UNIVERSITY OF BUCHAREST
MASTER SECURITY AND APPLIED LOGIC
DISCIPLINE, PRACTICAL MACHINE LEARNING

Usupervised learning
LDA & Birch

Paraschiv Stefan

January 2024

Contents

1 Introduction

This is a project that is for the Practical Machine Learning course, the main goal is to understand and to put in practice unsupervised machine learning methods. For this task I chose the Latent Dirichlet Allocation (LDA) method which does topic modelling and is very similar to clustering by finding topics that a document belongs to, on the basis of words that it contains. The second method I chose is Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) algorithm which is used to perform hierarchical clustering over particularly large data-sets. These methods are applied on a data-set which has two clusters.

1.1 Task description

For this project there are multiple requirements that need to be fulfilled, which are:

- Testing with two clustering models. Even though LDA isn't exactly a clustering method, it's very similar.
- At least two ways of representing features.
- Parameter tuning for both methods.
- Comparison of the unsupervised methods with a supervised method and random chance.
- Interpretation of the results for both methods.

These requirements have been completed and described in this document. There are also some additional requirements regarding the dataset, it must have at least 1000 samples and it can't be a popular dataset like MNIST.

1.2 Dataset

The dataset employed in this project is "Question Classification: Android or iOS?"[1]. While it might initially seem unconventional for a clustering task, I selected this dataset with a specific objective in mind. The primary goal is to utilize methods that can initially detect the two primary clusters (Android or iOS). Subsequently, the aim is to explore the capability of these methods to identify additional clusters, uncovering potential information specific to other categories within the sentences. This approach seeks to assess the adaptability of clustering methods in extracting nuanced patterns beyond the initially defined clusters.

The dataset comprises 51,370 entries in the training data and 11,008 entries in the test data. Each entry is characterized by a title, a body containing the question, a score provided by StackExchange users, the number of views, the label as a string (iOS/Android), and the label in numeric format (1 for iOS and 0 for Android). The dataset combines numerical features with text features, and for this task, only the text features will be utilized.

[1] https://www.kaggle.com/datasets/xhlulu/question-classification-android-or-ios

2 Data processing

2.1 Features

The features chosen for each method differ; for the LDA approach, I opted for Term Frequency Inverse Document Frequency (TF-IDF) and N-grams, while for the BIRCH approach, I selected TF-IDF and Word2Vec. The reason I refrained from using Word2Vec on LDA is due to its generation of negative values, which may not be comprehended by the LDA model.

Prior to creating these features, I initially dropped the numerical columns ('Id', 'Score', 'ViewCount', and 'Label') and merged the 'Title' column with the 'Body' column. Since LDA performs better on longer documents, I excluded entries with lengths under 30. In contrast, for the BIRCH approach, I retained such entries.

Beyond merging the columns, I commenced by cleaning the text, eliminating HTML tags, punctuation, and other characters that could introduce noise into the document. To ensure uniformity across all words, I applied stemming, reducing each word to its root form. For instance, the word *programming* becomes *program*, and *programmer* is also represented as *program*. This preprocessing step aids the model in capturing the essential meaning of words. As the two methods require different inputs, I will segment the feature creation into two subsections.

2.1.1 BIRCH features

The initial feature representation for this model is the **TF-IDF** (Term Frequency-Inverse Document Frequency), implemented using the scikit-learn Python library. Initially, processing all documents simultaneously was impractical due to the vast number of available features, as my laptop lacks sufficient memory and would crash. Consequently, after conducting experiments with various parameters for the TfidfVectorizer, the optimal settings for dimension reduction were empirically determined to be min_df=0.025 and max_features=60000.

TF-IDF is a method that converts text into a meaningful numerical representation, commonly employed for training machine learning algorithms for prediction. This method involves two steps: first, it calculates Term Frequency (TF), and then Inverse Document Frequency (IDF). In the initial step, TF represents a document as a vector of words with their respective frequencies. Higher frequencies indicate greater importance to the document.

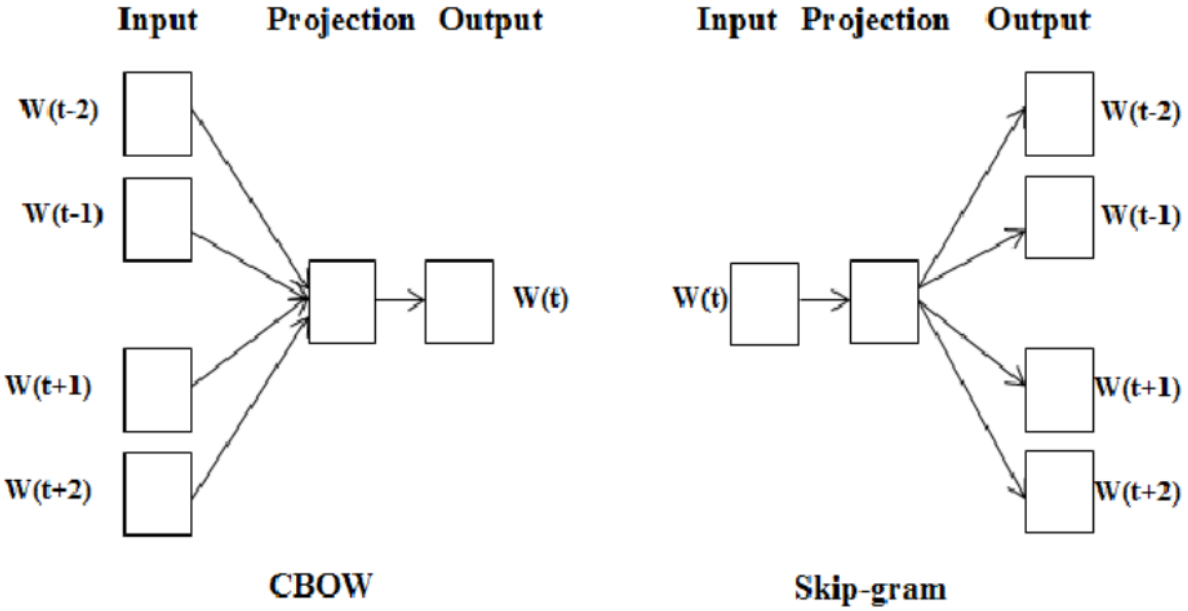
After creating these vectors, the IDF phase begins. This step assesses the commonality or rarity of a word across all documents. The closer the value is to 0, the more common the word is. Once these two values are determined, they are multiplied to yield a TF-IDF score. A higher score indicates greater relevance of the word. An example from our dataset is as follows:

