# Much Gracias: Semi-supervised Code-switch Detection for Spanish-English: How far can we get?

**Dana-Maria Iliescu***, **Rasmus Grand***, **Sara Qirko***, **Rob van der Goot**
IT-University of Copenhagen
dail@itu.dk, gran@itu.dk, saqi@itu.dk, robv@itu.dk

## Abstract

Because of globalization, it is becoming more and more common to use multiple languages in a single utterance, also called code-switching. This results in special linguistic structures and, therefore, poses many challenges for Natural Language Processing. Existing models for language identification in code-switched data are all supervised, requiring annotated training data which is only available for a limited number of language pairs. In this paper, we explore semi-supervised approaches, that exploit out-of-domain monolingual training data. We experiment with word uni-grams, word $n$-grams, character $n$-grams, Viterbi Decoding, Latent Dirichlet Allocation, Support Vector Machine and Logistic Regression. The Viterbi model was the best semi-supervised model, scoring a weighted F1 score of 92.23%, whereas a fully supervised state-of-the-art BERT-based model scored 98.43%.[1]

## 1 Introduction

Social platforms have been the cradle of the internet, driving vast amounts of communication among people from all over the world. As a consequence, the way people communicate in written text has changed, as now it is common to use, for example, abbreviations of words, emoticons, references to other users and use multiple languages within the same utterance. An annotated example sentence of this is the following tweet:

| **Word** | El | online | exercise | de | hoy | : |
|---|---|---|---|---|---|---|
| **Label** | es | en | en | es | es | other |

This phenomenon has caught particular interest in both sociolinguistics and Natural Language Processing (NLP) (Aguilar et al., 2020; Khanuja et al., 2020).

Classifying the language labels on the word level (i.e. code-switch detection) has shown to be beneficial to improve performance on downstream NLP tasks, like dependency parsing (Bhat et al., 2018) or lexical normalization (Barik et al., 2019). Previous work has shown that high performances can be achieved for this task for many language pairs (Molina et al., 2016; Banerjee et al., 2016). However, to the best of our knowledge, most previous work focused on supervised settings, restraining their usefulness to language pairs for which annotated datasets exist. Recent efforts to unify existing datasets have collected annotation for 4 (Aguilar et al., 2020) and 2 (Khanuja et al., 2020) language pairs, which confirms that annotated data is not available for most language pairs.

In supervised settings, recent transformer models (Vaswani et al., 2017; Devlin et al., 2019) have reached a new state-of-the-art (Aguilar et al., 2020; Khanuja et al., 2020), outperforming Bi-LSTMS and traditional machine learning methods used earlier (Molina et al., 2016; Banerjee et al., 2016). Yirmibeşoğlu and Eryiğit (2018) tackled this task in a semi-supervised setup as well, where they used character $n$-gram language models trained on monolingual data to predict perplexity on the target word for classification. They show that this obtains a micro-average F1 score of 92.9%, compared to 95.6% with a supervised CRF-model.

To overcome this limitation, **we focus on exploiting only mono-lingual datasets for performing word-level language identification in code-switched data. We refer to this setup as semi-supervised, since we have no data annotated for the task at hand (code-switch detection).** This enables the possibility to easily train models for new language pairs, and leads to the research question: *How do semi-supervised models compare and perform in the task of language identification in English-Spanish code-switched data?* (**RQ1**).

* Equal contributions
[1] https://github.com/RalleGr/msc-code-switching

Since supervised methods have the advantage of learning from annotated data, the second research question is: *How much can we reduce the gap in performance between the aforementioned semi-supervised models and a supervised state-of-the-art model?* (**RQ2**).

Previous work in similar setups have automatically generated code-switched data from monolingual datasets (Santy et al., 2021). We consider this approach to be orthogonal to ours, and Santy et al. (2021) exploit mono-lingual in-domain data, syntactic parsers and parallel sentences.

## 2 Datasets

In this section, we will first describe the manually annotated code-switched data that we use for evaluating our models, then we describe the monolingual data that we will use as "training" data. It should be noted that this is not real training data, as it is not annotated for the task at hand (thus the setting is semi-supervised).

### 2.1 Test data

To evaluate and compare our models, we use the Spanish-English (SPA-EN) part of the LinCE benchmark (a total of 32,651 posts equivalent to 390,953 tokens) (Aguilar et al., 2020). We chose this language pair because it has the challenge of increased similarity between the languages (Tristram, 1999). In the original data, 8 labels are used, from which we only focus on the 3 labels necessary for the language identification task: `lang1`, `lang2` and `other`, for English, Spanish and punctuation, numbers, symbols and emoticons, respectively. We use the default development and test splits for our experiments.

