Final Report

A lyric-based genre classification – A Naïve Bayes approach from naïve students.

Team: Grindset

Bùi Đức Khánh An – 20an.bdk@vinuni.edu.vn
Khau Liên Kiệt – 20kiet.kl@vinuni.edu.vn
Vương Đỗ Tuấn Thành – 20thanh.vdt@vinuni.edu.vn
Nguyễn Đại Nghĩa - 20nghia.nd@vinuni.edu.vn

Abstract:

Discrete Mathematics covers a variety of topics, one of which is probability. Experts in the field have produced various techniques applying the properties of probability. In this mini project, our team attempts to apply conditional probability to perform a basic classification. The goal is to develop a program implementing Naïve Bayes' classifier to differentiate songs into their genre using their lyrics. The paper will give details on constructing the algorithm from scratch and improve some features to aim for greater accuracy. The necessary background for Naive Bayes' classifier within its implementation is discussed rigorously in further sections.

1. Introduction:

The key concept of the project is to implement Naïve Bayes' Theorem for classifying songs into genres through the lyrics. The work procedure is divided into three steps:

- Handling raw data
- Featurizing
- Implementing Naïve Bayes' Theorem

In terms of **raw data handling**, the dataset is obtained from Tmthyjames, which contains *the song name*, *the lyrics*, *the year published*, *the artist*, *the genre*, etc. [6]. The dataset will be pre-processed and featurized into planned data types for later uses. Regarding the **featurizing**, its main function is to calculate the frequency of words appeared in each genre to derive the likelihood. The third procedure is to apply the **Naïve Bayes' Theorem** that gives prediction to the song's likelihood of the genre to be fallen into.

2. Mathematics background/Implementations:

2.1. Mathematics background

Naïve Bayes' classification is a probabilistic technique based on a strong assumption of independent features. For this ***genre classification problem***, the primary assumption is that *every word in the lyrics is independent of each other, given the genre they are in*. For example, the words "hip" and "hop" are independent given the lyrics are from the "*alt rock*" genre. Despite ignoring the context of the words, the model performs surprisingly well on some real-world problems, notably spam-filtering.

To clarify our purpose of the classification algorithm using **lyrics-based** approach, it generates the *probability of the genre given each word in the lyrics*. This calculation is according to the Bayes' Theorem, and we will elaborate more details on the event used in each component of the formula below.

$$P(genre_i|lyric) = \frac{P(lyric|genre_i) \times P(genre_i)}{P(lyric)}$$

With:

- $genre_i$: the event that the lyrics are in one of the genres - genre i
- $lyric$: the event of given lyrics to be classified

Since our lyric is composed of many words, the lyric can be made to be the intersection of many different words $lyric = w_1 w_2 w_3 \dots w_n$ with w_i represent the i^{th} word in the lyric. Given a specific genre, the independent assumption comes in as

$$P(w_1 w_2 \dots w_n | genre_i) = \prod_{k=1}^{n} P(w_k | genre_i)$$

with $P(w_k|genre_i)$ represents the frequency of word k[th] in genre i[th]

Hence, the probability of a genre given the lyric is

$$P(genre_i|lyric) = \frac{\prod_{k=1}^{n} P(w_k|genre_i) \times P(genre_i)}{P(lyric)}$$

Since we are only comparing the probability of whether the same lyrics belong to which genre, we can also conclude that

$$P(genre_i|lyric) \propto \prod_{k=1}^{n} P(w_k|genre_i) \times P(genre_i)$$

The larger the value of $\prod_{k=1}^{n} P(w_k|genre_i) \times P(genre_i)$, the more likely the lyric belongs to $genre_i$.

It must also be noted that, since each lyric contained hundreds of words within and the probability of each individual word is small, the product of $P(w_k|genre_i)$ might quickly approach 0. Therefore, to minimize this issue, we can use the log-function as it maintains the proportional property. Hence, we determine:

$$P(genre_i|lyric) \propto \log\left(\prod_{k=1}^{n} P(w_k|genre_i) \times P(genre_i)\right)$$
$$= \sum_{k=1}^{n} \log\big(P(w_k|genre_i)\big) + \log\left(P(genre_i)\right)$$

