by around 14%.

The training time (including disk IO) of the order 4 VM-SR on the largest dataset $D4$ is about 30 minutes, whereas KN takes about 6 minutes to train.

4 Conclusions

The main contribution of this paper consists of a novel probabilistic classifier, the VMM, which is based on the idea of combining predictions made by individual features into a mixture model whose components vary from instance to instance and whose mixing proportions reflect the predictive strength of each component. The main advantage of the VMM is that it is straightforward to incorporate rich features without sacrificing scalability or reliability of parameter estimation. Moreover, the VMM is simple to implement and works 'out-of-the-box' without feature selection, or any special tuning or tweaking.

Applied to language modeling, the VMM results in a state-of-the-art generative language model whose relative performance compared to N-Gram models gets better as one incorporates richer feature sets. It scales almost as well to large datasets as standard N-Gram models: training requires only two passes over the data as opposed to a single pass required by N-Gram models. Thus, the experiments provide empirical evidence that the VMM is based on a reasonable set of modeling assumptions, which translate into an accurate and scalable model.

Future work includes further evaluation of the VMM, e.g. as a language model within a speech recognition or machine translation system. Moreover, optimizing memory usage, for example via feature pruning or randomized algorithms, would allow incorporation of richer feature sets and would likely lead to further improvements, as indicated by the experiments in this paper. We also intend to evaluate the performance of the VMM on other lexical prediction tasks and more generally, on other classification tasks with similar characteristics.

Acknowledgments I would like to thank Mirella Lapata and Charles Sutton for their feedback on this work and Abby Levenberg for the preprocessed datasets.

References

Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. 2003. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3:1137–1155.

A. Berger, V. Della Pietra, and S. Della Pietra. 1996. A Maximum Entropy Approach to Natural Language Processing. *Computational Linguistics*, 22(1):39–71.

L. Bottou. 2004. Stochastic Learning. In *Advanced Lectures on Machine Learning*, Lecture Notes in Artificial Intelligence, pages 146–168. Springer Verlag, Berlin/Heidelberg.

S. Chen and J. Goodman. 1999. An Empirical Study of Smoothing Techniques for Language Modeling. *Computer Speech and Language*, 13:359–394.

S. Chen and R. Rosenfeld. 2000. A Survey of Smoothing Techniques for ME Models. *IEEE Transactions on Speech and Audio Processing*, 8(1):37–50.

M. Collins. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1–8, Philadelphia, PA, USA.

J. Goodman. 2001. A Bit of Progress in Language Modeling (Extended Version). Technical report, Microsoft Research, Redmond, WA, USA.

D. Klakow. 1998. Log-Linear Interpolation of Language Models. In *Proceedings of the 5th International Conference on Spoken Language Processing*, pages 1694–1698, Sydney, Australia.

R. Kneser and H. Ney. 1995. Improved Backing-off for M-Gram Language Modeling. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, pages 181–184, Detroit, MI, USA.

D. Lewis, Y. Yang, T. Rose, and F. Li. 2004. RCV1: A New Benchmark Collection for Text Categorization Research. *Journal of Machine Learning Research*, 5:361–397.

A. Mnih and G. Hinton. 2008. A Scalable Hierarchical Distributed Language Model. In *Advances in Neural Information Processing Systems 21*.

H. Ney, U. Essen, and R. Kneser. 1994. On Structuring Probabilistic Dependences in Stochastic Language Modeling. *Computer, Speech and Language*, 8:1–38.

B. Roark, M. Saraclar, and M. Collins. 2007. Discriminative n-gram Language Modeling. *Computer, Speech and Language*, 21:373–392.

R. Rosenfeld. 1994. *Adaptive Statistical Language Modelling: A Maximum Entropy Approach*. Ph.D. thesis, Carnegie Mellon University.

R. Rosenfeld. 1996. A Maximum Entropy Approach to Adaptive Statistical Language Modeling. *Computer, Speech and Language*, 10:187–228.

A. Stolcke. 2002. SRILM – An Extensible Language Modeling Toolkit. In *Proceedings of the 7th International Conference on Spoken Language Processing*, pages 901–904, Denver, CO, USA.

A. Van den Bosch. 2005. Scalable Classification-based Word Prediction and Confusible Correction. *Traitement Automatique des Langues*, 42(2):39–63.

F. Wood, C. Archambeau, J. Gasthaus, L. James, and Y. Teh. 2009. A Stochastic Memoizer for Sequence Data. In *Proceedings of the 24th International Conference on Machine learning*, pages 1129–1136, Montreal, Quebec, Canada.