Report 1: Language Detection

Alex Martorell i Locascio
Pim Schoolkate

March 2022

1 Introduction and Report structure

In this report, the first two sections give an overview of the problem, the algorithm and the structure of the data set. Section 4 proposes a detailed solution for the 1st exercise: It is a comprehension based task, as it is asked to interpret a confusion matrix and a PCA plot for the detected languages based on a previously defined vocabulary size (i.e. 1000 most common characters or words in the data set). The objective is to compare the results of language detection considering that the vocabulary or unigram vector a 1000 elements. Also included in Section 4 and Section 5 is the solution to Exercise 2, explaining the different preprocessing techinques applied as well as alternative classifiers to Naïve Bayes.

2 Algorithm

The algorithm for detecting the language works in two steps. First, a unigram vector is created, which in each element represents the amount of times the associated unigram is found in the corpus. This unigram can, by user definition, be either a word or a character. It then takes a subset of this vector which subsumes a defined amount of most common unigrams. Second, a classification algorithm is trained using this unigram vector and the provided training data. This results in a classification algorithm that, given a sentence, computes the probability of that sentence belonging to each language. Note that however, only meaningful probabilities for sentences that share unigrams with unigram vector can be computed. By default, this classifier is a Naive Bayes classifier, but can also be interchanged with other classification algorithms as described below.

3 Inspection of Data

In order to be able to interpret the results, the nature of the data has to be considered. For each language the dataset contains a 1000 observations, meaning that the complete dataset accounts for 22000 observations. This also implies that the data is balanced and thus when considering F1-scores, macro, micro, and weighted scores should provide similar results and thus can be interpreted equally. However, as the data is later randomly split in a training and test set, the weighted score will be used because of the introduced inbalances.

On closer inspection, the sentences in the corpus were found to be not perfect. For one, some of the sentences not in English, contain English sub-sentences or English loanwords. Secondly, some sentences are incorrectly labeled. For example, a few sentences containing English text are labeled as Latin. Because of these errors, care has to be taken when interpreting the results.

4 Exercise 1

The goal of this first exercise is to compare the language detection results. Two different possibilities are assessed: To train the data set with a vocabulary of 1000 words or 1000 characters.

4.1 Characters

In this section, the language detection based on characters is discussed. First, the base case with a 1000 characters is discussed, including the results and it's implications. After, the analysis with 6816 characters is discussed and analyzed. Both use a Naive Bayes classifier.

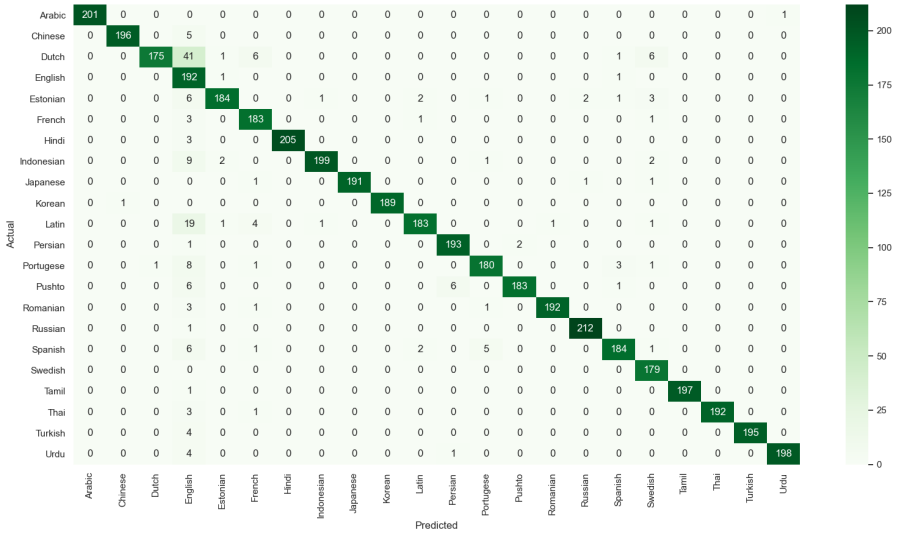

Figure 4.1: Confusion matrix of prediction of languages using a 1000 unigrams of characters

