Classifying Lyrics by Music Genre using Naive Bayes Classification

Thijs van den Hout
Radboud University
Nijmegen
t.vandenhout@student.ru.nl

ABSTRACT

Automatic classification of music genre has proven to be a difficult topic in music information retrieval. In particular classification solely on the basis of lyrics is a challenge. Previous research has focussed mainly on non-lyrical features for classifications, with a few exceptions. This project explores the possibilities and limitations of applying Multinomial Naive Bayes, a commonly used classification algorithm in the field of textual information retrieval, on the problem of classifying music genre by lyrics. A novel feature is the introduction of a certainty threshold before classification. The data set that is used distinguishes 5 genres and results of the experiment will be analyzed in terms of accuracy, misclassification errors and the certainty threshold trade-off.

1 INTRODUCTION

The automatic classification of music genre is a difficult problem in music information retrieval. Especially classification based purely on the lyrics of a particular song is complicated. Most research in the field of music classification makes use of audio signals, sometimes combined with textual information e.g. [2, 3, 5–7]. The automatic classification of music genre has many applications. Lyric database sites often sort their song lyrics by genre, some applications (e.g. Pandora radio) show complementary images to the music that is playing [2], and artists may want extra insight in the lyrics they create. In this research project, an attempt will be made to classify songs by genre using solely textual information: the song's lyrics. This will be done using a Multinomial Naive Bayes Classification. The data set that will be used is retrieved from Kaggle.com and contains roughly 250,000 song lyrics with their corresponding genre. The data set distinguishes 5 different genres: pop, hip-hop, rock, metal and country. A novel feature of the approach in this study is the inclusion of a certainty threshold. Since lyrics alone may not perfectly capture a genre, a certainty threshold may provide the user with more confidence about the classification.

2 BACKGROUND

Much research has been done in the field of music information retrieval. The classification of musical genre is often guided by multi-modal information by taking the audio signal into account. A few researchers have delved into the problem of genre classification purely by looking at the lyrics of songs [1, 4, 5, 8, 9]. A. Canicatti in a paper in the data mining conference of 2016 evaluated a number of classification methods and expresses the significance of the collection and preprocessing of data [1]. Hu et al. in 2009 published a paper in ISMIR in which they classified music mood by lyrics and audio and surprisingly showed that audio does not always outperform lyrics features [4]. This is an optimistic conclusion for the current research. Mayer et al. in 2008 combined lyics with other features such as rhyme and style for musical genre classification [5]. They tested their approach on various classification models and showed that stemming of words improves accuracy in Bayes classification. Their conclusion was that rhyme and style can indeed contribute to better classification. In a recent paper by Tsaptsinos, a hierarchical attention network was employed to prove the ability of neural models in lyrics-based music genre classification [8]. In their paper, they conclude their neural network approach outperformed classical non-neural models, though only compare it to a majority classifier and logistic regression. Finally, Ying et al. wrote in 2012 about the classification of mood and genre using lyrics features [9]. They explored the various lyrics feature such as POS tags and concluded, as may be intuitive, that mood classification obtained higher accuracies than genre classification. Based on the knowledge gathered from these papers, this study will continue to explore the possibilities of music genre classification based on song lyrics using Multinomial Naive Bayes classification.

3 AIM OF THE RESEARCH

The aim for this research is to create a classifier which can classify songs to their corresponding genre based on their lyrics. This classifier can be used to automatically assign a genre to lyrics in a number of applications. For example, a song lyric website could assign a genre to lyrics that have not been provided a genre label. Furthermore, the introduction of a certainty threshold for classification will reduce the error of the classifier, with the drawback of the model being more limited in its performance.

4 DATA

The data set that is used in this study is a data set from Kaggle[1], a data science platform on which data sets can easily be shared. The data set contains 362,237 songs with lyrics and genre label. After pre-processing and removing songs such that certain genres are not over-represented, the data set contains roughly 140,000 songs. This is a much larger number than is used in previous research that attempt genre classification based purely on lyrics.