### 2.2 Monolingual data

In order to perform semi-supervised code-switching detection, we use Wikipedia data, because it is available in many languages and easy to obtain. We extracted dumps from September 1st 2020 with Wikiextractor[2]. Without punctuation and numbers, the English dataset contains 420K distinct words and the Spanish dataset contains 610K distinct words.

It should be noted that there is a domain difference between the training and the dev/test data. However, collecting monolingual data from Twitter is non-trivial.[3] Furthermore, it should be noted that the Wikipedia datasets are not 100% monolingual, so there will be some Spanish data in the English dump and vice-versa. Both of these artefacts might have a negative effect on performance.

### 2.3 Automated annotation for monolingual tokens

Tokenization of the raw datasets is done using the English and Spanish SpaCy tokenization models[4], as it matches the tokenization of the development and test sets. Punctuation and non-word tokens (the `other` class) are identified with manually designed rules using regular expressions, and the python `emoji` package. Tokens that are not identified as `other`, are labeled with the corresponding label based on the language of the wikipedia.

## 3 Methods

### 3.1 Word uni-grams

We first clean the mono-lingual Wikipedia data by removing XML/HTML tags from the articles and special tokens that belong to the `other` class. We calculate the word probability based on the resulting data (word frequency/total number of words) using Laplace smoothing with a smoothing factor of 1.

### 3.2 Word n-grams

We also experiment with taking a larger context into account through bi-grams and tri-grams. Here, we divide the frequency of the $n$-gram containing the word with the frequency of the leading $(n-1)$-gram. The probability is computed this way for a given word in each language, and then the label with the highest probability is assigned to the word. Laplace smoothing with a factor of 1 is used. In our initial experiments, tri-grams showed very low performance, so we use bi-grams in the remainder of this paper.

### 3.3 Character n-grams

For this model, we calculate the joint log probability of words based on the monolingual training data, and assign the most probable label. We vary the $n$-gram size from 1 to 6 and use Laplace smoothing with a factor of 1.

[2] https://github.com/attardi/wikiextractor

[3] Twitter blog: Evaluating language identification performance

[4] https://spacy.io/

### 3.4 Viterbi decoding

The problem of code-switching can be represented as a Hidden Markov Model (HMM) problem, since a sentence can be seen as a Markov chain with hidden states that are the two different languages.

We use the Viterbi decoding algorithm (Forney, 1973) to find the most probable sequence of states given the observations - namely, to assign a language label (state) to each word (observation). We used eginhard's implementation[5] of the Viterbi algorithm and modified the starting and transition probabilities to the values specified below, which were found to be optimal using grid search on the development set using the range of initial probabilities for English from 0.1 to 0.9 with step size 0.1, transition probabilities for English from 0.05 to 0.95 with step size 0.05. The final hyperparameters are as follows:

- states: `lang1` and `lang2`, `other` tokens are identified based on heuristics (see Section 2.3);
- initial probabilities: 0.6 for English and 0.4 for Spanish;
- transition probabilities: 0.15 for transitioning to a different language and 0.85 for transitioning to the same language;
- emission probabilities: these are estimated through a relative probability model, the probability of the word being emitted from English, for example, is:

$$P(w) = \frac{P(w|EN)}{P(w|EN) + P(w|SPA)}, \quad (1)$$

  where $P(w|EN)$ and $P(w|SPA)$ are probabilities given by the dictionaries described in section 3.1. In case this is 0 (i.e. the word does not occur in our monolingual data), the emission probability is calculated by a relative character bi-gram probability.

### 3.5 Latent Dirichlet Allocation

Generally, Latent Dirichlet Allocation (LDA) aims to find the topics a document belongs to using the words in the document as features. In our case, the documents are the words, the features are character $n$-grams (with $n$ 1 to 5) and the topics are English and Spanish. The LDA algorithm does not output labels for the resulting clusters, so we select the top 10 words based on weight that represent best each cluster, and assign them a language using the word uni-gram method (Section 3.1). We use the Scikit Learn[6] implementation of LDA with the `TfidfVectorizer` and use only the first 100,000 words from each monolingual dataset, in order to reduce training time.

