Music Genre Classification from Lyrics
A multilingual perspective

Alice Perin
a.perin2@studenti.unipi.it
student ID: 626005

ABSTRACT

In this paper I present the data, methods, and results of my Text Analytics project: Music genre classification from lyrics. I performed classification tasks comparing traditional machine learning, neural networks, and transformer based methods.

1 INTRODUCTION

The foundation of the project is the "Multi-Lingual Lyrics for Genre Classification" dataset from Kaggle[1]. It provides almost 250 thousand lyrics in 33 different languages, of 10 different music genres. The goal of the project is to use different machine learning and deep learning methods for classification.

I handled the work as follows:
I started importing the whole dataset in the first notebook to understand the composition of the data and to filter the 7 most frequent languages (English, Portuguese, Spanish, Italian, Indonesian, French, and German) and analyze them separately.
Then, in another notebook, I have randomly selected 10 thousand rows for the English dataset and 10 thousand for Portuguese, creating 2 new (final) datasets.
I worked on each of the 7 language-specific datasets, using various machine learning and deep learning tools to classify the music genre label for the lyrics. And finally, I have used a transformer-based neural network (BERT) on the whole multilingual dataset for the same task.

The following notebooks' schema is to understand the structure of the work:

`Music_genre_classification` → main file, dataset splitting

`Random_10K_samples_df` → performs the random selections for the two largest datasets (English and Portuguese)

`df_language_song_genre_classification` → each file contains the classification tasks for one language

`language_3genres_classification` → each file contains the classification for just 3 of the most frequent genres per language

`BERT_multilanguage_classification`
`BERT_multilanguage_classification_3genres`
`BERT_multilanguage_classification_3languages` → perform the classification of the multilingual dataset

[1]https://www.kaggle.com/datasets/mateibejan/multilingual-lyrics-for-genre-classification?select=test.csv

2 DATA UNDERSTANDING

Notebook:
`Music_genre_classification`

Before starting to process the data, it is a good practice to clean and analyze the data you are going to work on. So I imported the .csv file containing the dataset and explored its shape, its variables and the content of the 'Genre', 'Language', and 'Lyrics' columns.
I immediately noticed the unbalance of the Genre distribution as there are a couple of labels that are much more consistent than the others, as it can be observed in table 1.

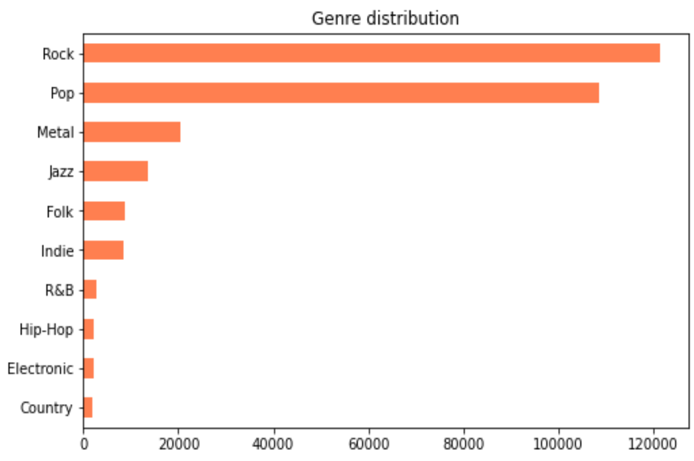


Table 1 Genre Distribution in the whole dataset

The unbalance stretches over the Language distribution as well, making me choose to work separately on the 7 most frequent languages, except for Romanian because its lyrics were unusable (below also the number of their values):

```
English           250197
Portuguese        30102
Spanish           3892
Romanian          1184 (not used)
Italian           808
Indonesian        737
French            644
German            478
```

After, I moved forward to the cleaning process. I removed the NaN values in the rows, used regular expressions to replace symbols, numbers and double spaces with single spaces and reset the index column.