RU-TxMM 2018, January 2019, Nijmegen, the Netherlands

[1] https://www.kaggle.com/gyani95/380000-lyrics-from-metrolyrics

Genre	Number of songs
Pop	40466
Hip-Hop	28251
Rock	38000
Metal	23759
Country	14387

Table 1: Genre distribution

4.1 Data set limitations

Inevitably, each data set will have its limitations. Particularly data sets that are not created for the research in mind. This data set contains a lot of valuable data, though, also has its limitations. Firstly, the data set contains only song lyrics from artists with starting letter A-G. The owner of the data set acquired the data with a web crawler and ran out of storage after the amount of data that can be found in this set. Regardless, this data is assumed to be representative because of its large number of songs. Secondly, the lyrics are obtained from MetroLyrics[2], a website where users can provide the lyrics to songs. This results in the lyrics being not necessarily objectively correct. Furthermore, the data set contains duplicate lyrics. For example, one song may be included in both its original version, acoustic version and covers, resulting in a duplication of lyrics. Finally, not every distinct genre contains the same number of songs; rock is over-represented in the data set, with nearly 50% of all data points before preprocessing. Nevertheless, this data set was chosen because of its size; to be able to compare the positive effects and trade-offs between a large corpus and smaller ones that are used in previous research.

4.2 Data preprocessing

The preprocessing of the data set is a very important important and time consuming task in data science. Especially with data sets that are not tailored to the study, extra care is required to make the data usable and reliable.

For preprocessing and data representation, the pandas package for Python was used. Panda's DataFrame data structure is very useful for operations on data sets. Firstly, all irrelevant columns and empty rows are dropped so we are left with lyrics and their corresponding genre. Next, rows for which the genre is not available or do not have a significant number of songs are removed. The genres *Indie* and *R&B* are merged with *Rock* and *Hip-Hop* respectively because of their similarities and relatively few number. As a result five genres remain: *Pop, Hip-Hop, Rock, Metal and Country*. Then, a number of the over-represented rock songs are randomly removed to make the classes roughly the same size. The distribution of genres can be seen in Table 1

From the lyrics, meta-information such as [Chorus] and [Verse] are removed. In some songs, information about the artist, song name or other meta information is included in the lyrics, which cannot automatically be removed because of its unstructured characteristic.

After these preprocessing steps, the data is split into a training and test set (80%-20%).

[2]http://www.metrolyrics.com/

5 METHODS

In this section, feature extraction, feature selection and classification methods are discussed.

5.1 Features

5.1.1 *TF-IDF.*

The most obvious set of features for any text classification task is the words that occur in the text. There exist various ways to include term features in a classification problem. The most simple feature vector is a vector of all unique words that occur in the collection of lyrics with as value either 1 or 0, whether the word occurs in the text or not, respectively. Logically, we can enrich this representation by counting the number of occurrences of each of those words.

A common representation of term occurrence in the field of text mining and information retrieval is TF-IDF. TF stand for term frequency, the number of occurrences of a term in the document. IDF signifies the inverse of the document frequency, one divided by the total number of occurrences of that term in the collection of documents. With this representation, the importance of a term in a document (TF) is combined with the uniqueness of the term in the collection (IDF). A term that occurs in very many documents is automatically penalized as it is divided by the total number of occurrences in the collection. This TF-IDF representation of words in the lyrics are used as features in the classification process.

5.1.2 *Text charactaristics.*

Certain statistical features in lyrics may also be useful in the classification of genre. Characteristics such as the use of long words and a large number of unique words may imply a higher complexity, which could be attributed to a particular genre. The same holds for the use of particular punctuation marks, number of lines of text and the average length of lines. All statistical features along with their description are presented in table 2.

5.1.3 *Rhyme.*

Since only textual information is considered in this classification application, no auditory features such as melody, key and beats per minute (BPM) are exploited. To still get a grasp of some melodic and rhythmic information, we can include rhyme features. As shown in previous research, rhyme structures can be effective features in the classification of genre [5]. In the study by Mayer et al., various rhyming schemes are included, such as AABB and ABAB, among others. In their study, they had a set of 397 songs, multiple orders of magnitude fewer than in the current study. This poses a limitation on the complexity of features that can be extracted from the data set within reasonable time; a trade-off for a large corpus. To still effectively take rhyme into account, a dictionary with all unique terms in the collection as keys, and their rhyme words as value was constructed before the extraction of features. Rhyme words were computed by a (slightly altered) script written by user *qpwo* on Github.[3] This script defines two words to rhyme if their phonemes