	address	...	web	week	went	wifi	window	wipe	wonder
0	0.0	...	0.000000	0.0	0.00000	0.216695	0.0	0.0	0.0

Observing the TF-IDF score for "wifi," which is 0.21, suggests its potential relevance in our representation. The min_df parameter, set to 0.025, filters out words appearing in less than 2.5% of the documents. Additionally, the max_features parameter constructs a vocabulary comprising the top 60,000 words, ranked by term frequency across the documents. Following the acquisition of these features, dimensionality reduction was further implemented using PCA.

The alternative feature representation is **Word2Vec**, utilizing the Gensim Python library implementation. This method encompasses two algorithms: Continuous Bag of Words (CBOW) and skip-gram. The distinction between these approaches lies in the prediction task. In the

CBOW version, the objective is to predict a target word based on the surrounding context words, while in the skip-gram version, the aim is to predict the surroundings based on a single word.



For this project, I deemed CBOW more suitable as it provides better representations for terms that are more frequently used. Before employing Word2Vec, sentence tokenization is necessary to create a One-Hot representation that the model can utilize. To reduce dimensionality and obtain a 1D vector of features, the approach involves using the average Word2Vec vector. Essentially, this entails seeking the 'average' meaning of a sentence. For instance, if many vectors from a sentence are close to each other, the average value will approximate the cluster and serve as a meaningful representation for the sentence. While this approach may not be suitable if word order is crucial, it is acceptable in this context where word order does not hold significance.

2.1.2 LDA features

For this model, I opted for **TF_IDF** and **n-gram** with tf_idf. The distinction lies in the use of the Gensim implementation instead of the sklearn version, although the difference is minimal. The rationale behind this choice is the necessity to cater to the LDA model's specific input requirements. To utilize the Gensim library, I initially had to create a dictionary of words (a unique list of words) and represent them as a Bag of Words (word frequency). While this approach involves additional steps compared to the sklearn implementation, it aligns with the LDA model's input specifications. Following this, I pruned words with a value below 0.025 to eliminate potential sources of noise.

For the second type of feature, I opted for **bigrams** as they provide a minimal context. Bigrams, consisting of sequences of two words, enable the creation of more informative features, such as "phone wifi," while still encompassing features generated from individual words. Bigrams are valuable for predicting the next most likely word based on probabilities. Following the extraction of bigrams, I employed tf-idf once again for feature representation.

2.2 Parameter Tuning

For parameter tuning the approach for the two models differ iterate through possible parameters and calculate a score for each possible combination. The evaluation for the two models differ.

For the **BIRCH** method, three possible metrics can be employed to evaluate the clusters: Davies-Bouldin Index, Silhouette Coefficient, and Calinski-Harabasz Index. The Davies-Bouldin Index averages cluster similarity, with a lower value indicating well-defined clusters. The Calinski-Harabasz Index, also known as the Variance Ratio Criterion, is calculated as a ratio of the sum of inter-cluster dispersion to the sum of intra-cluster dispersion for all clusters. A higher coefficient signifies better performance. The Silhouette Coefficient, which measures the 'density' of clusters, was chosen as the metric to evaluate the model's performance. The Silhouette Coefficient ranges from -1 to 1, with -1 indicating misassignment of clusters, 0 indicating overlapping clusters, and 1 indicating well-defined and separate clusters. The formula for this score is:

$Score = (b - a)/max(a, b)$

b - average inter-cluster distance (average distance between each point within a cluster)

a - average intra-cluster distance (average distance between all clusters)

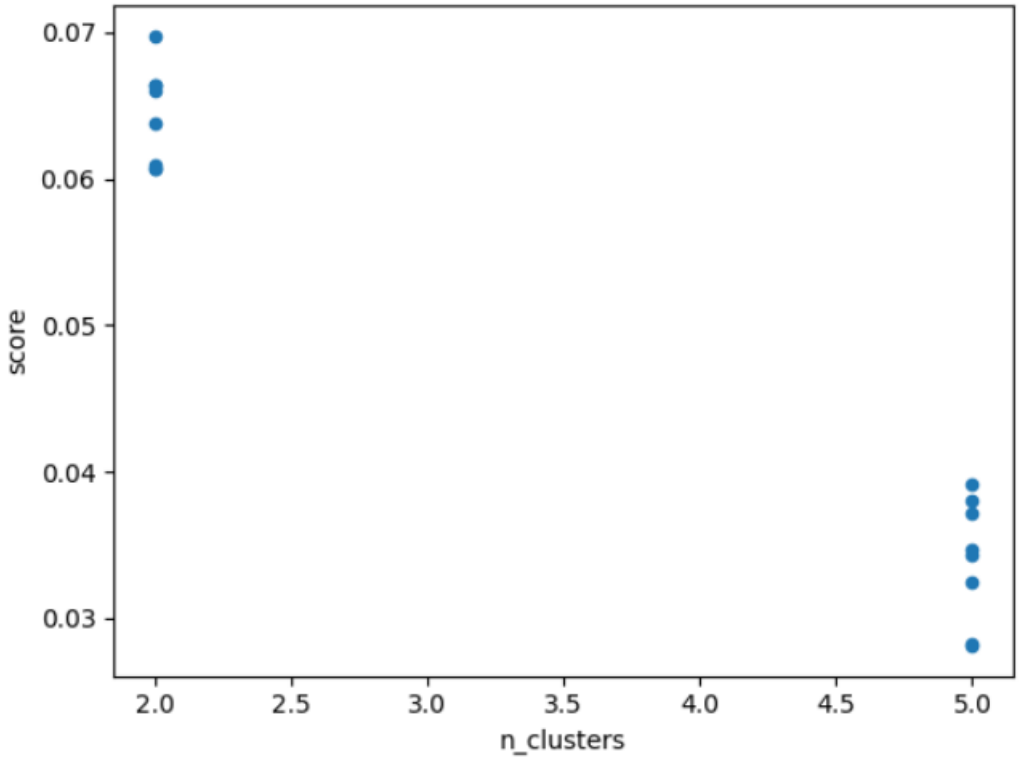
With this score I determined the best parameters for the BIRCH model, the parameters I considered to be important are: branching factor,number of clusters and threshold.
I will now present the best parameters for both of the features, **tf_idf** and **word2vec**. For the **word2vec** feature I tried the following values:

- branching factor - values tried [25,50]
- number of clusters - values tried [2,5]
- threshold - values tried [0,1,0.2,0.5,0.7]

In the table below we can see the results and combinations for our parameters for the **word2vec** features.

	branching_factor	n_clusters	threshold	score
0	25	2	0.1	0.066080
1	25	2	0.2	0.066368
2	25	2	0.5	0.060714
3	25	2	0.7	0.069774
4	25	5	0.1	0.028282
5	25	5	0.2	0.038001
6	25	5	0.5	0.032428
7	25	5	0.7	0.028157
8	50	2	0.1	0.066347
9	50	2	0.2	0.063820
10	50	2	0.5	0.061009
11	50	2	0.7	0.060734
12	50	5	0.1	0.039156
13	50	5	0.2	0.037159
14	50	5	0.5	0.034379
15	50	5	0.7	0.034750

The reason why branching factor doesn't have greater values is because of computational reasons. Graph for seeing the number of clusters for **word2vec**:



From the plot and table we can see pretty clear that the best values for this model are:
Branching factor:25-Number of clusters:2-Threshold:0.7

The observed poor scores with Word2Vec embeddings could be attributed to a potential issue where the word vectors have very close values. This becomes a concern, especially when averaging these vectors, as they might become even closer, making it challenging to capture meaningful distinctions.

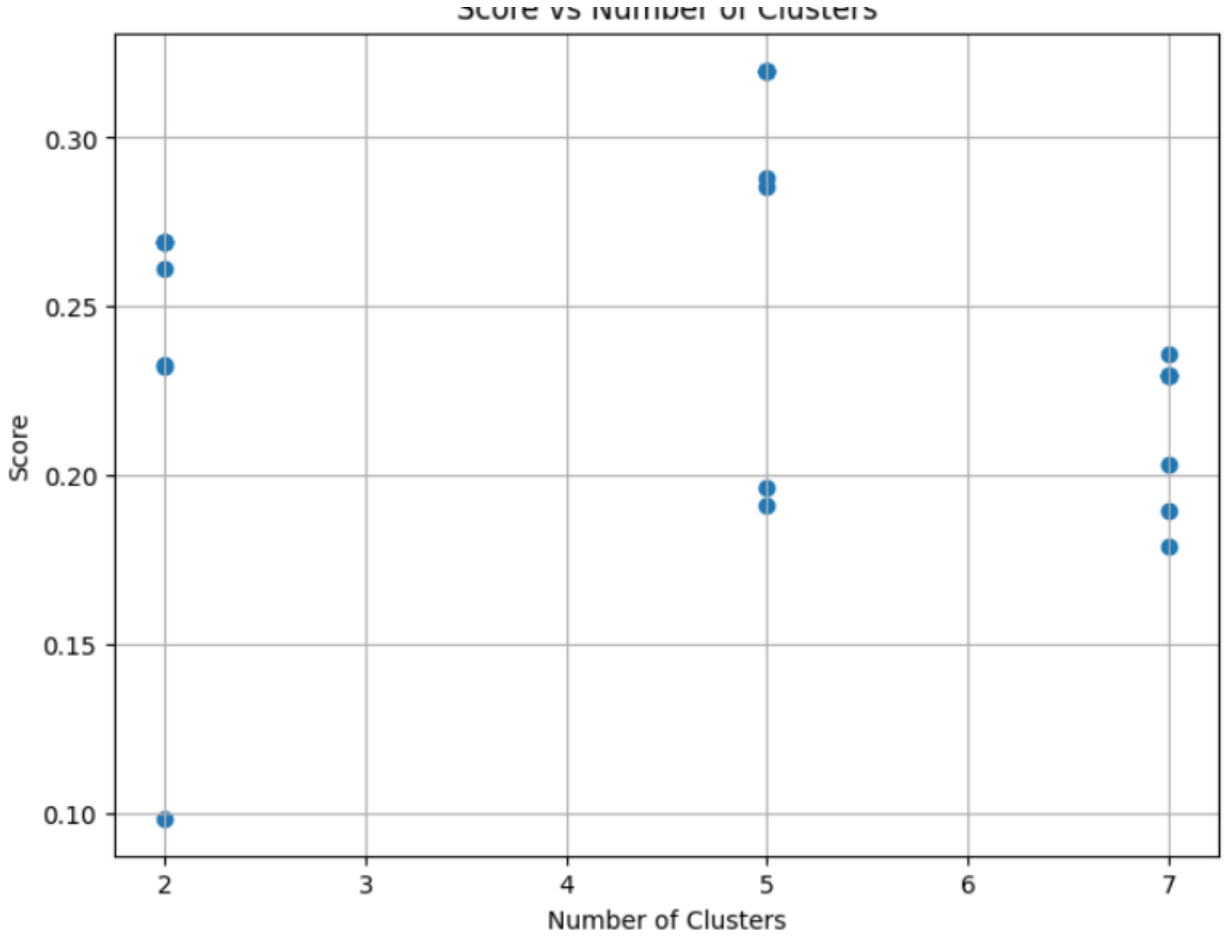
For the **tf_idf** features I tried the following values:

- branching factor - values tried [25,50,100,150]
- number of clusters - values tried [2,5,7]
- threshold - values tried [0,1,0.2,0.5]

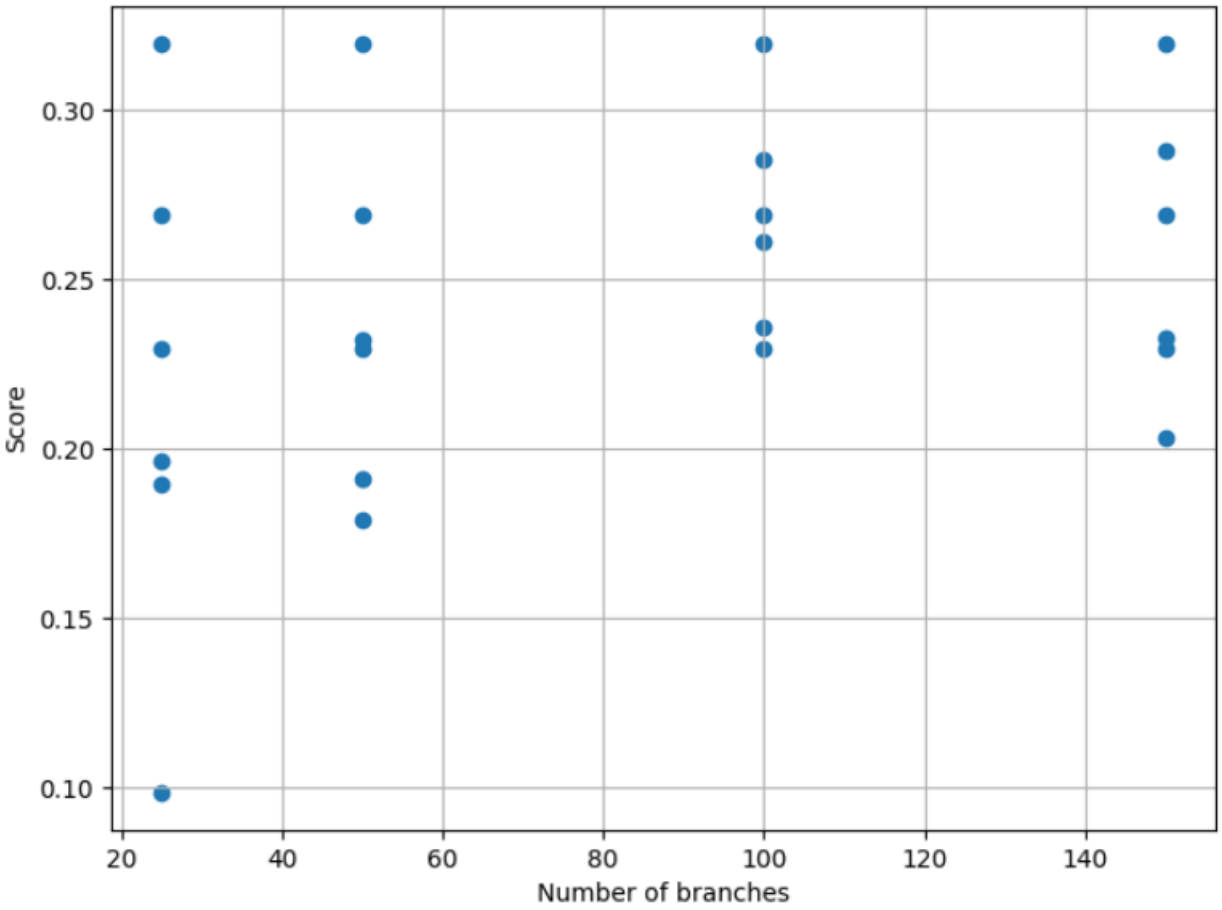
In the table below we can see the results and combinations for our parameters for the **tf_idf** features.

	branching_factor	n_clusters	threshold	score
1	25	2	0.1	0.4629216976919304
2	25	5	0.1	0.3301209687461191
3	25	7	0.1	0.3376851762023012
4	50	2	0.1	0.4629216976919304
5	50	5	0.1	0.3301209687461191
6	50	7	0.1	0.3376851762023012
7	100	2	0.1	0.4629216976919304
8	100	5	0.1	0.3301209687461191
9	100	7	0.1	0.3376851762023012

Graph for number of clusters:



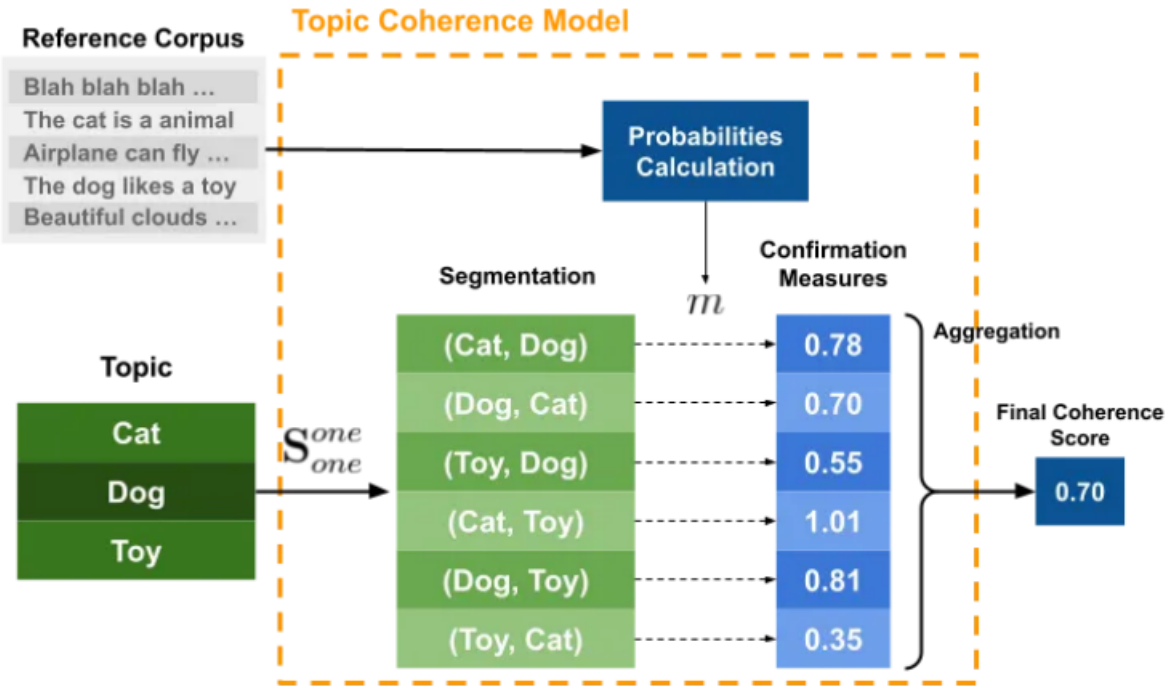
Graph number of branch factorization:



We can observe that for this feature the scores are better, another thing we can observe is that the threshold has a low value. If the BIRCH threshold is set to 0.1, it means that during the construction of the clustering structure, the algorithm will attempt to form subclusters in such a way that the diameter of each subcluster does not exceed the specified threshold of 0.1. In other words, the algorithm will try to create compact and relatively tight subclusters, ensuring that the points within each subcluster are not too far apart. Setting the threshold is a way to control the granularity of the clustering. A smaller threshold will result in more, smaller subclusters, potentially capturing finer details in the data but also increasing the computational cost. On the other hand, a larger threshold may lead to fewer, larger subclusters, which can be computationally more efficient but might overlook finer patterns in the data. Best parameters for BIRCH an tf_idf:

Branching factor:25-Number of clusters:2-Threshold:0.1

For the **LDA** model, I identified coherence and perplexity as the most suitable metrics for evaluating topics. I utilized both metrics during the hyperparameter tuning process. The reason for performing hyperparameter tuning for only one feature in this context is that the features are quite similar, and the tuned parameters will effectively apply to both features. The topic coherence measure operates as a pipeline, taking topics and the corpus as inputs and producing a single value that represents the topic coherence.



The calculation of the coherence score can be decided in multiple steps and ca be viewed as a pipeline:

- Segmentation - this is the process where we take the most important words of a topic and create subsets pairs from these words. We do this because when we calculate the final coherence score we want to see the relationship between each word from the subset created. For example if W = 'phone','android',root we can create with segmentation the subpairs ('phone','android','root'),... where phone is W1 and android and root is 'root'.
- Probability calculation -this phase happens after the segmentation step and calculates the occurence probabilities of a certain word or pair of words. These probabilities are used later to consolidate the score of the topic.
- Confirmation Measure - in this step we compute how well the subset W* supports the subset W', we do that with the help of the probabilities calculated earlier, and gen a confirmation score which is high if W' is occurring often in W*.
- Aggregation - the final step, this basically takes the confirmation measures and then aggregates them into a single value.

The Gensim library provides implementations for multiple coherence models: u_mass, c_v,c_uci,c_npmi. These models are very similar and differ in the way they calculate the segmentation , probability or the confirmation measure. For the LDA model I chose to use the c_v method of the coherence score.

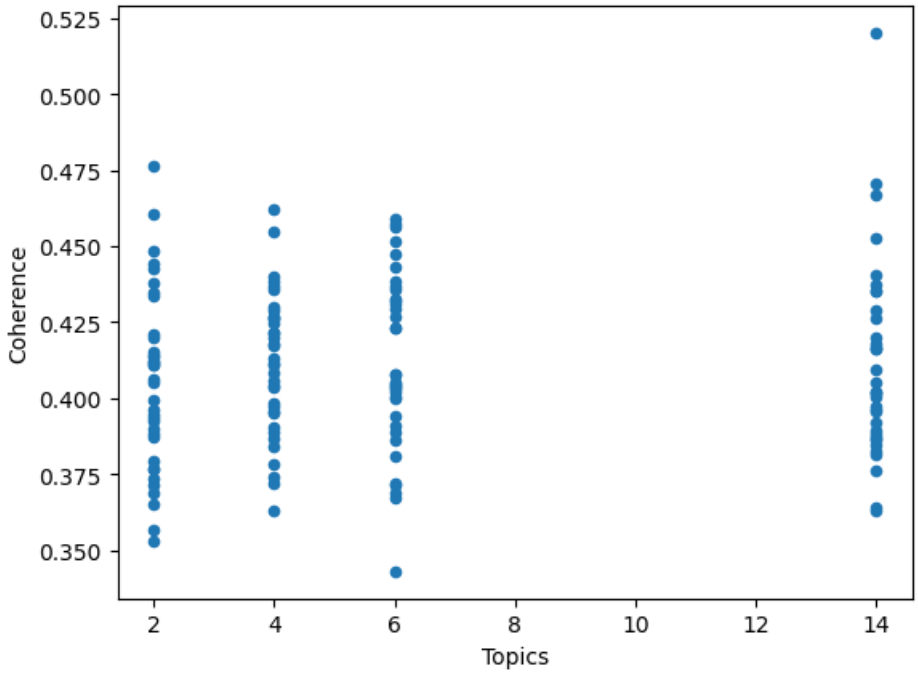
For the LDA model the parameters that were tuned are:

- number of topics - values tried: [4,2,6,14]
- alpha - values tried: ["asymmetric","symmetric",0.01,0.16,0.5,0.1]
- beta - values tried: ["symmetric",0.01,0.5,0.16,1,0.01]

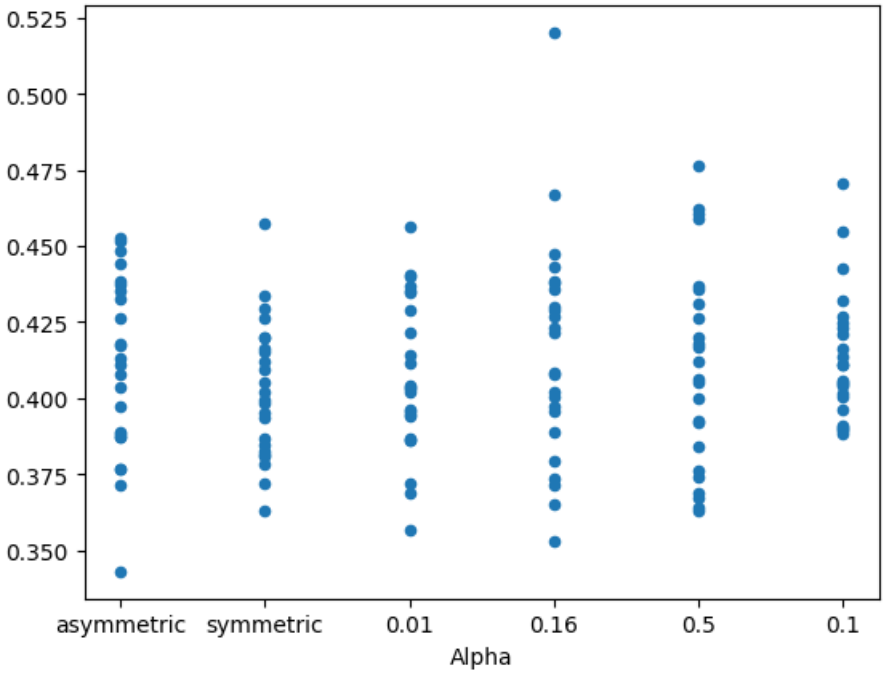
In the table below we have our scores:

	Topics	Alpha	Beta	Coherence
0	4	asymmetric	symmetric	0.426022
1	4	asymmetric	0.01	0.413166
2	4	asymmetric	0.5	0.403794
3	4	asymmetric	0.16	0.417149
4	4	asymmetric	1	0.397277
...	...	...	...	...
139	14	0.1	0.01	0.415971
140	14	0.1	0.5	0.470684
141	14	0.1	0.16	0.389263
142	14	0.1	1	0.405194
143	14	0.1	0.01	0.401411

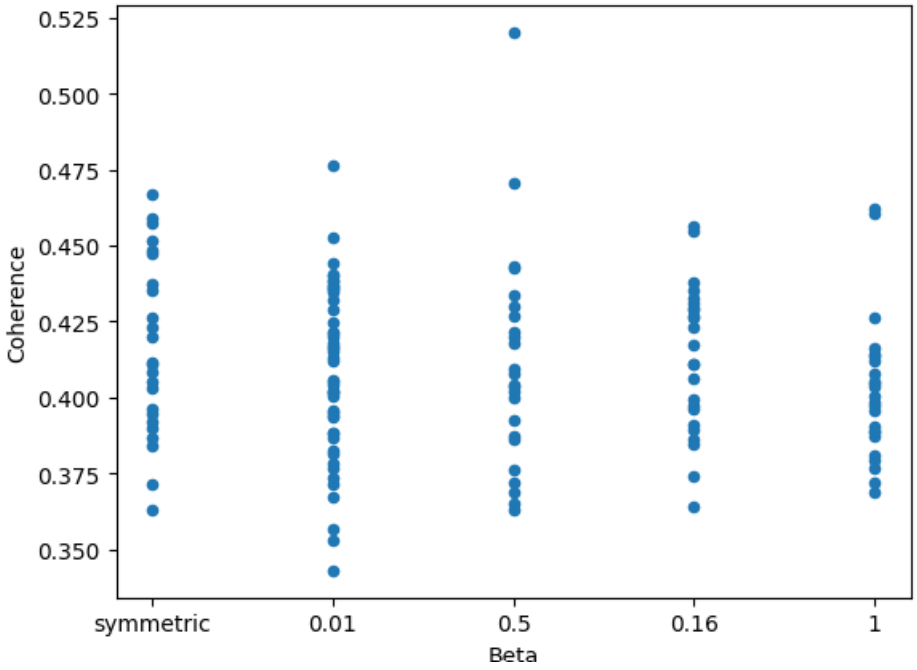
Graph for number of topics/clusters:



Graph for number of alpha:



Graph for number of beta:



From the plot and table we can see pretty clear that the best values for this model are:

Topics:14-Alpha:0.16-Beta:0.5

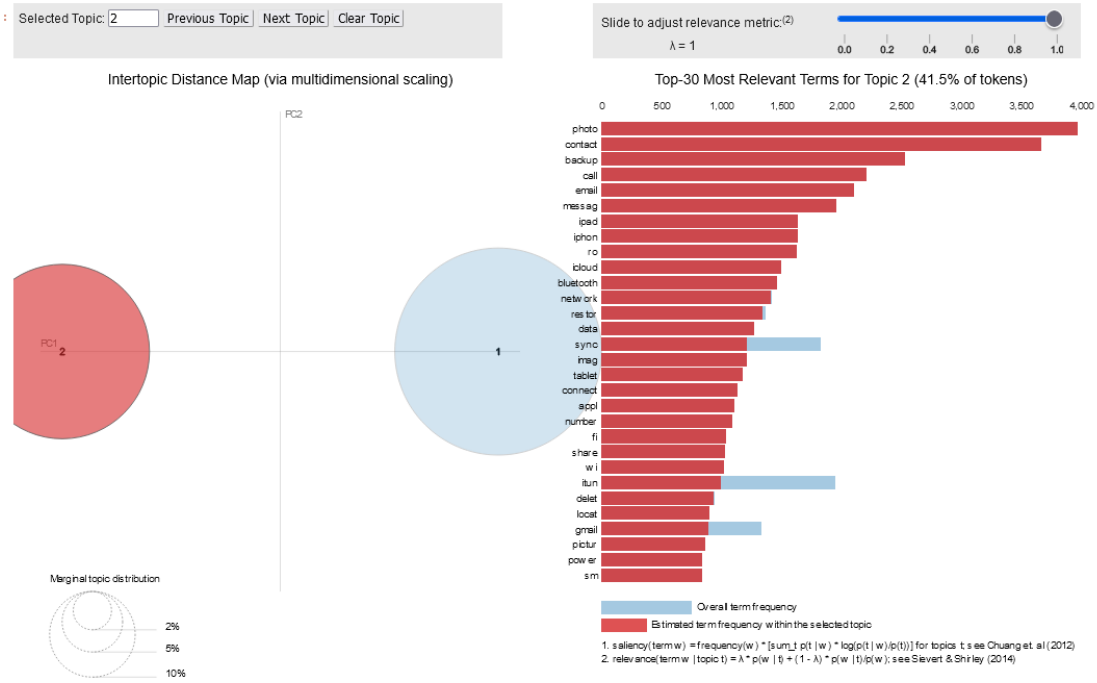
This will be an interesting experiment to see what topics the model found, the parameters we will be using will have the number of topics two since that is the task for this dataset.

Topics:2-Alpha:asymmetric-Beta:symmetric

3 Results

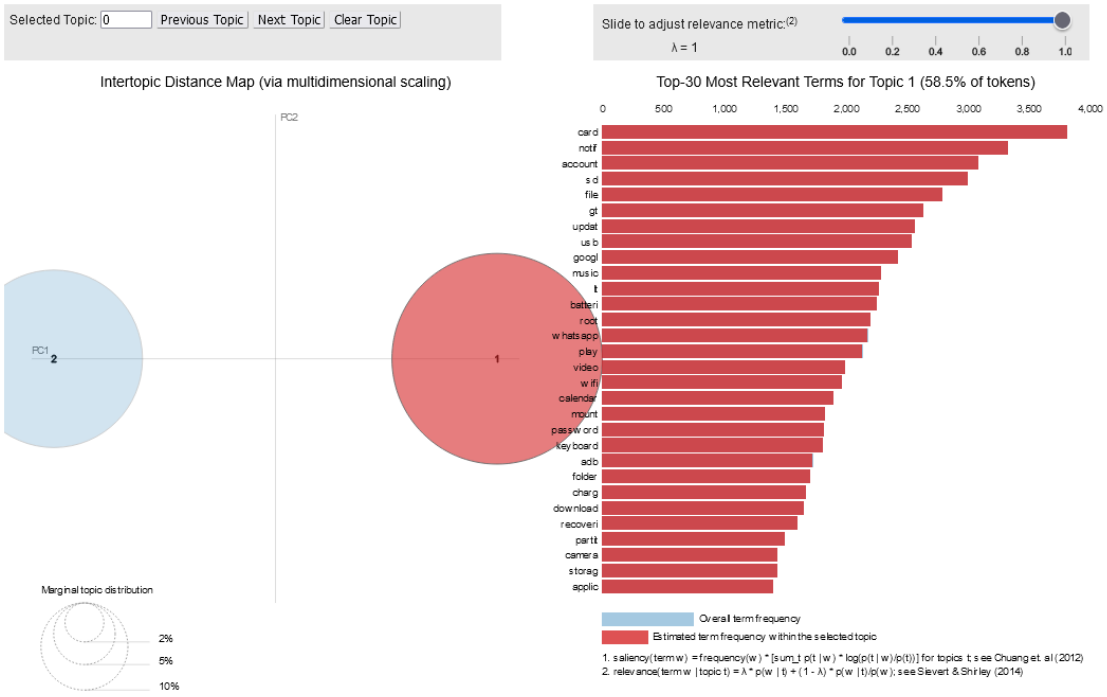
3.1 LDA approach

For the LDA model, I will initially address the obtained clusters before examining the results on the test data. As illustrated below, we observe two clusters, with the first cluster presented in the image below:



In the first topic these are the top 10 key words: [’0.011*”card” + 0.010*”notif” + 0.009*”account” + 0.009*”sd” + 0.008*”file” + 0.008*”gt” + 0.008*”updat” + 0.008*”usb” + 0.007*”googl” + 0.007*”music”’]

The second cluster is presented in the image below:



In the second topic these are the top 10 key words: [’0.019*”photo” + 0.018*”contact” + 0.012*”backup” + 0.011*”call” + 0.010*”email” + 0.009*”messag” + 0.008*”ipad” + 0.008*”iphon” + 0.008*”ro” + 0.007*”icloud”’]

After analyzing these topics, it becomes evident that the first topic pertains to Android, while the second topic is associated with iOS. I anticipated that the Android topic would include words such as ”android,” ”Samsung,” and ”Galaxy,” analogous to how the first topic encompasses ”iPad,” ”iPhone,” and ”iCloud.” Additionally, notable common topics, such as ”sync,” emerge from the analysis.