Finally, I filtered the dataset creating 7 datasets: one for each of the selected languages.

Notebook: `Random_10K_samples_df`

This notebook, imports the datasets for English and Portuguese and randomly selects 10 thousand rows from each one, creating the two final datasets for these 2 languages. The goal here is to reduce the number of rows, in order to work with smaller datasets.

3 CLASSIFICATION WITH TRADITIONAL MACHINE LEARNING TOOLS

Folders:
`Single-language classification`, `single-language classification 3 genres`

These two folders contain all the single-language classification processing through some of the traditional machine learning tools, and two types of neural networks (explained in the next section).
Namely: SVM (with tokenization and with parameter optimization), Naïve Bayes, Decision Tree, and Random Forest. An overview of the results of this section is outlined in table 2 and table 3.

For each of the languages I have imported the related dataset, split it into training set and test set (80%-20%) with *scikit-learn*'s *train_test_split* function, tokenized and vectorized the lyrics with *spacy* (except for Indonesian language) and *CountVectorizer*. The tokenization and vectorization have been carried out using an ad-hoc function that lemmatizes the tokens (using *spacy*'s language-specific model) and combines the lemmas into bigrams and trigrams. This function is used as *CountVectorizer*'s analyzer in the vectorization part.

For Indonesian, I had to adopt a different library, because *spacy* does not support this language, and the multilanguage tokenizer didn't work. So, I chose the lemmatizer and tokenizer of the *nlp-id* module of Kumparan's NLP Services[2] and fit it into the vectorization function.
Once I have assigned the vectorized lyrics to the variable `x_train_tok`, I moved on to the core step of the classification task.

○ SVM – Support Vector Machine

The first machine learning algorithm I tested is SVM (Support Vector Machine), a supervised learning method for (also) classification tasks. I used *LinearSVC* as the algorithm's learner, that works well in large datasets and follows one-vs-the-rest scheme for multiclass classification. The weighting function is *TfidfTransformer*.
I tested the algorithm first with un-tokenized lyrics, then with tokenized lyrics (`x_train_tok`, `x_test_tok`) and then with tokenized lyrics and optimized parameters (*GridSearch*). The SVM results in table 2 are relative to the tokenized and optimized results.

○ Naïve Bayes

The second algorithm I used is Naïve Bayes, a probabilistic based on Bayes' theorem. I passed the tokenized lyrics and used *MultinomialNB* as learning algorithm for multiclass problems. In general, it doesn't perform well comparatively to SVM.

○ Decision Tree

The third tool is Decision Tree, which solves the classification problem by turning the data into a tree representation through the learning algorithm *DecisionTreeClassifier*.

○ Random Forest

The last one is Random Forest, which performs better than the other tools relatively to this dataset, as it can be observed in table 2.

To better understand how these tools work and how much the complexity of the dataset is important for the performance of the algorithm, I applied SVM and Random Forest on English, Portuguese, Spanish and

