Large-Scale Language Classification

Writing a Detector for 200 Languages on Twitter

Jordan Cazamias
jaycaz@stanford.edu

Chinmayi Dixit
cdixit@stanford.edu

Martina Marek
martinam@stanford.edu

ABSTRACT

Identifying the language of a text is an important requirement for any other processing on written text. While the focus so far has been mainly on language detection for long written text, short social media post like tweets are becoming more important. Furthermore, most language classifiers are designed to work only with dozens of languages. In this paper, we scale up the language identification to work on 200 languages, and additionally present the results of the classifier on a twitter data set of 70 languages. The classifier reached close to 95% when tested on 200 languages, and the results on the twitter data set are just short of 90%. Additionally, we made some optimizations to ensure that classifications are made in a timely manner, suitable for use on the web.

1. INTRODUCTION

Many automated language detectors exist, but most only work on the order of dozens of languages and focus on European languages. Meanwhile, hundreds of languages are spoken around the world, and many of these, although we never heard about them, are spoken by millions of people. As more and more people around the world, especially in Asia, but also Africa, begin to use social media websites like Twitter and Facebook, these languages are becoming better represented online. So, NLP technologies on these sites need to be equipped to handle the main spoken languages of the world, not just the most popular few.

We intend to build a language classifier for Twitter, training on a corpus of approximately 200 languages. Specifically, this classifier should be able to make a decision between these languages when given only a single tweet as input. Working with tweets will present some unique challenges. Because tweets are only 140 characters long, at most, we cannot gather statistics that require a large count of words. Also, dealing with colloquial language and misspellings will need to be considered.

1.1 Related Work

Chew, Mikami, and Nagano 2011 built a language identifier that can recognize 182 languages with 94% accuracy to be able to detect the language of web pages. They used an improved n-gram model as the features, and a corpus containing the Universal Declaration of Human Rights and Biblical texts as their data.
Grefenstette 2014 also used n-gram features as well as short words to identify the language of a given text. They showed that, especially on short sentences, n-gram features outperform word-based features. Their tests were done on the ECI corpus from the European Corpus Initiative on 9 different European languages.
Souter et al. 1994 investigated the effectiveness of three different sets of features, unique character strings, frequent words and bi- and trigram character features, showing that trigrams outperform frequent word features and all outperform the unique character strings by far.

A challenging task in Language Identification is the classification of closely related languages. As a result, the DSL (Discriminating between Similar Languages) shared task was defined (Zampieri et al. 2014). To tackle the task, different groups used mainly linear models and character or word based features. Porta and Sancho 2014 for example used a maximum Entropy classifier with word and/or character based n-gram models of varying size, depending on the language group that was to be classified. King, Radev, and Abney 2014 used a naive Bayes classifier and also experimented with word and character n-grams of varying length, and reported similar results. A more thorough discussion of the results can be found in Zampieri et al. 2014.

2. DATASETS

There are many translation text data corpora out there, but few cover more than several dozen up to a hundred languages. The ones which do cover the number of languages we need typically don't include actual data from Twitter. To combat this issue, we used several datasets for different purposes.

2.1 Big Dataset

Our largest dataset covered all the languages we intended to classify. As a base, it included texts that are translated into every language; for instance, religious texts like the Quran, as well as the Universal Declaration of Human Rights. For the more popular languages, it also included text from web sources such as Wikipedia articles. The data was

strictly in plaintext; however, because much of the text was extracted from websites, certain metadata such as HTML tags were included. Efforts were made to clean up this metadata as much as possible before training.

2.1.1 Simulated Twitter Data

As most sentences in this data set were much longer than usual twitter data (which is typically 140 characters long), we simulated twitter data by breaking up the sentences into shorter paragraphs of a specific word length. With this, we were able to evaluate the performance of our classifier depending on the sentence length of its input.

2.2 Twitter Dataset

We gathered pre-labeled twitter data using : `https://blog.twitter.com/2015/evaluating-language-identification-performance`. This data set covers 70 languages. We had approximately 80,000+ lines of tweets which we further divided into Train, Test and Dev data (60%, 20%, 10%).

2.3 DSL Shared Task

Discriminating between related languages is a challenging task, and therefore we used the DSL shared task data set from 2014 to test our classifiers on their performance on similar languages (Zampieri et al. 2014). The data set contains 6 groups of closely related languages. We concentrated our evaluation on the first three groups (Bosnian, Croatian and Serbian; Indonesian and Malaysian; Czech and Slovakian) as those appeared in our first data set, while the last three groups distinguish between languages spoken in different parts of the world (like American and British English). The data set contains 18,000 sentences for each languages in the training set, and 2,000 sentences per languages in the test set.