The scores for **n-gram**:
Perplexity: -7.54051327966975
Coherence Score: 0.5940579270471378

The scores for **tf_idf**:
Perplexity: -8.1031138346081
Coherence Score: 0.410347110138202

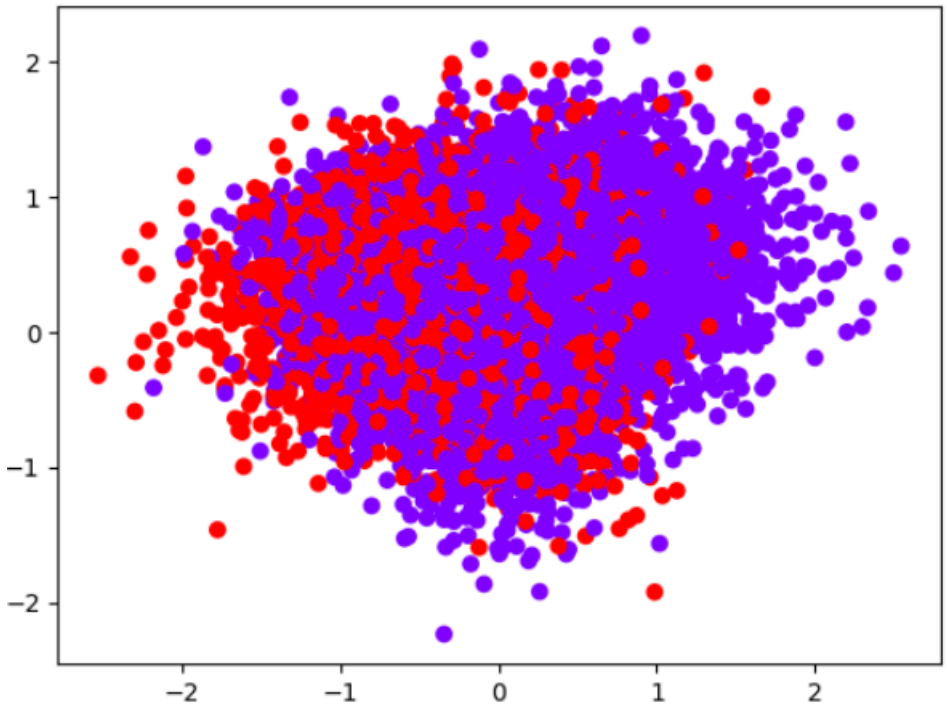
To benchmark it against a supervised baseline, I computed the accuracy of the model. When encountering an unpredicted document, the output comprises two probabilities (given the two clusters) with corresponding scores for each topic. The higher probability determines the assigned topic or class.

With this model, I achieved an accuracy of 51% and a macro F1-score of 47% using **tf_idf**. Similarly, with **n-grams**, I attained an accuracy of 51% and an F1-score of 48%.

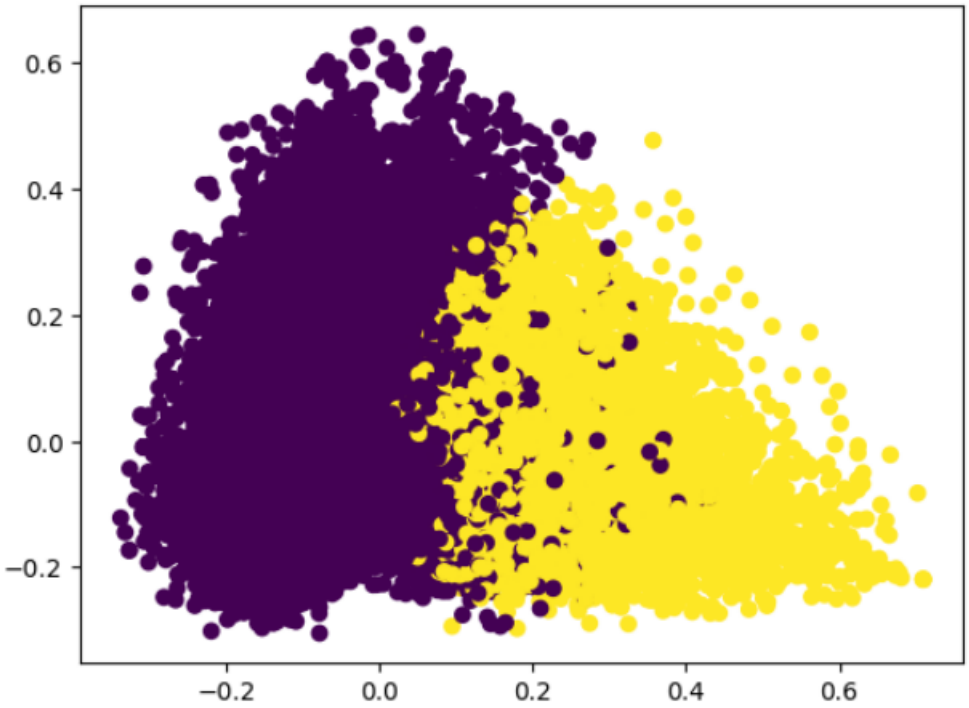
3.2 Birch approach

For the Birch model I want to discuss the clusters that I obtained and then see the results on the test data.

For the **word2vec** representation, the 'clusters' are really bad.



These are the clusters obtained with the best feature, **tf_idf**.



As observed, the clusters are distinctly separated, yielding the following scores: Silhouette score: 0.3112585001852664

In comparison to a supervised baseline, I computed the model's accuracy, which stands at 82

3.3 Compare with supervised basline and random chance

In this section, I will focus exclusively on the optimal outcomes obtained from unsupervised models, specifically the BIRCH model achieving an accuracy of 82% and the LDA model achieving an accuracy of 51%. As a point of reference in the supervised baseline, I employed the Random Forest model, yielding an accuracy of 75%, and a random chance assignment (where vectors are randomly assigned with 1 and 0), resulting in an accuracy of 72%.