Discriminating Between Similar Nordic Languages

René Haas, Leon Derczynski
IT University of Copenhagen
{renha, leod }@itu.dk

Abstract

Automatic language identification is a challenging problem and especially discriminating between closely related languages is one of the main bottlenecks of state-of-the-art language identification systems. This paper presents a machine learning approach for automatic language identification for the Nordic languages, which often suffer miscategorisation by existing state-of-the-art tools. Concretely we will focus on discrimination between the six Nordic languages: Danish, Swedish, Norwegian (Nynorsk), Norwegian (Bokmål), Faroese and Icelandic.

Keywords: DSL, NLP, Language Identification, Machine Learning

1. Introduction

Automatic language identification is a challenging problem and especially discriminating between closely related languages is one of the main bottlenecks of state-of-the-art language identification systems.(Zampieri et al., 2014)

This paper presents a machine learning approach for automatic language identification for the Nordic languages. Concretely we will focus on discrimination between the six Nordic languages: Danish, Swedish, Norwegian (Nynorsk), Norwegian (Bokmål), Faroese and Icelandic.

This papers explore different ways of extracting features from a corpus of raw text data consisting of Wikipedia summaries in respective languages and evaluates the performance of a selection of machine learning models.

Concretely we will compare the performance of classic machine learning models such as Logistic Regression, Naive Bayes, Support vector machine, and K nearest Neighbors with more contemporary neural network approaches such as Multilayer Perceptrons (MLP) and Convolutional Neural Networks (CNNs).

After evaluating these models on the Wikipedia data set we will continue to evaluate the best models on a data set from a different domain in order to investigate how well the models generalize when classifying sentences from a different domain.

2. Related Work

The problem has been investigated in recent work (Goutte et al., 2016)(Zampieri et al., 2015) which discuss the results from two editions of the "Discriminating between Similar Languages (DSL) shared task". Over the two editions of the DSL shared task different teams competed to develop the best machine learning algorithms to discriminate between the languages in a corpus consisting of 20K sentences in each of the languages: Bosnian, Croatian, Serbian, Indonesian, Malaysian, Czech, Slovak, Brazil Portuguese, European Portuguese, Argentine Spanish, Peninsular Spanish, Bulgarian and Macedonian.

3. The Dataset

The first step when constructing a language classifier is to gather a dataset. For this purpose we use a small script using the Wikipedia API for Python.[1]

The script helped download the summaries for randomly chosen Wikipedia articles in each of the languages which are saved to as raw text to 6 `.txt` files of about 10MB each.

After the initial cleaning, described in the next section, the dataset contains just over 50K sentences in each of the language categories. From this two datasets with exactly 10K and 50K sentences respectively are drawn from the raw dataset. In this way the datasets we will work with are balanced, containing the same number of data points in each language category.

Throughout this report we split these datasets, reserving 80% for the training set and 20% for the test set we use when evaluating the models.

3.1. Data Cleaning

This section describes how the dataset is initially cleaned and how sentences are extracted from the raw data.

Extracting Sentences The first thing we want to do is to divide the text into sentences. This is generally a non-trivial thing to do. My approach is to first split the raw string by line break. This roughly divides the text into paragraphs with some noise which we filter out later.

We then extract shorter sentences with the sentence tokenizer (`sent_tokenize`) function from the NLTK(Loper and Bird, 2002) python package. This does a better job than just splitting by `'.'` due to the fact that abbreviations, which can appear in a legitimate sentence, typically include a period symbol.

[1] `https://pypi.org/project/wikipedia/`

Cleaning characters The initial dataset have a lot of characters that do not belong to the alphabets of the languages we work with. Often the Wikipedia pages for people or places contain the name in the original language. For example a summary might contain Chinese or Russian characters which are arguably irrelevant for the purpose of discriminating between the Nordic languages.

To make the feature extraction simpler, and to reduce the size of the vocabulary, the raw data is converted to lowercase and stripped of all characters with are not part of the standard alphabet of the six languages.

