Language Identification of Tweets

Avia Efrat (ID. 300928538)

Final project report for Machine Learning Applications, Tel Aviv University, 2019

1 Introduction

1.1 The Problem

The dataset consists of 80,000 tweets, of which there are 10,000 tweets in each of the following eight languages: Dutch, English, French, Indonesian, Italian, Portuguese, Spanish and Tagalog. Each entry in the dataset (tweet) contains only the tweet's text and its unique id. The goal is to use this data to develop a model which, given a tweet, predicts the language that this tweet is written in.

1.2 What is "Language Identification" Exactly?

No, I'm not going address here the grand philosophical question of what constitutes a language. But I think this question needs to be addressed in the context of our problem before diving into research and experimentation.

1.2.1 Relying solely on our dataset vs. using external resources

Ideally, in a language identification task, one would expect that for each language there exists a dedicated dictionary or a phrase-list that serves as a reference. Such dictionaries would have been built beforehand, probably by relying on existing trustworthy dictionaries, or by scraping vast corpora. Each of these dictionaries would serve as a reference as to the language of each word. It still wouldn't be enough to completely solve our task though, as tweets tend to be more free-formed than regular text (more on that in section 2.2.1).

However, in our task we are limited to using only our dataset. Therefore, the labeling of our tweets is the only information we can rely on when trying to predict a tweet's language. Specifically, if a tweet containing a word w is labeled as language l, then we are to assume that w is part of language l's vocabulary. Unfortunately, from a quick inspection, it seems that our data is not mistake-free, and its labeling can sometimes be seen as rather arbitrary.

1.2.2 Obvious mistakes

Consider tweet 42376[1], labeled as "Italian":

spooky indicator

[1] I identify tweets in a different manner then the long "tweet_id" field. It is according to the index assigned to the tweet in either "raw_X_train.csv" or "raw_X_test.csv" in the "data" directory, which is created after loading the initial dataset. however, I will always present a tweet's contents when addressing it in this way, so there is no need to open any files directly.

Without any prior knowledge in English or Italian (meaning relying blindly on the dataset), there is no other way but viewing this labeling as perfectly legitimate. But since I possess knowledge external to the dataset, this labeling seems to be incorrect. I know for a fact that both of these are English words, and "spooky" doesn't strike my as Italian. And indeed, after verifying with an Italian dictionary, both these words are not in Italian (there is an "indicatore" in Italian, meaning "indicator"). Furthermore, when searching this tweet online[2], you encounter a twitter profile[3] whose tweets are solely in English, and its "bio" section contains a phrase in Portuguese. Furthermore, upon inspecting this profile more thoroughly, I doubt that this profile is even organic. Starting from 06/02/2018 and ending at 27/04/2018, this profile almost solely tweeted, several times a day, a variation of the same sentence. The other posts were a few retweets. Furthermore, between 02/08/2017 and 11/08/2017, this user has twitted 444 times (44+ tweets a day), 442 of them are of the form "spooky <noun>" (e.g. "spooky basis"). The other 2 are just "http://www.dubme.tv" (an inactive domain as of today).

Finding such tweets requires simply creating a vocabulary for each language in the dataset, searching for tweets that all of their words belong to two different dictionaries, and then manually inspecting for errors. That is not the only example of such a mistake. I choose to present this one as it involves words in English, which is the lingua franca of our field, and as such is easy to demonstrate errors with.

1.2.3 Tweets with no real language content

Consider the following tweets:

