HW №2 Report

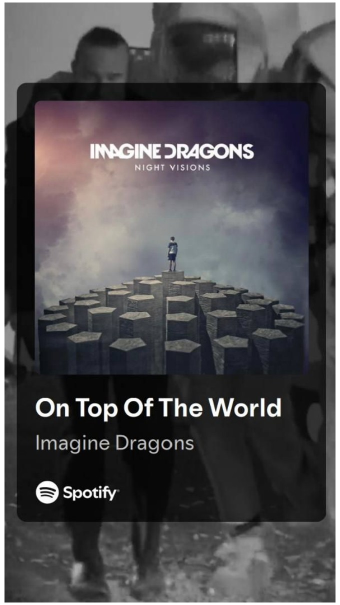
By Korablina Maya, group 233

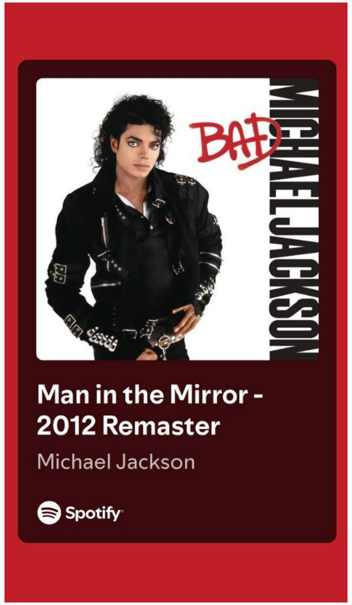
Part 1: TF-IDF Lyric Analysis

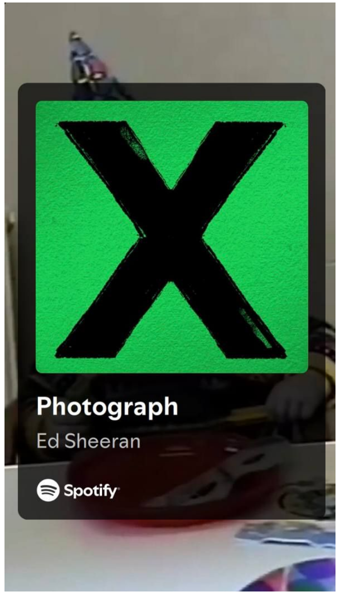
Objective:

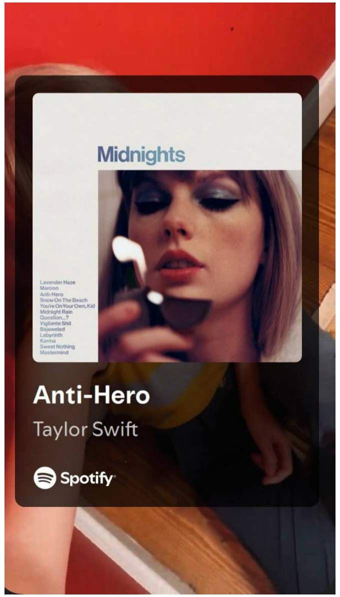
- To study the TF-IDF method for text vectorization To compare different vectorization approaches
- To conduct a statistical analysis of text data

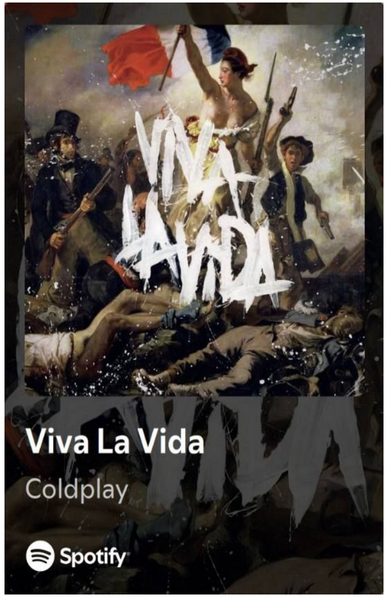
Songs I have chosen





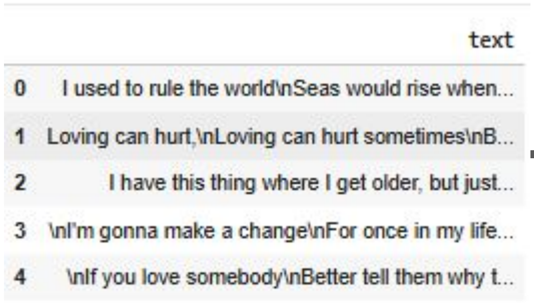




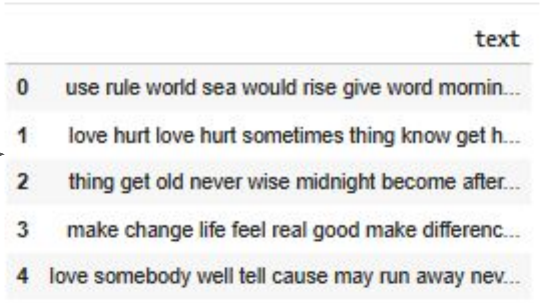


I thought that 3 songs is not really enough, so I took 5

Preprocess



	text
0	I used to rule the world\nSeas would rise when...
1	Loving can hurt,\nLoving can hurt sometimes\nB...
2	I have this thing where I get older, but just...
3	\nI'm gonna make a change\nFor once in my life...
4	\nIf you love somebody\nBetter tell them why t...