In this way we only accept the following character set

```
'abcdefghijklmnopqrstuvwxyzáäåæéíðóöøúýþ '
```

and replace everything else with white space before continuing to extract the features. For example the raw sentence

```
'Hesbjerg er dannet ved sammenlægning
 af de 2 gårde Store Hesbjerg
 og Lille Hesbjerg i 1822.'
```

will after this initial cleanup be reduced to

```
'hesbjerg er dannet ved sammenlægning
 af de gårde store hesbjerg
 og lille hesbjerg i ',
```

We thus make the assumption that capital letters, numbers and characters outside this character set do not contribute much information relevant for language classification.

4. Baselines

4.1. Baseline With langid.py

As a baseline to compare the performance of the models in we compare with an out of the box language classification system. “langid.py: An Off-the-shelf Language Identification Tool." (Lui and Baldwin, 2012) is such a tool.

Out of the box langid.py comes with with a pretrained model which covers 97 languages. The data for langid.py comes from from 5 different domains: government documents, software documentation, newswire, online encyclopedia and an internet crawl.

We evaluated how well langid.py performed on the Wikipedia dataset. Since langid.py returned the language id “no" (Norwegian) on some of the data points we restrict langid.py to only be able to return either “nn" (Nynorsk) or “nb" (Bokmål) as predictions. It is a quite peculiar feature of the Norwegian language that there exist two different written languages but three different language codes.

In Figure 1 we see the confusion matrix for the langid.py classifier. The largest errors are between Danish and Bokmål and between Faroese and Icelandic. We see that langid.py was actually able to correctly classify most of the Danish data points however approximately a quarter of the data points in Bokmål was incorrectly classified as

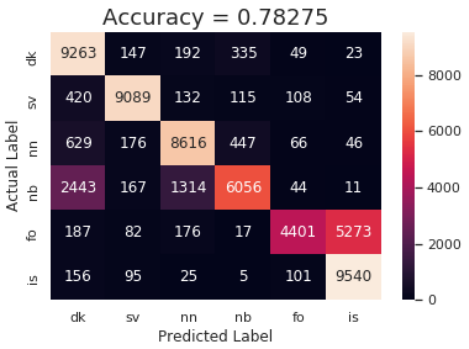


Figure 1: Confusion matrix with results from langid.py on the full dataset with 300K data points.

Model	Encoding	Accuracy
Knn	cbow	0.780
Log-Reg	cbow	0.819
Naive Bayes	cbow	0.660
SVM	cbow	0.843
Knn	skipgram	0.918
Log-Reg	skipgram	**0.929**
Naive Bayes	skipgram	0.840
SVM	skipgram	**0.928**
Knn	char bi-gram	0.745
Log-Reg	char bi-gram	0.907
Naive Bayes	char bi-gram	0.653
SVM	char bi-gram	0.905
Knn	char uni-gram	0.620
Log-Reg	char uni-gram	0.755
Naive Bayes	char uni-gram	0.614
SVM	char uni-gram	0.707

Table 1: Overview of results for the dataset with 10K data points in each language.

Danish and just under and eighth was classified as Nynorsk.

Furthermore langid.py correctly classified most of the Icelandic data points however over half of the data points in Faroese was incorrectly classified as Icelandic.

4.2. Baseline with linear models

In the Table 1 we see the results for running the models on a dataset with 10K data points in each language category. We see that the models tend to perform better if we use character bi-grams instead of single characters.

Also we see that logistic regression and support vector machines outperform Naive Bayes and K-nearest neighbors in all cases. Furthermore for all models we get the best performance if we use the skip-gram model from FastText.

Comparing the CBOW mode from FastText with character bi-grams we see that the CBOW model is on par with bi-grams for the KNN and Naive Bayes classifiers while bi-grams outperform CBOW for Logistic Regression and support vector machines.

5. Our Approach

5.1. Using FastText

The methods described above are quite simple. We also compare the above method with FastText, which is a library for creating word embeddings developed by Facebook (Joulin et al., 2016).

In the paper "Enriching Word Vectors with Subword Information" (Bojanowski et al., 2016) the authors explain how FastText extracts feature vectors from raw text data. FastText makes word embedding using one of two model architectures: continuous bag of words (CBOW) or the continuous skip-gram model.