> 1498 (English): "@RoaringNurse @1daywithoutus @Unite4Europe @unisontweets @The3Million @AlfonsLuna @elperiodico @ReporterVic... https://t.co/m3kr36GF95"
>
> 13951 (Spanish): "RT @GaiaGaiaVentu: @imaneBeF96 @Benji_Mascolo @fedefederossi @Infedeseyes @soulofbenjamin @abrazandotini @La_Martyy @anchorpenc @_Jessica_M..."
>
> 20165 (French): "@diabate33 @CIES_Football @sudouest @girondins @girondins33 @SudOuest_Sport @AdrienMonk64 @Fcgb765 @diego_rolan... https://t.co/4ygwDJBXAO"
>
> 35773 (Indonesian): "RT namjoohyukpic: https://t.co/X8sxUwU0ZW"
>
> 37135 (Indonesian): "RT @ImSenyoritaVice: <lots of SPARKLE emojis>[4]"
>
> 46310 (Italian): "@danielleediz KSJSJJSJDJSJ"[5]
>
> 59178 (Dutch) "RT @Ass2Day: @DeucesWild00 @EvePornSaint @GGgirls0 @lovelygirls200 @wtgbythesea @MsCatherine00 @NymphoThought @Throatmydick @fickfotzchen88..."
>
> 65577 (Portuguese) "RT @kimcarsom: @maryrock180457 @andinho4020111 @Mellyssa57 @seudevan @karlaRejane28 @ednilson02 @edy769 @SergioTokita @MaryT79718239 @PETER..."

There are about 90 of these types of tweets, more than 0.1% of the data. Above, I presented one in each language, except Indonesian where I also Included a tweet mainly composed of "sparkle"

[2] https://twitter.com/ANewBrigade/status/836447435253755904. Although I presented my findings regarding this tweet and its profile, Generally I find the need of online validation of the data out of scope for this project. I fully understand that data preprocessing and validation is an important part of data science, and I even find it quite enjoyable as it requires creative insights. However, I think that adding a web interface is a bit too much, especially considering I'm doing this project alone =)

[3] https://twitter.com/ANewBrigade.

[4] Adding emojis to LaTeX ended up being not so trivial.

[5] In an improbable coincidence, the string "jjsj" appears in 12 different tweets in Italian, and in no other languages.

emojis, and Tagalog where I didn't find these kind of tweets. What do all of these tweets contain? A combination of nonsense, twitter handles and links. Handles are usually a variation on proper nouns (names of persons, organizations, etc.), sometimes a rather convoluted one.

Handles Generally, I don't believe that handles should count as part of a language, for several reasons. First, even if some handles are easily tokenized into proper names, such as "@Lorna_Sophia" → "Lorna Sophia" and even "@SergioTokita" → "Sergio Tokita", that does not mean they should consist as part of their labeled language. Britney, for example, is not an English word according to the Oxford English Dictionary. The same reasoning applies even more clearly when considering handles like "@MaryT79718239" or "@fickfotzchen88". Second, even when considering easily tokenized non proper noun handles, since English can be considered the Internet's lingua franca, handles tend to be in English even when the user's native language is not English (or at least when the tweet is not labeled as English). For example, considering the list of tweets that started this section, we get that "lovely", "girls", "deuces" and "wild" are words in Dutch[6]. Third, tagging a user with a handle does not necessarily mean that the tweet itself is in a specific language. One could still argue, that even if reasons 1 through 3 from the previous paragraph are sound, handles and their naming patterns could nonetheless help us in determining a tweet's language, even if the handles themselves are not part of a language. I address this suggestion in section 2.2.3.

As a side note, many tweets of this pattern (a long list of handles) seem to have common and repeating handles. Given their lack of any real content, analyzing these tweets using social network algorithms could provide interesting results, especially given the (not-so) recent rise of large scale bot farms. But that is a matter for a different kind of project...

URLs and Hashtags Furthermore, all of the above discussion regarding handles could also be applied to URLs and hashtags. For example, one could suggest using the country domain of a URL to deduce a tweet's language, even though there is no guarantee for such a match. Moreover, one could argue that posting a URL in a tweet is actually posting the URL's content. However, I doubt that this view is consistent with relying only on the dataset itself, so I'm not going to further explore this issue. Hashtags, however, may be a different matter, as they tend to be more coupled with the labeled language. Although they are often shortened version of their 'real' meaning, many times they are composed of what could be generally considered as valid words. As such, they can provide more 'immediate' information of a tweet's language than URLs. And even though their content too tends to lean more towards English, I decided to give hashtags a special treatment, described in throughout section 2.1.

