Feature Hashing for Language and Dialect Identification

Shervin Malmasi
Harvard Medical School, Boston, MA
Macquarie University, Australia
shervin.malmasi@mq.edu.au

Mark Dras
Macquarie University
Sydney, Australia
mark.dras@mq.edu.au

Abstract

We evaluate feature hashing for language identification (LID), a method not previously used for this task. Using a standard dataset, we first show that while feature performance is high, LID data is highly dimensional and mostly sparse ($>99.5\%$) as it includes large vocabularies for many languages; memory requirements grow as languages are added. Next we apply hashing using various hash sizes, demonstrating that there is no performance loss with dimensionality reductions of up to 86%. We also show that using an ensemble of low-dimension hash-based classifiers further boosts performance. Feature hashing is highly useful for LID and holds great promise for future work in this area.

1 Introduction

Language Identification (LID) is the task of determining the language of a text, at the document, sub-document or even sentence level. LID is a fundamental preprocessing task in NLP and is also used in lexicography, machine translation and information retrieval. It is widely used for filtering to select documents in a specific language; *e.g.* LID can filter webpages or tweets by language.

Although LID has been widely studied, several open issues remain (Hughes et al., 2006). Current goals include developing models that can identify thousands of languages; extending the task to more fine-grained dialect identification; and making LID functionality more readily available to users/developers. A common challenge among these goals is dealing with high dimensional feature spaces. LID differs from traditional text categorization tasks in some important aspects. Standard tasks, such as topic classification, are usually performed within a single language, and the maximum feature space size is a function of the single language's vocabulary. However, LID must deal with vocabulary from many languages and the feature space grows prodigiously.

This raises immediate concerns about memory requirements for such systems and portends implementation issues for applying the systems to dozens, hundreds or even thousands of languages. Recent LID work has reported results on datasets including over 1,300 languages (Brown, 2014), albeit using small samples. Such models are going to include an extraordinarily large feature space, and individual vectors for each sample are going to be extremely sparse. LID is usually done using n-grams and as the number of languages and/or n gets larger, the feature space will become prohibitively large or impractical for real-world use.

For high dimensional input, traditional dimensionality reduction methods (*e.g.* PCA, LDA) can be computationally expensive. Feature selection methods, *e.g.* those using entropy, are simpler but still expensive. Recently, feature hashing has been shown to be a very effective dimensionality reduction method (Weinberger et al., 2009). It has proven to be useful in numerous machine learning applications, particularly for handling extremely high dimensional data. It also provides numerous other benefits, which we describe in §2.1.

Although hashing could be tremendously useful for LID, to our knowledge no such experiments have been reported to date. It is unclear how collisions of features from different languages would affect its application for LID. Accordingly, the aims of the present work are to: (1) evaluate the effectiveness of hashing for LID; (2) compare its performance to the standard n-gram approach; (3) assess the role of hash size (and collision rate) on accuracy for different feature types; and (4) determine if ensemble methods can boost performance.

Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Short Papers), pages 399–403
Vancouver, Canada, July 30 - August 4, 2017. ©2017 Association for Computational Linguistics
https://doi.org/10.18653/v1/P17-2063

2 Related Work

2.1 Language and Dialect Identification

Work in language identification (LID) dates back to the seminal work of Beesley (1988), Dunning (1994) and Cavnar and Trenkle (1994). Automatic LID methods have since been widely used in NLP research and applications. Recently, attention has turned to discriminating between close languages, such as Malay-Indonesian and Croatian-Serbian (Ljubešić et al., 2007), or even dialects/varieties of one language, *e.g.* Arabic dialects (Malmasi et al., 2015). This has been the focus of the "Discriminating Similar Language" (DSL) shared task series in recent years. In this work we use data from the 2016 task (Malmasi et al., 2016).

The 2016 task used data from 12 different languages/dialects. A training and development set consisting of 20,000 sentences from each language and an unlabelled test set of 1,000 sentences per language was used for evaluation. Most participants relied on multi-class discriminative classifiers trained with word unigrams and character n-grams (Malmasi et al., 2016).

2.2 Feature Hashing

Feature hashing is a method for mapping a high-dimensional input to a low-dimensional space using hashing (Weinberger et al., 2009). Hashing has proven to be simple, efficient and effective. It has been applied to various tasks including protein sequence classification (Caragea et al., 2012), sentiment analysis (Da Silva et al., 2014), and malware detection (Jang et al., 2011).

