
Markovian Time Series Models for Language Identification

Arnold Binas

Department of Computer Science
University of Toronto
Toronto, ON M5S 3G4
abinas@cs.toronto.edu

Abstract

In this paper, we compare the performance of several Markovian time series models for language identification of written text. In particular, we focus on the Hidden Markov Model which has only rarely been applied to the language classification problem. We test our classifiers on six languages and analyze our findings.

1 Introduction

Language identification has many applications in the increasingly global information domain. Among many other examples, it could be used by search engines to categorize web pages, by word processors to select the appropriate spell checker to use, for email routing or filtering, or for automatic online translation services. Language identification of *large* text samples is essentially a closed problem. Commercially available systems, such as Xerox's CA language identifier¹, achieve close to perfect success rates for dozens of languages.

The vast majority of existing language identifiers uses character n -gram models of the target languages, i.e. simple Markov chains. For example, Cavnar and Trenkle use n -gram-based rank order statistics for $n = 1 - 5$ to compile profiles of target languages and compare them to profiles of test cases, achieving a 99.8% error rate on language classification of test sequences 300 or more characters long [1]. Xerox's CA calculates the probabilities of a test sentence in separate trigram models for each target language to determine its best guess [2]. Although other techniques have been implemented, n -grams have proven not easy to beat. MacNamara et al. compared a neural networks-based identifier to an n -gram-based one and found n -grams to be superior [3]. Ahmed et al. found a Naïve Bayes classifier inferior to one that uses bigram rank order statistics [4]. Another advantage of n -gram models is that they can be trained very efficiently as the problem of estimating a trigram's maximum likelihood probability is convex and a closed-form solution exists.

N -gram models can, however, take up large amounts of memory and need large amounts of training data for large n . They also offer no further insight into a language except being able to identify it. Large n (> 2) are usually required to achieve good performance. Space considerations become important when a high number of target languages is considered, each requiring their own model. One possible way to reduce storage space requirements

¹<http://www.xrce.xerox.com/competencies/content-analysis/tools/guesser>

is to use Aggregate Markov models (AGMMs), which can be thought of as a regularizing intermediate between uni- and bigrams [5]. AGMMs can also be used to discover classes of characters which might be interesting from a linguistic point of view. To the best of our knowledge, AGMMs have not yet been considered for use in language identification.

Little work, too, has been done on employing Hidden Markov models (HMMs) for the language identification task. Ueda and Nakagawa have considered the use of HMMs in identifying languages in text [6]. They found a 7-state HMM to perform at approximately the level of a trigram model on their data. Ueda and Nakagawa’s results have gone virtually unnoticed in the NLP community and are not easily accessible. Neither have they been reproduced elsewhere in the literature. Just as AGMMs, Hidden Markov models can also be used to discover classes of characters that might be typical for a given language.

In this paper, we compare three different Markovian time series models for language identification in written text: n-grams (unigram and bigram), AGMM, and HMM. We train and test our models on text data from news and sports news websites in various languages and focus on the language identification of very short character sequences. We furthermore show that language identification of larger sequences is a rather easy task, with essentially zero error being achievable even with simple models and moderate amounts of training data. Section 2 below briefly introduces the models used and how they were regularized. In Section 3 we describe our evaluation methodology and the data used in the experiments. In Section 4 we talk about the experiments and findings before we conclude in Section 5.

2 Model Specifications

In this section we briefly review the different Markovian time series models that we compare: simple Markov models, the Aggregate Markov model, and the Hidden Markov model. We assume familiarity with these models and refer the reader to the standard literature for more elaborate treatments of the basic models. Here, we will focus on the Hidden Markov model and applying Laplace smoothing to an HMM’s output parameters in order to be able to train on a modest amount of training data. As a brief technical note, for numerical reasons, we internally actually keep the logarithms of model parameters.

All models are used in a similar way to build language identifiers. First, for each language, one model of each type is built by maximizing the likelihood of the training character sequence. When the language of a test sequence is to be guessed, the likelihood of that sequence in each of the models is determined. The language corresponding to the model in which the test sequence has the highest likelihood is returned as the result.

2.1 Two Simple Markov Models

The simplest time series model is the Markov model. An n^{th} order Markov model looks back n states to determine the probability of transitioning to a given next state. The simplest such model is the unigram (0^{th} order). The unigram does not consider any previous states but, applied to our task at hand, simply looks at the probability distribution of the characters of the alphabet. More sophisticated is the bigram (1^{st} order). In bigrams, the probability of a character depends on its predecessor (the single previous state).