The skip-gram and CBOW models are first proposed in (Mikolov et al., 2013) which is the paper introducing the word2vec model for word embeddings. FastText builds upon this work by proposing an extension to the skip-gram model which takes into account sub-word information.

Both models use a neural network to learn word embedding from using a context windows consisting of the words surrounding the current target word. The CBOW architecture predicts the current word based on the context, and the skip-gram predicts surrounding words given the current word.(Mikolov et al., 2013)

5.2. Using A Convolutional Neural Network

While every layer in the MLP is densely connected such that each of the nodes in a layer is connected to all nodes in the next layer, in a convolution neural network we use one or more convolutional layers.

Convolutional Neural networks are very popular for image recognition but they can also be used for text classification (Jacovi et al., 2018).

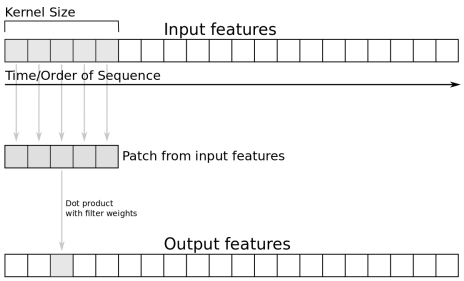


Figure 2: Diagram of Convolutional Neural network.

The basic premise of a convolutional layer is illustrated in Figure 2.[2] In a CNN you have a filter which slides over the input. The CNN then takes the dot product of the weights of the filter and the corresponding input features, before applying the activation function.

[2]Source: https://realpython.com/python-keras-text-classification/

Model	Encoding	Accuracy
MLP	char bi-gram	0.898
CNN	char bi-gram	**0.956**
MLP	char uni-gram	0.697
CNN	char uni-gram	0.942

Table 2: Overview of results for the neural network models for the data set with 10K data points in each language.

Model	Encoding	Accuracy
Knn	skipgram	0.931
Logistic Regression	skipgram	**0.933**
Naive Bayes	skipgram	0.806
SVM	skipgram	0.925

Table 3: Overview of results for the data set with 50K data points in each language.

6. Results

6.1. Results with neural networks

In the following we evaluate the initial results for the neural network architectures which can be seen in Table 2. Here we compare the result of doing character level uni- and bi-grams using the Multilayer Perceptron and Convolutional Neural Network. We see that the CNN performs the best, achieving an accuracy of 95.6% when using character bi-grams. Both models perform better using bi-grams than individual characters as features while the relative increase in performance is greater for the MLP model.

6.2. Increasing the size of the data set

Usually the performance of supervised classification models increase with more training data. To measure this effect we increase the amount of training data to 50K sentences in each of the language categories. Due to much longer training times only the "baseline models" are included with the skip-gram encoding from FastText which we saw achieved the highest accuracy.

Table 3 show that the accuracies for the logistic regression model and the K nearest neighbors algorithm improved slightly by including more data. Unexpectedly, performance of the support vector machine and naive Bayes dropped a bit with extra data.

Even when including five times the amount of data the best result, logistic regression with an accuracy of 93.3%, is still worse than for the Convolutional Neural Network trained on 10K data points in each language.

Model	Encoding	Accuracy
MLP	char bi-gram	0.918
CNN	char bi-gram	**0.970**

Table 4: Overview of results for the data set with 50K data points in each language.

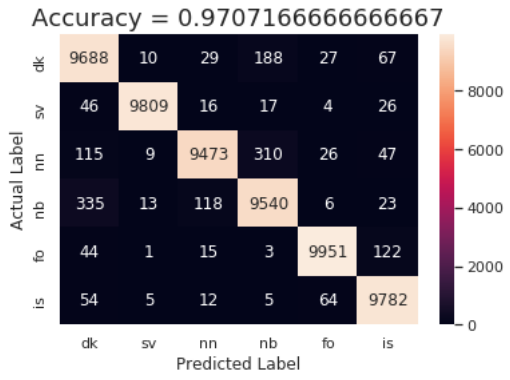


Figure 3: Confusion matrix with results from the convolutional neural network on the full data set with 50K data points in each language.

