Data Mining - Homework 4

Valerio Trenta - matr. 1856471

January 12, 2020

Where each network can be found:

- CNN with simple embedding (paragraph 2) → folder *simple_cnn*, file **cnnlyricsclassification.py**;
- BERT (paragraph 3) → folder *bert*, file **berttextclass.py**;
- CNN with word2vec embedding (paragraph 4) → folder *w2v_cnn*, file **cnnword2vecclasslyrics.py**;

1 Preprocessing

The **preprocessing** part is basically the same for every network, except the one which exploits the **BERT** model. With *preprocessing*, apart from the stop words removal and the tokenization of each lyric, we also mean the way the dataset is handled: as suggested by the hint that was provided in the text, the dataset we are going to exploit is heavily unbalanced due to many different reasons.
First, a little preprocessing is carried out when loading the dataset: we immediately discard every column that is useless to our target, so we only select the two columns *"lyrics"* and *"genre"*. Then, for each lyric, we lower the case of the words contained in it and immediately remove some special characters such as parenthesis which often recur in the lyrics, and the words *"intro"*, *"verse"* and *"chorus"*.
One may also notice that many lyrics are actually "empty" - meaning, the *lyrics* field for the song only contains a *"NaN"* value - and some other contain few words in it as if they were truncated so we filter the dataset by only retrieving in a dictionary the *"valid"* lyrics that may contribute to our classification task, that is to say the lyrics with at least **10** words.
We then carry out the preprocessing by removing the stop words and the punctuation and obtaining a "bag of tokens" for each lyric.
At this point, we still have a very large dataset containing more than 250.000 lyrics, and what we notice is that the dataset itself is still very unbalanced: for example, classes such as *"Electronic"* and *"Jazz"* only contain about 7.000 lyrics, while *"Rock"* contains more than 100.000 lyrics.

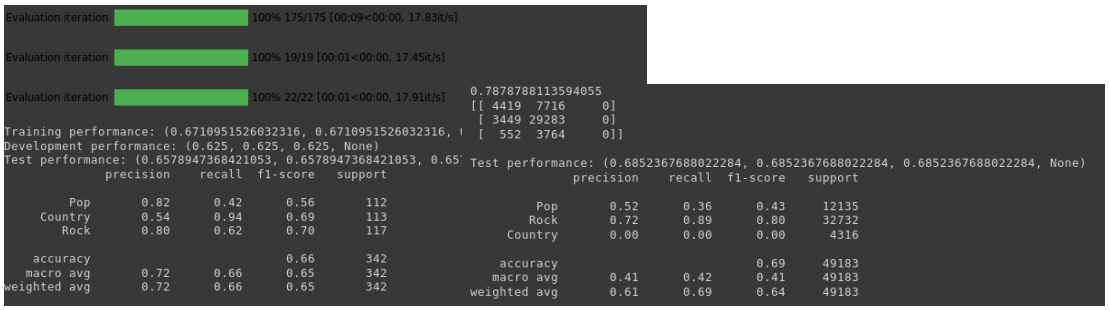


Figure 1: From left to right, results from BERT and the Word2Vec CNN implementation with 3 classes and extremely unbalanced dataset.

To carry out a classification on a balanced dataset, we decide to proceed as follows: we select five out of the twelve classes that we originally had, excluding a priori *"Not Available"* and *"Other"* since they would be pretty useless for obvious reasons. So we decide to pick these five classes: *"Rock"* (about 100.000 lyrics), *"Hip-Hop"* (about 25.000 lyrics), *"Electronic"* (about 7.000 lyrics), *"Jazz"* (about 7.000 lyrics) and *"Country"* (about 14.000 lyrics).
Now, for each of the classes we only select exactly 1.500 lyrics, so that each of them has the same number of samples and we have a dataset with a total of 7.500 songs.
Then, the dataset we obtained is split into training, development (only used in the BERT model, otherwise joined with training) and test sets according to the **sklearn** library (10% for test, the remaining for training and development).
Both the CNN with word2vec embedding and the BERT model that we are going to see in the next paragraphs have been exploited to provide an example which should clarify why this dataset needs to be balanced as aforementioned. Both the networks took as input lyrics from three arbitrarily selected classes: *Pop*, *Rock* and *Country*, with the first one having about 40.000 samples, the second one about 100.000 and the latter 14.000. As we expected, both the models have encountered huge problems when dealing with the classification of the **Country** lyrics, because they are actually obfuscated by the huge amount of samples of the other two classes: the CNN with the word2vec embedding couldn't actually classify lyrics from the *Country* class (see confusion matrix, third column, and precision/recall values to zero), while the BERT model, even if actually more accurate than the first one, shows by the results in terms of precision metric that lyrics from the *Country* class were much harder to classify with respect to the lyrics from the other two classes which had much more samples, because half of the elements classified as "Country" were actually elements from other classes (see figure above).