When analyzing a unigram with length 1000 and without any modifications of the code, the Naive Bayes classifier achieves an weighted F1-score of 0.95741. The amount coverage of the vocabulary is 98.1%, meaning that nearly all characters in the corpus are features in the vocabulary. The accompanying confusion matrix as can be seen in figure 4.1, shows that the algorithm does fairly well. However, it seems to sometimes misclassify Dutch as English. One good reason for this, is the similarity between the English and Dutch language, together with Scandinavian language, German and Frysian, all stemming from the same Germanic language. One of the similarities of Dutch and English (which sets it apart from German) is the omission of the High German consonant shift, which was the process of sounds like "p" changing into "f" or "pf" as in the case with apple (English), appel (Dutch), and Apfel (German). Another, is the near absence of the Germanic umlaut in both languages with the rare exception of Dutch in cases of the plural of words ending in double "e", like "zee" (ocean), with plural "zeeën" (oceans), or "idee" (idea) with plural "ideeën". Although more similarities (and dissimilarities) exist, these two are most important for considering the misclassification of the language in the model.

Another misclassification is observed again when predicting English as Latin. As discussed in the data description however, this is largely due to the errors in the dataset, and thus should not be a major reason for concern.

The Principle Component Analysis (PCA) plot (see figure 4.2) of the reduced dimensions based on

the train set, including the test data points, shows that, more or less, three categories of languages can be distinguished. First, it can be observed that the first dimension can be interpreted as the difference between languages which use the Latin alphabet and languages which do not. This dimension is able to explain 31.31% of the variance in the data. The second dimension is harder to interpret in terms of meaning, but does account for 13.8% explained variance. However, when only considering the Asian languages, it seems to be able, to some extend, distinguish the difference between them.