While unigram probabilities for all states can usually be completely estimated from training data, limited amounts of training data do not generally exhibit all possible bigrams or higher order n-grams given the alphabet. Moreover, some n-grams are simply not used in a given language (consider, for example, the trigram *szx* in English). We apply conventional Laplace (“plus-one”) smoothing when training our n-gram models to avoid zero probabilities of n-grams that do not occur in the training data but might occur in a test sequence.

2.2 The Aggregate Markov Model

Saul and Pereira have introduced the Aggregate Markov model (AGMM) for a similar task, language modeling based on sequences of words rather than characters [5]. They showed that an AGMM’s complexity is intermediate between the complexities of uni- and bigrams and can be interpreted as regularizing bigram models by constraining their transition matrices to be of lower rank. In our application, an AGMM first probabilistically maps characters to *categories* of characters and then categories to the next output character. The category is the latent variable of the model. AGMMs are trained by maximizing the likelihood of the training data using the EM algorithm.

It is important to note the significance of building similar models for all languages when using them for classification. More specifically, we can only meaningfully compare the likelihoods of a test sequence in all models if all models use the same number of possible character categories. I.e. the domain size of the latent variable in each model should be the same. The optimal number of categories can be set via a validation data set.

2.3 The Hidden Markov Model

Hidden Markov models have been hugely successful in a plethora of applications, including another important NLP domain, speech recognition. The simplest HMM (which we employ in our experiments) consists of a 1st order Markov chain that is *hidden* behind a probabilistic output function. That is, a given hidden state transitions to the next hidden state with a certain probability. Then the actual observable output is produced according to the probability distribution of possible outputs given the underlying hidden state. In our application, states might represent character categories and outputs are actual characters from the language’s alphabet.

Being a latent variable model, an HMM is also trained by the EM algorithm. In the M-step of the algorithm, the estimate of the output probability of a given character is the ratio of the frequency with which that character is output in state s in the training data and the count of all characters ever output in s (Equation 1), where $\hat{A}(s, c)$ is the estimated probability that character c is output in state j , $\gamma(s, t)$ is the probability that the HMM was in the hidden state s when outputting the t^{th} character of the training sequence, and y_t is the t^{th} character of the training sequence.

$$\hat{A}(s, c) = \frac{\sum_{t|y^t=c} \gamma(s, t)}{\sum_{t=1}^{\tau} \gamma(s, t)} \quad (1)$$

Note that an HMM with only one hidden state therefore behaves as a unigram. A problem arises when not all characters of the alphabet occur in the training sequence— \hat{A} will have zero-entries. In order to circumvent this problem and to avoid overfitting output parameters to the training sequence, we use Laplace smoothing in the M-step for the output parameter matrix A . Observe that in Equation 1, the sum in the numerator is exactly the number of times the character c is output in state s and the denominator is exactly the number of times state s is passed through in the training sequence. Laplace smoothing therefore adds one to the numerator and $|A|$ to the denominator of Equation 1, where $|A|$ is the number of characters in the alphabet. Equation 2 shows the Laplace-smoothed M-step estimate for \hat{A} . This smoothing technique corresponds to placing Dirichlet priors on the output parameters [7].

$$\hat{A}(s, c) = \frac{\sum_{t|y^t=c} \gamma(s, t) + 1}{\sum_{t=1}^{\tau} \gamma(s, t) + |A|} \quad (2)$$

For our purpose of building language classifiers, as with the AGMM, we can only meaningfully compare likelihoods of test sequences when using similarly structured models for each language. I.e., we want to train models with the same number of hidden states for all languages. Again, the optimal number of hidden states can be set by validation.

3 General Methodology

In this section, we talk about the data, how it was acquired, and how it was used in our experiments. We consider only languages that are based on the Roman script. While in an application of language identification we would use hints such as diacritics (accented letters: e.g. é in French, or umlauts: ä in German), we convert all non-standard Roman letters to their base form in order to allow us to better and more generally evaluate our techniques. I.e., we drop all accents and ornaments from non-standard characters and consider only their standard equivalents (é becomes e, ä becomes a, etc.). We furthermore consider common word and sentence separators (Table 1). This results in a total of 34 modeled ASCII characters; other characters potentially occurring in text are dropped during pre-processing.