3. IMPLEMENTATION

3.1 Algorithm

3.1.1 Data Preprocessing

Big Dataset.

To remove the noise from our training/testing data, we made the following exclusions:

1. Discarded non-alphabet characters such as punctuation, digits, etc.

2. Some files had a lot of noise in them, especially HTML source code. Languages that had a lot of such noise in them were therefore often misclassified as each other, although not being related what so ever (like English and Chinese). To counteract this, we excluded all paragraphs that contained < or > in them. This made sure that any HTML tags would be excluded.

Twitter Dataset.

For twitter, we found that the users were generous in their use of hashtags, -handles, and URLs. As these did not contribute any value in recognizing the language used, we removed all words with # and in them, as well as strings formatted like a URL.

3.1.2 Bayes Classifier

For training/classification, we use a Naive Bayes classifier:

$$p(L_k|x_1, x_2, \cdots x_n) = p(L_k) \prod_{i=1}^{n} p(x_i|L_k)$$

During training, it creates a probabilistic model of each language and can estimate the conditional probabilities of a feature appearing with a certain class (language). During training, we retain the most frequent 5000 features to limit the number of features. During classification, if a feature is unknown, it gets a minimal probability assigned. As we assume that each language is equally likely to occur, the prior probabilities $p(L_k)$ are the same and can be omitted.
During classification, the features are extracted from the sentence and then for each language, the probability of the sentence being in that language is computed with the previously computed conditional probabilities. The language with the highest probability is the output of the classifier $\widehat{L}_k$. To avoid underflow errors, the log probabilities are taken:

$$\widehat{L}_k = \underset{L_k}{arg\max}\ p(L_k|x_1, x_2, \cdots x_n) = \underset{L_k}{arg\max} \sum_{i=1}^{n} log(p(x_i|L_k))$$

3.1.3 Logistic Regression

In contrast to a Naive Bayes, which as a generative model models the whole probability distribution, Logistic Regression is a discriminative model that models the conditional distribution directly.
Logistic regression takes a set of input data and a set of labels and maps a relationship between the two. During training, the algorithm calculates the weights on each of the features. For each training data point, we have a feature vector $x^{(i)}$ and an observed class of $y^{(i)}$. The probability of this class is p if $y = 1$ or $1-p$ if $y = 0$. So, the likelihood estimation using logistic regression will be $if there are only 2 classes$:

$$L = \prod_{i=1}^{n} p(x_i)^{y_i} (1 - p(x_i))^{(1-y_i)}$$

When there are more than 2 classes, the following formula is used for the conditional probabilities:

$$Pr(Y = c \mid X = x) = \frac{e^{{\beta_0}^{(c)} + x.\beta^{(c)}}}{\sum_c e^{{\beta_0}^{(c)} + x.\beta^{(c)}}}\}$$

During classification, the probabilities are used to find the likelihood of each of the classes. The class with the highest likelihood is chosen in the end.

3.1.4 RNN

Recurrent Neural Networks (RNNs) have made quite a splash in the Machine learning world thanks to some impressive use cases (`http://karpathy.github.io/2015/05/21/rnn-effectiveness/`). In short, RNNs are best suited for making predictions on input data that is sequential in nature. Since we are already treating our input as a sequence of characters, we reasoned that an RNN could be an effective tool for classifying tweet languages.

Implementation.

For the implementation of the RNN, we use the Deeplearning4j library *DeepLearning4j - Deep Learning for Java*. We

Figure 1: Simplified representation of the RNN

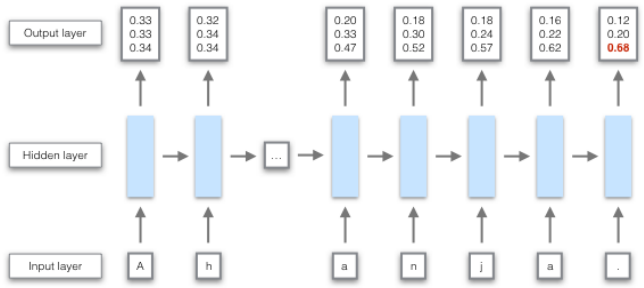


Table 1: Most frequent n-grams for some selected languages

	Most frequent n-grams
English	_they, _them, _the_, allah, _you_, n_the, _and_
French	s_de_, _est_, pour_, tion_, ment_, _des_, _pour
Spanish	tros_, _dios, s_de_, _de_l, o_que, s_que, _los_
German	_die_, allah, n_und, _der_, _sie_, _ist_, _und_
Czech	_jsme, jsme_, jest_, _kdož, jich_, _jest, _bůh_
Slovak	alebo, balík, _použ, lebo_, _ktor, enie_, _balí
Japanese	言った, あなたが た, なたがたの, ならない。, のである。, わたしたち