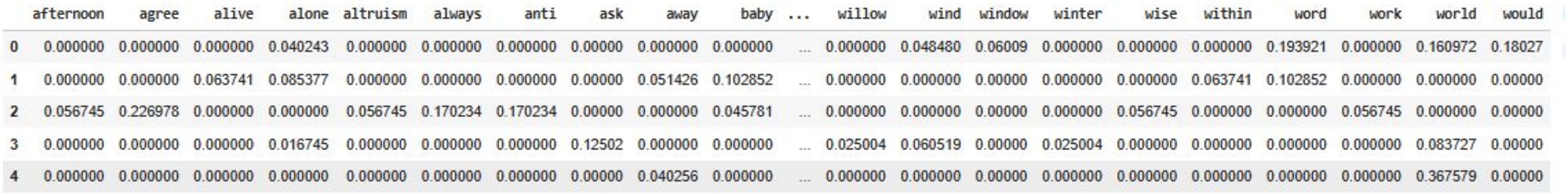


	text
0	use rule world sea would rise give word mornin...
1	love hurt love hurt sometimes thing know get h...
2	thing get old never wise midnight become after...
3	make change life feel real good make differenc...
4	love somebody well tell cause may run away nev...

- **I tokenized my texts.** In this task token = word
- **I deleted all stopwords**. I added some words to stoplist after analysys, like “oh”, “lalala” that are common for songs but are meaningless
- **I removed all symbols but letters** using regular expressions
- **I used WordNetLemmatizer** to transworm words to their infinitive forms. This process required me to identify part of speech (noun, verb etc) and after that make the transformation.

TF-IDF

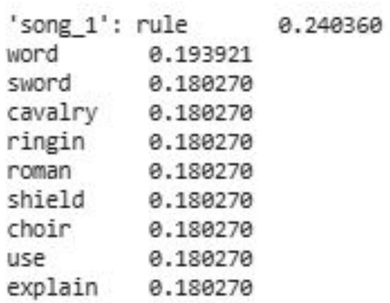
I used TfidfVectorizer from sklearn to calculate tf-idf for every word in every text



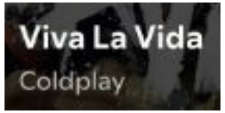
	afternoon	agree	alive	alone	altruism	always	anti	ask	away	baby	...	willow	wind	window	winter	wise	within	word	work	world	would
0	0.000000	0.000000	0.000000	0.040243	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	...	0.000000	0.048480	0.06009	0.000000	0.000000	0.000000	0.193921	0.000000	0.160972	0.18027
1	0.000000	0.000000	0.063741	0.085377	0.000000	0.000000	0.000000	0.00000	0.051426	0.102852	...	0.000000	0.000000	0.00000	0.000000	0.000000	0.063741	0.102852	0.000000	0.000000	0.00000
2	0.056745	0.226978	0.000000	0.000000	0.056745	0.170234	0.170234	0.00000	0.000000	0.045781	...	0.000000	0.000000	0.00000	0.000000	0.056745	0.000000	0.000000	0.056745	0.000000	0.00000
3	0.000000	0.000000	0.000000	0.016745	0.000000	0.000000	0.000000	0.12502	0.000000	0.000000	...	0.025004	0.060519	0.00000	0.025004	0.000000	0.000000	0.000000	0.000000	0.083727	0.00000
4	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.040256	0.000000	...	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.367579	0.00000

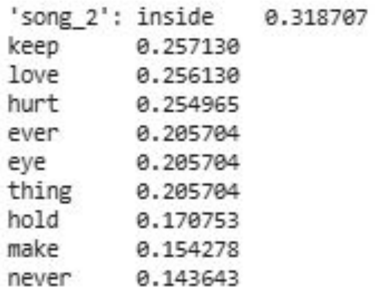
I had a total of 294 unique words. TF-IDF shows importance of a word to the particular document. The higher the metric, the more important the word is to the song

Most important words to each song according to TF IDF

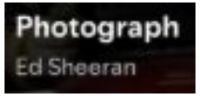


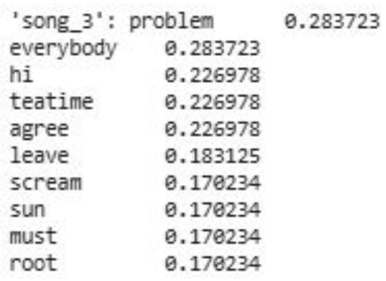
```
'song_1': rule        0.240360
word          0.193921
sword         0.180270
cavalry       0.180270
ringin        0.180270
roman         0.180270
shield        0.180270
choir         0.180270
use           0.180270
explain       0.180270
```