This method uses a hash function $h(x)$ to arbitrarily map input to a hash key of a specified size. The hash size, *e.g.* 2^{18}, determines the size of the mapped feature space. Hash functions are many-to-one mappings. Collision occur when distinct inputs yield the same output, *i.e.* $h(a) = h(b)$. The collision rate is affected by the hash size. From a learning perspective, collisions cause random clustering of features and introduce noise; unrelated features map to the same vector index and may degrade the learner's accuracy. However, it has been shown that "the interference between independently hashed subspaces is negligible with high probability" (Weinberger et al., 2009).

A positive by-product of hashing is that it eliminates the need for a feature dictionary. In NLP, bag-of-words models require a full pass over the data to identify the vocabulary for each feature type (*e.g.* n-grams) and build a feature index. Eliminating this has many benefits: it simplifies implementation of feature extraction methods, reduces memory overhead, and facilitates distributed computing. Global statistics, *e.g.* totals and per-class feature counts, are usually required for feature selection and dimensionality reduction methods. Feature hashing may eliminate the need for full processing of the data to calculate these.

3 Data and Experimental Setup

Our methodology is based on the results of 2016 DSL Shared Task (Malmasi et al., 2016) and we use their dataset. The DSL task is performed at the sentence level, making it more challenging.

3.1 Data

A key shortcoming in LID research has been the absence of a common dataset for evaluation (Hughes et al., 2006), a need that has been met by the corpora released as part of the DSL shared task series. We use the DSLCC 3.0 corpus from the 2016 DSL task.[1] This allows us to compare our findings to that of the 17 participants. Using a standard, publicly available dataset also facilitates replicability of our results. The 2016 task used data from 12 different languages and varieties,[2] including training/development sets composed of 20,000 sentences per language. An unlabelled test set of 1,000 sentences per language was used for evaluation. The total sentences for training and testing are 240k and 12k, respectively. We report our results on the standard test set.

3.2 Classifier and Evaluation

Participants applied various methods, but the task organizers note that linear classifiers, particularly SVMs, were the most successful (Malmasi et al., 2016). This is unsurprising as SVMs have been very successful for text classification and we adopt this method. The data is balanced across classes, so accuracy is used as the evaluation metric.

3.3 Features

Most DSL entries use surface features, with words and high-order character n-grams being particularly successful. We apply character n-grams of order 1–6 (`CH1-6`) and word unigrams (`WD1`).

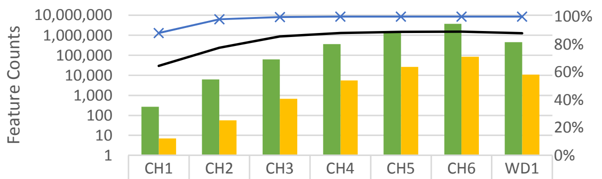
[1] http://ttg.uni-saarland.de/resources/DSLCC/

[2] Bosnian (BS), Argentine Spanish (ES_AR), Peninsular Spanish (ES_ES), Mexican Spanish (ES_MX), Canadian French (FR_CA), Hexagonal French (FR_FR), Croatian (HR), Indonesian (ID), Malay (MY), Brazilian Portuguese (PT_BR), European Portuguese (PT_PT) and Serbian (SR).

4 Feature Performance & Dimensionality

In our first experiment we examine the feature space in our dataset and establish the memory requirements and accuracy of the feature types we use (character 1–6 n-grams and word unigrams).

For each feature type we extract the training vectors and use it to train a linear SVM model which is used to classify the standard test set. We report the feature's accuracy on the test set along with some statistics about the data: the number of features in the training data, the number of out-of-vocabulary (OOV) features[3] in the test data, and the sparsity[4] of the training data matrix. These results are shown in Figure 1.