1.2.4 The curious case of Zara Larsson

I'll warp the discussion of tweets with questionable language content with the most blatant example I found. Zara Larsson, in case you wondered, is a rather successful Swedish singer born in 1997. As a small scale celebrity, the fact that the string "Zara Larsson" appears in the dataset is not noteworthy. What is perhaps more notable, is the fact that it appears in more than 0.2% of the tweets. But still, it could be attributed to uneven sampling when creating the dataset. However, looking into it, it turns out that 55 of them are labeled "Spanish", and all but one are of the form "#VideoLove Zara Larsson <different URL>". The other 117 are all labeled "Portuguese", and all

[6] Actually, tweet 59178 contains a few more new "Dutch" words, but I didn't feel the need to restate them.

but one are of the form "googuns_lulz: #VideoLove Zara Larsson <different URL>". At the risk of repeating myself, relying on tweets with such content does not seem to fall into accordance with our intuitive understanding of what is language identification.

1.2.5 So, what is "language identification" in the context of this project?

Taking the "rely on the dataset alone" requirement too literally, all the discussions in section 1.2 become moot. After all, if we can only rely on the labels, then the notion of a language l is just the set of all the text labeled as l. Since such an approach clearly leads to a simple and degenerate solution, I'll allow myself to take into consideration insights that could not have arisen from looking only at the dataset, but that reflect our basic intuitions regarding language.

The main benefit we gain from section 1.2 is not a conclusive definition of what, and what not, constitutes a language. Rather, it focuses our attention on the fact that whatever we decide to include within a language, we should be aware of the implicit assumptions behind our decisions, and preferably be able to supply some reasoning supporting them. During this project I will do my best to adhere to this principle.

1.3 Motivation Behind Choosing This Project

My main interest in ML/AI is natural language processing, so naturally I leaned towards a project of this kind. In addition, and perhaps even a more determining factor in my decision, is that I wanted a project that will force me to dedicate a significant time to text preprocessing. This aspect of the data science pipeline is far less glorified than the algorithmic part, and I thought it was a good opportunity for me to tackle it head-on, even if not on a very large scale, or as thoroughly as a production-grade process.

1.4 Code

The project's repository can be found at https://github.com/aviyoop/aml2019, with a README file detailing the project's structure and basic running examples.

2 Solution

2.0.1 General settings

The dataset is split with a 0.2 test size. The split was stratified. A constant random seed was used to enable comparability between different runs. All runs were performed on my personal laptop, using Intel Core i7-5500U CPU @ 2.40GHz×4.

2.1 A (very) Quick and (very) Naïve Approach

As a first step, I'm interested to see the performance of a basic solution. From there, hopefully I'll gain some insights on how to proceed. This approach will involve no preprocessing, and a simple word tokenization method. However, since tweets are not your traditional text (recall sections 1.2.2 and 1.2.3), even my naïve approach will include some minimal considerations of their special features.

2.1.1 Tokenization method

Taking inspiration from NLTK's TweetTokenizer[7], but changing a few things myself, I will tokenize each tweet in the following manner:

Each token is either a "word" or a "hashtag". Following the discussion in section 1.2.3, I decided to try to treat hashtags as language elements, but to make them distinct from 'regular' words.

"words" are defined as sequences of Latin letters (including accents like ä or ñ, and common digraphs like Dutch's "IJ" letter), apostrophes and hyphens. a "word" cannot end with an a apostrophe or an hyphen. In addition, words are not case sensitive, as none of dataset's languages have a strong dependence on capitalization[8]. Notice that this simple approach transforms handles such as "@Princess_Caroline" to the words "princess" and "caroline". I address the handle issue on later sections. Here I preferred to keep the tokenization method as simple as I could.

As for "hashtags", surprisingly there is no official documentation defining the structure of a valid hashtag. My approximating solution was defining a "hashtag" according to the information I could extract from the JavaScript source code of Twitter's tweet parsing package[9].

2.1.2 Feature creation

The tweets of each language are tokenized, and based the on the extracted tokens, two vocabularies are created for each language. One vocabulary includes all the "words" of a language, and the other contains all the "hashtags". Following, 16 features are created, 8 for "words" (one for each language) and 8 for "hashtags". Each "word" feature represents the proportion of a language's words out of the total number of words in the tweet. Similarly, 8 "hashtag" features are created. It is important to note that vocabularies are created only according to the training data, but the feature creation is applied to the test data as well as to the training data.