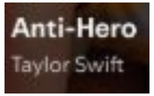


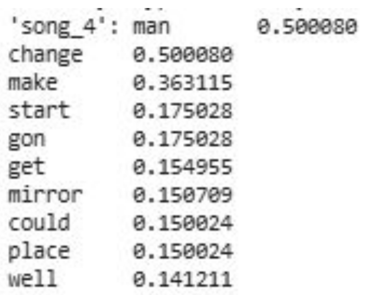
```
'song_2': inside    0.318707
keep        0.257130
love        0.256130
hurt        0.254965
ever        0.205704
eye         0.205704
thing       0.205704
hold        0.170753
make        0.154278
never       0.143643
```



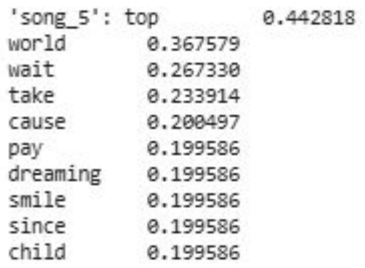


```
'song_3': problem      0.283723
everybody      0.283723
hi             0.226978
teatime        0.226978
agree          0.226978
leave          0.183125
scream         0.170234
sun            0.170234
must           0.170234
root           0.170234
```

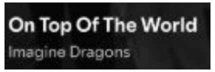




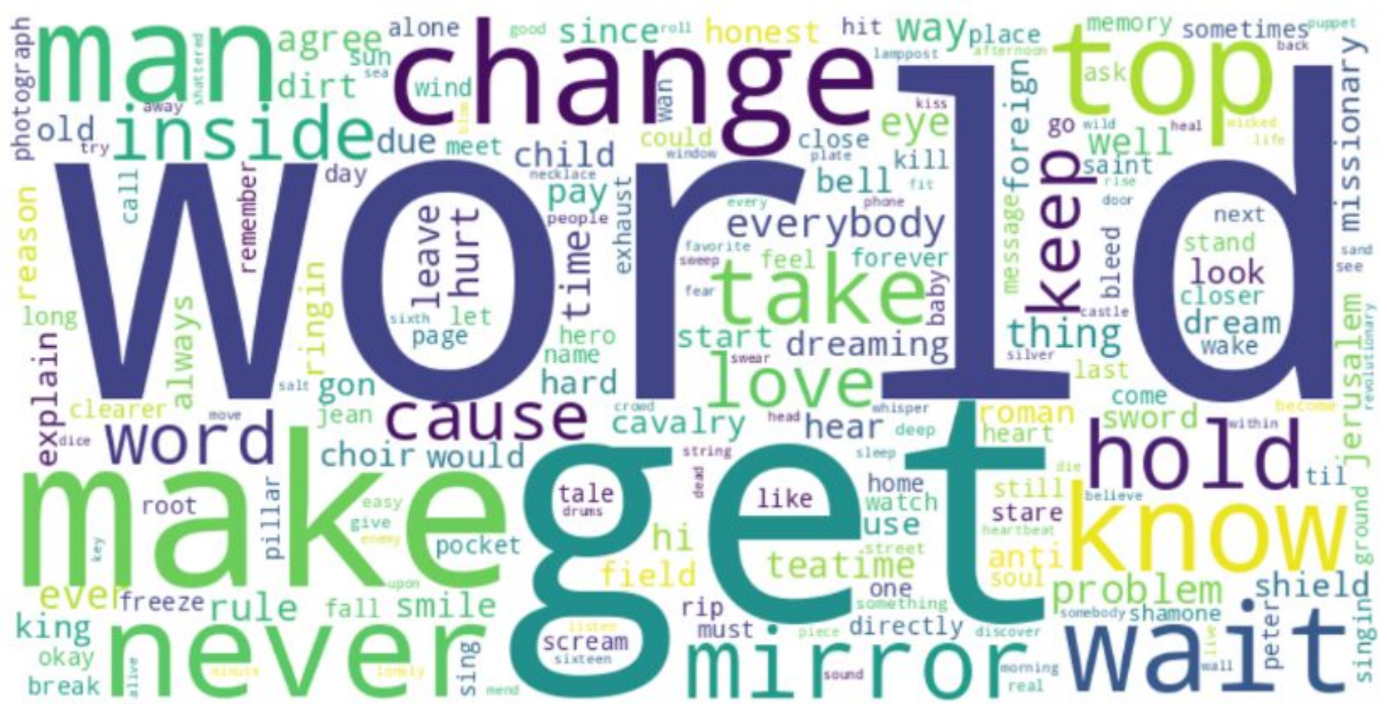
```
'song_4': man        0.500080
change      0.500080
make        0.363115
start       0.175028
gon         0.175028
get         0.154955
mirror      0.150709
could       0.150024
place       0.150024
well        0.141211
```