initialize an RNN and feed in a character at a time. These characters are represented by sparse vectors that are 1 at the index reserved for that character, and 0 at all others. As we have approx. 250 unique characters in the first group, the dimension of these sparse vectors is 1 x 250.
This input vectors go through the input layer with a tanh activation function. The input layer then feeds into a hidden layer of size 50 that uses the same activation function. For both these layers, we use LSTM cells as they have proven to be most effective with RNNs (as they solve the vanishing error back flow, see Hochreiter and Schmidhuber 1997).
Then comes the output layer with a softmax function (a maximum entropy classifier), that outputs the likelihood of each language at each given time step.
During classification, we look for the highest probability over the whole sequence, and decide for that language. Ideally, this should look like in figure Cho et al. 2014.

To speed up the training process, we spent quite a bit of time trying to run our RNN on a GPU using the CUDA functionality of the DeepLearning4J library. Unfortunately, as it turns out, this CUDA component was in the middle of a heavy rewrited by its developers and was not in a stable enough state to get working. This not only added more time, but hampered our ability to perfect the RNN's hyperparameters, as the time cost of an in-depth hyperparameter search would have been too great to perform without the GPU speedup.

3.2 Features

3.2.1 Frequency Features

The most intuitive set of features was simply to take the words in each language and build a dictionary of works ranked by their number of usages. In order to use less memory, only a subset of the most frequent words could be stored for each language. This approach was also described in **??**. One problem of this approach is that it only works for scripts that have spaces between words. For languages like Chinese or Japanese, one would first need to run another classifier to decide which characters form one word to be able to create a dictionary.

3.2.2 N-gram Features

A more suitable set of features for this problem is character n-grams. In part, this is because of the whitespace issue mentioned in the previous section. It is also well-suited to the task because working with Twitter data provides a convenient 140-character cap to the input data. However, special pre-processing needs to be done, in addition to the steps mentioned in the previous section, before training and classification can be performed effectively. Users' @-handles and hashtags are commonly in English, or at least in Latin characters, regardless of the language being spoken, so these tokens need to be ignored to avoid polluting our n-gram feature counts. Emoticons or emojis (i.e. pictographic symbols defined in Unicode) may also be ignored, or they may be included if the use of certain emoji is indicative of a certain language. Lastly, whitespace characters were replaced with an underscore character.

Rather than store the counts of all our n-grams, which would have required a huge amount of data storage and slowed down our system considerably, we performed our counts and only stored the top k results for each language. We tried different values of k, ranging from 1000 to 5000.

As for the most effective size of n-gram to use, we simply ran our n-gram classifier on a range from size 2 to size 6 to find the most effective size. We also tried collecting n-grams of multiple sizes. The results are discussed in the next section.

Some of the most frequent n-grams can be seen in table 1. Usually short, frequent words, like pronouns, articles and conjunctions, make it to the top. However, in some languages like Czech, pronouns are used less frequently as the information about the gender is already contained in the verb. Instead, frequent words like "be" ("jsme") make it to the top. Also, frequent syllables, like "tion_" in French, make it to the top.
For languages that use a different script, like Japanese, the type of n-grams can differ. In the Japanese script, each character is equal to 2 or 3 English characters, and therefore, while there are still only 5 characters per n-gram, much longer words can make it to the top. So the n-grams for Japanese contain mainly combinations of pronouns and prepositions or conjugated verbs in combination with prepositions. As a big part of our training data is the Quran, various variations of the word "Allah" or God show up in most languages.