### 3.6 Support Vector Machine

For our Support Vector Machine (SVM) model, we consider the monolingual data (Section 2.2) to be the gold training data, without tokens from the `other` class. Using `TfidfVectorizer`, we extract character $n$-gram features from each word, with $n$ 1 to 5. We use the Scikit Learn implementation with all default parameters and select the first 100,000 words from each dataset.

### 3.7 Logistic regression

We use Logistic Regression in a weakly-supervised manner, the same as with SVM, where we consider the first 100,000 words from each Wikipedia dataset to be the gold training data. Again, we use `TfidfVectorizer` to extract character $n$-gram features, with $n$ 1 to 5, and rely on the default Scikit Learn implementation.

### 3.8 Ensemble model

We also experiment with ensembling the previous methods, where we use a simple majority voting. We compare using all models, to using the best 3 and the best 5 models, as well as an oracle.

## 4 Results

To evaluate the performance of our models, we use weighted F1 score[7]. As found in Table 1, Viterbi has the overall best performance scoring 95.76% on validation data and 92.23% on the test data. Word uni-grams, character $n$-grams, SVM and Logistic Regression models achieve results with a range of weighted F1 score from 90.34% to 92.19% on validation data and a range from 87.80% to 88.95% on test data.

We compare our performance to a supervised BERT-based classifier as implemented by the MaChAmp toolkit 0.2 (van der Goot et al., 2021).

[5] https://github.com/eginhard/word-level-language-id/

[6] https://scikit-learn.org/

[7] over only the classes of interests; as mentioned in Section 2.1, we only focus on 3/8 labels of the LinCE data

| Model | Dev | Test |
|---|---|---|
| Word uni-grams | 91.32% | 88.25% |
| Word n-grams | 50.50% | 49.04% |
| Character n-grams | 92.19% | 88.95% |
| Viterbi | 95.99% | 92.23% |
| LDA | 64.40% | 62.84% |
| Support Vector Machine | 91.39% | 88.74% |
| Logistic regression | 90.34% | 87.80% |
| **Ensemble Model** | | |
| All models | 92.15% | 88.99% |
| Models 3, 4 and 6 | 93.72% | 90.27% |
| Models 1, 3, 4, 5 and 6 | 92.96% | 89.64% |
| Oracle* | 98.47% | - |
| **Supervised** | | |
| MaChAmp | 99.24% | 98.43% |

Table 1: Models evaluated using weighted F1 score on validation and test data. * Made use of gold labels

We use multilingual BERT and all default settings. Results in Table 1 show that there is still a performance gap between the semi-supervised approaches and this state-of-the-art supervised model. When comparing common confusions of our best semi-supervised model (Viterbi) to the output of MaChAmp, we found that there was more confusion in the Viterbi model about `other`, where 213 words were classified as `lang1` and 60 as `lang2` instead, compared to just 3 and 1 in MaChAmp. Full confusion matrices can be found in the appendix.

The majority voting ensembling models do not lead to improved performance. However, the oracle ensemble, which always picks the correct label if it is available from one of the models, shows that there is potential in improving the selection method for ensembling.

## 5 Discussion

When inspecting the performances of the models per class (see also Table 2 in the appendix), we found that, for the development dataset, all models have a better F1 score for English than for Spanish and, for the test dataset, the other way around. This might be due to a discrepancy between the label distribution of the two datasets and is a significant aspect to be investigated in future work.

Regarding the LDA model, its low performance can be explained by the results of (Zhang et al., 2014), which show that for the task of language filtering, the performance of LDA decreases when the dominating language decreases under 70% of the whole text. This is also the case in our experiments, where the test data had a 54% English and 46% Spanish ratio. Furthermore, the amount of evidence per sample is rather low compared to the normal use of LDA (it is commonly used on the document level).

For character $n$-grams, we observed that the more we increased the value of $n$, the better results we got, up until $n = 6$. The higher order $n$-grams performed better with around 12% difference in validation weighted F1 score, as we can capture groups of letters that are representative for a language, e.g. 'tion' in English and 'cion' in Spanish. This model achieves good results also because it addresses the problem of misspelled words. For word $n$-grams, using tri-grams resulted in worse predictions than using bi-grams with around 11% difference in validation weighted F1 score.

For LDA, SVM and Logistic Regression models we tried to vectorize data with `CountVectorizer` from Scikit Learn, which gives the term-frequency for each $n$-gram in a word. However, `TfidfVectorizer` performed approximately 1% better in LDA and Logistic Regression and 4% for SVM in validation data. This was then the preferred vectorizer in all models, as it helps decreasing the impact of very frequent character $n$-grams that are not expressing much value and gives more importance to less frequent character $n$-grams.