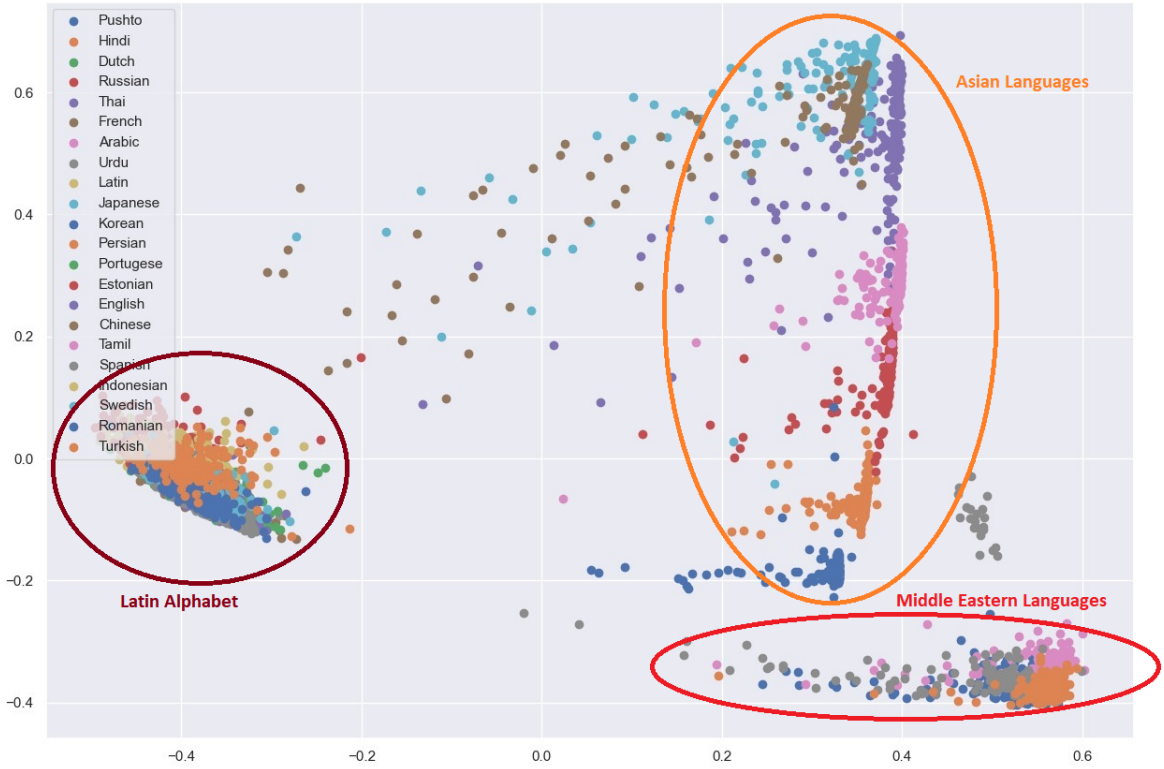

Figure 4.2: PCA of characters using a 1000 unigrams of characters

Comparing to analyzing 1000 unigrams, another analysis was performed using all 6818 characters as unigrams. As the coverage of the 1000 unigrams was 98.1%, the remaining 5818 unique characters only make up 1.9% of the corpus, meaning the net gain of information is little. Without any preprocessing or other modifications, the weighted F1-score yielded 0.92019, worse compared to using 1000 unigrams. When looking at the confusion matrix, it becomes clear that besides English, Dutch now also gets confused with Swedish. Thus the inclusion of more unigrams does not seem to benefit the results of the algorithm. Rather, it appears that certain unigrams, previously not included in the unigrams vector, have significant influence on the probability for the language being Swedish. An example could be the previously discussed "ë". Besides, due to the increase in unigrams, the influence on the probability of a language has decreased, meaning that previously influential unigrams are now overshadowed by the additional unigrams.

4.2 Words

Instead of individual characters, the unigram vector is built using most common words.

A look at the confusion matrix displayed in 4.3 shows worse results than if the unigram vector contains characters. This is further supported by the weighted F1-score of 0.88457. First of all, it is

observed that many sentences are predicted as English when they are actually not. The explanation for this is that many sentences contain a few English words as a sub-sentence. Another reason is that they are mistakenly labeled as English beforehand. Results for Chinese, Japanese and Korean are not good, because the algorithm is incapable of classifying most of the sentences in that language, labelling a lot of them as Swedish. Part of the reason behind this is that the vocabulary contains very few words of reference for these languages.

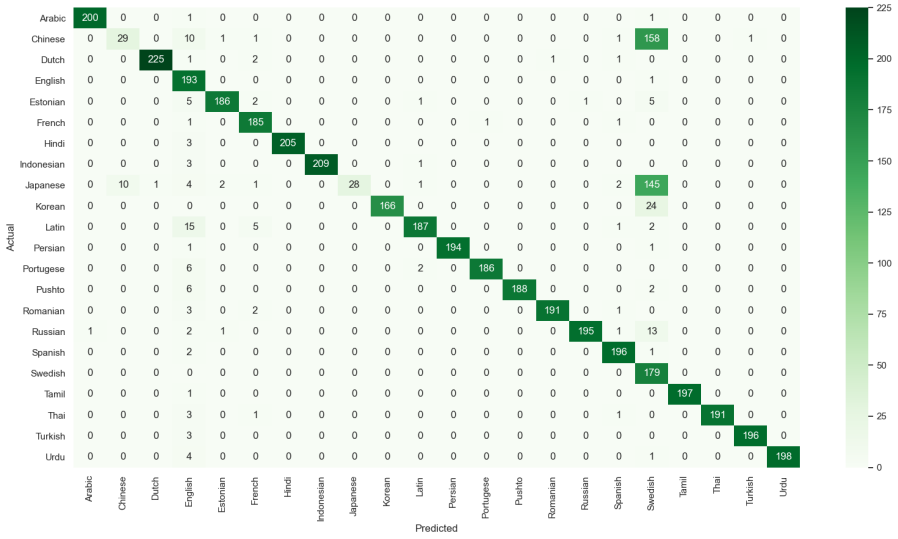

Figure 4.3: Confusion matrix of prediction of languages using a 1000 unigrams of words