Example: Consider this simple dataset, containing only two languages, and no hashtags:

Train: "I smell gas" (English) ; "Huelo a gas" (Spanish)
Test: "I smell food" (English) ; "Huelo la comida" (Spanish)

The vocabularies created are {"i", "smell", "gas"} for English and {"huelo", "a", "gas"} for Spanish. The resulting feature values are:

	prop. of English words	prop. of Spanish words
"I smell gas"	1.0	0.3333
"Huelo a gas"	0.3333	1.0
"I smell food"	0.6667	0.0
"Huelo la comida"	0.0	0.3333

Table 1: Simple word-based features

2.1.3 Algorithm

Since we are dealing with structured data, the obvious go-to algorithm will be a boosting one. Since none of the features is categorical, CatBoost is a lesser option. XGBoost and LightGBM

[7] http://www.nltk.org/api/nltk.tokenize.html#nltk.tokenize.casual.TweetTokenizer
[8] As opposed, for example, to German, whose nouns are required to be capitalized
[9] https://github.com/twitter/twitter-text/blob/master/js/pkg/twitter-text-2.0.5.js

are expected to have similar performance, but LightGBM is a lot faster[10]. Given this project revolves much more around experimentation than around a possibility of a negligible improvement in accuracy, I decided to focus the algorithmic effort on LGBM alone, and leave more time to preprocessing and feature considerations.

2.1.4 Training

After creating the features from section 2.1.2, the next step was hyperparameter tuning. Since there are only 16 features, speed is not expected to be a issue, leaving me with accuracy-related hyperparameters. My grid included two hyperparameters, each with three possible values: the default value, a value above it, and a value below it. Specifically, I used *learning_rate* values of [0.05, 0.1, 0.2], and *num_leaves* values of [15, 31, 63]. I used a LGBM's native cross validation API[11], a 5-fold validation, and the *multiclass* objective (softmax loss function)[12]. I addition, I utilized LGBM's *early_stopping_rounds* feature, which stops cross validation if the score on the validation set does not improve for n consecutive rounds. I set $n = 5$, which is half of the default value, since my training set is rather small.

The differences between the 9 different hyperparameter combination were quite minuscule, with less then 0.005 difference between the highest and the lowest loss. The best hyperparameters were $learning\_rate = 0.05$ and $num\_leaves = 15$, with an optimal number of boosting rounds of 110. The total training time, including the hyperparameter tuning, took about 2 minutes.

2.1.5 Results and discussion

Train Accuracy	Test Accuracy
99.122%	78.167%

Table 2: Performance of words and hashtags tokenization

Above all else, the first thing that comes to mind is that even this simple model achieves about 78% accuracy, far more than the expected 12.5% accuracy of random predication. That being said, it is possible that there is still room for improvement, as the large difference between the train and the test accuracy could suggest a problem of overfitting. Some of that difference could be attributed to the inherent advantage of the training data being sole source of the vocabularies, resulting in every token in the training data to contribute to the information encoded in the features. That being said, a possible overfitting indicator is the $num\_leaves = 15$ choice in the cross validation, as a large value of this parameter is linked to overfitting[13]. However, before re-tuning hyperparameters, it is still worth investigating other contributing factors to this difference, as all the cross validation results were very similar.

After examining the vocabularies, it seems that the hashtag vocabularies are much more sparse than the "word" ones, with many hashtags only appearing once, or very similar hashtags with varying forms of shortening. Given the rather short time that the training took, and the questionable status of hashtags as language components to begin with (see section 1.2.3), it could be

[10] see https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db for CatBoost's performance when there are no categorical variables, and on speed comparison between XGBoost and LightGBM.
[11] https://lightgbm.readthedocs.io/en/latest/Python-API.html#lightgbm.cv
[12] https://lightgbm.readthedocs.io/en/latest/Parameters.html#objective
[13] https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html#deal-with-over-fitting

the case that hashtags actually contribute more noise than actual information. Applying the same validation and training process, but this time using only 'regular' words, (no hashtags) we get

Train Accuracy	Test Accuracy
99.076%	86.862%

Table 3: Performance of words tokenization, without hashtags

Just ignoring the hashtags, we got 8.5% improvement on the test set, with less than 0.5% drop in training accuracy. This indicates that it would probably be beneficial to pay further attention to the unique properties of tweets (as opposed to more 'regular' text) when trying to build our language identification model. This is the main goal of section 2.2.