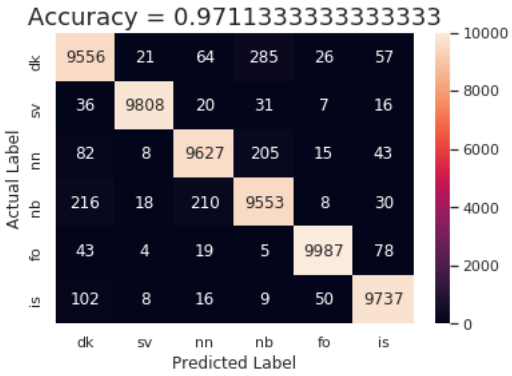


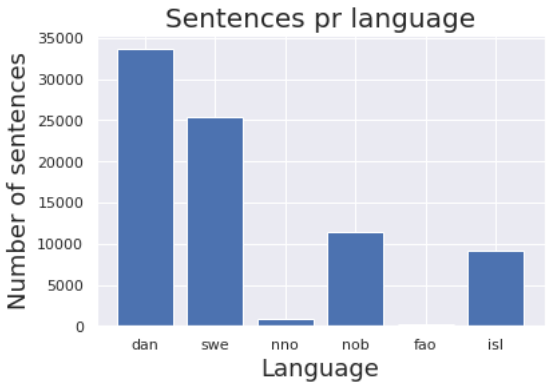
Figure 4: Confusion matrix with results from a supervised FastText model on the full data set with 300K data points.

In Table 4 we see the results for running the neural networks on the larger data set. Both models improve by increasing the amount of data and the Convolutional Neural Network reached an accuracy of 97% which is the best so far.

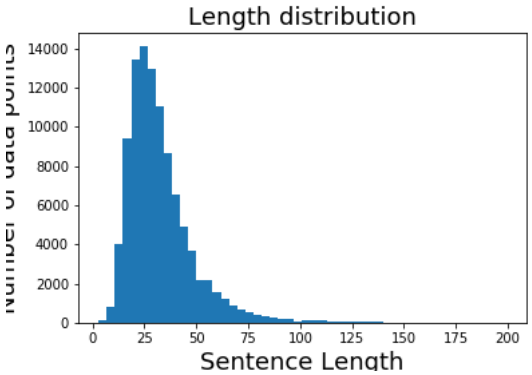
In Figure 3 we see the confusion matrix for the convolutional Neural Network trained on the full Wikipedia data set with 300K data points pr language. We see that the largest classification errors still happens between Danish, Bokmål and Nynorsk as well as between Icelandic and Faroese.

6.3. Using FastText supervised

FastText can also be used for be trained supervised and be used for Classification. In the Paper Bag of Tricks for Efficient Text Classification (Joulin et al., 2016) the authors show that FastText can obtain performance on par with methods inspired by deep learning, while being much faster on a selection of different tasks in Tag prediction and Sentiment Analysis. The confusion matrix from running the FastText supervised classifier can be seen in Figure 4. We see that FastText is roughly on par with the CNN.



(a) Distribution of the number of sentences in each language in the Tatoeba data set.



(b) Distribution of the length of sentences in the Tatoeba data set.

Figure 5: Distribution of the lengths and language classes of Tatoeba sentences.