From the derived formula, we need to calculate the sum of log-functions and then compare them. The larger the result is, the more likelihood the song belongs to that genre or explicitly we need to evaluate:

$$\arg\max_{i \in \{1,\dots,I\}} \left(\sum_{k=1}^{n} \log\big(P(w_k|genre_i)\big) + \log\left(P(genre_i)\right)\right)$$

2.2. Implementation

The implementation for this project is divided into three main parts:

1. **Handling raw data** - importing the data and manipulating the dataset into appropriate form
2. **Featurizing** - manipulating the data into appropriate data structures for later uses
3. **Implementing Naïve Bayes' Theorem** - implementing from scratch the Naive Bayes algorithm

2.2.1. Handling raw data

The raw data set comes in the **csv** format and has the following appearance:

	album	artist	id	lyric	song	year	genre	ranker_genre	album_genre
0	The Johnny Cash Show (1970)	Johnny Cash	0	These hands aren't the hands of a gentleman th...	These Hands	1970.0	Gospel	Country	NaN
1	Rock the World (2000)	Bubbles	0	(Spoken) Every night I look up into the sky an...	I Have A Dream	2000.0	NaN	pop	Bubblegum Pop
2	The Johnny Cash Show (1970)	Johnny Cash	1	These hands raised a family these hands built ...	These Hands	1970.0	Gospel	Country	NaN
3	Recipe for Hate (1993)	Bad Religion	1	I know a man who doesn't have many friends	My Poor Friend Me	1993.0	Melodic Hardcore|Punk Rock	punk rock	Punk Rock
4	Punk in Drublic (1994)	NOFX	1	Friday night we'll be drinkin' Maneshevitz	The Brews	1994.0	Hardcore Punk|Melodic Hardcore|Punk Rock|Skate...	punk rock	Punk Rock
...	...	...	...	...	...	...	...	...	...
2778354	Edit	Usher	633518	Bring it all together like contact	Don't Hurt Em	NaN	R&B	rhythm and blues	NaN
2778355	The Hangover (2015)	Obie_Trice	633519	Ain't about college now it's about a dollar	Chuuuurch	2015.0	Hip Hop	Hip Hop	NaN
2778356	Edit	Usher	633519	Fold it up two times then creep back	Don't Hurt Em	NaN	R&B	rhythm and blues	NaN
2778357	The Hangover (2015)	Obie_Trice	633520	She turnt up, hottest in the city, most popular	Chuuuurch	2015.0	Hip Hop	Hip Hop	NaN
2778358	Edit	Usher	633520	Killin' them with a flow so cold	Don't Hurt Em	NaN	R&B	rhythm and blues	NaN

2778359 rows × 9 columns

For this project, the ***ranker_genre*** column is used to classify the songs based on its lyrics. But the original dataset has some caveats.

```
# Merge screamo, punk rock, heavy metal to alt rock to simplify
classification
# Too many genres can cause some trouble in classifying
data_csv['ranker_genre'] = np.where(
    (data_csv['ranker_genre'] == 'screamo')|
    (data_csv['ranker_genre'] == 'punk rock')|
    (data_csv['ranker_genre'] == 'heavy metal'),
    'alt rock',
    data_csv['ranker_genre']
)

# All label to lower case
data_csv['ranker_genre'] = np.where((data_csv['ranker_genre'] == 'Country'),
'country',  data_csv['ranker_genre'])
data_csv['ranker_genre'] = np.where((data_csv['ranker_genre'] == 'Hip Hop'),
'hip hop',  data_csv['ranker_genre'])

# For debugging
data_csv['ranker_genre'].value_counts()
```

Code is adapted from [6]

In the ***ranker_genre*** column, "*screamo*," "*punk_rock*," and "*heavy_metal*" are quite similar and can be grouped under "*alt_rock*" genre. This step is essential as too many genres can cause several issues for classifying the accurate genre.