But before focusing on analyzing tweet structure, you may wonder why I didn't try to improve the word-based features. Such improvements could be TF-IDF related, such as creating a unique feature for each word, addressing the relative frequency of a word within its language, and adding an "unknown" token. As I see it, the goal of this section was to perform a quick and dirty first attempt, and gain initial insight. Creating TF-IDF-like features would have taken a substantial amount of time to train (especially considering 5-fold cross validation), and that was not my goal here. However, in section 2.3 I will take a more granular approach to feature creation, aiming at a final push to improve accuracy.

2.2 The Uniqueness of Tweets

As we saw at the end of the last section, ignoring the hashtags proved to be very beneficial to our model. In this section, we will show several issues common to tweet text, and discuss several approaches to dealing with them.

2.2.1 Some notable properties of tweets

Below is a non exhaustive list of properties that are prevalent in tweets, but are much less common in more traditional text corpora. Discussing some of them, my examples will be in English, but it is very reasonable to think that these patterns apply to all other languages as well.

Twitter-unique entities: Hashtags, handles and URLs were already mentioned in section 1.2.3. An additional entity, and a very common one, is "retweets", which are comprised of the letters "RT", followed by a possible handle and a colon.

Slang and abbreviations: slang such as "finna[14]", and common abbreviations such as "omg" and "lmao".

General onomatopoeic utterances: Could be view as part of the previous item. "oomf" and "pfft" are notable examples. Others include "haha" and "jaja" (the Spanish/Portuguese equivalent of "haha"). However, these tend to vary considerably in length, which leads to the next group of examples:

[14] https://www.urbandictionary.com/define.php?term=finna

Character and syllable repetition: Some written utterances, such as "haha", take many different writing forms, most of them just repetitions of the syllable "ha". Even the exact phrase "hahahahahahahahaha" appears 22 times in the data, with the total amount of the "haha" variants in the thousands. In addition, there are many other character repetitions, usually used as emphasizers. One of the countless examples is "Daaaaaaaaammmmmnnn" from tweet 220.

Typos and spelling errors: Given the causal nature and "immediateness" of tweets, typos, which are unintentional writing errors, frequently occur. For similar reasons, spelling errors are a lot more common than in traditional texts. Note that some of these typos and errors, as a result of the chaotic nature of Internet orthography, can sometimes "evolve" to a new word by themselves, either with a different meaning or as an alternative spelling, once again intersecting with the "Slang and abbreviations" section.

Emojis and ASCII art: Long before Twitter, ASCII art was used in online texts, either as emoticons (":]" etc.), l33t, or just as general art. These are still utilized today in tweets, even ASCII art (see tweet 37023). But more notably, emoticons' modern relative, the emojis, are widely used within tweets. Emojis are a very interesting test case in my opinion, as scrutinized research of emoji types and frequencies can suggest differing relations between cultures, languages, and emotional expressivity. However, as emojis are not natural components of language, that the hashtags experimentation proved to provide mostly noise, and that generally I find adequate emoji/emoticon analysis to require a substantial effort, I decided to ignore emojis altogether in my model.

2.2.2 Tackling these Twitter-specific properties

Some items of the aforementioned list, such as Twitter-unique entities and some of the utterances and slang, could be reasonably addressed using word-based tokenization. Others, especially, character and syllable repetition, are probably an insurmountable challenge given my available computational power. Those will require a different approach, or at least an addend on top of the word tokenization. This addend will be various ways of text preprocessing, and will be presented in the following paragraphs. The former, the different approach, will comprise section 2.3.

2.2.3 Text preprocessing

Observing the data again after going over section 2.2.1, I devised the following preprocessing procedures, in order to tackle at least some of the issues. This list is by no means a complete one, and should be viewed as an initial attempt to improve the model, guided mostly by observing the data and searching for common patterns that might hinder language identification ability.

Removing retweets Retweets are very common in the dataset, occurring in a about half of the tweets. Since they only contain the sequence "RT <handle> : ", I perceive them as containing mostly irrelevant information.