	CH1	CH2	CH3	CH4	CH5	CH6	WD1
🟩 Feat Count	272	6.1k	61.5k	358.3k	1.4m	3.7m	443.9k
🟨 Test OOV	7	57	0.7k	5.5k	26.0k	84.6k	10.7k
×Sparsity (%)	88.17	98.04	99.7	99.94	99.98	99.99	99.99
—Test Acc. (%)	64.49	77.49	85.77	88.21	89.09	89.2	87.93

Figure 1: Details of our feature spaces. Training set feature counts and test set OOV counts are shown as bars (left axis, logarithmic scale). Training data sparsity and test set accuracy are shown as lines (in %, right axis).

These results reveal a number of interesting patterns. Character n-grams perform better than word unigrams. The winner of the 2016 DSL task combined various character n-grams into an SVM model to obtain 89.4% accuracy on the test set. We obtain the best result of 89.2% using character 6-grams alone. The character n-gram feature grow rapidly, from 61k trigrams to 3.7m 6-grams. While accuracy plateaus at 6-grams, there is only a 1% improvement from 4- to 6-grams, but a huge feature space increase. The number of OOV test features also increases, but is relatively small.

The sparsity analysis is also revealing, showing that for many features only around 0.1% of the training matrix contains non-zero values. We can expect that this sparsity will rapidly grow with the addition of more classes to the dataset. This poses a huge problem for practical applications of LID for discriminating large number of languages.

In this regard, we can also assess how the dimensionality cumulatively increases as languages are added, shown in Figure 2. Features increase even as similar languages are added; we expect this trend to continue if more classes are added.

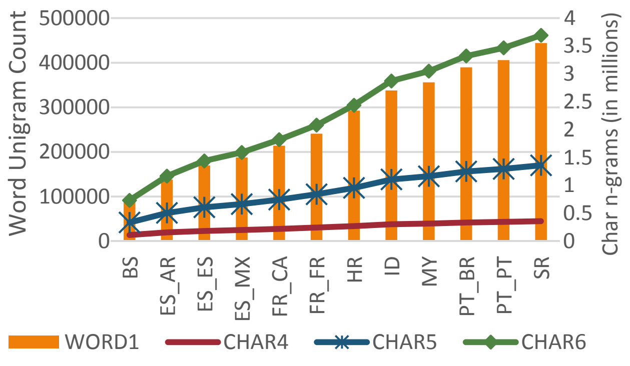


Figure 2: Feature growth rate as classes are added. Words (bars, left axis) and character n-grams (lines, right axis) grow as languages are added.

5 Feature Hashing Performance

Having established baseline performance for our features using the standard approach, we now experiment with applying feature hashing to the same data in order to evaluate its effectiveness for this task and compare it to the standard approach.

We also assess the effect of hash size on the feature collision rate, and in turn, on classification accuracy. To do this we test each feature[5] using hash sizes in the range 2^{10} (1024) to 2^{22} (2.1m) features, which covers most of our feature types. Our hash function is implemented using the signed 32-bit version of MurmurHash3.[6]

We report each feature's accuracy at every hash size, with the smallest hash that yields maximal accuracy considered to be the best result. Each feature is compared against its performance using the full feature space (baseline). These results, along with the reduction in the feature space for the best results, are listed in Table 1.

Our first observation is that every feature matches baseline performance at a hash size smaller than its full feature space. This demonstrates that feature hashing is useful for LID.

We can also assess the effect of feature collisions using the results, which we plot in Figure 3. We note that at the same hash size, features with a larger space perform worse. That is, with a 2^{12} hash size, CHAR4 outperforms 5- and 6-grams. This is evidence of performance degradation due to hash collision. However, we see that when using an appropriately sized hash, feature collisions between languages do not degrade performance.

[3] *i.e.* features present in the test set but not the training set.
[4] Matrix sparsity is the proportion of zero-valued elements.
[5] Except character unigrams which only have 272 features.
[6] https://github.com/aappleby/smhasher