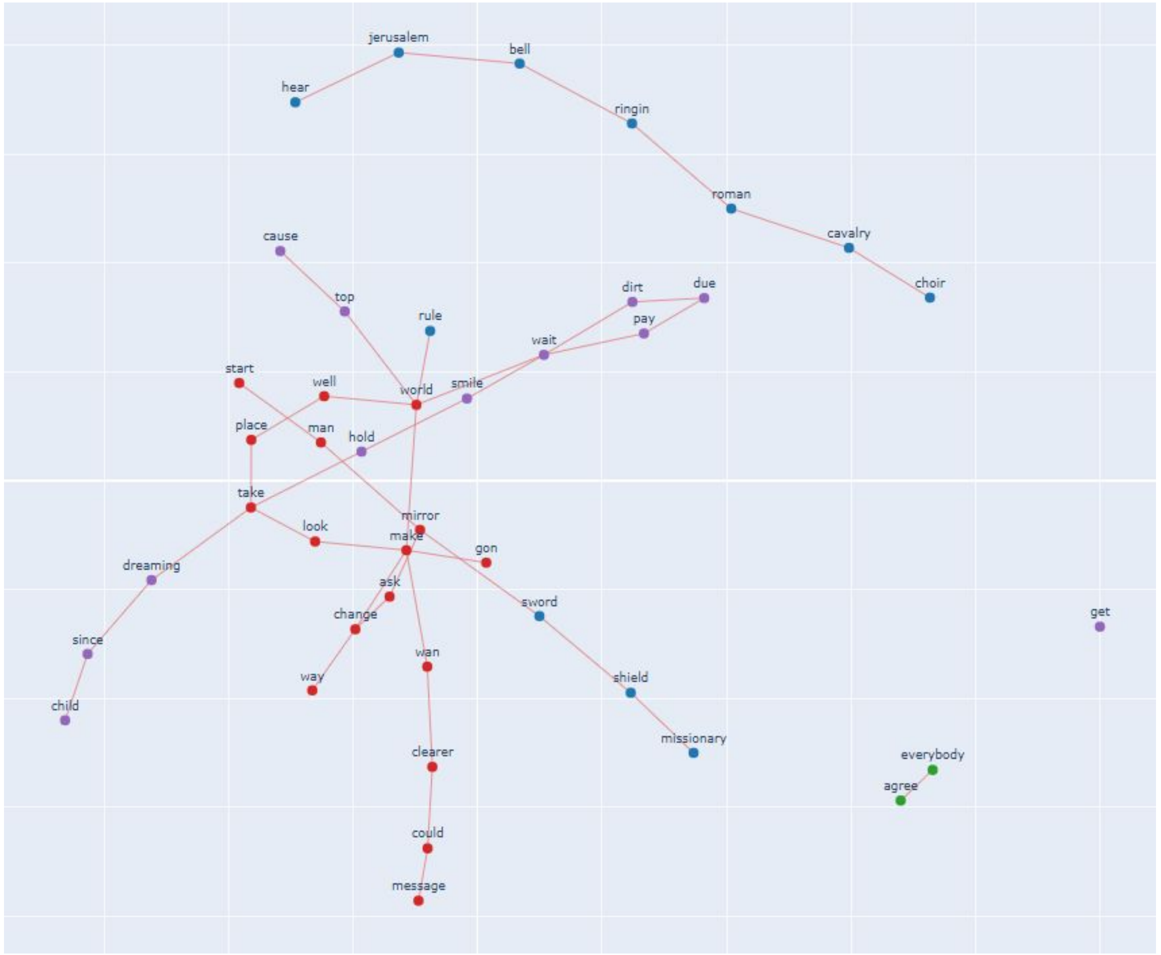
```
'song_5': top          0.442818
world        0.367579
wait         0.267330
take         0.233914
cause        0.200497
pay          0.199586
dreaming     0.199586
smile        0.199586
since        0.199586
child        0.199586
```