Each row in the code contains one line of the lyrics, so the lyrics of the same song must be joined together for easier access. Fortunately, **pandas library** provides a ***.groupby()*** function for grouping different row entries into one entity, and a ***.join()*** function is used to concatenate the lyric rows.

```
# Data is available as 1 lyric per row. Merge to 1 song per row
group = ['song', 'year', 'album', 'genre', 'artist', 'ranker_genre']
lyrics_by_song = data_csv.sort_values(group).groupby(group).lyric.apply(' '.join).apply(lambda x:
x.lower()).reset_index(name='lyric')

# Remove non-alphanumeric characters
lyrics_by_song["lyric"] = lyrics_by_song['lyric'].str.replace(r'[^\w\s+]','')

# For debugging
lyrics_by_song
```

Code is adapted from [6]

Since the project is a lyric-based classifier, we will eliminate unnecessary columns:

```
# Clean unecessary column
group = ['song', 'ranker_genre', 'lyric']
lyrics_by_song = lyrics_by_song[group]

# For debugging
print(lyrics_by_song.shape)
lyrics_by_song
```

2.2.2. Featurizing

First, we setup the necessary constants for later use

- ***MIN_LYRIC_LEN***: any song with less than 400 characters is considered meaningless and hence, removed
- ***TRAIN_FRACTION***: split the train-test set with ratio of 80%-20%
- ***RANDOM_SEED***: random seed to recreate the random sampling

```
# Constant
MIN_LYRIC_LEN = 400
TRAIN_FRACTION = 0.8
RANDOM_SEED = 150

# Only accept song with more than 400 characters
# Since shorter songs usually contain meaningless phrases
lyrics_by_song = lyrics_by_song[lyrics_by_song.lyric.str.len() >
MIN_LYRIC_LEN]
lyrics_by_song = lyrics_by_song.reset_index(drop=True)

# Split all the lyric words
lyrics_by_song['words'] = lyrics_by_song['lyric'].str.split()
```

Code is inspired from [6]

As for any prediction task, the data must be split into the training set and the test set:

```
from sklearn.utils import shuffle

# List of genres for classifying
genres = list(set(lyrics_by_song["ranker_genre"]))
print(genres)

# Train and test set
train_dataframe = pd.DataFrame()
test_dataframe = pd.DataFrame()

for genre in genres:
  # Split train - test by ratio of 8 - 2
  lyrics_by_song_qualified =
lyrics_by_song[(lyrics_by_song.ranker_genre==genre)]
  train_set = lyrics_by_song_qualified.sample(frac = TRAIN_FRACTION,
random_state = RANDOM_SEED)
  test_set = lyrics_by_song_qualified.drop(train_set.index)
  train_dataframe = train_dataframe.append(train_set)
  test_dataframe = test_dataframe.append(test_set)

# Shuffle and reset train - test index
train_dataframe = shuffle(train_dataframe).reset_index(drop=True)
test_dataframe = shuffle(test_dataframe).reset_index(drop=True)

# For debugging
print(train_dataframe.head())
print(train_dataframe.shape)
print(test_dataframe.head())
print(test_dataframe.shape)
```

Code is adapted from [6]

Creating a word vocab list to be used in later sections to check if words in the test set are also in the train set:

```
# Create a word vocabulary - words must be unique
word_vocab = []
for words in train_dataframe['lyric_words']:
  word_vocab.extend(words)
word_vocab = set(word_vocab)

# For debugging
word_vocab
```

2.2.3. Implementing Naïve Bayes' Theorem

To calculate the $P(genre_i|lyric)$, we need to calculate the values of $P(w_k|genre_i)$ and $P(genre_i)$. To facilitate the calculation, we create a dictionary named ***genre_data*** to store values for later use

- We first use a counter to count the ***word_occurence*** and the number of words in each genre, which will be used for $P(w_k|genre_i)$

- We can straightforwardly calculate $P(genre_i)$ by using $\frac{number\ of\ songs\ in\ genre_i}{number\ of\ songs\ in\ total}$ as in the ***prob*** value

```
genre_data = dict()
for genre in genres:
  words = []
  train_dataframe.loc[train_dataframe['ranker_genre'] == genre,
'lyric_words'].apply(words.extend)
  genre_data[genre] = {
       'word_occurence': Counter(words), # count all the word occurence in
the genre
       'count': len(words),
       'prob': train_dataframe['ranker_genre'].value_counts()[genre] /
train_dataframe.shape[0]
  }

# For debugging
genre_data
```

To avoid the case of a word, existed in this genre but not in the other genre that leads to the probability of 0, we will apply the **Laplace smoothing** to $P(w_k|genre_i)$

$$P(w_k|genre_i) = \frac{occurence\ of\ w_k + \alpha}{number\ of\ words\ in\ genre + \alpha \times K}$$