4. RESULTS

4.1 Baseline

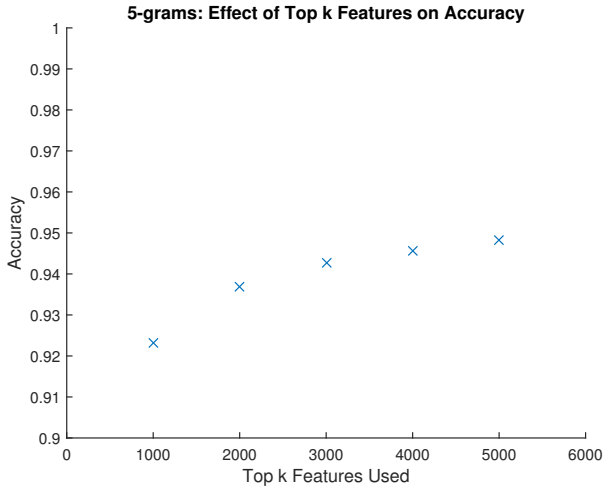
Our baseline system selected the most frequent word from each language. If any of the most frequent words were found in the input text, it would be classified into the language. As expected, this baseline has a low performance of just 22,3%, as shown in table 2.

Frequency classifier.

	Baseline	Frequency	Bayes
Accuracy	0.223	0.824	0.948

Table 2: Accuracy for the different classifiers, tested on 1000 test sentences per language, trained on the first 10000 paragraphs

Figure 2: Accuracy depending on the number of features (5-grams) used



Using the top 100 most frequent words in place of the one most frequent word increased the span for the algorithm. This in turn led to an improvement on the baseline by approx. 60% (see table 2).

4.2 Bayes

With Bayes classification, we tested the output with different n-gram lengths. The accuracy for each is shown in table 3. Based on the accuracy, 5 was the best performing length for n-grams. The accuracy was almost 95% using 10000 paragraphs for training and 1000 for testing.
We also tested how using ranges of n-grams (meaning we used bigrams, trigrams,... at the same time) would influence accuracy. Interestingly, using just 5-grams still had the best accuracy, although using 3-5 grams came very close in accuracy. Including bigrams as well decreased the accuracy, though, as bigrams overall appear more frequently than other n-grams, and would therefore be more likely to included in the feature set. Since bigrams are the shortest, they are also the most likely to be shared across languages, and therefore reduced the accuracy.

We tested the accuracy and speed of the algorithms by varying certain parameters. Figure 4 shows the effect of increase in number of features on the accuracy. As expected, the accuracy increased with the increase in number of features; but, the plot shows that we may have been getting close to a saturation point after which further improvement would be negligible.

Since the long-term goal is to be able to classify Twitter data, which usually consists of shorter sentences, we tested the algorithm with simulated twitter data (as described in section 2.1.1) on varying sentence lengths. The performance is stable with 15 words and more, and starts dropping when the sentences are shorter. The accuracy still stays over 90%

Figure 3: Using different ranges of n-grams, from 2-to-5 all the way to 5-to-5

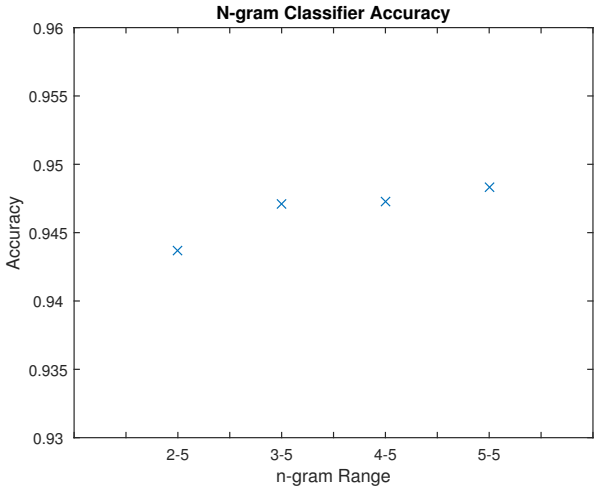


Figure 4: How the accuracy of our n-gram classifier holds up as we use smaller and smaller sentences