2 Exercise 1.1: simple CNN

We are now going to design an **embedding layer** through which we are going to embed (encode) the words the network is going to process and the layers - the ones performing the cross-correlation operation - that are needed to process the inputs. Embedding is a crucial and fundamental part, since a **Convolutional Neural Network** cannot process directly words, but needs numerical values - vectors - to actually perform its task. The following one is a naive implementation in terms of embedding - we are going to improve this part - and also in terms of network exploited, since it's composed of a single convolutional layer. The simplest way to encode the words for each lyric is by picturing each word as a vector with **C** components, with C = number of classes we have, 5 in our case. So if word **x** belongs to a lyric in the second class and to another lyric in the fourth class, its representation will be **x** = [0 1 0 1 0]. Thus, we build up a vocabulary where we store each word appearing in our corpus only once and encode it this way. Then, in order to provide this vocabulary as a lookup-table - the vectors of the vocabulary will be then "freezed", meaning they won't change as the network processes the inputs - to the **embedding layer** of our network, we transform it into a matrix where each row represents a different word as its corresponding vector. We represent each song in our dataset by replacing each token with the corresponding index of the row in the vocabulary, so each lyric will be composed of a list of indexes that will match against the embedding layer and thus will be turned into the vector representing the word. Labels are not one-hot encoded, PyTorch functions only need the index (an integer) for each class.

This way of representing our corpus is very simple, but also not so efficient: of course, if a lyric contains a word that only occurs in one of the five classes, then it will be easy to classify the lyric in this single class, but if the lyric doesn't have an "unique" word, then the classification is going to be very hard, and the network is likely to fail it, as the results seem to suggest. So, we know that this embedding is actually **not** the best possible - actually, it is likely to be one of the worst possible - since we are losing *a lot* of information: we only know whether a word from a lyric is exploited or not in one or more of the five genres, but that's it: we know nothing about the context of the word nor about its semantic relationship with other words. An implementation exploiting tf-idf is likely to be much more accurate, and we should expect this implementation to have poor results, but this was the first step I decided to take in order to better understand how to embed the words for the CNN and how to build up the CNN itself.

Since not every song has the same number of words, we established a prefixed length of **100**, so that shorter songs have been padded with zeros to reach it and longer songs have been cut. This was necessary in order to provide the songs to the neural network with the same length.

The network that is exploited for this task has just one convolutional layer with a single input channel, **100** output channels and a kernel of the same size of the size of the output and vectors embedding the words in our vocabulary (so (5x5) in our case), meaning the convolutional layer will slide on the **100** words of each

lyric by considering five of them at a time (just like a 5-gram model over the words), and so, since it performs a one-dimensional convolution (one channel as input), the output of this operation will once again be a vector of size 100, so the fully connected layer will have as input 100 max-pooled values, one value from each of the 100 output channels, that will be computed after applying max-pooling to the vector of size 100 coming from the convolution. Dropout is only applied when the network is training itself.
The activation function for the convolutional layer is a **ReLU**, which simply turns the values found in the convolutional layer for each group of five words to zero if they are negative, while it will maintain the original value if it is positive. The final layer is fully connected with all the 100 output channels of the convolutional layer and has an output of size 5, that is the number of the classes: indeed, the target of this final layer is to produce the scores for each of the five classes. At this point, the activation function for the output layer is obviously the **SoftMax**, which returns as output a vector that is still of size 5, so representing the score for each class, but the score is a value between 0 and 1, so that it represents the *probability* of the lyric to belong to each of the 5 classes, and the one that is more likely to be the class of the input is obviously the one with the highest probability: it's the output activation function that has to be exploited in multiclass classification problems and, for the very same reason, the **loss function** exploited is the **Cross Entropy Function**, because at each training step it computes the difference between the five scores returned and the ones that were expected, so it basically returns the distance between the probability distribution that we get after the **SoftMax**, and the one that is expected.
The **Adam** optimizer was exploited during the train step because it is one of the most efficient optimizers and setting the learning rate to 0.001 usually speeds up the training - and this seemed to be the case.