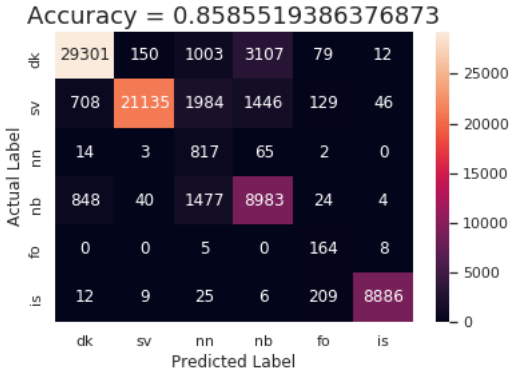


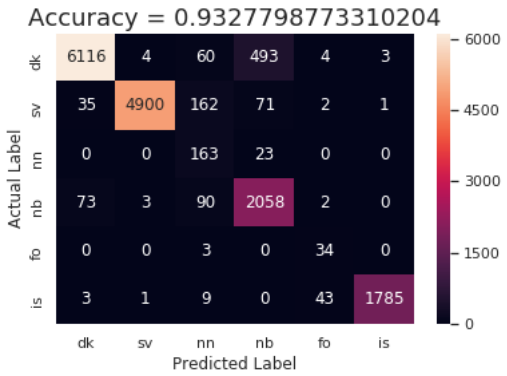
Figure 6: Confusion matrix for FastText trained using only character level n-grams on the Wikipedia data set and evaluated on the Tatoeba data set.

6.4. Evaluating the models on another data set

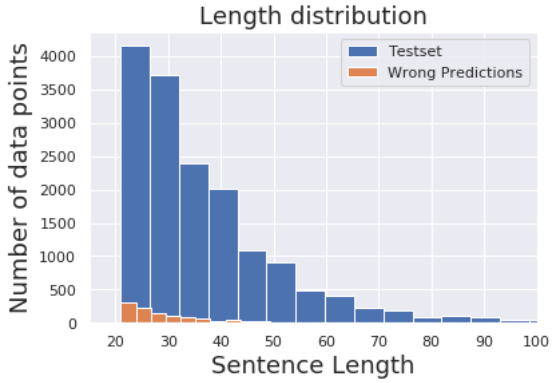
It would be interesting to see how the two best performing models generalize by testing on a data set different from the they have been trained on (the Wikipedia data set).

I downloaded an additional data set from Tatoeba[3] which is a large database of user provided sentences and translations. In Figure 5a we see the number of sentences in each

[3] tatoeba.org/



(a) Confusion matrix for FastText trained using only character level n-grams on the combined Wikipedia/Tatoeba data set and evaluated on the Tatoeba data set.



(b) Distribution of sentence lengths Tatoeba test set along with the mis-classified sentences.

Figure 7: Error analyses.

language for all sentences in the Tatoeba data set. Observe that we have very few samples in Nynorsk and Faroese.

The language used in the Tatoeba data set is different from the language used in Wikipedia. The Tatoeba data set mainly consists of sentences written in everyday language. Below we see some examples from the Danish part of the Tatoeba data set.

```
Hvordan har du det?

På trods af al sin rigdom
og berømmelse, er han ulykkelig.

Vi fløj over Atlanterhavet.

Jeg kan ikke lide æg.

Folk som ikke synes at latin er det
smukkeste sprog, har intet forstået.
```

We see that the performance drops quite a lot when shifting to another domain. For reference the accuracy of langid.py on this data set is 80.9% so FastText actually performs worse than the baseline with an accuracy of 75.5% while the CNN is a bit better than the baseline with an accuracy of 83.8 %

One explanation for the drop in performance is that the sentences in the Tatoeba data set is significantly shorter than the sentences in the Wikipedia data set as seen in Figure 5b. As we saw in the previous section both models tend to mis-classify shorter sentences more often than longer sentences. This and the fact that the "genre" of sentences are different might explain why the models trained on the Wikipedia data set does not generalize too well to the Tatoeba data set without a drop on performance.

The CNN uses character bi-grams as features while, with the standard settings, FastText uses only individual words to train. The better performance of the CNN might indicate that character level n-grams are more useful features for language identification than words alone.

To test this we changed the setting of FastText to train using only character level n-grams in the range 1-5 instead of individual words. In Figure 6 we see the confusion matrix for this version of the FastText model. This version still achieved 97.8% on the Wikipedia test set while improving the accuracy on the Tatoeba data set from 75.4% to 85.8% which is a substantial increase.

Thus using character level features seem to improve the FastText models ability to generalize to sentences belonging to a domain different from the one it has been trained on.

6.5. Retraining on the combined data set

To improve the accuracy over the Tatoeba data set we retrained the FastText model on a combined data set consisting of data points from both the Wikipedia and Tatoeba data set.