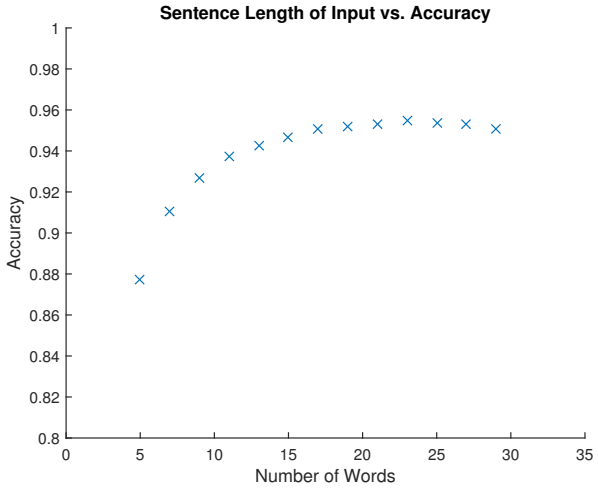
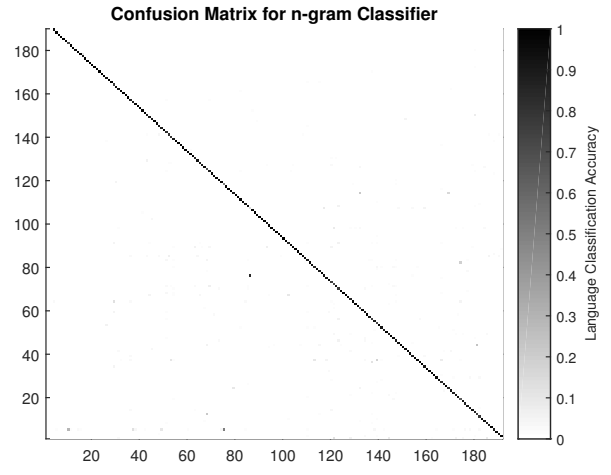


Figure 5: Our entire confusion matrix, summarizing the overall classification accuracy of our system



for 7 words or more though, and only drops slightly below 88% when the sentences consisted of 5 words only. The results can be seen in figure 4.

4.2.1 Analysis of the Confusion Matrix

For a detailed analysis we looked at the performance of our classifier by language and computed a confusion matrix (see figure 4.2.1) to be able to see which languages are most likely to be missclassified as each other. 62 languages had an accuracy of 99% or higher, while only 14 languages scored lower than 80%. 26 languages scored between 80-90%.

The languages that were either completely or nearly completely correctly classified usually either had a unique writing system, a unique set of characters only used by this language, or were not related with any other language in our set and could therefore be classified based on their very unique n-grams. An example of the latter is Oromo, a language spoken by more than 40 million people in east Africa (`https://en.wikipedia.org/wiki/Oromo_language#Sounds_and_orthography`), that, although it uses Roman characters and no special characters, can be easily told apart from the other languages in our set by it's unique style of writing with many double characters: "Baayyinaa ummataan garuu sadarkaa sadaffaarra jirti, kunis Eshiyaa fi Afrikaatti aanteti.".

The languages with the lowest performance were pairs of closely related languages that shared many n-grams with each other. By far the worst performance was Malay with an accuracy of only 15.7%. Malay is very closely related to Indonesian, and was misclassified as Indonesian over 80% of the time.

In a similar fashion, Serbian reached an accuracy of only 68.1% and was most often (22.7%) misclassified as Croatian, to which it is closely related. In 3.6% of the cases it was misclassified as Bosnian, the third of these closely related language that also appear as one group of the DSL shared task, as do Malay and Indonesian.

It also often misclassified Danish as Norwegian (23.7%). Some research on those two languages showed, although they differ in pronunciation, their writing system shares the same special characters (æ, ø and å) and they have similar vocabulary (`http://blog.jensen-localization.com/en/2013/07/are-danish-and-norwegian-languages-so-close-as-we-imagine.html`). These results show that while the Bayes classifier works well overall, it is not able to capture the subtle differences between languages that share not only the same alphabet, but are also similar in structure and vocabulary.

Some of the error is not due to closely related languages, though, but simply to noisy data. We noticed that some files, like English and Chinese, have many HTML tags in them, as parts of their data was copied from online resources. Therefore Korean and Chinese, although both having a unique set of characters, don't have close to perfect accuracy, but achieve only 72% and 57.6%, respectively. The rest of the time both are misclassified as Iloko, a language spoken by people in the Philippines, although neither is related to it (`https://en.wikipedia.org/wiki/Ilocano_language`). But in all three data sets, HTML tags appear, and therefore their learned language models are similar.

English, although achieving a solid accuracy of 98.1%, is misclassified as Chinese in 1.7%, also due to the HTML tags that appear in the English data set.

Czech and Slovak, although usually considered as very closely related languages, and a group that also appear in the DSL shared task, had both solid results of 98.6% and 92.8%, respectively. While those two languages are mutually intelligible, both languages have their own unique set of special characters which makes it easier for the classifier to build up differing language models (`https://en.wikipedia.org/wiki/Comparison_of_Slovak_and_Czech`).