Word cloud



World is the most important word in amont the others throughout all songs



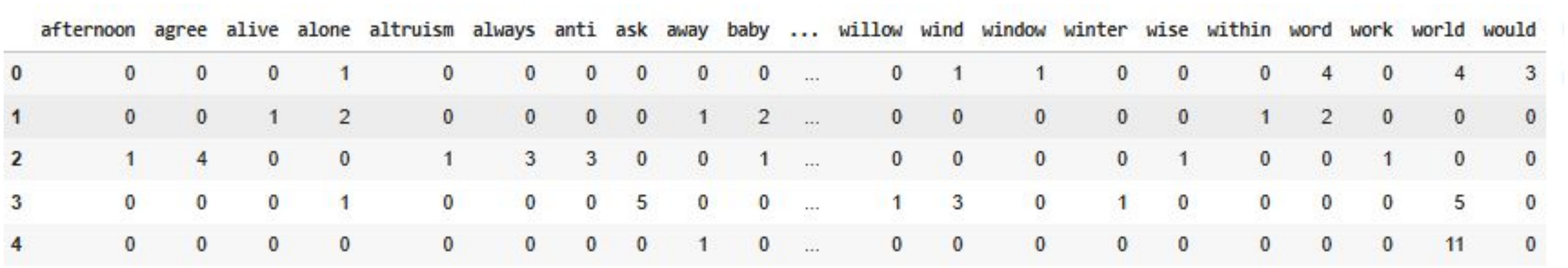
Bigram and trigram graph

I wanted to see how all the selected songs were related in meaning. To do this, I took the 30 most common bigrams and trigrams from words with significant meanings according to TFIDF.
It turned out that "Viva La Vida" and "Anti-Hero" have distinct meanings, while "Man in the Mirror," "Viva La Vida," and "On Top of the World" share a common meaning related to "world"

Ed sheeran, sorry(
Your sond's bigrams and trigrams are not very common(

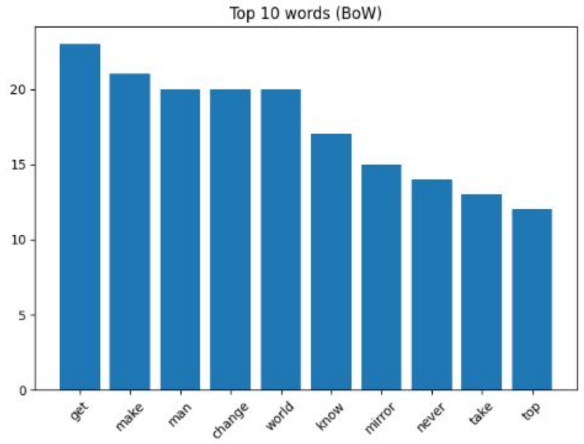
TF-IDF vs BagOfWords

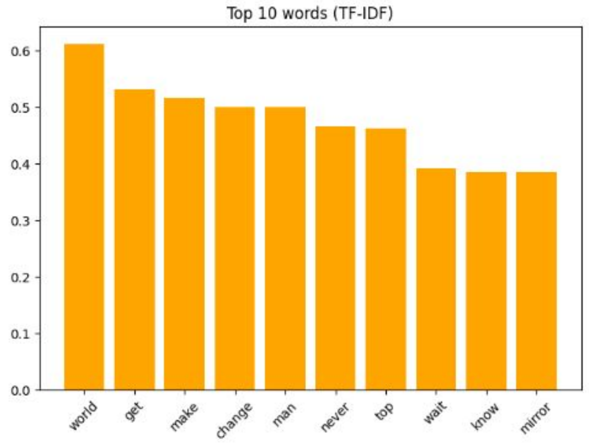
I used CountVectorizer from sklearn to implement BoW vectorization



	afternoon	agree	alive	alone	altruism	always	anti	ask	away	baby	...	willow	wind	window	winter	wise	within	word	work	world	would
0	0	0	0	1	0	0	0	0	0	0	...	0	1	1	0	0	0	4	0	4	3
1	0	0	1	2	0	0	0	0	1	2	...	0	0	0	0	0	1	2	0	0	0
2	1	4	0	0	1	3	3	0	0	1	...	0	0	0	0	1	0	0	1	0	0
3	0	0	0	1	0	0	0	5	0	0	...	1	3	0	1	0	0	0	0	5	0
4	0	0	0	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	11	0

This method calculates number of occurrences of word in text





In BoW, **get** is the leading word because it appears frequently across different songs

But TF-IDF does not place it in the top position, which means the word is frequent but not unique.In TF-IDF, the top word is **world**

This means that it appears less often, but it is highly characteristic of specific texts (for example, *Viva la Vida* → “rule the world”)

Some words rise in the TF-IDF ranking even though they are not dominant in BoW (for example, **wait**). This indicates that these words occur less frequently but strongly distinguish a particular document

TF-IDF vs Word2Vec

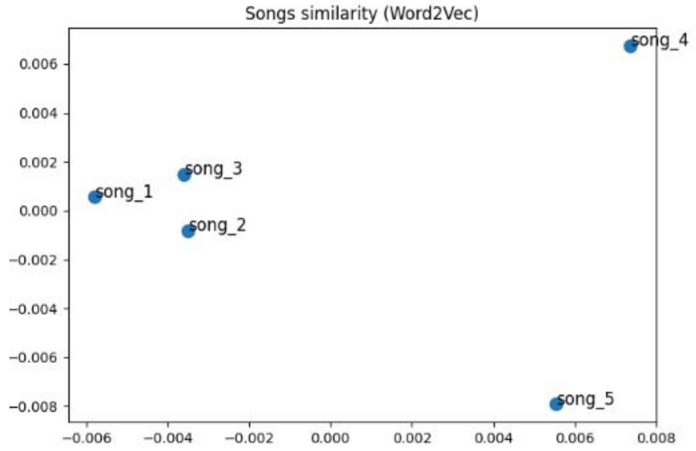
We can’t really compare these methods directly, as TF-IDF counts metric for each word and dimensions depend on total number of unique words in all documents

Word2Vec built different – this method transforms each text in n-dimensional vector (n is hyperparameter).