The FastText model achieved an accuracy of 97.2% on this combined data set and an accuracy of 93.2% when evaluating this model on the Tatoeba test set alone - the confusion matrix can be seen in Figure 7a.

As was the case with the Wikipedia data set the mis-classified sentences tend to be shorter than the average sentence in the data set. In Figure 7b we see the distribution of sentence lengths for the Tatoeba test set along with the mis-classified sentences.

In the Tatoeba test set the mean length of sentences is 37.66 characters with a standard deviation of 17.91 while the mean length is only 29.70 characters for the mis-classified sentences with a standard deviation of 9.65. This again supports the conclusion that shorter sentences are harder to classify.

7. Analysis

7.1. Principal Component analysis and t-SNE

To gain additional insight on how the different word embedding capture important information about each of the language classes I thought it would be interesting

to try and visualize the embeddings using two different techniques for dimensionality reduction.

No matter which way we choose to extract the feature vectors they belong to a high dimensional feature space and in order to do visualization we need to project the feature vectors down to 2d space.

To do this I have implemented two different methods: Principal Component Analysis (PCA) which i will compare with T-distributed Stochastic Neighbor Embedding (t-SNE). Here we will begin with a brief explanation of the two techniques and proceed with an analysis of the results.

Principal Component Analysis The first step is to calculate the covariance matrix of the dataset. The components of the covariance matrix is given by

$$K_{X_i,X_j} = E[(X_i - \mu_i)(X_j - \mu_j)] \tag{1}$$

where X_i is the ith component of the feature vector and μ_i is the mean of that component.
In matrix form we can thus write the covariance matrix as

$$K(\mathbf{x},\mathbf{z}) = \begin{bmatrix} \operatorname{cov}(x_1,z_1) & \dots & \operatorname{cov}(x_1,z_n) \\ \vdots & \ddots & \vdots \\ \operatorname{cov}(x_n,z_1) & \dots & \operatorname{cov}(x_n,z_n) \end{bmatrix} \tag{2}$$

The next step is to calculate the eigenvectors and eigenvalues of the covariance matrix by solving the eigenvalue equation.

$$\det(Kv - \lambda v) = 0 \tag{3}$$

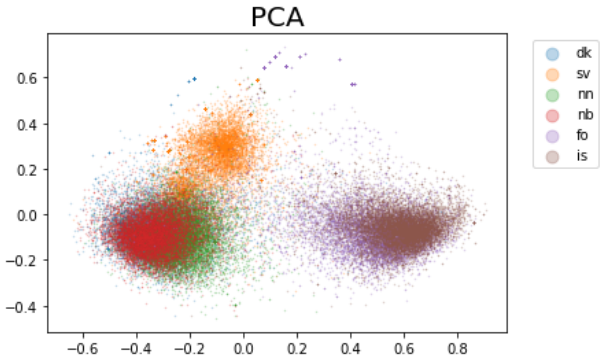
The eigenvalues are the variances along the direction of the eigenvectors or "Principal Components". To project our dataset onto 2D space we select the two eigenvectors largest associated eigenvalue and project our dataset onto this subspace.

In Figure 8 we see the result of running the PCA algorithm on the wikipedia dataset where we have used character level bigrams as features as well as the cbow and skipgram models from FastTest.

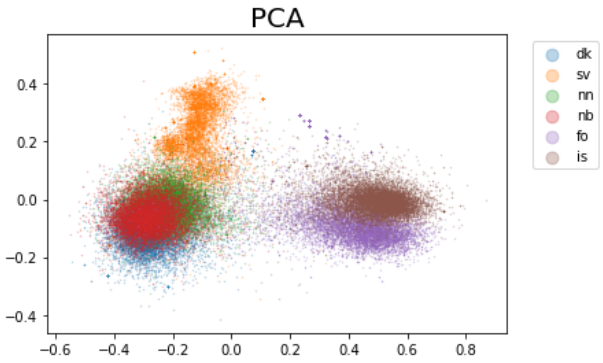
In the figure for encoding with character level bi-grams the PCA algorithm resulted in two elongated clusters. Without giving any prior information about the language of each sentences the PCA is apparently able to discriminate between Danish, Swedish, Nynorsk and Bokmål on one side and Faroese and Icelandic on the other since the majority of the sentences in each language belong to either of these two clusters. With the FastText implementations we observe three clusters.