Table 1: Characters used for language modeling.

| 1–26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |
|------|---------|-------|----|----|----|----|----|----|
| A–Z | newline | space | . | ? | ! | , | ' | “ |

We consider the following six languages: English, French, Spanish, German, Dutch, and Indonesian, which has officially adopted the Roman writing system. The first five are all European languages, with two pairs of somewhat similar languages: French and Spanish as well as Dutch and German. We consider these pairs to evaluate our models in potentially difficult cases. We also include one non-European language, Indonesian, which is highly unrelated to any of the other five, providing an easy test case for our models.

For each language, we acquired several articles each from news and sports news websites from the respective countries. 15,000 characters in total have been acquired for each language, and sentences have been randomly shuffled to avoid training on text in only one topic and testing on text samples talking about a different area. With an abundance of data for language identification—the Internet is overflowing with it—we can afford separate training, validation, and test sets. We chose a non-trivial training set size that is still small enough to allow training the non-convex models in a reasonable amount of time. The validation set is used to set the latent variable parameters (number of classes for the AGMM, number of hidden states for the HMM), and the test set provides a benchmark for comparing the performance of all our models. Using the data sets described above, we obtained test error values for pairwise language identification as well as executing one larger, and more real-life test, in which all six languages were valid targets.

3.1 The Data

As mentioned above, the available data was a 15,000 character long sequence of text per language. For each language, this data set was split into training, validation, and test sets with the even ratio of 5,000:5,000:5,000 characters. We chose to keep the training data size relatively small for two reasons. Firstly, we were bounded by computation time constraints when optimizing the non-convex models (AGMMs and HMMs). Secondly, we tried to artificially keep the error rate up. A combination of training on lots of data and testing on long sequences gives essentially zero error for all reasonably sophisticated models, which

is great for practical applications, but not of much use when trying to compare different models. Figure 1 shows the character distributions of the English and Indonesian training data. Note the frequently occurring spaces (character 28) in both languages and a particularly high number of *A*'s in the Indonesian language. Histograms such as the one in Figure 1 are what unigram models are based on. Figures 2 and 3 show the possible character combination patterns for the same languages. A dot in position (i, j) means that the bigram ij occurs in the language's training data. Note that the plot does not visualize frequencies but simply which characters can follow which other characters. Even so, potentially characteristic differences can already be seen across the different languages.

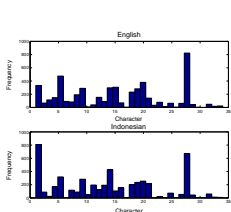


Figure 1: Character distribution in English and Indonesian training sequences.

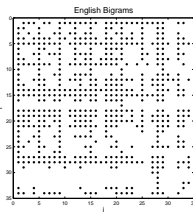


Figure 2: English bigrams occurring in training data.

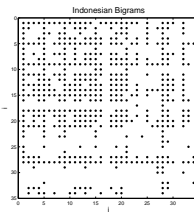


Figure 3: Indonesian bigrams occurring in training data.

The validation set, used to set latent variable domain sizes, and the test set are used as follows. The respective 5,000 characters in the data set are split into $\frac{5,000}{K}$ sequences of length K , not paying attention to word or sentence boundaries. This can be done for testing purposes since a transition from, say, a space to a letter may give away just as much information about the language as a letter-to-letter transition in the middle of a word. We separately test for $K = 10, 30, 50, 100, 200$. It is expected that shorter test sequences yield higher error rates than longer ones. We test on several different test sequence lengths in order to find one for which error rates are high and distinguishable enough to allow the comparison of several different models.

3.2 Evaluation Protocols and Error Measure

For each language's training set, we train a unigram model, a bigram model, ten AGMMs, and ten HMMs with latent variable domain sizes of 1, 3, 5, 7, 10, 15, 20, 25, 30, and 34. The set of all unigram models forms a language identifier, as does the set of all bigram models. For a given latent variable domain size, the sets of all such AGMMs and all such HMMs form one language classifier each. From the ten AGMM classifiers, we chose the one that performed best on the validation data. We picked the best HMM identifier analogously. Validation and testing were conducted as follows. We evaluated our language classifiers by testing each classifier on an equal amount of test/validation data from each target language (5,000 characters per language). The error measure is simply the proportion of wrongly classified sequences.

4 Experiments and Empirical Analysis

We first built one model each per language and model type. For the AGMMs and HMMs, when training a model for a given language and with a given latent variable domain, we executed six random restarts and found the (locally) optimal parameters for each. The set of parameters yielding the highest likelihood of the training data was kept. Figure 4 plots

the HMM log likelihood of the English training sequence as it is maximized by the EM algorithm for ten random restarts. The arrival at at least two distinct local maxima can be observed. This is expected as the problem of optimizing HMM parameters is not convex. Note also that the likelihood converges fairly quickly (after around 50 EM iterations). This means that a local maximum is found early, again possibly preventing the discovery of a better one farther away.