4.3 Logistic Regression

For the testing of the Logistic Regression classifier we could use only 1000 features for each language, as we ran into "out of memory" issues when training on more features (and even when using 1000 features with 6-grams). As expected, the results are a little bit lower when training on a smaller feature set. We therefore reran our tests on the Bayes classifier to be able to compare the results of both to each other. The results can be seen in figure 6. As both are linear classifiers and both use the same set of features, the performance is very similar. Only for bigram features the performance of the Logistic Regression classifier is much worse than the Naive Bayes, and drops to 26.3%. We are not sure why there is such a huge gap, since both classifiers use the exactly same set of features and the same data set for both training and testing.

Additionally, logistic regression significantly reduced the classification time (measured for testing 200,000 sentences) by 1/3rd. This is shown in Figure 7.

4.4 Twitter Data

With the Twitter data, we received an accuracy in the range of 80% to 89.7%. Our tests on the simulated twitter data (section 4.2) showed that the accuracy stays quite high as long as the sentences have 7 words or more, which is true for most tweets. The lower performance is therefore not only due to the shorter length of the sentences. There are many reasons for the lower performance in this case. First of all, there was less data to train the classifier on. Secondly, the nature of the data is more difficult. Thirdly, the tweets contained a high number of emojis, special characters etc. They also spelled words in a non-standard way (abbreviations, spelling mistakes, leaving out special characters out of

Figure 6: Comparison of the performance of the Bayes and Logistic Regression classifier on different n-grams. 1000 Features were used.

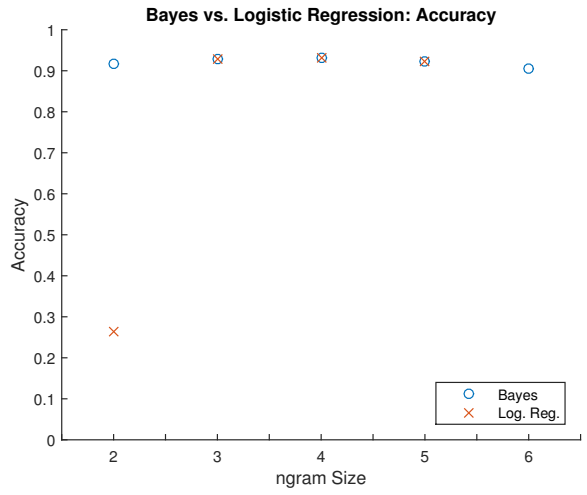


Figure 7: Comparison of the classification time of 200,000 sentences the Bayes and Logistic Regression classifier with different n-grams. 1000 Features were used.

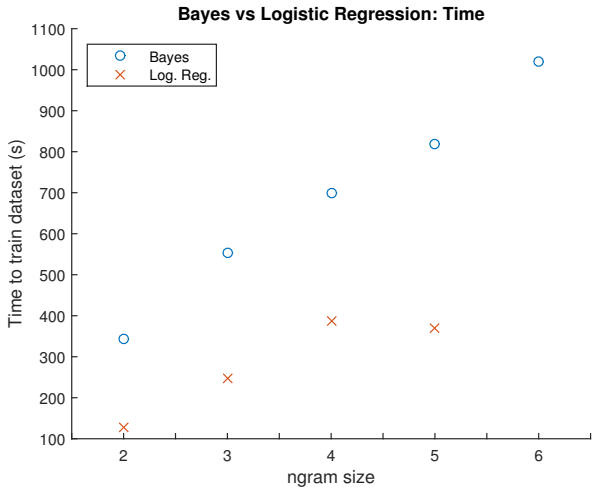
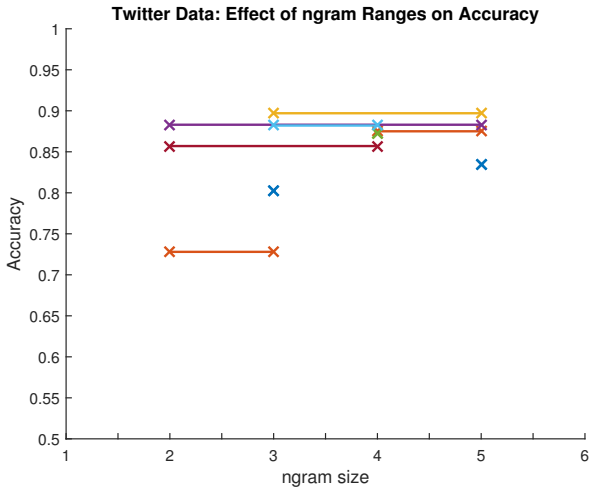


Figure 8: Performing a beam search can be done without a serious degradation in accuracy



laziness). This would cause some noise during the training process. Also, the length of each tweet was limited to only a few words once the hashtags and references were removed, reducing the sentence length further. As the sentence length was lower, accurate classification became more difficult.