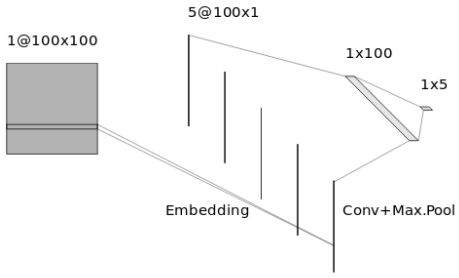
3 Exercise 1.2: BERT

Same preprocessing steps have been applied to the input for the BERT network, except that the tokenization and stopwords removal part was handled directly by the net itself and its tokenizer. I will not discuss about the implementation of this network, it was done by merely following the tutorial that was provided: the *"bert-base-uncased"* pre-trained **BERT** model was chosen to handle this classification task, and by exploiting this pre-trained model with pre-trained weights for the words, the results that we obtained and reported in the **Results** paragraph were far more accurate and better than the ones obtained with the previous naive network.
Notice that by choosing another pre-trained model, such as *"bert-large-uncased"* which is the largest one, we could get even better results.
The final layer on top of the pre-trained model is the one that is exploited in the provided notebook - **BertForSequenceClassification** - which is actually meant to provide a classification for sentences, that in our case are lyrics.

We provide this final layer the number of classes we have in our problem, so that it can exploit them to provide the class scores of each lyric we give it as input.

4 Can we do better? Exercise 1.1 with Word2Vec

Hard to do better than BERT model, but we can surely do better than the previous naive implementation of the CNN, and get closer to the BERT's results. We could probably make our previous network *deeper* by adding more convolutional layers with different kernel sizes
[Convolutional Neural Networks for Sentence Classification], so to consider in each of them 3, 4 and then 5 words at a time in one lyric, and this would eventually improve the performance of the network, but in order to see even a little improvement, to exploit the **word2vec** model for embedding is actually enough: the improvements we can obtain by exploiting this model are due to the fact that the vectors that represent the words and are found by this pre-trained neural network *already highlight the contextual relationships between the words in our corpus*, so that words sharing a similar or same context are actually represented as vectors that are close to each other in the vector space.
This is what has been changed in this implementation (Figure 2, next page) of the network with respect to the previous, naive one: the embedding layer is now built up by exploiting the **word2vec** model provided by the **gensim** library, so we are basically exploiting pre-trained weights for the words from another neural network. Each word is represented as a vector with **100** components, and this **dense** representation is actually more accurate than the previous one as the results suggest. Each word in the **word2vec** model is represented by taking into account a *window* of size 5, so the model predicts the current word from the words that are in a range distance of five from it, just like the convolutional layer takes into account 5-grams due to its kernel size (5x100). So, the model which was exploited is the **Continuous Bag Of Words** (parameter "sg = 1"), the one that is set by default by the library exploited. We get different results by exploiting the other model, the **skip-gram** - just a little improvement in the overall accuracy.
It is also important to highlight that each word appearing in the corpus is taken into account by the model: the default for the library is to consider only words appearing more than five times in the whole corpus, but this way we would lose information about words that may uniquely identify one song of one genre - which was maybe the only good point of the previous implementation of the CNN.
As can be seen from the results described in the next paragraph, this implementation actually improves the overall accuracy of the classification, as well as the precision and recall values for each class, with respect to the previous implementation of the CNN.




```
Net(
  (embedding): Embedding(43477, 100)
  (conv): Conv2d(1, 100, kernel_size=(5, 100), stride=(1, 1))
  (fc1): Linear(in_features=100, out_features=5, bias=True)
  (softmax): Softmax(dim=1)
)
```

Figure 2: Representation of the CNN exploited with word2vec embedding. Notice that for practical reasons only 5 of the vectors coming from the convolutional layer are reported.