In order to verify this, for each unigram in the unigram vector, the language with the highest probability is taken as "the most associated language" for that unigram. By counting how many times a certain language is associated with a unigram, it is verified that Chinese, Japanese, and Korean are under represented, versus, for example, Tamil and Hindi. Linguistically, although these results heavily rely on the data set provided (which is per se biased), we can say that since East-Asian languages used a character-based writing system, characters are repeated a lot less and therefore do not show in the vocabulary tally. Figure 4.4 shows a visual perspective on the amount of representation for each language. For comparison, the same graph was made for characters and can be found in the appendix 8.1.

A more in-depth analysis of the actual vocabulary words showed that out of the 11 chinese words, 8 were actually misclassified, showing a substancial bias in our dataset and one of the possible reasons why sentences in East-Asian languages are wrongly predicted: Not only does the vocabulary have few words of the aforementioned languages to provide accurate predictions, but some of them are not even actual words.

Thus when provided with Chinese, Japanese, or Korean sentences, the sentences do not contain words that are in the vocabulary, meaning that the classifier basically has to guess. One would expect that, when guessing, the wrong predictions are equally distributed over the other languages. But this is not the case, as by default it seems to prefer Swedish. After re-running the analysis with a different seed, this preference shifted to a different language, meaning that this bias is likely a minuscule difference in probabilities of all languages when no word corresponds to one in the vocabulary, which means that one, in this case Swedish always has the "highest" probability.

The PCA plot (see figure 4.5 of the reduced dimensions based on the train set, including the test

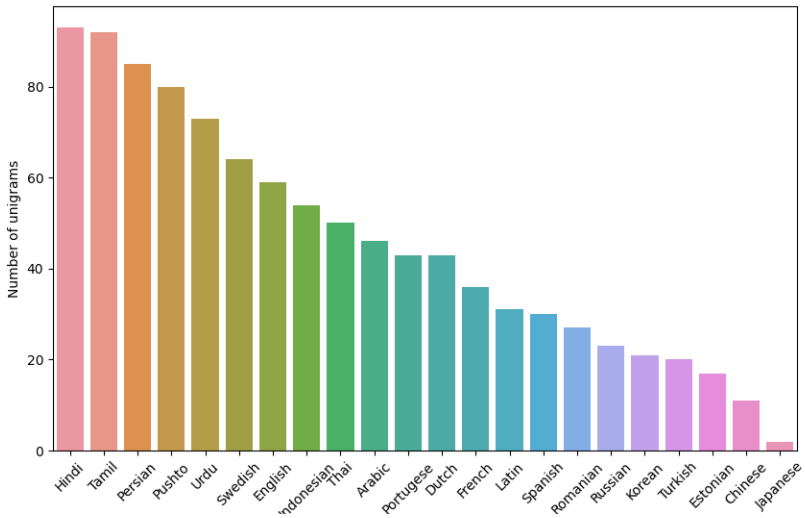

Figure 4.4: Number of words in the vocabulary per language.

data points is not able to capture the explained variance well, as the first dimension only accounts for 7.88%, and the second only 3.64%. Some kind or relation seems to be present on both axis (namely, the points are spread in some pattern), but due to the low explainability of the dimensions, no conclusions can be drawn from this plot.