Given the language models, we conducted one experiment in which all six languages were valid target languages, as well as a series of tests in which all possible distinct pairs of languages were tested pairwise, allowing us some insight into their relatedness.

4.1 Six Target Languages

In this subsection we present the results of language identification tests with six target languages. Figure 5 visualizes the intuitive fact that longer test sequences result in a lower identification error. The test error for all classifiers is plotted, i.e. unigram-based, bigram-based, and AGGM- and HMM-based for all latent variable domain sizes. Figure 6 shows the HMM classifier validation error for all numbers of hidden states and all lengths of test sequences. It shows that at validation time, most of the time the 20-state classifier is chosen.

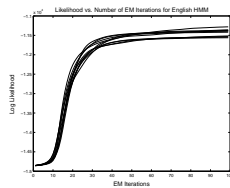


Figure 4: Expectation-maximization of English HMM with 10 random restarts.

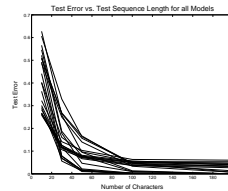


Figure 5: Test error versus length of test sequence for all classifiers.

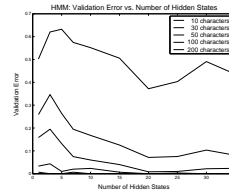


Figure 6: HMM classifier validation error versus number of hidden states.

Table 2 lists the error values of all models for all tested lengths of character sequences. The “winning” classifiers are shown in bold font. For the AGGM- and HMM-based classifiers, the best-performing numbers of categories/states as determined by validation are also shown. Figure 7 shows a bar plot of the second row of Table 2, the test errors of different models for 30-character long test sequences. Figure 8 compares the HMM classifier test errors for all numbers of hidden states with the unigram- and bigram-based classifiers for the same number of characters in a test sequence. Note that the best HMM-based classifier almost always performs at the level of the bigram-based one or slightly better. Note also that the HMM classifier works best with about 20 states. The true optimum might lie somewhere around this number as we only investigated a limited set of possible numbers of hidden states. For the test sequence length of 200, the 34-states model was chosen as a tie in the validation phase. The 20-states HMMs also result in a zero test error rate on 200-character sequences. The AGMM-based identifier’s performance is surprisingly poor. With its low capacity, the unigram-based classifier, as expected, always performs worse than the bigram-based one.

4.2 Binary Language Identification

Table 3 shows pairwise test error results of the best-performing HMM-based classifier for all binary combinations of the six languages considered. We show here the test errors for very short (10 characters; corresponding to about two words) test sequences as longer se-

Table 2: Test performance summary—six languages.

| Test case length | Unigram | Bigram | AGMM | HMM |
|------------------|---------|---------------|--------------------|--------------------|
| 10 | 0.5080 | 0.2920 | 0.2587 (25) | 0.3633 (20) |
| 30 | 0.2685 | 0.0679 | 0.1165 (25) | 0.0562 (20) |
| 50 | 0.1667 | 0.0183 | 0.0667 (25) | 0.0133 (20) |
| 100 | 0.0400 | 0.0000 | 0.0367 (25) | 0.0000 (20) |
| 200 | 0.0133 | 0.0000 | 0.0333 (1) | 0.0000 (34) |

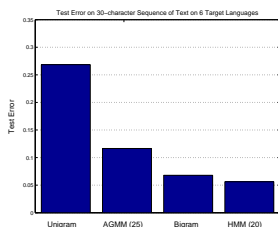


Figure 7: Six-language identification test error for all classifiers on 30-character test sequences.

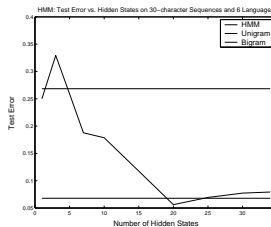


Figure 8: Test error of HMM classifier versus number of hidden states for 30-character sequences. Uni- and bigram test errors are shown as horizontal lines.

quences yield very small or zero test errors, making a meaningful comparison impossible. Interpreting a higher test error as a higher similarity of the two languages tested, we can make a few expected observations. Separating Indonesian and any of the European languages gives a low test error, confirming that indeed those languages are not very related. Two pairs of languages that stand out as being quite similar (high test error) are French and Spanish as well as Dutch and German. The binary language identification errors for 30-character long sequences are all at most 3% and those for longer sequences are essentially zero. Binary language separation with HMMs is thus a fairly easy task even for relatively short writing samples.