With α is the smoothing parameter and K is the number of dimensions ($K = size\ of\ vocab$ in this case). Since we will apply the log to this parameter, we can do it here to optimize the performance as $np.log2$ works fast and best on integer parameters.

$$\log_2 P(w_k|genre_i) = \log_2(w_k\ occurence + \alpha) - \log_2(number\ of\ words\ in\ genre + \alpha \times K)$$

For words that are not in our word_vocab, we simply ignore its probability and set $P(w_k|genre_i) = 1$ or $\log_2 P(w_k|genre_i) = 0$. This is convenient since we do not have to do anything else to remove these cases.

```
# Since by multiplying too many small probability, the result will become 0
# Therefore, we will use log2 function to avoid this situtation

alpha = 1
K = len(word_vocab)
def prob_log2_lyric_in_genre(lyric_words, genre):
  prob_lyric = 0

  for word in lyric_words:
    # for word not in the vocab, it would be much easier just to drop it
    if word in word_vocab:
      # using Laplace smoothing with number of features = number of word in
vocab
      prob_lyric += np.log2(genre_data[genre]['word_occurence'][word] +
alpha) - np.log2(genre_data[genre]['count'] + alpha * K)
  return prob_lyric
```

With all the necessary data in hand, the genre prediction is simple as we only need to choose the genre that has $\sum_{k=1}^{n} \log\big(P(w_k|genre_i)\big) + \log\left(P(genre_i)\right)$ max

2.3. Improvements

A n-gram model considers the set of all sequences of n words within a document. In our case, the documents are the lyrics of the song. N-grams are used to improve the classification tasks in NLP. For example, in sentiment analysis, the sentence "This movie is really fascinating" has the following analysis:

Model	Combination
Unigram	["This", "movie", "is", "really", "fascinating"]
Bigram	["This movie", "movie is", "is really", "really fascinating"]
Trigram	["This movie is", "movie is really", "is really fascinating"]

In the unigram model, both 'really' and 'fascinating' both occurs once, and there is a chance that our model would classify the sentence as positive. Intuitively, a bigram model can improve the accuracy because the phrase "really fascinating" is present, signaling a positive sentiment with a higher degree.

After a few attempts with the unigram model, we decided to take a step further by taking the ***idea of bigram model***, which is to create a list of every 2 words instead of 1 word.

```
def bigram(lyric_words):
  return list(ngrams(lyric_words, 2))

lyrics_by_song['lyric_words'] = lyrics_by_song['words'].apply(bigram)
lyrics_by_song
```

This approach reduces the "naive-ness" in the previous approach by considering the word locality, which might express the featured characteristic of one song, which yields to a better result.

The implementation process is as straightforward as before with no change in implementation. Instead of considering separate words previously, we consider the 2-word pairs as independent features.

3. Results and discussion

3.1. Results

The accuracy of the model is calculated by taking the proportion of *the number of correctly predicted songs* over *the total number of songs in the datas*et. After training the model with more than 2.7 million lines of lyrics, we successfully predict the genre of around 83% of 11983 sample songs in the test from our dataset.

	song	ranker_genre	lyric	words	lyric_words	predicted
0	Maria	alt rock	she moves like she dont care smooth as silk co...	[she, moves, like, she, dont, care, smooth, as...	[(she, moves), (moves, like), (like, she), (sh...	alt rock
1	Angel from Montgomery	country	i am an old woman named after my mother my old...	[i, am, an, old, woman, named, after, my, moth...	[(i, am), (am, an), (an, old), (old, woman), (...	country
2	Good Times	country	when i ran to the store with a penny and when ...	[when, i, ran, to, the, store, with, a, penny,...	[(when, i), (i, ran), (ran, to), (to, the), (t...	country
3	Hawaiian Wedding Song	country	this is the moment ive waited for i can hear m...	[this, is, the, moment, ive, waited, for, i, c...	[(this, is), (is, the), (the, moment), (moment...	country
4	Full Disclosure	alt rock	oh i dont want one i dont want one i dont want...	[oh, i, dont, want, one, i, dont, want, one, i...	[(oh, i), (i, dont), (dont, want), (want, one)...	alt rock