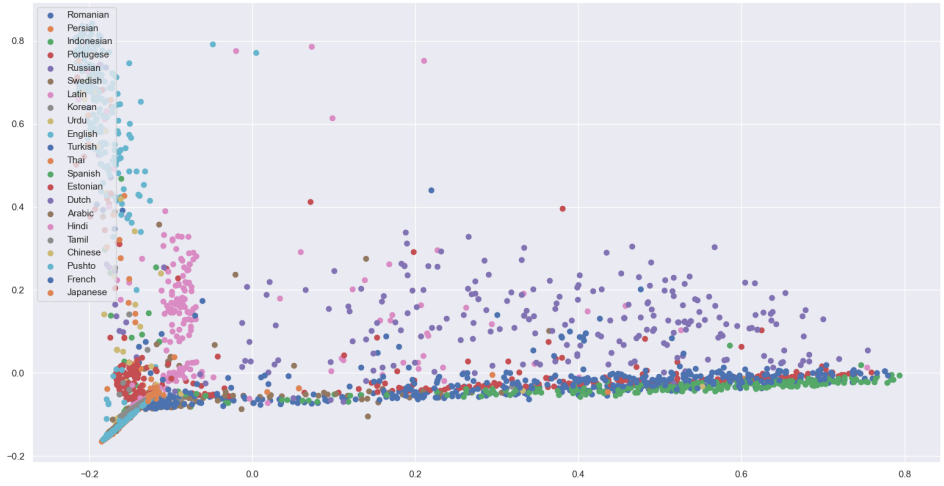

Figure 4.5: PCA of characters using a 1000 unigrams of words

The quantity of words in the vocabulary was increased to 25000 to see how this would effect the F1-scores and the coverage. The coverage of the vocabulary compared to the case of a 1000 words doubled and and now accounts for 52.0%. The weighted F1-score yields 0.91909, a slight improvement

over the 1000 words, but still significantly worse than using characters as unigrams. The coverage is still significantly lower than for characters. A worsening of explanability in the PCA dimensions was also found. In conclusion, character-based language detection yields better results in when using the Naive Bayes classifier, mainly due to the substantial difference in coverage.

5 Preprocessing

For this project, five preprocessing techniques were considered, of which the method and the individual results will be discussed first. Please note that the implications and limitations of these methods are discussed in their dedicated chapters.

5.1 Lowercases

A very simple and straightforward technique is to change all characters to lower cases. Upon inspection of the data, no capital letters were found, rendering this step nearly useless. However, it is still done, just in case some exceptions are present in the data.

5.2 Punctuation

Another uncomplicated technique is to remove all punctuation. This could be meaningful, however, as the count vectorizer function from the sklearn package does this already, doing it seperately does not make much sense.

5.3 Stemming

Stemming was considered as a valuable option as it would be able to dramatically reduce the variability of the words in the sentences. Most language specific stemmers depend on the language of the text, thus, as the objective is to predict the language, these can't be used. Therefore, it was opted to use the Porter stemmer, which works with certain heuristics on the English language and translates positively to other languages. In general, what the Porter algorithm does is to reduce similar words like "walks", "walked", and "walking" to the base of these words "walk". Because this stemmer does not make use of some predefined vocabulary, the Porter stemmer is also applicable to other languages, even though it might not make sensible reductions as these reductions in most cases still reduce to a single base word. The major benefit of this, is that it reduces the plurality of words of languages that use the Latin alphabet, which should allow Asian languages (when using words as unigrams) to be more represented in the vocabulary of the Naive Bayes algorithm that is being used.

5.4 Stop words

The concept of *stop words* refers to words that do not add meaning to the text that is being analyzed (i.e. the, an, a... in English). An "official" list of stop words for each language does not exist, as it is a subjective matter and should be obtained by a statistical study. The NLTK corpus module provides a list of stop words for a few languages, although it does not cover all of the ones contained in the data set. Unfortunately, this does not improve or change the results at all: It is believed that stop words work better if the goal is to do tasks such as sentiment analysis, where stop words become void of meaning.