5 Results

To see the results obtained, please open the three pictures under the folder *results*: *performance_BERT.png, performance_naive_embedding_cnn.png, performance_w2v_cnn.png*. We can draw some conclusions from them: first, the way we embed the words can improve the results we get from the Convolutional Neural Network implementation. These results are not as good as the ones we get from the pre-trained BERT model, and this can be seen just by taking a look at the confusion matrix and the values of precision and recall for each class. There is something else we should take into account: some classes are actually easier to define from the network's point of view. For example, the "Hip-Hop" class is more "verbose", meaning it contains more words. In every network, almost every "Hip-Hop" song from the test set was classified correctly, so the recall value is the highest (0.93, in the naive implementation) but the precision is so low (0.35, still naive implementation) because lots of other songs from other classes were misclassified as "Hip-Hop" songs, and this may be mainly because of the high number of words that are actually contained in the lyrics of "Hip-Hop" songs, so we should expect that eventually words that are used in a lyric from another class are almost definitively used also in this class: indeed, most of the "Rock" and "Jazz" lyrics were misclassified as "Hip-Hop", and their recall values are very low because of this.
This behavior is highlighted in the results of the naive implementation, while the word2vec embedding of the second implementation is able to mitigate the strong influence of the verbosity of "Hip-Hop" lyrics, with the BERT pre-trained model still being the best one in terms of results, but the difference is minimal. Another important result can be drawn from the confusion matrix of the word2vec-

embedded CNN and the BERT model: in the results of the first one, the network seems a little confused about the difference between "Rock" and "Electronic" lyrics: 58 out of 147 "Rock" songs were correctly classified, but 52 out of 147 were misclassified as "Electronic". The very same but converse behavior can be found in BERT: 37 out of 124 "Electronic" songs were correctly classified, but 55 out of 124 were classified as "Rock" songs, so it seems the networks still can't tell the differences between the lyrics of these two genres, and this may be due to the fact that we should provide a larger amount of lyrics to them.
Still, since the limit of 1.500 songs per genre was set due to the unbalanced nature of the dataset, there isn't much we can do to improve these results: "Electronic" genre has only 7.000 lyrics in the original dataset, so that should be as far as we can go to set the limit. The ideal condition would be to have a number of 40.000 valid lyrics per genre.
Indeed, I also performed a classification with the *word2vec-embedded* CNN with each class in the dataset containing 5000 lyrics, so with a dataset that is still balanced, and with more samples to train the network: the results can be found under the folder *results* in the image *w2v_cnn_performance_25000_5_classes.png*, and we can see that the accuracy increases (0.62), but the underlying pattern we have already discussed (network is a bit confused with respect to certain lyrics coming especially from "Electronic" and "Rock" classes, but also from "Jazz" and "Country") is still present.
I also noticed a little improvement when setting the **word2vec** model from the CBOW to the **skip-gram** one and the **window** parameter to 10, so that the embedding is done by exploiting the word currently processed by the network in the lyric to predict the words in a range of distance of 10 from it, predicting the context from the word. The performances are located under the folder *results*, file *w2v_cnn_performance_skip_gram_7500_5_classes.png*, and we can see the improvement by the values of *precision* and *recall* of each class - except "Electronic" - and the overall accuracy: these results are the best we can get from the word2vec-embedded CNN and the ones which are actually closer to the performance of the BERT model. The improvement might be due to the fact that the **skip-gram** model should actually have a better understanding (and so, a better representation) of very rare words in the corpus.
A classification with only 3 classes, without "Electronic" and "Rock", was carried out with the word2vec-embeddedd CNN, and the results actually showed better performances as expected - see the fourth and last picture under the *results* folder, named *performance_w2v_cnn_3_classess_1500.png*

References

[Convolutional Neural Networks for Sentence Classification]
https://arxiv.org/pdf/1408.5882.pdf, Yoon Kim.