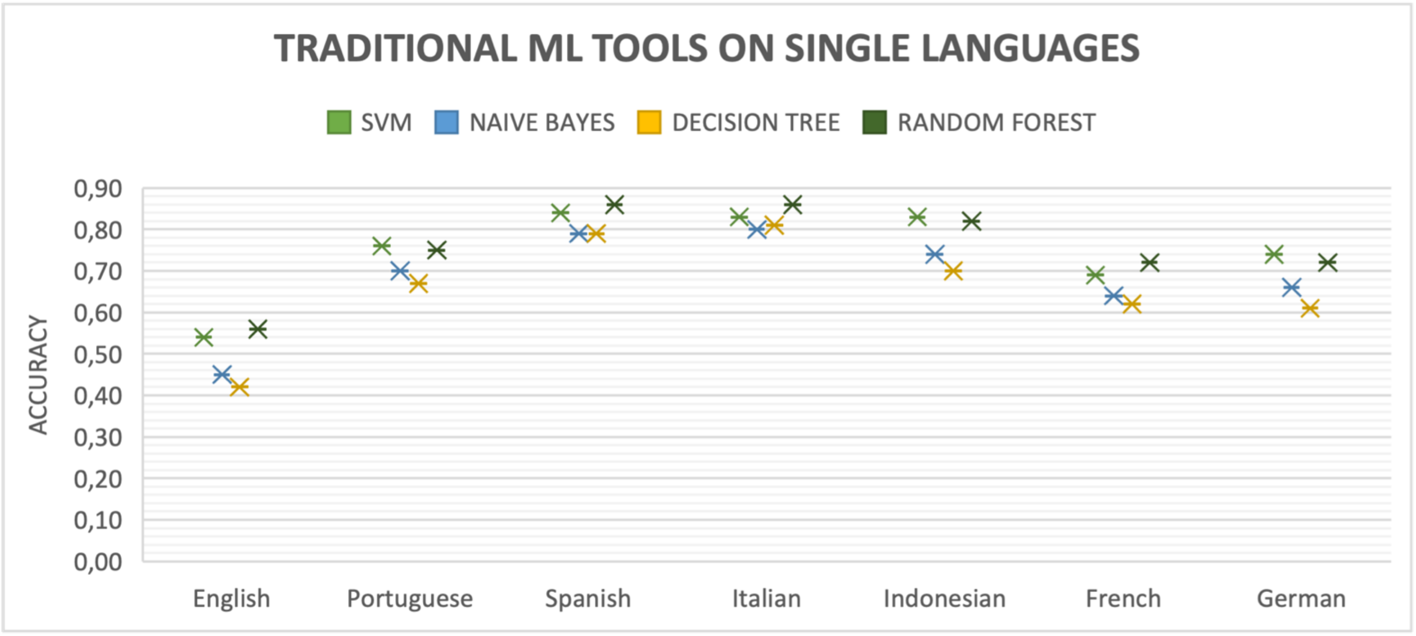


Table 2 Traditional ML Tools on Single Languages

[2] https://pypi.org/project/nlp-id/#description

Indonesian datasets making a comparison between the all-genres-dataset and a new 3-genres-dataset, extracted by filtering the original one. Each new notebook (see `single-language classification 3 genres`), processes the lyrics in the same way as described above, but only on the 3 most frequent music genres for each language.

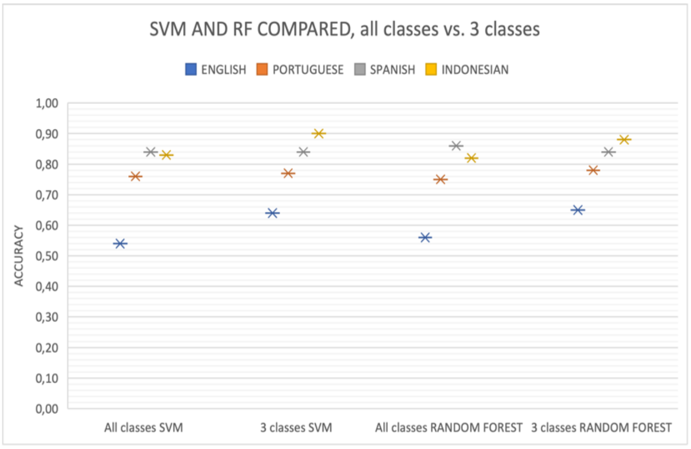


Table 3 SVM and RF compared, all classes vs. 3 classes

Although it does not make substantial improvements, the reduction of the classes from 10 for English, 8 for Portuguese, 9 for Spanish and 7 for Indonesian to only 3 labels have made a small, positive difference in the models' overall accuracy.