5.5 Lemmatization

Similar to stemming, lemmatization searches for the root form of a word, but does so in a different way. Instead of, in the case of the Porter stemmer, using a heuristic to find a root form of a word, a lemmatizer makes use of a predefined corpus of words which it uses to find the root node. Thus, in order to apply lemmatization, the language of the text needs to be known. However, as the preprocessing of the X data is not able to use any Y data, this method is not applicable to this problem. Knowing this full well, it was still chosen to apply the English WordNet lemmatizer to all sentences, just like with the Porter Stemmer, in order to see what effect this would have on the results, and in general how lemmatization works in practice.

5.6 Preprocessing results

Table 1 shows the results once the three preprocessing steps detailed above are applied considering that the vocabulary is an array of 1000 words. As it is easy to see, preprocessing steps have barely no effect on the coverage or the F1 score. The hypothesised increase in coverage from the stemming did not come to fruition, which could be a possible explanation for the poor increase in performance. A possible hypothesis to why this happens is that the sentences are already quite clean, as well as the size of the vocabulary, which contains the more common 1000 words across the 22 languages.

Legend:
S= Stemming, **L**= Lemmatization, **SW** = Stopwords

	None	S	L	SW
Number of unigrams	1000	1000	1000	1000
Coverage	0.2577	**0.2674**	0.2622	0.2587
Micro F1 score	**0.8921**	0.8916	0.8918	0.8918
Macro F1 score	0.8812	0.8811	**0.8813**	0.8813
Weighted F1 score	0.8846	0.8845	**0.8847**	0.8846
PCA 1	7.88%	8.054%	**8.24%**	7.88%
PCA 2	3.64%	3.52%	3.64%	**3.65%**

Table 1: Preprocessing results with a vocabulary of 1000 words. Best results are highlighted in black.

6 Classifiers: Experiments and results

6.1 Naïve Bayes

The Naïve Bayes classifier works based on the statistical likelihood of a unigram belonging to a specific language. Thus, for each element of the unigram vector, n amount of probabilities are computed, n being the length of L, the languages in the corpus. The computation for each of the probabilities is done as follows:

$$P(L_i = l|W_j = w) = \frac{\#(L_i = l|W_j = w)}{\#(W_j = w)}$$

Where W_j is the jth unigram in the unigram vector, and L_i is the ith language.

Thus, when predicting, for a given sentence, if it contains one or more unigrams which are in the unigram vector, it sums the probabilities for each language and chooses the language with the highest probability.

6.2 Linear Support Vector Classifier

A more complicated classification method, the Linear Support Vector Classifier (LSVC) builds the optimal line or hyperplane that allows for classification of data points in ^{p} into classes. In this Language detection example, observe that our data points cannot be plotted intuitively because the number of dimensions is high. As a matter of fact, each data point contains the probability of that sentence belonging to one of the 22 languages. ($d = 22$). The term linear refers to kernel function. Non linear kernels are considered for points that cannot be separated in a finite dimensional space, but this is not the case.

6.3 Decision Tree

As a classification method, the tree is constructed based on the training set and the leaves are the 22 languages in the data set. The different nodes are the 1000 words from the vocabulary, each with its corresponding Gini impurity:

$$G_i = 1 - \sum_{i=0}^{c-1} [P(i \mid N)]^2$$

where $P(i|N)$ is the probability of belonging to class i for each node. Note that each node puts a and c is the number of classes (In this case $c = 22$). In other words, it displays the probability of a misclassification. The node with lowest Gini impurity is selected to become the root of the tree, and so forth. Decision trees are often likely to overfit, and thus care should be taken when interpreting the results. In this report, no further attention will be paid to this.

6.4 Multiple layer Perceptron

The Multiple Layer Perceptron, from now on Perceptron, builds a shallow hidden neural network, with one hidden layer. The input layer has an equal number of nodes to the amount of unique unigrams in the unigram vector. The output layer has n amount of nodes, one for each of the languages in the corpus, and computes the softmax equation, meaning that the results of this layer can be interpreted as a likelihood of a language being the correct one. The hidden layer, in this case was chosen to have 100 nodes and use the Rectified Linear Unit activation function. The Perceptron was trained with the Adam optimizer, without any regularization.