[3]https://github.com/qpwo/python-rhyme

Feature	Description
Word count	number of words / average number of words
punctuation count (period, question mark, comma, apostrophe / double quote, colon, hyphen)	count of each punctuation mark / number of words
digits [0-9]	count / number of words
average word length	average number of characters split by whitespace
number of words	in document / average over collection
fraction of unique words	nr unique words / total number of words
words per line	average number of words per line in the lyrics
unique words per line	average number of unique words per line in the lyrics
syllables per word	average number of syllables per word

Table 2: Lyrics characteristics features

from the stressed syllable to the end are the same. This accurate heuristic allows for a faster computation of rhyme words than the cmudict package provided by nltk (Natural Lanuage ToolKit), a renowned Python toolkit for datascience. The rhyme features included in the classification are presented in table 3.

Feature	Description
AA rhyme	The number of occurrences of the end of one line rhyming with the end of the next line, normalized by the number of lines
AxA rhyme	The number of occurrences of the end of one line rhyming with the end of the second subsequent line, normalized by the number of lines
in-line rhyme	the fraction of words in the same line that rhyme with another word in that line

Table 3: Rhyme features

5.1.4 *POS tags.*

Part-of-speech (POS) tags such as noun, verb, adjective, etc. are often examined in natural language classification. The use of certain word-types may be more abundant in one genre than another. These part-of-speech tags are extracted from the lyrics and then vectorized as was done with the words, this time using the normalized counts and not the TF-IDF values. These features are then filtered to the best 40% of features, since many POS tags are indifferent to genre and will not contribute to the classification.

Because of the large corpus size, the process of extracting POS tags takes over half an hour for all 35 million words. This limits the ability to test many configurations of these features such as grouping certain varieties of POS tags together (all forms of nouns, singular, plural, uncountable, etc. collapsed to a single feature).

5.1.5 *Stemming.*

The stemming of words comprises the removal of word endings to transform a word to its stem form. For example, the words *listening, listened and listeners* are all transformed to simply *"listen"*. This reduces the dimensionality of the features as fewer unique terms are considered. For the stemming of words, PorterStemmer from the nltk package was used. Unlike in previous research, the stemming of the data had no significant effect on the classification accuracy, often it was slightly worse than without stemming. The fact that stemming of words had a negative effect implies a high value in word-endings. Stemming of the words also took quite long, which is why in the testing of other feature sets, the stemming of data was also omitted.

5.1.6 *Selection.*

After extracting the features that were mentioned in the preceding sections, we obtain a set of features that is rather large. This can be attributed mostly to the TF-IDF features since it includes the TF-IDF value for each unique word in the collection of lyrics. Despite the fact that stop-words and very uncommon words are already removed in the process, many of these words are likely to not include much information about the genre of a particular song. Predictors with such small prediction capacity should be removed. To accommodate this fact, a feature selector is employed. This selector takes the top x% of features that predict the genre best. For experimental purposes, this value is set to the default 10%.

5.2 Classification

Multinomial Naive Bayes is used for classification of the lyrics. This classifier was chosen because of its ability to quickly fit large data sets and still be accurate. Often times in text classification, support vector machines work best, however, their complexity is more than quadratic in the sample size. Therefore it is difficult to fit a data set larger than 10,000 samples. Decision tree classifiers performed worse than multinomial naive bayes classifiers in previous research on lyrics-based genre classification [5]. Furthermore, decision trees are more prone to overfitting on the data without extensive pruning. In the tests conducted in this research, decision tree classifier consistently misclassified around 10% more documents than the Naive Bayes classifier.

To realize the classification threshold mentioned in the introduction, test documents are not directly classified to their most probable class. First, the probability of assignment for each class is computed. Normally, the document will be assigned to the class with the highest probability, however large or small it is. If the probability for the most probable class is still rather low, there is a large chance of misclassification. To limit the chance of misclassification, we can introduce a probability threshold which must be surpassed before classification. The implications of this approach is a rise in accuracy at the cost of fewer documents being classified. Such a threshold may be useful the case where misclassification errors should strictly be avoided. Documents that as a result are not classified should then be checked by hand.

5.2.1 Implementation.
The implementation of this project was done in Python (Jupyter Notebook), largely with the use of sklearn datascience packages for feature extraction, selection and pipelining. NLTK packages were used for word tokenization and POS tagging. The pipeline for feature extraction, selection and classification used in this project can be seen in the appendices. The complete code used in this project can be found on the author's Github page.[4]

6 RESULTS

In this section, the results in terms of classification accuracy, feature contribution and certainty threshold trade-off will be discussed.