Table 3: HMM: Binary language separation on 10-character sequences of text—test error.

| | Indonesian | Spanish | French | Dutch | German |
|-------------------|------------|---------|--------|--------|--------|
| English | 0.1120 | 0.1470 | 0.1690 | 0.1680 | 0.1770 |
| Indonesian | | 0.1110 | 0.0900 | 0.1100 | 0.1030 |
| Spanish | | | 0.2100 | 0.0990 | 0.1090 |
| French | | | | 0.1250 | 0.1650 |
| Dutch | | | | | 0.2200 |

4.3 Observations from Model Parameters

The latent variable models we trained can be used to discover categories of characters in a language. More specifically, we take the class that is most probable for a given character in an AGMM to be its category. The state for which the output of a given character is most probable is the HMM-based category for that character. Table 4 lists all categories

of characters discovered by a 10-class Aggregate Markov model for the English language. Table 5 lists character categories discovered by a 10-state HMM. Note that both models have exclusive categories covering almost all vowels (categories 5 and 7 in both cases). Many of the consonants also appear among similar neighbors in both model’s categories. The HMM does a fairly good job in grouping non-letter characters together, namely in category 8.

Table 4: English character categories discovered by a 10-class AGMM.

| | | | |
|---|---------|----|-------------|
| 1 | CFLP! | 6 | N |
| 2 | HKMRVWZ | 7 | EU |
| 3 | BJOQ | 8 | newline . ? |
| 4 | T | 9 | X space ’ |
| 5 | AI | 10 | DGSY, “ |

Table 5: English character categories discovered by a 10-state HMM.

| | | | |
|---|-------|----|-----------------|
| 1 | SY, ’ | 6 | FLN |
| 2 | H | 7 | EO |
| 3 | MRVXZ | 8 | newline . ? ! “ |
| 4 | DGKT | 9 | BCJPW |
| 5 | AIU | 10 | Q space |

5 Conclusions and Future Work

We have compared several common Markovian time series models for the use of language identification. We showed that a first-order simple Markov model classifier is slightly outperformed by its corresponding Hidden Markov model classifier on a multiple-target language classification task. It was demonstrated how binary language classification errors can be used to infer the degree of relationship between two languages. We also showed how latent variable models can be used to discover meaningful character classes. It remains to mention that the full potential of HMM-based language identifiers has not yet been realized in our experiments, which raises three possibilities of future work. Firstly, more effort could be spent on finding a good local maximum in the training set likelihood when optimizing HMM parameters. Secondly, more numbers of hidden states could be tried as our experiments suggest that the optimal number of hidden states is not at all intuitive. Lastly, more sophisticated regularization techniques could be used. Absolute discounting and shrinkage come to mind as worthy candidates.

References

- [1] William B. Cavnar and John M. Trenkle. N-gram-based text categorization. In *Proc. of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, Las Vegas, US, 1994.
- [2] Greg Grefenstette. Comparing two language identification schemes. In *Proc. of the 3rd International Conference on Statistical Analysis of Textual Data (JADT 95)*, Rome, Italy, 1995.
- [3] Shane MacNamara, Pádraig Cunningham, and John Byrne. Neural networks for language identification: A comparative study. *Information Processessing and Mgmt.*, 34(4):395–403, 1998.
- [4] Bashir Ahmed, Sung-Hyuk Cha, and Charles Tappert. Language identification from text using n-gram based cumulative frequency addition. In *Proc. of CSIS Research Day*, pages 12.1–12.8, Pace University, NY, 2004.
- [5] Lawrence Saul and Fernando Pereira. Aggregate and mixed-order Markov models for statistical language processing. In *Proc. of the Second Conference on Empirical Methods in Natural Language Processing*, pages 81–89, Somerset, New Jersey, 1997.
- [6] Yoshio Ueda and Seiichi Nakagawa. Diction for phoneme/syllable/word-category and identification of language using HMM. In *Proc. of the First International Conference on Spoken Language Processing (ICSLP 90)*, pages 1209–1212, 1990.
- [7] Dayne Freitag and Andrew Kachites McCallum. Information extraction with HMMs and shrinkage. In *Proc. of the AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999.