6.5 K-Nearest Neighbor

The function computes K-nearest neighbors with $k = 5$ as a default value, meaning it decides if a sentence belongs to a language or another based on majority voting of the 5 nearest neighbors. Note that the metric used is the Euclidean distance unless it is otherwise specified. In order to compute the nearest neighbors, it chooses between a few algorithms, comparing their performance.

6.6 Comparison of results

6.6.1 Classification results

Table 2 summarizes results obtained for different classification Algorithms. As can be seen nearly all classifiers outperform the Naive Bayes classifier (see bold weighted F1-scores), except for the decision tree and k-nearest neighbour with regards to characters. This was to be expected, as Naive Bayes is a simple (yet extremely powerful) technique. The best classifier turns out to be the Perceptron. As this is a neural network, it can be improved in terms of architecture, depth, and activation functions,

however, the big downside is that it is, and will remain, a black box that will not allow to explain the inaccuracies that it still generates.

Legend:
NB= Naive Bayes, **LSVC**= Linear Support Vector Classifier, **DT** = Decision Trees, **MLP** = Multilayer Perceptron, **KNN** = K-nearest neighbors, **RF** = Random Forest

	NB	NB	LSVC	LSVC	DT	DT	MLP	MLP	KNN	KNN
	Words	Char	Words	Char	Words	Char	Words	Char	Words	Char
N.o.Uni	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
Mic. F1	0.89204	0.95522	0.92978	0.96568	0.90386	0.9352	0.9252	0.9722	0.9097	0.9543
Mac. F1	0.8812	0.95783	0.9256	0.96633	0.90163	0.9355	0.92128	0.972699	0.908	0.955
W. F1	0.88458	0.95741	**0.926**	**0.9662**	**0.902136**	0.93493	**0.922**	**0.97258**	**0.9091**	0.95496

Table 2: Results based on classification algorithm. Best results are highlighted in black

7 Conclusions

One of the main insights gained while working on this task was to acknowledge the myriad of preprocessing techniques and classifying algorithms that can be applied in order to attempt to improve the language detection problem. By construction of the dataset, it was clearly proven that at least for all the preprocessing techniques that were tried, no significant improvements were obtained. This seems to happen because the data set is already quite prepared for this problem, and the sentences were clean enough to not need sophisticated preprocessing. The main conclusion to obtain from the first exercise is that detection by character provides much better coverage and F1 scores than detection through words due to lower variety of characters versus the high variety of words.

The comparison to other classifiers showed that choosing the proper tool for job in this case is important. In cases of very little data, Naive Bayes is said to outperform more complicated algorithms, but since the data provided was abundant enough, a different option should be chosen. In the case of this report, this turned out to be a Neural network, based on the metrics used. However, depending on requirements, a more explainable algorithm can be used.