The classification accuracy obtained with all feature groups and no probability threshold is 60.3%. The baseline is the majority classifier, which always classifies a document as the most frequent class in the data. The baseline accuracy is 27.9%. The confusion matrix in figure 1 depicts the classification accuracies per class and their misclassification rates. The TF-IDF features, as expected, capture most of the information used in the classification process. POS tags did not contribute significantly, adding only 0.3% to the accuracy score, which can be interpreted as random. Rhyme features, albeit few in number, contributed more, with around 3%. Text characteristics features contributed a mere 1% on average.

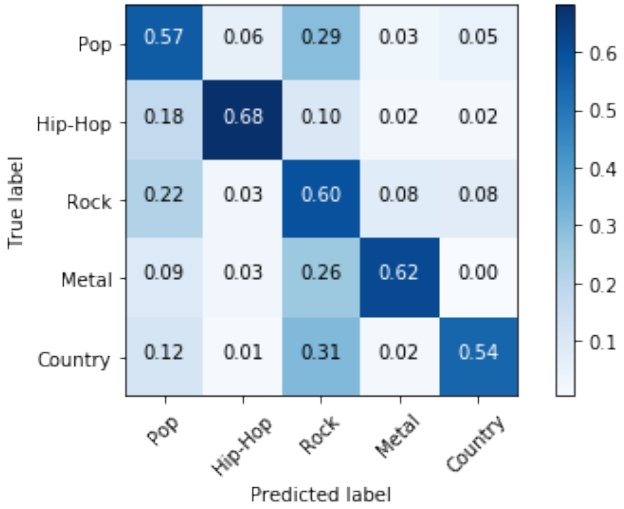

Figure 1: Confusion Matrix

The inclusion of a certainty threshold has an effect of both accuracy and of course the number of documents that are automatically classified. In the graph presented in figure 2 this trade-off is visualized. the x-axis represents the probability threshold which must be surpassed by the genre with the highest probability according to the classifier.

[4]https://github.com/tvdhout/lyrics-classification

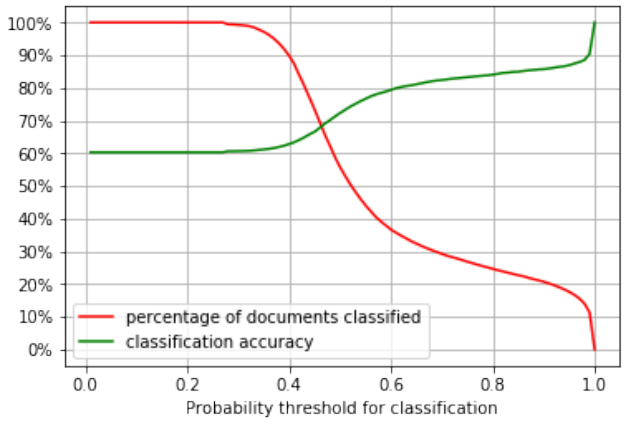

Figure 2: Certainty threshold trade-off

7 CONCLUSION

A classification accuracy of 60.3% over 5 classes is a definite improvement over the baseline. It also matches previous research results, for example [5], in which an average accuracy of around 30% was obtained in a classification problem with 10 genres. They achieved half the accuracy with double the number of classes. From the confusion matrix we can get an idea which genres are most difficult to distinguish from each other. Hip-Hop proved to be the best distinguishable from the other genres given the features, which may be expected due to its unique abundant use of colloquialisms and recognizable slurs. Pop and Rock are quite similar, which is observable in the matrix. The same holds for Rock and Metal, between which lie a broad gradient of sub-genres. A surprisingly large misclassification rate is observed between country and rock, solely unidirectional. This could be attributed to the smaller number of country songs in the corpus.

From the plot depicting the trade-off between classification percentage and accuracy as a result of the certainty threshold we can extract useful information. It is clear the number of classified documents drops very quickly as the threshold reaches above 0.4. A 10% accuracy increase from 60% to 70% results in nearly half of the documents not being classified. In certain scenarios where classification errors should strictly be avoided, such a certainty threshold may be helpful, but doubtfully so in this application.