The fact that the oracle model has a 3% higher weighted F1 score than the best model (in validation data), suggests that there is room for improvement for the ensemble model with other methods than majority voting. Improvements on the single models could be achieved by using bigger monolingual datasets of the same size or selecting a corpus that is more similar to the test set (social media-like posts), which is not as easy to query as Wikipedia articles. The overall performance of the models can also be slightly improved by a more complex method for the `other` class (the existing rule-based method scored an F1 of 96.76, see Table 2 in the appendix).

The training efficiency of the Viterbi model and the supervised model were measured in a Windows Sub-system for Linux environment on an i7-7700K processor with 16GB ram. We ran the MaChAmp model in this environment and it completed in 53,990 seconds. In comparison, the Viterbi training

completed in 1,805 seconds, which is an improvement of almost 30 times faster than the MaChAmp model.

## 6 Conclusion

In this study we evaluated different types of models, namely word uni-grams, word $n$-grams, character $n$-grams, Viterbi Decoding, Latent Dirichlet Allocation, Support Vector Machine and Logistic Regression, for the task of semi-supervised language identification in English-Spanish code-switched data. We found that most of the models achieved promising results, however, the Viterbi model performed the best with a weighted F1 score of 95.76% on validation data and 92.23% on test data (**RQ1**). Using this model, one can potentially train CS-detection for many more language pairs as previously possible. Furthermore, since the majority voting did not lead to improvements, we experimented with an Oracle model, which showed that by combining results form our models, the best score we could achieve is 98.47% on validation data. Even though the results were good, our models still underperformed compared to the supervised MaChAmp model, that scored 99.24% weighted F1 score on validation data and 98.43% on test data (**RQ2**). There is also a clear take away that, by using simpler, faster approaches like ours and when top performance is not crucial, one can avoid the extensive process of human-annotation and long training time that are needed by finetuning these large transformer models on supervised data.

## References


Gustavo Aguilar, Sudipta Kar, and Thamar Solorio. 2020. LinCE: A centralized benchmark for linguistic code-switching evaluation. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 1803–1813, Marseille, France. European Language Resources Association.

Somnath Banerjee, Kunal Chakma, Sudip Kumar Naskar, Amitava Das, Paolo Rosso, Sivaji Bandyopadhyay, and Monojit Choudhury. 2016. Overview of the mixed script information retrieval (MSIR) at FIRE-2016. In *Forum for Information Retrieval Evaluation*, pages 39–49. Springer.

Anab Maulana Barik, Rahmad Mahendra, and Mirna Adriani. 2019. Normalization of Indonesian-English code-mixed Twitter data. In *Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019)*, pages 417–424, Hong Kong, China. Association for Computational Linguistics.

Irshad Bhat, Riyaz A. Bhat, Manish Shrivastava, and Dipti Sharma. 2018. Universal Dependency parsing for Hindi-English code-switching. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 987–998, New Orleans, Louisiana. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

G. D. Forney. 1973. The Viterbi algorithm. In *The viterbi algorithm*, 3, pages 268–278.

Simran Khanuja, Sandipan Dandapat, Anirudh Srinivasan, Sunayana Sitaram, and Monojit Choudhury. 2020. GLUECoS : An evaluation benchmark for code-switched nlp. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, page 3575–3585.

Giovanni Molina, Fahad AlGhamdi, Mahmoud Ghoneim, Abdelati Hawwari, Nicolas Rey-Villamizar, Mona Diab, and Thamar Solorio. 2016. Overview for the second shared task on language identification in code-switched data. In *Proceedings of the Second Workshop on Computational Approaches to Code Switching*, pages 40–49, Austin, Texas. Association for Computational Linguistics.

Sebastin Santy, Anirudh Srinivasan, and Monojit Choudhury. 2021. BERTologiCoMix: How does code-mixing interact with multilingual BERT? In *Proceedings of the Second Workshop on Domain Adaptation for NLP*, pages 111–121, Kyiv, Ukraine. Association for Computational Linguistics.

Hildegard L. C. Tristram. 1999. *How Celtic is Standard English?* Nauka.

Rob van der Goot, Ahmet Üstün, Alan Ramponi, Ibrahim Sharaf, and Barbara Plank. 2021. Massive choice, ample tasks (MaChAmp): A toolkit for multi-task learning in NLP. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 176–197, Online. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Zeynep Yirmibeşoğlu and Gülşen Eryiğit. 2018. Detecting code-switching between Turkish-English language pair. In *Proceedings of the 2018 EMNLP Workshop W-NUT: The 4th Workshop on Noisy User-generated Text*, pages 110–115, Brussels, Belgium. Association for Computational Linguistics.