Removing handles Although some of them are already removed as part of retweets removal, they are still very common, with many tweets comprising of mostly handles. Following the discussion in sections 1.2.3 and 1.2.5, I wanted to see the effect of ignoring them.

Removing URLs Same with handles, but even more clearly, URLs, which in the dataset are of the form "https://t.co/<random characters>", are perhaps the most obvious element to entirely consist of noise, at least when dealing with word-based tokenization. And given that they are present in almost half of the tweets, ignoring them should contribute to a model's performance.

Normalizing repetitions Unlike the previous preprocessing methods, this one deals with elements that are clearly perceived as language components. The abundance of repetition variance in words is a clear candidate to hinder performance. The normalization was applied in the following manner: first, I reduced every sequence of 2 or more consecutive identical letters, into 2. *aaaah* $\rightarrow$ *aah*. Some languages include rare words with more then 2 consecutive identical letters, but I thought reducing them was worth the price of eliminating all other nonsensical sequences. Second, I reduced every sequence of 2 or more consecutive non-letters into 1. !!! $\rightarrow$!. Last, I reduced every sequence of 2 or more consecutive identical character 2grams, into 2. *jajajaja* $\rightarrow$ *jaja*. Even with these steps, there were still words that were not handled in a satisfactory manner. As an example, consider the oh-so-conventional token *Jdjsjskdnshs* from tweet 42789. As a side note, the entire tweet reads "Jdjsjskdnshs https://t.co/JpSsqIUpOD", which serves a reminder to our discussion of the definition of language identification in the context of our problem.

2.2.4 Training

The dataset was preprocessed with the methods from the previous section, in a sequential manner. i.e, first the retweets were removed, than the URLs, and so forth. Between each two steps, I trained a model, tuning hyperparameters in the same manner as in section 2.1.4. Since there were $2^4 = 16$ different combinations of possible preprocessing steps, I chose to present only a partial subset, while the other combinations could be examined in future research. In addition, even that hashtags didn't prove helpful before, I decided to give them one last change, thinking that maybe the preprocessing could have some effect on their utility.

2.2.5 Results and discussion

	Train Acc.	Test Acc.
no preprocessing	99.076%	86.862%
+remove retweets	98.856%	86.974%
+remove handles	98.160%	87.055%
+remove URLs	97.836%	87.270%
+reduce repetition	97.819%	87.283%

	Train Acc.	Test Acc.
no preprocessing	99.122%	78.167%
+remove retweets	98.909%	78.855%
+remove handles	98.397%	83.197%
+remove URLs	98.160%	85.037%
+reduce repetition	98.080%	85.268%

Table 4: Effects of preprocessing, without hashtags (left) and with hashtags (right)

Although utilizing preprocessing indeed steadily improved the test accuracy (as well as the overfitting) for every additional method applied, the effect was minuscule on the non-hashtag features - a little more the 0.5% overall improvement. Applying preprocessing on the hashtag features proved a lot more beneficial. Especially the 4.3% improvement when removing the handles. However, since using hashtag features seems as a lesser choice, I chose not to discuss here possible explanations for these improvements. A few things that could be beneficial to inquire are[15] the

[15] These were extracted from confusion matrices. I didn't include them as I didn't view this discussion important enough considering the 12-page limit. Confusion matrices could be recreated using the *evaluation.py* module.

18% accuracy improvement in Indonesian when removing handles, and the 13% improvement in French when removing URLs.

Before proceeding, I'll present a short discussion of our best obtained results yet - using only word features, with maximal preprocessing. As can be seen in Figure 1, the model has a small bias towards predicting English as a tweet's language. But since English is "the language of the Internet", and that in fact many dataset entries contain English words in addition to words in their labeled language, this seems as a reasonable result.