For both cbow and skipgram we see a distinct cluster of Sweedish sentences. When comparing the two FastText models we see that the t-SNE algorithm with skipgrams seems to be able to separate the Faroese and Icelandic data points to a high d ecree compared with the cbow model. Also for the cluster identified with the sentences with Danish, Bokmål and Nynorsk the skipgram models seem seem to give a better separation, however to a lesser degree than with the two former languages.

(a) Character bigram



(b) Fasttext cbow



(c) Fasttext skipgram

Figure 8: Dimensionality reduction using PCA

t-SNE The T-distributed Stochastic Neighbor Embedding method was first proposed in 2008 in the paper "Visualizing Data using t-SNE"(van der Maaten and Hinton, 2008).
In the paper the authors explain the theory behind the algorithm which I will make a brief summary of here.
Suppose you pick a data point x_i, then the probability of picking another data point x_j as a neighbor to x_i is given by

$$p_{ji} = \frac{\exp(||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(||x_i - x_k||^2/2\sigma_i^2)} \tag{4}$$

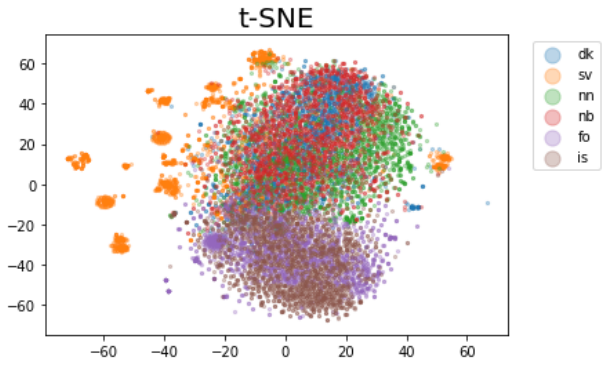
Now having this probability distribution the goal is to find the low-dimensional mapping of the data points x_i which we denote y_i follow a similar distribution. To solve what is referred to as the "crowding problem" the t-SNE algorithm uses the Student t-distribution which is given by

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k\neq l}(1 + ||y_k - y_l||^2)^{-1}} \tag{5}$$

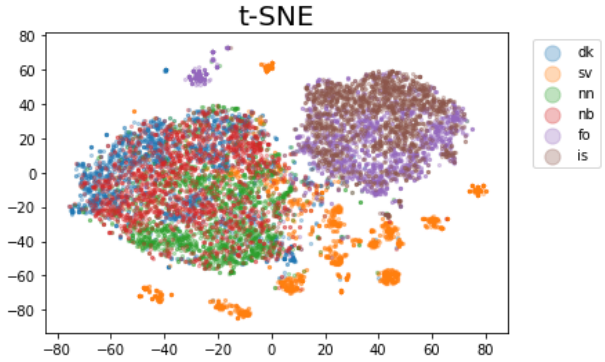
Now finally for optimizing this distribution is done by using gradient decent on the Kullback-Leibler divergence which is given by

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1} \quad (6)$$

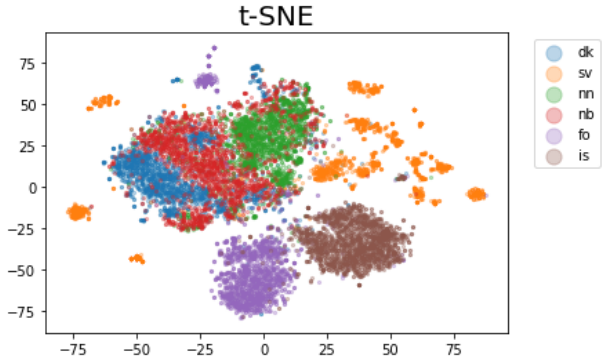
The result from running the t-SNE algorithm on the Wikipedia dataset can be seen in Figure 9. As was the case with the PCA algorithm it appears that the encoding with FastText seem to capture the most relevant information to discriminate between the languages, especially the skip-gram mode seems to do a good job in capturing relevant information.