4 CLASSIFICATION WITH NEURAL NETWORKS

For this section, I have worked on the same notebooks of the previous part.
I chose to test the classification task on two kinds of neural network: LSTM (Long-Short Term Memory) and CNN (Convolutional Neural Network).
I decided to take into consideration two very different neural architectures, in order to test their effectiveness on textual data.
The pre-processing of the data consists in transforming the textual labels (Rock, Pop, Metal...) into numerical labels, for both the training set and test set. Then I have converted the numerical labels in one-hot vectors since they are required in input by neural networks.
After this, the lyrics assigned to `x_train` variable are converted into sequences of integers and padded to make sure all the sequences have the same length (maximum length: 200).

○ LSTM – Long Short Term Memory

As for LSTM, some languages have slight changes in the model's architecture but basically this is the combination of hyperparameters that I have adopted for this kind of network:

```
model = Sequential()
model.add (Embedding(20000, 128))
model.add (LSTM(20, dropout=0.4,
recurrent_dropout=0.4))
model.add (Dropout (0.4))
model.add (Dense(32,
activation='relu'))
model.add (Dropout(0.2))
model.add (Dense(number_of_labels,
activation='softmax'))
model.compile(loss='categorical_cross
entropy',optimizer=SGD(learning_rate=
0.01, momentum=0.8),
metrics=['accuracy'])
```

Batch size	64
Epochs	10
Validation split	0,2
Embedding size	128

Table 4 Plain LSTM hyperparameters

Only for English and Portuguese, I have used **FastText**. I used it just on two languages for a matter of memory usage.
I started with uploading the files which contain the pre-trained models: *crawl-300d-2M.vec* for English and cc.pt.300*.vec* for Portuguese.
The pre-processing differs in the creation of the embedding matrix using the uploaded embeddings, that are used as the initial

weights of the network. On top of that, I have trained the network.
The hyperparameters of this network are a little different from the plain one.
The Embedding layer is composed by:
- a matrix of `nb_words` (minimum number between the number of features and the length of the word index) and the fixed embedding size (fixed to 300 for FastText embeddings)
- the non-trainable weights of the network (`embedding_matrix`)

```
model.add(Embedding(nb_words,
embed_size, weights =
[embedding_matrix], trainable =
False))
```

Batch size	64
Epochs	10
Validation split	0,2
Embedding size	300

Table 5 FastText LSTM hyperparameters

Also, I have changed the hidden layer to 64 units (instead of 32).

Then, for every language I have trained a **Word2Vec** model and used it to fine-tune an LSTM. The pre-processing consists in importing *Word2Vec* model from *Gensim,* re-converting the string labels into integer labels, and split the whole dataset into sentences before creating a test and training set again. By receiving the sentences in input, the model creates a 300 size embedding for each word type, then used to create the embedding matrix. Then the words in the lyrics are encoded into integers and padded, and the labels are converted to one-hot vectors.

```
model=Sequential()
model.add(Embedding(input_dim=vocab_si
ze, output_dim=embed_dim,
input_length=max_lyrics_len, trainable
= False,
embeddings_initializer=Constant(embed_
matrix)))
model.add(LSTM(12, dropout=0.4,
recurrent_dropout=0.4))
model.add(Dropout(0.4))
model.add(Dense(24,
activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(number_of_labels,
activation='softmax'))
model.compile(loss='categorical_crosse
ntropy',
optimizer=SGD(learning_rate=0.01,
momentum=0.8), metrics=['accuracy'])
```

Batch size	64
Epochs	5
Validation split	0,2
Embedding size	300

Table 6 Word2Vec LSTM hyperparameters

As can be observed in table 7, LSTM with pre-trained FastText embeddings did not perform very well in general, but in the English case it worked a bit better than plain and Word2Vec versions. Keeping aside English and German with 0 accuracy on both LSTM's versions, the mean accuracy of plain LSTM is 0,70 (otherwise 0,50 with English and German) while the Word2Vec version performs 0,72 (otherwise 0,51) on average.