Word2Vec is not about the “meaning” of the text. It is about similarity of texts calculated by some closure estimator like cosine similarity.

I calculated cosine sim and used PCA to display 100-dimensional vectors in 2-dimensional space

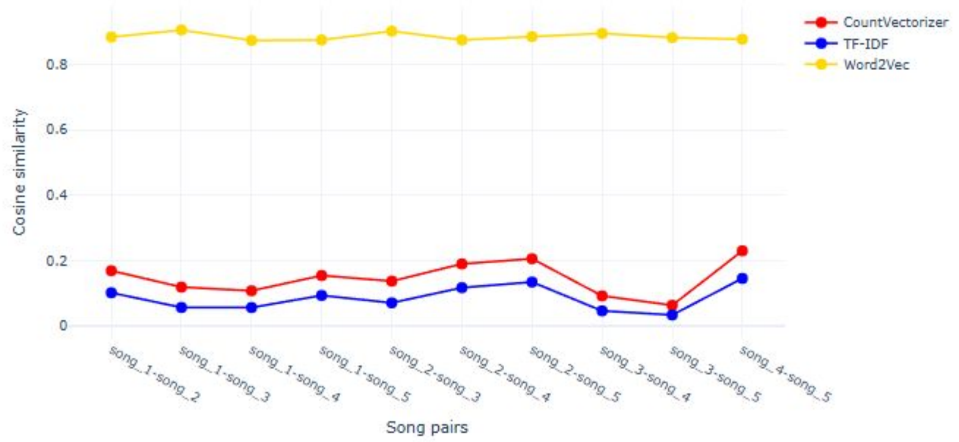
“Viva La Vida”, “Photograph” and “Anti-Hero” (1, 2, 3) are close because they share introspective, emotional, and narrative language
"Man in the Mirror" (4) differs slightly due to its focus on self-improvement and social themes.
"On Top of the World" (5) is the most distinct, with optimistic, success-oriented vocabulary that sets it apart from the more reflective songs.



We often use vectorizations to calculate similarities (for ex. in recommendation systems)

BoW is the worst method for similarities, because it only calculates words → songs with same set of words, but different meaning would be considered similar (that is clearly shown on the graph)

TF IDF and W2V are better for the similarity. In fact, TF-IDF is slightly better on a small amount of documents as it considers word importance



Conclusion on part 1

I used the TF-IDF method to find the most important words in five song lyrics. After preprocessing the texts, TF-IDF showed which words are meaningful for each song, not just frequent.

When comparing TF-IDF with Bag-of-Words, it became clear that Bag-of-Words is less useful because it only counts how often words appear. TF-IDF works better for a small set of texts since it considers word importance. Word2Vec was also tested, and it helped show which songs are similar in meaning by placing them close in vector space.

Overall, TF-IDF and Word2Vec give a more accurate picture of text meaning and similarity, while Bag-of-Words is too simple for this type of analysis.

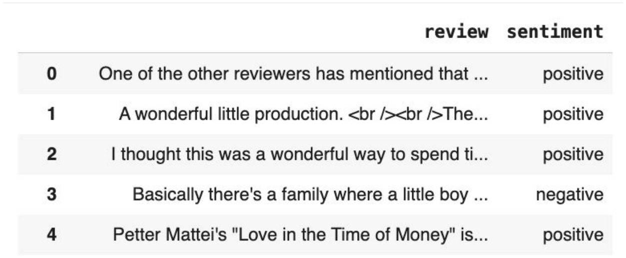
Part 2: Sentiment Classification with BERT

Objective:

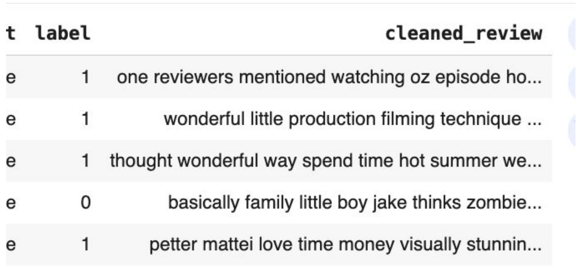
- to use a pre-trained BERT model to classify the sentiment of movie reviews.

Preprocess

I work with reviews. They can be positive or negative, depending on a context



	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. <br /><br />The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive



t	label	cleaned_review
e	1	one reviewers mentioned watching oz episode ho...
e	1	wonderful little production filming technique ...
e	1	thought wonderful way spend time hot summer we...
e	0	basically family little boy jake thinks zombie...
e	1	petter mattei love time money visually stunnin...