8 DISCUSSION

As mentioned before, data accuracy is a vital cog in the appropriately named field of data science. Since the data set used in this research is created without supervision or quality control of any kind, it functions as an approximation of the necessary data for such classification exercises. Many lyrics are riddled with spelling mistakes, non-words, word contractions and expansions (e.g. babyyyy, miiine), and excessive or inexistent use of punctuation. Future research can undoubtedly profit from a more carefully constructed data set. It would be interesting to see if other classification problems can benefit more form a certainty threshold. Future research is also advised to delve deeper into the possible inclusion of features such as readability measures, verse and chorus distinction and more expansive rhyme schemas.

REFERENCES

[1] Anthony Canicatti. 2016. Song Genre Classification via Lyric Text Mining. In *Proceedings of the International Conference on Data Mining (DMIN)*. The Steering Committee of The World Congress in Computer Science, 44.

[2] Michael Haggblade, Yang Hong, and Kenny Kao. 2011. Music genre classification. *Department of Computer Science, Stanford University* (2011).

[3] Xiao Hu and J Stephen Downie. 2010. Improving mood classification in music digital libraries by combining lyrics and audio. In *Proceedings of the 10th annual joint conference on Digital libraries*. ACM, 159–168.

[4] Xiao Hu, J Stephen Downie, and Andreas F Ehmann. 2009. Lyric text mining in music mood classification. *American music* 183, 5,049 (2009), 2–209.

[5] Rudolf Mayer, Robert Neumayer, and Andreas Rauber. 2008. Rhyme and Style Features for Musical Genre Classification by Song Lyrics.. In *Ismir*. 337–342.

[6] Cory McKay, John Ashley Burgoyne, Jason Hockman, Jordan BL Smith, Gabriel Vigliensoni, and Ichiro Fujinaga. 2010. Evaluating the Genre Classification Performance of Lyrical Features Relative to Audio, Symbolic and Cultural Features.. In *ISMIR*. 213–218.

[7] Robert Neumayer and Andreas Rauber. 2007. Integration of text and audio features for genre classification in music information retrieval. In *European Conference on Information Retrieval*. Springer, 724–727.

[8] Alexandros Tsaptsinos. 2017. Lyrics-based music genre classification using a hierarchical attention network. *arXiv preprint arXiv:1707.04678* (2017).

[9] Teh Chao Ying, Shyamala Doraisamy, and Lili Nurliyana Abdullah. 2012. Genre and mood classification using lyric features. In *Information Retrieval & Knowledge Management (CAMP), 2012 International Conference on*. IEEE, 260–263.

Appendices

A IMPLEMENTATION PIPELINE

The complete code for this project can be found on the Github page https://github.com/tvdhout/lyrics-classification These appendices will show the most important parts regarding feature extraction and classification pipelining.

A.1 TF-IDF features

This code is used to extract TF-IDF features from the data.

```
vectorizerTFIDF = TfidfVectorizer(stop_words='english', sublinear_tf=True, max_df=0.5, min_df
    =2, ngram_range=(1,1))

selector10 = SelectPercentile(percentile=10) #select best 10% of features

tfidf = Pipeline([
                        #('stemmer', stemTransform), #stemming (not used)
                        ('vectorize', vectorizerTFIDF), #vectorization
                        ('selection', selector10) #feature selection
])
```

A.2 Text characteristics features

This code is used to extract text characteristics from the data

```
prondict = cmudict.dict()

def numsyllables(word): #inspiration: https://datawarrior.wordpress.com/2016/03/29/flesch-
    kincaid-readability-measure/
    try:
        return sum([1 if x[-1].isdigit() else 0 for x in prondict[word.lower()][0]])
    except KeyError:
        return 1

def StatsVectorizer(data):
    print("Stats")
    features = np.zeros((len(data),14), dtype=float)
    avg_nr_words = float(np.mean([len(text.split()) for text in data]))

    progress = IntProgress(min=0, max=len(data)) #progress bar
    display(progress)

    for i in range(len(data)):
        progress.value+=1
        whitespace = re.compile('\s')
        nr_words = len(whitespace.split(data[i]))
        if(nr_words<2):
            continue
        features[i][0] = data[i].count('.')/nr_words
        features[i][1] = data[i].count('?')/nr_words
        features[i][2] = data[i].count(',')/nr_words
        features[i][3] = (data[i].count('"')+data[i].count('\''))/nr_words
        features[i][4] = data[i].count(':')/nr_words
        features[i][5] = data[i].count('-')/nr_words
        features[i][6] = data[i].count('\n')/nr_words
        features[i][7] = nr_words/avg_nr_words #nr words / mean
        features[i][8] = np.mean([len(x.split()) for x in data[i].split('\n')]) #mean nr
            words per line
        features[i][9] = len(set(whitespace.split(data[i])))/nr_words #unique words fraction
```

```
        features[i][10] = np.mean([len(set(whitespace.split(x)))/(len(whitespace.split(x)))
            if len(whitespace.split(x))>0 else 0 for x in data[i].split('\n')]) #unique words
            per line
        features[i][11] = np.mean([len(x) for x in whitespace.split(data[i])]) #word length
        features[i][12] = sum(1 for c in data[i] if c.isdigit())/nr_words
        features[i][13] = sum([numsyllables(word) for word in whitespace.split(re.sub('[^\w\s
            ]','',data[i]))])/nr_words #avg nr syllables

    return features

Statsfeatures = FunctionTransformer(StatsVectorizer, validate=False)
```