Wei Zhang, Robert AJ Clark, and Yongyuan Wang. 2014. Unsupervised language filtering using the latent dirichlet allocation. In *Fifteenth Annual Conference of the International Speech Communication Association*.

# A Appendix

| | Validation F1 score | | | Test F1 score | | |
|---|---|---|---|---|---|---|
| **Model** | **English** | **Spanish** | **Other** | **English** | **Spanish** | **Other** |
| 1. Word uni-grams | 91.05% | 88.77% | 96.76% | 87.42% | 89.94% | 95.84% |
| 2. Word n-grams | 45.41% | 31.98% | 96.76% | 40.61% | 35.73% | 95.84% |
| 3. Character n-grams | 91.84% | 90.19% | 96.76% | 88.56% | 90.68% | 95.84% |
| 4. Viterbi | 96.09% | 95.48% | 96.76% | 93.13% | 94.71% | 95.84% |
| 5. Latent Dirichlet Allocation | 59.37% | 53.09% | 96.76% | 53.15% | 57.74% | 95.84% |
| 6. Support Vector Machine | 90.84% | 89.21% | 96.76% | 88.07% | 90.55% | 95.84% |
| 7. Logistic regression | 89.65% | 87.74% | 96.76% | 86.74% | 89.41% | 95.84% |
| **Ensemble Model** | **English** | **Spanish** | **Other** | **English** | **Spanish** | **Other** |
| All models | 91.92% | 90.00% | 96.76% | 88.36% | 90.91% | 95.84% |
| Models 3, 4 and 6 | 93.55% | 92.31% | 96.76% | 90.35% | 92.34% | 95.84% |
| Models 1, 3, 4, 5 and 6 | 92.79% | 91.17% | 96.76% | 89.31% | 91.69% | 95.84% |
| Oracle* | 98.96% | 98.81% | 96.76% | - | - | - |
| **Supervised** | | | | | | |
| MaChAmp | 99.14% | 99.08% | 99.78% | 98.42% | 99.00% | 99.84% |

Table 2: Models evaluated using F1 score per class for validation and test data

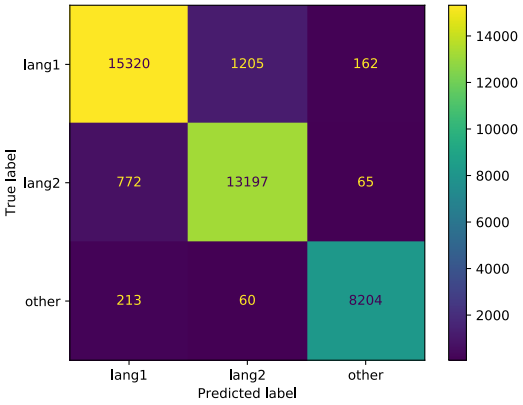


Figure 1: Confusion matrix of Viterbi predictions on test data

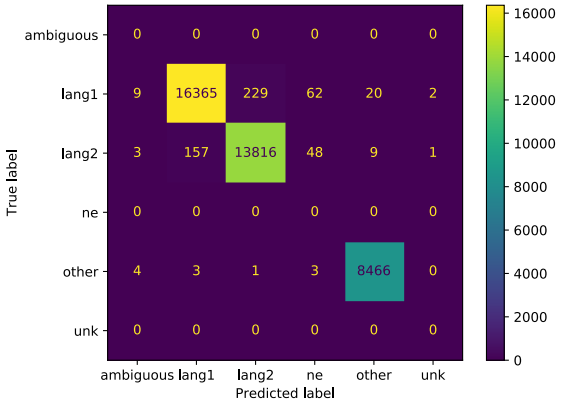


Figure 2: Confusion matrix of MaChAmp predictions on test data

Figure 1 and 2 show the confusion matrices of the Viterbi and MaChAmp model. It can be noted that the confusion matrix for MaChAmp model has more than the three labels we used, because it was trained on part of the original training set presented in Section 2.1. This set contained 8 classes, and, thus, occasionally, the model mistakenly predicted some of these classes. It can be seen that there was more confusion in Viterbi model about `other`, where 213 words were classified as `lang1` and 60 as `lang2` instead, compared to just 3 and 1 in MaChAmp, which also had 7 other misclassifications.