(a) Character bi-gram



(b) FastText CBOW



(c) FastText skip-gram

Figure 9: Dimensionality reduction using t-SNE

Here we recover some interesting information about the similarity of the languages. The data points in Bokmål lies in between those in Danish and Nynorsk while Icelandic and Faroese have their own two clusters which are separated from the three former languages.

This is in good agreement with what we already know about the languages. Interestingly the Swedish data points and quite scattered and the t-SNE is not able to make a coherent Swedish cluster.

This does not however mean that the Swedish datapoint are not close in the original space. Some care is needed when interpreting the plot since t-SNE groups together data points such that neighboring points in the input space will tend to be neighbors in the low dimensional space.

If points are separated in input space, t-SNE would like to separate them in the low dimensional space however it does not care how far they are separated. So clusters that are far away in the low dimensional space are not necessarily far away in the input space.

7.2. Discussion

We used the dimensionality reduction techniques PCA and t-SNE to make visualizations of feature vectors obtained by making a one-hot encoding with character bi-grams and with the two modes from FastText.

These unsupervised techniques was able to separate the sentences from Wikipedia into different clusters. Without any prior knowledge about the actual language of each sentence these techniques indicated that the six languages can be divided into three main language categories: (1) Danish Nynorsk Bokmål (2) Faroese Icelandic and (3) Swedish.

Generally the supervised models had the largest errors when discriminating between languages belonging to either of the language groups mentioned above.

For the "classical" models we saw that Logistic Regression and support vector machines achieved better performance than Knn and Naive Bayes, where the latter performed the worst. This was true in all cases irrespective of the method of feature extraction.

Additionally we saw that when we used feature vectors from the FastText skip-gram model the classification models achieved better results than when using either FastText CBOW or character n-grams.

Generally we saw that increasing the number of data points lead to better performance. When comparing the CNN with the "classical" models however the CNN performed better than any of the other models even when trained on less data points. In this way it seems that the CNN is able to learn more from less data when compared to the other models.

8. Conclusion

This paper presented research on the difficult task of distinguishing similar languages applied for the first time

to the Scandinavian context. To address this, we first created a datasets. We compared four different "classical" models: K nearest Neighbors, Logistic regression, Naive Bayes and a linear support vector machines with two neural network architectures: Multilayer perceptron and a convolutional neural network.

The two best performing models, FastText supervised and CNN, saw low performance when going off-domain. Using character n-grams as features instead of words increased the performance for the FastText supervised classifier. By also training FastText on the Tatoeba dataset as well as the Wikipedia dataset resulted in an additional increase in performance.

The data from this task will be released openly, as well as a CodaLab instance for others to participate directly.

9. References

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching word vectors with subword information.

Goutte, C., Léger, S., Malmasi, S., and Zampieri, M. (2016). Discriminating similar languages: Evaluations and explorations.

Jacovi, A., Sar Shalom, O., and Goldberg, Y. (2018). Understanding convolutional neural networks for text classification. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 56–65, Brussels, Belgium, November. Association for Computational Linguistics.

Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016). Bag of tricks for efficient text classification.

Loper, E. and Bird, S. (2002). Nltk: The natural language toolkit. In *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics*.

Lui, M. and Baldwin, T. (2012). Langid.py: An off-the-shelf language identification tool. In *Proceedings of the ACL 2012 System Demonstrations*, ACL '12, pages 25–30, Stroudsburg, PA, USA. Association for Computational Linguistics.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space.

van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605.

Zampieri, M., Tan, L., Ljubesic, N., and Tiedemann, J. (2014). A report on the dsl shared task 2014. In *VarDial@COLING*.

Zampieri, M., Tan, L., Ljubešic, N., Tiedemann, J., and Nakov, P. (2015). Overview of the dsl shared task 2015. In *Joint Workshop on Language Technology for Closely Related Languages, Varieties and Dialects*, page 1.