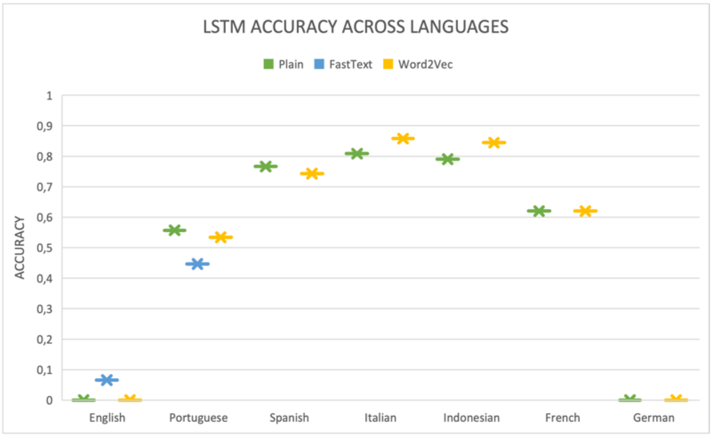


Table 7 LSTM Accuracy across Languages

○ CNN – Convolutional Neural Network

The second neural architecture I used is CNN, with the following hyperparameters:

`model = Sequential()`
`model.add(Embedding(20000, 128))`
`model.add(Dropout(0.4))`
`model.add(Conv1D(filters=10, kernel_size=3, padding='valid', activation='relu', strides=1))`
`model.add(GlobalMaxPooling1D())`
`#vanilla dense layer`
`model.add(Dense(10))`
`model.add(Dropout(0.4))`
`model.add(Activation('relu'))`
`#output layer`
`model.add(Dense(number_of_labels))`
`model.add(Activation('softmax'))`
`model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])`

Batch size	64
Epochs	20
Validation split	0,1
Embedding size	128

Table 8 Plain CNN hyperparameters

As I did with LSTM, I used **FastText** on English and Portuguese with the same hyperparameters as the plain CNN except for the Embedding layer, in which I added the embedding matrix (created also for the LSTM) to set the weights of the network. The embedding size is 300 as well.
On the other hand, in the **Word2Vec** version, I have increased the dimension of the hidden layer to 32 and added another layer of 32 neurons. Every other parameter remains unchanged.
Compared to LSTM results, CNN performs better across all languages. In general, the plain version works better than Word2Vec's, with an average of 0,62 for plain CNN and 0,48 for Word2Vec CNN.

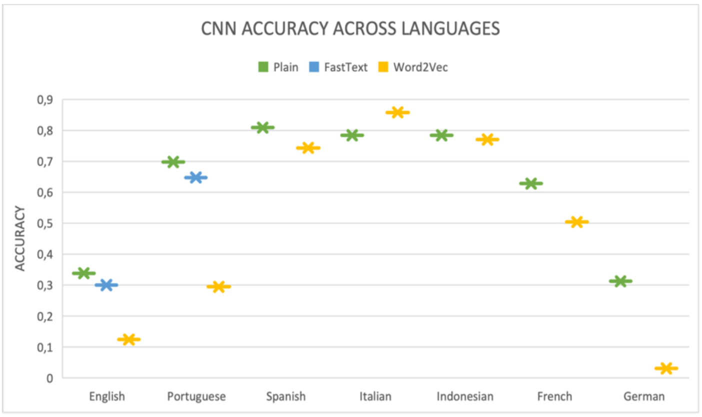


Table 9 CNN Accuracy across Languages

Also for this section I was curious about trying neural networks with only 3 genres.
In this case as well, I have observed slight changes: the major changes emerge in the English language, where the average delta in accuracy is 0,26.