- I processed data the same way I did i part 1. The only difference – I did not standardize words (as bert can process diff. forms of word without troubles)
- I also replaced sentiment as numbers (0 - negative, 1 - positive)
- I splitted data to train and test ()

Setting up BERT

- I downloaded a pre-trained BERT model from HuggingFace
- I also downloaded BERT tokenizer where input_ids and attention_mask are set (i use max_len=256 and batch_size 16 in my tokenizer)
- I created 2 data loaders for train and test to prepare data for my model

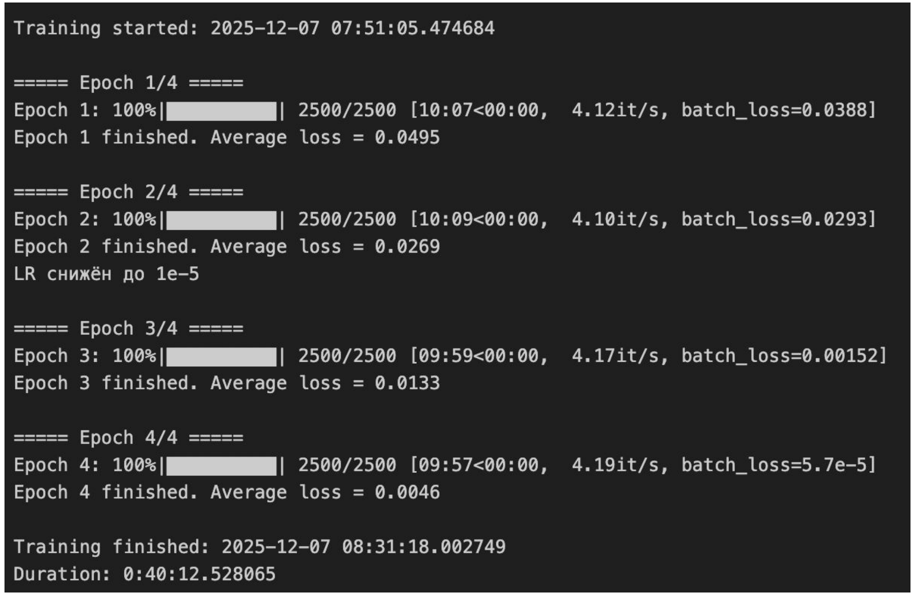
Doing some fine-tuning

I made for my BERT model a classic training cycle with some add-ons:

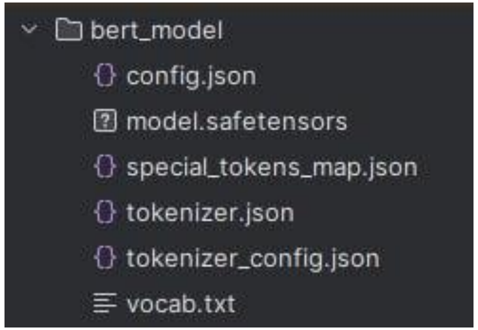
- Trained on gpu
- 4 epochs
- 2500 batch per epoch
- decreased learning rate after 2-nd epoch – smaller steps, more accurate learning

My model was training for 40 minutes on rtx 3070, that is a really great result!

I also saved my model for future uses



```
Training started: 2025-12-07 07:51:05.474684

===== Epoch 1/4 =====
Epoch 1: 100%|██████████| 2500/2500 [10:07<00:00,  4.12it/s, batch_loss=0.0388]
Epoch 1 finished. Average loss = 0.0495

===== Epoch 2/4 =====
Epoch 2: 100%|██████████| 2500/2500 [10:09<00:00,  4.10it/s, batch_loss=0.0293]
Epoch 2 finished. Average loss = 0.0269
LR снижён до 1e-5

===== Epoch 3/4 =====
Epoch 3: 100%|██████████| 2500/2500 [09:59<00:00,  4.17it/s, batch_loss=0.00152]
Epoch 3 finished. Average loss = 0.0133

===== Epoch 4/4 =====
Epoch 4: 100%|██████████| 2500/2500 [09:57<00:00,  4.19it/s, batch_loss=5.7e-5]
Epoch 4 finished. Average loss = 0.0046

Training finished: 2025-12-07 08:31:18.002749
Duration: 0:40:12.528065
```

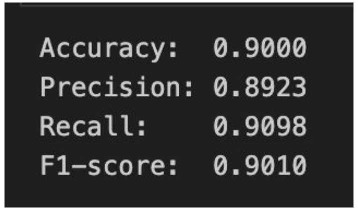


```
bert_model
    config.json
    model.safetensors
    special_tokens_map.json
    tokenizer.json
    tokenizer_config.json
    vocab.txt
```

And what about metrics?

Accuracy = 0.9 → just what I need!