		Hash Size													
Feature	**Baseline**	$\mathbf{2^{10}}$	$\mathbf{2^{11}}$	$\mathbf{2^{12}}$	$\mathbf{2^{13}}$	$\mathbf{2^{14}}$	$\mathbf{2^{15}}$	$\mathbf{2^{16}}$	$\mathbf{2^{17}}$	$\mathbf{2^{18}}$	$\mathbf{2^{19}}$	$\mathbf{2^{20}}$	$\mathbf{2^{21}}$	$\mathbf{2^{22}}$	**△ feats**
CHAR2	0.77	0.74	0.76	**0.77**	0.77	0.77	0.77	0.77	0.77	0.77	0.77	0.77	0.77	0.77	-33%
CHAR3	0.86	0.74	0.79	0.82	0.84	0.85	**0.86**	0.86	0.86	0.86	0.86	0.86	0.86	0.86	-47%
CHAR4	0.88	0.70	0.76	0.80	0.83	0.86	0.86	**0.88**	0.88	0.88	0.88	0.88	0.88	0.88	-82%
CHAR5	0.89	0.65	0.72	0.77	0.81	0.84	0.86	0.87	0.88	**0.89**	0.89	0.89	0.89	0.89	-81%
CHAR6	0.89	0.58	0.67	0.73	0.78	0.82	0.84	0.87	0.87	0.88	**0.89**	0.89	0.89	0.89	-86%
WORD1	0.88	0.70	0.74	0.78	0.81	0.83	0.85	0.86	0.87	**0.88**	0.88	0.88	0.88	0.88	-41%

Table 1: Test set accuracy for hashed features at each hash size. Baseline is accuracy without hashing. Best result (w/ smallest hash) per row in bold. Last column is the best result's reduction in dimensionality. We observe that every feature matches its baseline at a hash size smaller than its full feature space.

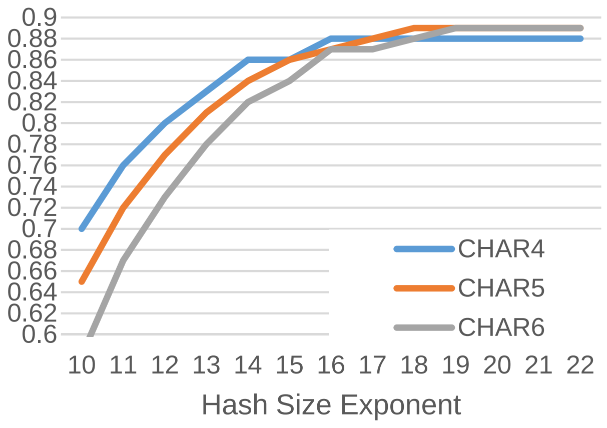


Figure 3: Performance (left axis) of 3 features at different hash sizes. Higher order n-grams perform worse at lower hash sizes due to collisions.

We also analyze the memory reduction achieved via hashing by calculating the relative difference in dimensionality between the best result and the full feature set, listed in the last column of Table 1. We see very significant reductions of up to 86% in dimensionality without any performance loss. Character 4-grams yield very competitive results (0.88) with a large feature space reduction of 82% using a hash size of 2^{16}.

6 Hashing-based Ensemble Classifier

Ensemble classifiers combine multiple learners with the aim of improving accuracy through enhanced decision making. They have been applied to many tasks and shown to achieve better results compared to single-classifier methods (Oza and Tumer, 2008). By aggregating the outputs of multiple classifiers their outputs are generally considered to be more robust. Ensembles have been successfully used for LID, *e.g.* winning the 2015 task (Malmasi and Dras, 2015a). They also achieve state-of-the-art performance for Native Language Identification (Malmasi and Dras, 2017). Could an ensemble composed of low-dimension hash-based classifiers achieve competitive performance?

In order to assess this we created an ensemble of our features with a hash size of 2^{16}. Evaluating against the test set, a hard voting ensemble achieved 88.7% accuracy while a probability-based combination obtained 89.2%. Comparing to the winning shared task accuracy of 89.4%, this is an excellent result given that only 65,536 features were used by our system. Ensemble combination boosted our best single-model 2^{16} hash size result by 1.1%. This highlights the utility of ensemble methods for hashing-based feature spaces. It also shows that model combination can compensate for small performance losses caused by hashing.

7 Discussion and Conclusion

We presented the first application of feature hashing for language identification. Results show that hashing is highly effective for LID and can ameliorate the dimensionality issues that can impose prohibitive memory requirements for some LID tasks. We further showed that reduced feature spaces with as few as 65k features can be combined in ensemble classifiers to boost performance. We also demonstrated the effect of hash collision on accuracy, and outlined the type of analysis needed to choose the correct hash size for a given feature.