Dataset	LSTM	LSTM FT	LSTM W2V	CNN	CNN FT	CNN W2V
ENGLISH 3-classes	0,4997	0,5224	0	0,244	0,5736	0,5487
ENGLISH all-classes	0	0,066	0	0,3385	0,3	0,124

Table 10 Comparison between English all-classes and three-classes NN accuracy

5 CLASSIFICATION WITH BERT

Folder: `BERT`

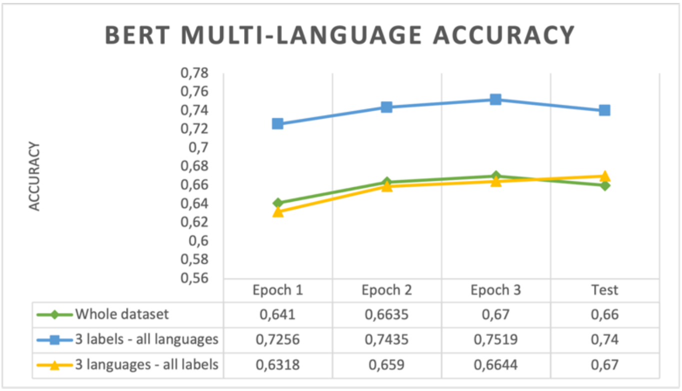
For the last part of the project, I have tested the whole 33-language and 10-genres dataset on a transformer based neural network: **BERT**. Other than the 7 languages already described, the dataset includes lyrics in: Swahili, Tagalog, Somali, Catalan, Turkish, Dutch, Slovak, Croatian, Norwegian, Slovenian, Afrikaans, Danish, Swedish, Estonian, Finnish, Polish, Czech, Hungarian, Vietnamese, Russian, Lithuanian, Latvian and Korean.
I chose to use the model *bert-base-multilingual-cased,* which is pre-trained on 104 languages.

The pre-processing consists in concatenating the special [CLS] and [SEP] tokens to the sequences, tokenizing them, creating the integer encodings and padding them to fixed lengths (128 for the training sequences, 512 for the test sequences). Then, after splitting the integer sequences into train and validation inputs, they are wrapped and converted into tensors.
These are the hyperparameters that I used for the training:

Batch size	32
Epochs	3
Weight decay	0,01
Learning rate	2e-5

Table 11 BERT hyperparameters

Like for the other tools, I have tested this BERT model on the 3 most frequent labels and then on the 3 most frequent languages. Table 12 explains that reducing the number of labels, BERT gets more accurate in the prediction, although 0,74 is not a relatively high score for accuracy.
On the other hand, the reduction of the number of languages, does not improve the test accuracy.



	Epoch 1	Epoch 2	Epoch 3	Test
Whole dataset	0,641	0,6635	0,67	0,66
3 labels - all languages	0,7256	0,7435	0,7519	0,74
3 languages - all labels	0,6318	0,659	0,6644	0,67

Table 12 BERT MultiLanguage accuracy

CONCLUSIONS

In this project I used various types of classifiers with the purpose of comparing them and figuring out which one works best with the dataset I chose to train them on.
The highest accuracy value is 0,9 using SVM for 3-labels Indonesian dataset. The lowest is 0, in English and German datasets using LSTM networks. Between these two extremes, the algorithm that performed better is on Italian language reaching 0,858 with a CNN fine-tuned on Word2Vec. Beside these, many other values were inconsistent.
A few considerations: in exploring the dataset, I noticed that many lyrics contained onomatopoeias, whole lines in languages other than the one indicated in the dataset, and other scattered noise that I could not eliminate because it was confusable with the lyric itself (e.g., the indication of the chords a, b, c, d...). In addition, the number of musical genres presented is very unbalanced (also within languages), thus making it difficult for the algorithms to learn and recognize some classes. Considering this, one can perhaps understand why the accuracy of the classification tools on this dataset is not that high. After all, data quality is important too.
To conclude, there is no best algorithm, but the high accuracy depends on many factors such as the combination of algorithm and language (e.g. neural networks do not perform well with English and German lyrics), and the balance in distribution of the labels to predict: that's why the highest accuracy in the 3-genres datasets.