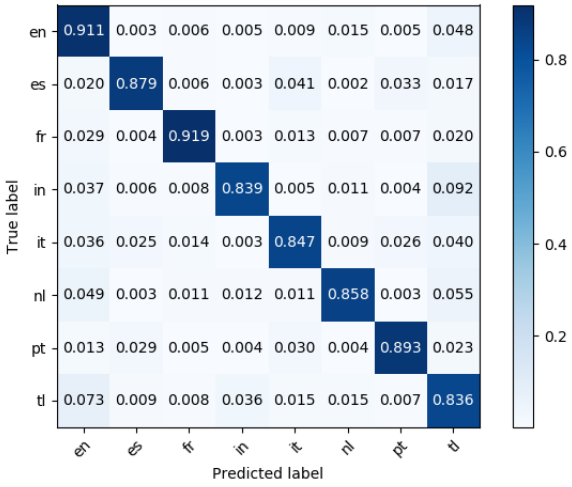

Figure 1: Confusion matrix using 'word' features and full text preprocessing

Perhaps a more surprising result is Tagalog being a close second in that regard. A possible explanation is the American rule of the Philippines until 1946, and the strong U.S influence that still exists there today. Indeed, Figure 1 shows that 54% of the model's failures to recognize English were "Tagalog" predictions, and in the opposite direction, that number is 45%. Furthermore, notice that Dutch has a similar relation, although to a lesser extent, with both English and Tagalog. The Netherlands's very high English proficiency rate could further explain the model's tendency towards predicting Tagalog.

Other noticeable result is the relatively high confusion between Portuguese, Spanish, and Italian, which was to be expected. Far less expected is the model's impressive accuracy when predicting French. I expected French to be part of the aforementioned confusion group as well. Perhaps France's reputation of language elitism, even towards English, is not a complete myth... Finally, the most noticeable outlier was predicting "Tagalog" when in fact it was Indonesian. Indonesian was not confused with English as much as Tagalog (and Dutch). To my understanding, Tagalog is indeed the closest to Indonesian compared to the other languages, but they are not a close as their relative confusion indicates.

Provided the above preprocessing introduced only a marginal improvement to the model, two main future paths could be considered. The first is introducing more methods of preprocessing. One example could be further normalizing all words that include more than 2 *h*s and 2 *a*s (and only those letters), into a standardized *haha* word, and similarly with *jaja*. A more ambitious method could be trying to remove typos via word proximity and frequency statistics, but given we cannot rely on any external source, this could prove very complex, and even introduce mistakes not currently present in the dataset. The second path is considering that given the more casual nature of tweets, maybe we are nearing the limit of what word-based tokenization has to offer, and that adding preprocessing could only provide us with additional marginal improvements. In the next and final section I'll discuss a finer-grained tokenization method, and compare its results to those of word-based tokenization.

2.3 Character-Based Tokenization

2.3.1 Motivation

To better deal with the unorthodox nature of tweet text, I suggest a more flexible tokenization approach - character based tokenization. Aside from different words, different languages also exhibit different character frequencies, and this information could be utilized by counting characters, or "1grams". Moreover, some languages contain unique letters (mostly accents, such as *ä*), further contributing to the distinction between languages. More complex language properties could also be utilized, as languages also differ on ngram frequencies. Other than well-known English examples as "th", "ing", and "ough", additional notable ones are "sch" in German or "ij" in Dutch.

Furthermore, Character-based tokenization can better handle some of the properties mentioned in section 2.2.1, such as typos, spelling errors, alternative spellings and character repetitions. With regards to the first three, word tokenization will treat word pairs like "threatening" and "threatning" or "rationalization" and "rationalisation" as two completely different words, while character-ngram tokenization will reflect the fact that these words are actually very similar. Moreover, spelling variations also tend to be in accordance with a language's ngram distributions. Character repetitions are a bit trickier, but if repetitions are reduced and long ngrams are also considered, repetitions could be analyzed as a variant on regular ngram frequencies.

2.3.2 Feature creation

1grams through 5grams were extracted from the training set. For 1grams, the allowed characters were all the letters, and some language specific symbols such as "¿" and "№"[16]. For longer ngrams, some additional preprocessing was applied. First, all characters except letters, intra-word hyphens, and letter-adjacent apostrophes were replaced with spaces (" "). Then all space sequences longer than 1 were reduced to a single space, and finally a space was added to the beginning and end of each tweet if the tweet didn't already start with a space. This was in order to be able to capture not just an ngram's frequency, but whether that ngram was in the beginning of a word (starts with a space), in the middle of a word (contains no spaces) or at the end of a word (ends with a space). Longer ngrams were not extracted, both from wanting to be at least partially able to compare the performance of word-based tokenization to our current one, and from time considerations, as the training is already expected to be much longer. Since using all extracted ngrams was infeasible, the top 500 of each length were selected using the chi squared test in order to find the ngrams most significant to language distinction. Finally, for each selected ngram, I calculated its relative frequency (out of all ngrams with the same length in the tweet).