Testing with different n-gram lengths, we could see that the best performance was found at the n-gram length 3-5. Figure 8 shows the comparison of the accuracy for each of the n-gram selections.

4.5 Optimization

Since both our features space as well as our class space is extremely big, the classification of each sentence requires the Bayes classifier to compute the probability for each of the 200 languages based on all 5000 features. While this is still quick for a single sentence, testing 1000 sentences from each language (so 200,000 sentences in total) takes a considerate amount of time. If one would now want to scale this up and classify millions of twitter tweets that come in every day, the algorithm needs to run faster.

For this, we implemented a beam search style algorithm, that first takes a subset of our feature space to decide for the most likely n languages, and then outputs the most likely language based on the full set of features out of this list.

Of course some accuracy is lost when using this approach, but the speed of the algorithm improves considerably. When using the top 100 n-grams as features for the first classification round, the accuracy drops by approx. 5% (depending on the beam size used), but the algorithm runs nearly twice as fast. Results can be seen in figure 9 and 10.

When using the most common 1000 n-grams in the first subset, the accuracy decreased by less than 1%, while still gaining more than 30% in speed. The results are shown in figure 9 and 10.

Since we noticed that the Logistic Regression classifier was much faster during classification time, we implemented another 2-step classificator that first uses a Logistic Regression classifier with 1000 features to output the n most likely languages, and then used the Bayes classifier with 5000 features to decide for the most likely language out of this set. In this

Figure 9: Performing a beam search can be done without a serious degradation in accuracy

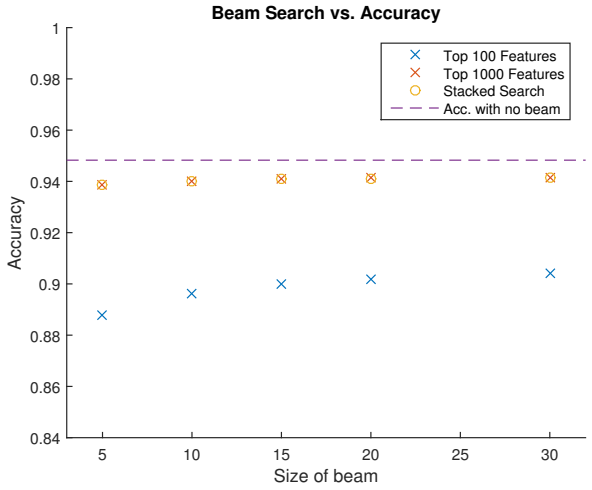


Figure 10: Performing a beam search can significantly improve classification time

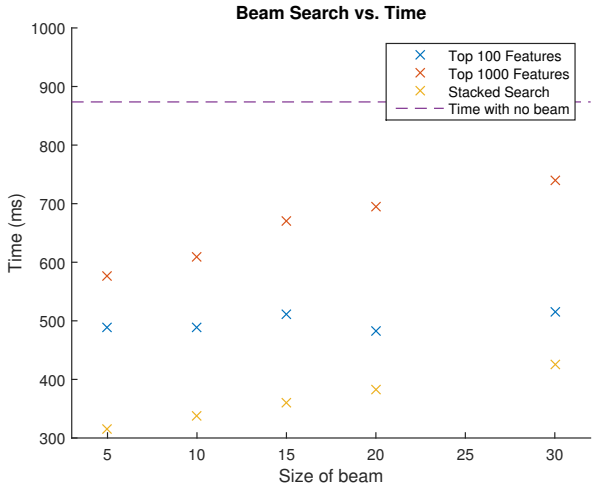


Table 3: Performance of the Bayes classifier on the DSL shared task set with 5-grams and 5000 features

	Bosnian, Croatian, Serbian	Indonesian, Malaysian	Czech, Slovakian
Accuracy	0.867	0.950	0.995

Table 4: Performance of the Logistic regression classifier on the DSL shared task set with 5-grams and 5000 features

	Bosnian, Croatian, Serbian	Indonesian, Malaysian	Czech, Slovakian
Accuracy	0.854	0.959	0.999

way, we benefited form the speed of the Logistic Regression classifier (the classification time dropped to nearly a third), while still maintaining the high accuracy of the Bayes classificator (the performance drops by just 1%, even on small beam sizes). Results can be seen in figure 9 and 10.