3.2. Discussion

When it comes to the final step, we have already implemented the same Naïve Bayes' Theorem to classify songs based on emotion. However, due to the insufficient labeled dataset for emotion (around 1000 songs only), the accuracy of this task is not as promising as we expected (around 60%, which is not significant enough). Therefore, we are heading to improve the accuracy of our main **lyrics' classifier.**

In terms of improvements, there are two technique trials: **N-gram language model** and **Data Preprocessing with Word Lemmatization and Removing Stop Words.** The detailed explanation for N-gram method is clearly elaborated under section 2.3 (Improvement). When it comes to the remaining approach, although we found that some words have the same meaning, their notations are different. For instance, the plural words are those which lemmatization could be applied such that "songs" and "song" contain the same meaning, thus we could reduce those and represent them as one word "song." This implementation could also convert other verb tenses into the simple tense. In addition, we also managed to remove words in natural language that have little meaning, such as "is," "an," "the," which are known as stop words. These words provide less significantly unique information that can be used for classification; thus, they are proven to be abundant. Therefore, these two approaches could clean our "lyric words" component, utilizing the dictionary of the English language.

To apply these concepts in our code, fortunately, ***NTLK library*** has provided the implementation in practice.

```
# Clean lyric words by lemantizing and stop words
lemmatizer = WordNetLemmatizer()
stop_words = stopwords.words('english')
stopwords_dict = Counter(stop_words)

def lemmatize_stop(lyric_words):
  # Lemantizing
  for i in range(len(lyric_words)):
    lyric_words[i] = lemmatizer.lemmatize(lyric_words[i])

  # Stop word
  lyric_words = [lemmatizer.lemmatize(word) for word in lyric_words if word
not in stopwords_dict]
  return lyric_words

lyrics_by_song['words'].apply(lemmatize_stop) # Uncomment this line to use
lemmatizing and stop words

# For debugging
lyrics_by_song
```

However, in comparison with not using these two approaches, the accuracy of applying them has no significant change, or even decreases. The table below illustrates the results of the afore statement:

	Unigram	**Bigram**
Doing nothing	77.37%	83.12%
Lemmatization	76.79%	83.14%
Stop words	77.37%	83.13%
Both lemmatization and stop words	76.79%	83.14%

At the final stage, our team decides that model with only ***Bigram*** method yields the best result since it accounts for the most promising accuracy and shortest runtime due to fewer calculations in the raw code.

4. Conclusion

The idea of the Bayesian method is illustrated in this report and is stated the reason why we apply it. We concentrate on developing a profound algorithm for categorizing music by genre based on its lyrics.

We gather the data by crawling online sources and form a list of songs with unique features, especially genre, needed for classification. Then, we count the total number of words in each genre and filter them with our classifier algorithm to optimize the accuracy of the calculation afterwards.

Implementing the Naive Bayes' Theorem, we predict the genre of one song based on the highest probability that its lyrics belong to that genre. During the improvement stage, after considering some methods to increase the accuracy, *Bigram* method is chosen as it results in the highest accuracy overall.

5. References

[1] P. Horbonos, "How to build and apply Naive Bayes classification for spam filtering," *Towards Data Science*, Jan. 31, 2020. https://towardsdatascience.com/how-to-build-and-apply-naive-bayes-classification-for-spam-filtering-2b8d3308501

[2] V. Jayaswal, "Laplace smoothing in Naïve Bayes algorithm," *Medium*, Nov. 22, 2020. https://towardsdatascience.com/laplace-smoothing-in-na%C3%AFve-bayes-algorithm-9c237a8bdece

[3] T. J. Dobbins, *Lyrics Wikia*. 2020. [Online]. Available: https://github.com/tmthyjames/cypher

[4] T. Vu, "Naive Bayes Classifier," *Tiep Vu's blog*, Aug. 08, 2017. https://machinelearningcoban.com/2017/08/08/nbc/

[5] M. Garvey, "Naïve Bayes Spam Filter — From Scratch," *Towards Data Science*, Dec. 02, 2020. https://towardsdatascience.com/na%C3%AFve-bayes-spam-filter-from-scratch-12970ad3dae7 (accessed Nov. 18, 2021).

[6] "Using Naive Bayes to Predict a Song's Genre Given its Lyrics Grid-Searched." https://tmthyjames.github.io/2018/february/Predicting-Musical-Genres/ (accessed Nov. 18, 2021).