2.3.3 Training

This time, as I'm aiming for maximal accuracy and not just experimenting, I'll fix the learning rate to 0.05, and tune just the *num_leaves* parameter, as in section 2.1.4. Preprocessing will be done in the same manner as in section 2.2.4. Total training time was about 12 hours, with cross validation taking about 85% of the time.

[16] For the full list, see the *constants.py* module.

2.3.4 Results and discussion

The 5 to 6 percentage points improvement strengthens my hypothesis that character-based tokenization is better suited for handling tweets. As in word tokenization, applying more preprocessing gradually improved performance, but here preprocessing proved much more valuable. While preprocessing improved the performance of word tokenization only by 0.48%, character tokenization was improved by 2.31%, even more impressive when considering character tokenization performed better to begin with. In addition, unlike word tokenization, preprocessing didn't gradually decrease the training accuracy.

	Train Acc.	Test Acc.
no preprocessing	99.934%	91.374%
+remove retweets	99.922%	91.674%
+remove handles	99.950%	92.514%
+remove URLs	99.909%	93.338%
+reduce repetition	99.901%	93.482%

Table 5: Accuracy of character-based tokenization with gradually increasing preprocessing

Comparing the two confusion matrices, we see that the improvement was rather uniform across all languages, with the relative accuracy ordering remaining identical. However, the model is now less biased towards English, and my guess is this stems from the fact that not using word tokenization reduced the effect of English words regularly occurring within other tweets. The most common confusion is still between Tagalog and Indonesian, and by a wide margin. As to my understanding, their lexical similarity is not greater than that of between Spanish and Portuguese, so it should be interesting to investigate this result. In addition, while English and Tagalog still exhibit similar confusion patterns between themselves, Dutch is now closer to this pair. Moreover, Spanish and Portuguese are now relatively more confused with each other than with Italian, perhaps indicating that maybe Italian shares more exact words with Spanish and Portuguese, but Spanish and Portuguese have more common letter patterns between themselves.

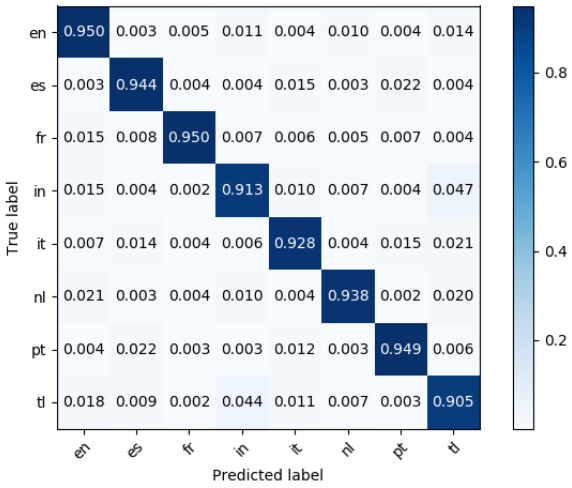

Figure 2: Confusion matrix using ngram features and full text preprocessing

Finally, To more accurately compare word and character based tokenizations, we would have to select the top 2000 words (to have a similar number of features) in the same manner as described in section 2.3.2, and compare the results. As I'm nearing the end of allowed pages, I'll leave it as a future suggestion. But personally, I believe only very thorough preprocessing methods will be able to negate the stronger effect of tweet-related noise on word-based tokenization. In addition, using only 2000 different words from all the languages combined 'covers' less data then using 2000 different ngrams, especially without extensive preprocessing. Another possible suggestion for improvement is to try and combine ngram and word features, with hope of utilizing the advantages of each. Other suggestions, such as using other variants of tf-idf, or emoji analysis, were already mentioned in the report.