8 Appendix

8.1 Code

8.1.1 preprocessing.py

```
def preprocess(sentences, labels, stemming=True, lemmatizing=True, remove_stopwords=
    True):

    preprocessed = pd.Series([])
    stop_words = []

    if remove_stopwords == "True":
        stop_words = generateStopwords(labels)

    for sentence in sentences:
        sentence = removePunctuation(sentence)
        sentence = lowerCharacters(sentence)

        if remove_stopwords == "True":
```

```
            removeStopwords(sentence, stop_words)

        if stemming == "True":
            stemmer = getPorterStemmer()
            sentence = stemSentence(sentence, stemmer)

        if lemmatizing == "True":
            lemmatizer = getWordNetLemmatizer()
            sentence = lemmatizeSentence(sentence, lemmatizer)

        sentence = pd.Series([sentence])
        preprocessed = pd.concat([preprocessed, sentence], ignore_index=True)
    return preprocessed,labels

def lowerCharacters(sentence):
    return sentence.lower()

def removePunctuation(sentence):
    return sentence.translate(str.maketrans('', '', string.punctuation))

def getWordNetLemmatizer():
    return nltk.stem.WordNetLemmatizer()

def tokenizeSentence(sentence):
    return nltk.tokenize.word_tokenize(sentence)

def getPorterStemmer():
    return nltk.stem.PorterStemmer()

def generateStopwords(labels):
    unique_labels = labels.unique()
    for i in range(len(unique_labels)):
        unique_labels[i] = unique_labels[i].lower()

    common_languages = list(set(unique_labels).intersection(nltk.corpus.stopwords.
    fileids()))
    stop_words = []
    for i in range(len(common_languages)):
        stop_words.extend(set(nltk.corpus.stopwords.words(common_languages[i])))
    return stop_words

def removeStopwords(sentence, stop_words):
    word_tokens = tokenizeSentence(sentence)
    filtered_sentence = [w for w in word_tokens if not w in stop_words]
    return "".join(filtered_sentence)

def stemSentence(sentence, stemmer):
    token_words = tokenizeSentence(sentence)
    stem_sentence = []
    for word in token_words:
        stem_sentence.append(stemmer.stem(word))
        stem_sentence.append(" ")
    return "".join(stem_sentence)

def lemmatizeSentence(sentence, lemmatizer):
    token_words = tokenizeSentence(sentence)
    lemma_sentence = []
    for word in token_words:
        lemma_sentence.append(lemmatizer.lemmatize(word))
        lemma_sentence.append(" ")
    return "".join(lemma_sentence)
```

8.1.2 classifiers.py

```
def vocabulary_dist(X_train, y_train, X_test):
```

```
    print("========")
    print("Computing vocabulary distribution:")
    print("========")
    trainArray = toNumpyArray(X_train)

    clf = MultinomialNB()
    clf.fit(trainArray, y_train)
    y = clf.feature_log_prob_
    ylab = np.reshape(clf.classes_, (22,1))

    lang_column = np.reshape(np.array([i for i in range(0,22)]), (22,1))
    xmax = np.argmax(y,axis=0)
    lang = np.concatenate((ylab,lang_column), axis=1)

    vocab_count = np.bincount(xmax)
    df = pd.DataFrame({"Language" : lang[:, 0].tolist(), "Number of unigrams":
    vocab_count.tolist()})
    df=df.sort_values(by= ["Number of unigrams"], ascending=False)
    plt.figure(figsize=(10,6))
    # make barplot
    sns.barplot(x='Language', y="Number of unigrams", data=df)
    plt.xticks(rotation=45)
    plt.show()
```

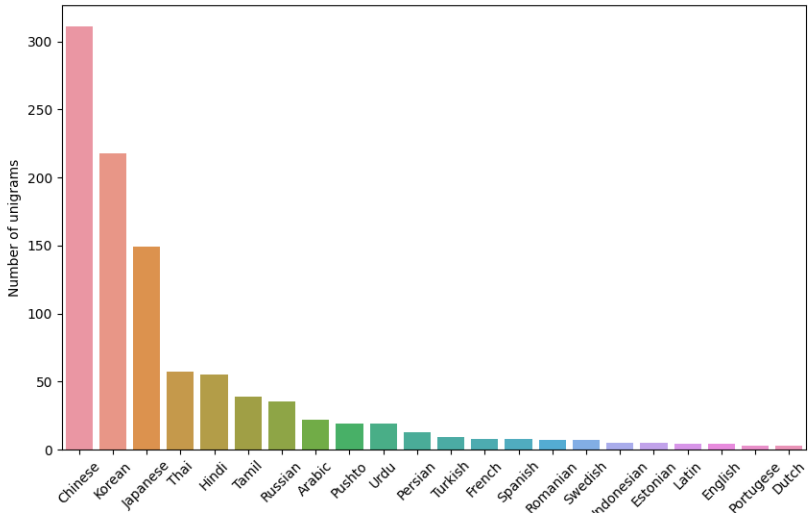

Figure 8.1: Number of chars in the vocabulary per language.