A.3 Rhyme features

This code is used to extract rhyme features from the data.

```
def RhymeVectorizer(data):
    print('Rhyme')
    features = np.zeros((len(data),4), dtype=float)

    #progress bar
    progress = IntProgress(min=0, max=len(data))
    display(progress)

    for i in range(len(data)):
        progress.value+=1

        #removes punctuation
        doc = re.sub('[^\w\s]','',data[i].lower())

        #last word of each line
        end_words = [re.sub('[\W]','',line.split()[-1]).lower() if len(line.split())>0 else '
            ' for line in \
                    doc.split('\n')]

        #rhymes dictionary for this document to eliminate the need for many searches in all
            rhymes, just in case
        doc_rhymes = dict([(x, rhymes[x]) if x in rhymes else (x,[]) for x in re.compile('[\s
            ]+').split(doc)])

        #end of line rhymes
        for j in range(len(end_words)-1):
            try:
                features[i][0] += 1 if end_words[j+1] in doc_rhymes[end_words[j]] else 0 #AA
            except: #key error on '', will only try this once
                continue
            try:
                features[i][1] += 1 if end_words[j+2] in doc_rhymes[end_words[j]] else 0 #AxA
            except: #last word index will be out of bounds (+2), skip these features
                continue

        #within line rhyme
        for line in doc.split('\n'):
            for j in range(len(line.split())): #word 1 rhymes with...?
                for k in range(j, len(line.split())): #word 2
                    features[i][2]+=1/len(line.split()) if line.split()[k] in doc_rhymes[line
                        .split()[j]] else 0
```

```
        #normalize
        for f in range(len(features[i])):
            features[i][f]/=len(doc.split('\n'))

    return features

#Create transformer from function, used in the pipeline
Rhymefeatures = FunctionTransformer(RhymeVectorizer, validate=False)
```

A.4 POS features

This code is used to extract POS features from the data.

```
def POSvectorizer3(data):
    '''
    returns the POS tags for all the words in the lyrics
    '''
    print("POS")
    progress = IntProgress(min=0 , max=len(data)) #progress bar
    display(progress)

    for i in range(len(data)):
        progress.value += 1
        data[i] = data[i].lower()
        data[i] = re.sub('[^\w\s]','',data[i])
        data[i] = '␣'.join([x for _,x in pos_tag(word_tokenize(data[i]))])

    return data

getPOS2 = FunctionTransformer(POSvectorizer3, validate=False)
vectorizerPOS = TfidfVectorizer(vocabulary=list(load('help/tagsets/upenn_tagset.pickle').keys
    ()), use_idf=False, lowercase=False)
selector40 = SelectPercentile(percentile=40)

pos = Pipeline([
                    ('getTags', getPOS2), #get POS tags
                    ('vectorize', vectorizerPOS), #vectorization
                    ('selector', selector40) #feature selection
])
```

A.5 Final Pipeline

This code shows the final pipeline which is fit to the data.

```
features = FeatureUnion([
                        ('vectorizer', tfidf),
                        ('stats', Statsfeatures),
                        ('Rhymefeatures', Rhymefeatures),
                        ('POS', pos),
                        ])
mnb = MultinomialNB()
pipeline = Pipeline([
    ('features', features), #extract features
    ('classifier', mnb), #classify
])
```