Hashing provided dimensionality reductions of up to 86% without performance degradation. This is impressive considering that no feature selection or analysis was performed, making it highly efficient. This reduction facilitates model loading and training as we also showed that LID data is extremely sparse, with over 99% of our training matrix cells containing zeros. This is particularly useful for limited memory scenarios (*e.g.* handheld or embedded devices). It may also enable the use of methods requiring dense data representations, something often infeasible for large datasets.

Another key advantage of hashing is that it eliminates the need for maintaining a feature dictionary, making it easy to develop feature extraction modules. This greatly simplifies parallelization, lending itself to online learning and distributed systems, which are important issues for LID systems in our experience.

Hashing also holds promise for facilitating the use of deep learning methods for LID. In the 2016 DSL task, such systems performed "poorly compared to traditional classifiers"; participants cited "memory requirements and long training times" spanning several days (Malmasi et al., 2016). Feature hashing has recently been used to compress neural networks (Chen et al., 2015) and its application for deep learning-based text classification may provide insightful results.

There are also downsides to hashing, including the inability to interpret feature weights and model parameters, and some minor performance loss.

Future work in this area includes evaluation on larger datasets, as well as cross-corpus experiments, which may also be insightful. The application of these methods to other text classification tasks, particularly those dealing with language varieties such as Native Language Identification (Malmasi and Dras, 2015b), could also provide a deeper understanding about how they work.

References

Kenneth R Beesley. 1988. Language identifier: A computer program for automatic natural-language identification of on-line text. In *Proceedings of the 29th Annual Conference of the American Translators Association*. page 54.

Ralf D Brown. 2014. Non-linear Mapping for Improved Identification of 1300+ Languages. In *EMNLP*.

Cornelia Caragea, Adrian Silvescu, and Prasenjit Mitra. 2012. Protein sequence classification using feature hashing. *Proteome science* 10(1):S14.

William B. Cavnar and John M. Trenkle. 1994. N-Gram-Based Text Categorization. In *Proceedings of SDAIR-94*. Las Vegas, US, pages 161–175.

Wenlin Chen, James T Wilson, Stephen Tyree, Kilian Q Weinberger, and Yixin Chen. 2015. Compressing neural networks with the hashing trick. In *ICML*. pages 2285–2294.

Nadia FF Da Silva, Eduardo R Hruschka, and Estevam R Hruschka. 2014. Tweet sentiment analysis with classifier ensembles. *Decision Support Systems* 66:170–179.

Ted Dunning. 1994. *Statistical identification of language*. Computing Research Laboratory, New Mexico State University.

Baden Hughes, Timothy Baldwin, Steven Bird, Jeremy Nicholson, and Andrew MacKinlay. 2006. Reconsidering language identification for written language resources .

Jiyong Jang, David Brumley, and Shobha Venkataraman. 2011. Bitshred: feature hashing malware for scalable triage and semantic analysis. In *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, pages 309–320.

Nikola Ljubešić, Nives Mikelić, and Damir Boras. 2007. Language indentification: How to distinguish similar languages? In *Information Technology Interfaces*. pages 541–546.

Shervin Malmasi and Mark Dras. 2015a. Language Identification using Classifier Ensembles. In *Proceedings of LT4VarDial 2015*.

Shervin Malmasi and Mark Dras. 2015b. Multilingual Native Language Identification. In *Natural Language Engineering*.

Shervin Malmasi and Mark Dras. 2017. Native Language Identification using Stacked Generalization. *arXiv preprint arXiv:1703.06541* .

Shervin Malmasi, Eshrag Refaee, and Mark Dras. 2015. Arabic Dialect Identification using a Parallel Multidialectal Corpus. In *Proceedings of PACLING 2015*. pages 209–217.

Shervin Malmasi, Marcos Zampieri, Nikola Ljubešić, Preslav Nakov, Ahmed Ali, and Jörg Tiedemann. 2016. Discriminating between Similar Languages and Arabic Dialect Identification: A Report on the Third DSL Shared Task. In *Proceedings of the 3rd Workshop on Language Technology for Closely Related Languages, Varieties and Dialects (VarDial)*. Osaka, Japan, pages 1–14.

Nikunj C Oza and Kagan Tumer. 2008. Classifier ensembles: Select real-world applications. *Information Fusion* 9(1):4–20.

Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. 2009. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, pages 1113–1120.