F1 score for the positive class is close to the accuracy, indicating a balance between precision and recall



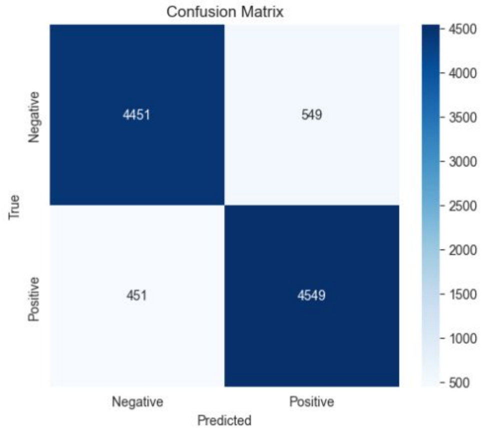
```
Accuracy:  0.9000
Precision: 0.8923
Recall:    0.9098
F1-score:  0.9010
```

According to confusion matrix, diagonal elements (TP + TN) are very large, meaning the model almost always classified correctly.

FP and FN almost equal (the difference is very small), meaning:

- The model distinguishes positive and negative reviews well.
- The model is not biased toward either class.

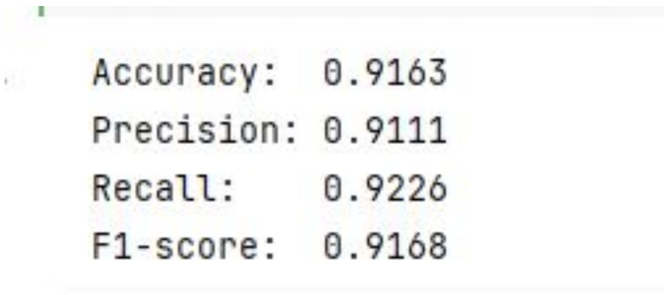
The balance between precision and recall is roughly equal



But can we make it better?

Instead of using fixed learning rate, we can adjust learning rate at each step

The training process uses a learning rate schedule with a warmup and cosine decay. During warmup, the learning rate gradually increases from zero to its peak value, helping stabilize early training. After that, cosine decay smoothly reduces the learning rate toward zero, allowing the model to converge more accurately.



```
Accuracy:   0.9163
Precision:  0.9111
Recall:     0.9226
F1-score:   0.9168
```

Result: **accuracy = 0.9163 (+ 1, 63% 🔥)**

F1-score = 0.9168 is almost identical to the accuracy

Precision = 0.9111 and r**ecall = 0.9226** are closely aligned, **the model performs consistently** across both classes

Now, this is awesome!

Can we make it EVEN MORE better?

Bert was pretrained on regular texts, meaning all stopwords were not removed.

I've decided to try training pipeline from previous slide, but on data where preprocess just removed html-tags and punctuation, everything else left untouched

Result: accuracy = 0.9294 🔥🔥🔥

Precision = 0.9240 🔥

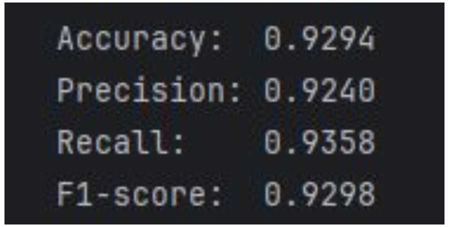
Recall = 0.9358 🔥

My model shows a strong ability to capture relevant sentiment signals in the data!!

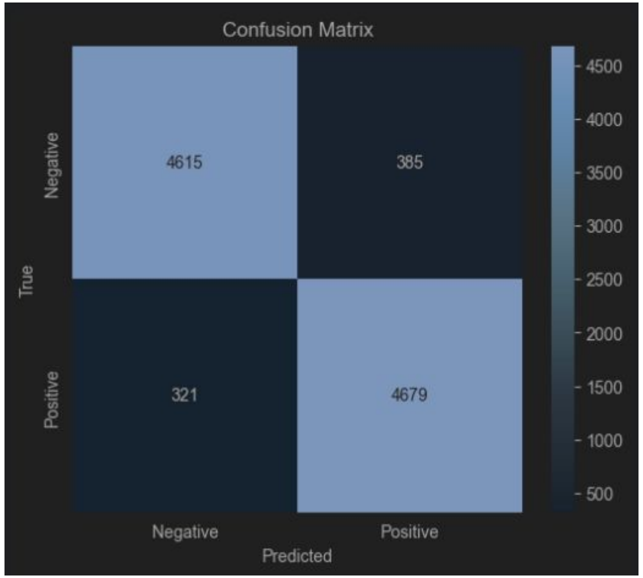
F1-score = 0.9298🔥

F1 is nearly identical to accuracy, showing a strong balance between precision and recall and confirming overall model stability!!

And finally, the numbers from conf. matrix indicate low misclassification rates for both classes!



```
Accuracy:  0.9294
Precision: 0.9240
Recall:    0.9358
F1-score:  0.9298
```


Confusion Matrix
True
Negative
Positive
Predicted
Negative
Positive