4.6 DSL shared task

As our classifier performed poorly on some very similar languages, like Malay and Indonesian, or Serbian, Bosnian and Croatian, we used the DSL shared data set to see how the performance could be improved for those languages.
In a first step, we tested our existing classifiers on this data, and then used an RNN to try to improve the performance where the linear classifiers were struggling.

4.6.1 Bayes

The results in table 3 show that our 5-gram Bayes classifier works well for the groups Czech and Slovakian, as well as for Malay and Indonesian, while it scores significantly lower on the first group, Serbian, Bosnian and Croatian.

4.6.2 Logistic Regression

Similarly, when classifying Indonesian from Malaysian and Czech from Slovakian the Logistic Regression classifier performed very well with 95.9% and 99.9%. During classification of Bosnian, Croatian and Serbian the performance was a comparatively lower 85.4%.

4.7 RNN

When training the RNN on a small data set, and testing on the same data set, it overfitted quickly with an error rate of nearly zero percent and showed that the approach could potentially work really well. The result vector starts off with equally distributed probabilities over all labels, and then gets more sure of each label when the sequence progresses.
Unfortunately, though, when training on the full data set, the algorithm did not converge, but stagnated around 30% accuracy. This could be due to the fact that the hyper parameters are not perfectly tuned, but we did not have enough time to test out all sorts of different parameters. Also, due to the long training time, we could train the RNN only for a couple of hundred iterations, which is most likely not enough.
Furthermore, maybe assigning each character in a sequence a label is a too strongly constrained, and instead there should

be only one label outputted at the end of each sentence. This could be achieved with a Encoder/Decoder-style RNN as used in Machine Translation nowadays, where the input sequence is first encoded into a feature vector by a first RNN, and then decoded by a second RNN which then decides for a label, as described in **??**. This could be an interesting thing to try out in the future.

5. CONCLUSION

5.1 Outlook & Future Work

So far, our classification system has performed better than expected, with an accuracy in the 90 percent range and a reasonable classification time. As we anticipated, there were issues classifying certain, very similar languages, but that is to be expected from any large scale language classification system. The question becomes, then, how to proceed further.

There are two main goals we would like to achieve. First and foremost, we would like to push up our accuracy. Simply improving the overall accuracy, however, is not enough to give us an effective system. There are specific "problem languages" that have an unusually high chance of misclassification, and if speaking members of those languages cannot rely on the classification system, then our system has failed to be a globally effective system. Therefore, further approaches for training on the DSL corpus would be a good first step.

The RNN, of course, is one way to do that. Perhaps, with the right balance of hyperparameters, an RNN system would work well on the DSL corpus. As such, perhaps a hybrid classification system would be useful, where an n-gram classifier is used to classify most languages and an RNN is used for the most problematic languages.

The other important component is time. Since this system is intended for Twitter data, it would most likely be used in a web setting and therefore, classification time needs to be as quick as possible while still maintaining an acceptable level of accuracy. Our stacked beam search was an important step in this direction, and there are likely additional optimizations that we could make.

6. ACKNOWLEDGMENTS

We would like to thank Dr. David Jurgens for inspiring us to take on this problem and for his continued mentorship throughout this process.

References

Chew, Yew C., Yoshiki Mikami, and Robin L. Nagano (2011). "Language Identification of Web Pages Based on Improved N-gram Algorithm". In: *International Journey of Computer Science Issues* 8.3.

Cho, Kyunghyun et al. (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *CoRR* abs/1406.1078. URL: `http://arxiv.org/abs/1406.1078`.

DeepLearning4j - Deep Learning for Java. `http://deeplearning4j.org/about.html`. Accessed: 2015-11-19.

Grefenstette, Gregory (2014). "Comparing two language identification schemes". In: pp. 58–67.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long Short-Term Memory". In: *Neural Comput.* 9.8, pp. 1735–1780. ISSN: 0899-7667.

King, Ben, Dragomir Radev, and Steven Abney (2014). "Experiments in Sentence Language Identification with Groups of Similar Languages". In: Dublin, Ireland, pp. 146–154.

Porta, Jordi and Jose Luis Sancho (2014). "Using Maximum Entropy Models to Discriminate between Similar Languages and Varieties". In: Dublin, Ireland, pp. 120–128.

Souter, Clive et al. (1994). "Natural Language Identification using Corpus- Based Models". In: *Journal of Linguistic* 13.

Zampieri, Marcos et al. (2014). "A Report on the DSL Shared Task 2014". In: *Proceedings of the First Workshop on Applying NLP Tools to Similar Languages, Varieties and Dialects*, pp. 58–67.