U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

CAC Project

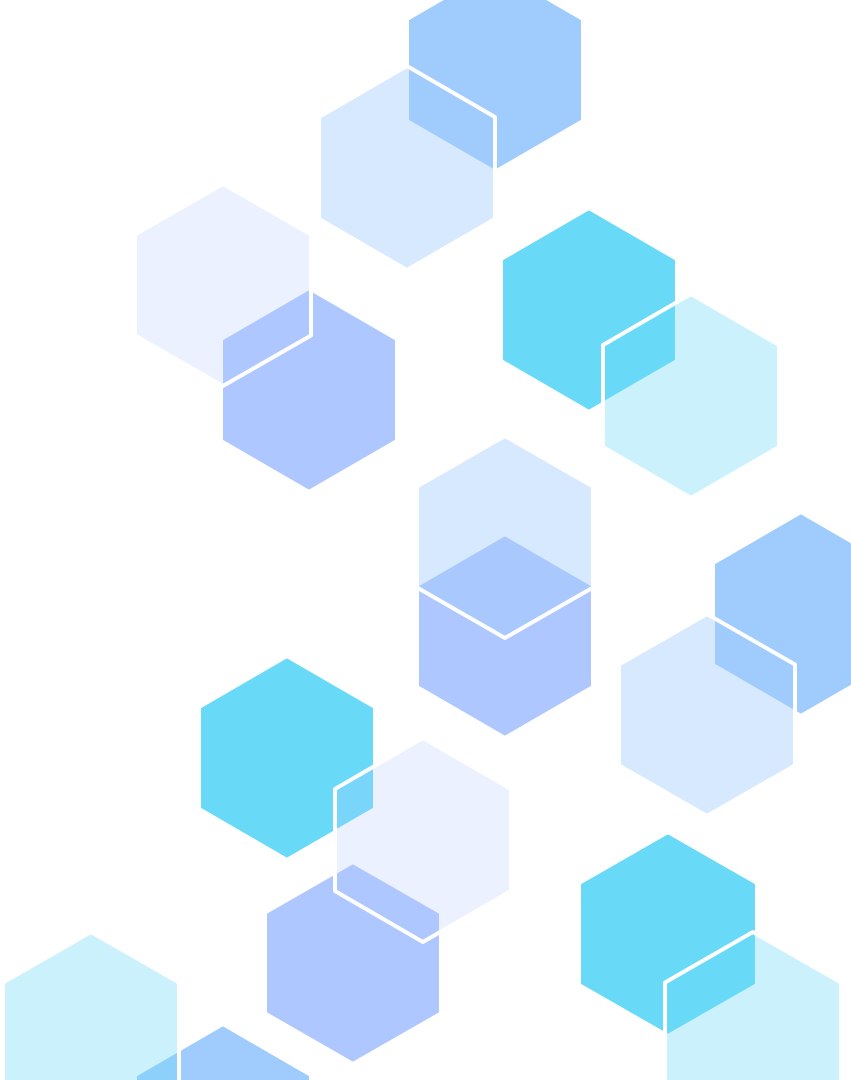
Topic Modelling
Time Series

Group H
André Barbosa – 202007398
José Araújo - 202007921
José Ribeiro - 202007231

Table of contents

01

Introduction



Project Description

NLP

Creation of recipe topics using NLP tools and analysis.

+

Time Series

Usage of Time Series and predictive models to predict the popularity of topics.

Project Goals

- Creation of topics resorting to NLP techniques.
- Predicting the popularity of a recipe topic using time series data.
- Analyze and take conclusions on the time series data.

02

Dataset

Hummus Dataset

Data Files

- The Hummus Dataset, used in this project, contains three csv files:
    - members.csv
    - recipes.csv
    - reviews.csv

Data Preprocessing

For preprocessing, firstly, we needed to pre-process the **ingredients** column of the recipes dataset.

The final pre-processed version had therefore an additional column named **ingredients_pp**, which consisted of a list of ingredients (without their quantities) of each recipe.

```
import ast

# x for the row, ing_or_quant for the result column to return, ingredients (0) or
def ing_process(x, ing_or_quant):

    try:
        ing_list = ast.literal_eval(x)
    except:
        print(x)
        return None

    try:
        res = list(ing_list.values())[0]
    except:
        print(ing_list)
        return None

    # for the ingredients return the


    return [x[ing_or_quant] for x in res]


recipes['ingredients_pp'] = recipes['ingredients'].apply(ing_process, args=(0,))
```

Data preparation/preprocessing

When sampling data, we went for 2 options:

- **10,000** recipes with the highest number of reviews;
- **10,000** of the most recent recipes.

Then for the NLP models, we cleared the data that didn't have any **description** or **ingredients**, and based on these two, we separated data by 3 attribute modes:

- Description only;
- Ingredients only;
- Both Ingredients and Description

03

NLP

Text processing

In order to apply the vectorizer later on, we applied 3 different processing techniques individually to the data, **lemmatization**, **stemming** and **tokenization**.

Firstly, only keep lowercase letter characters:

```
text_sample = text_sample.apply(lambda x: re.sub('[^a-zA-Z]', ' ', x))
text_sample = text_sample.apply(lambda x: x.lower())
```

Then apply the processing:

Lemmatization:

```
def clean_text_lemmatization(text_sample):
    lemmatizer = WordNetLemmatizer()
    sw = set(stopwords.words('english'))
    text_sample = text_sample.apply(lambda x:
        ' '.join([lemmatizer.lemmatize(w,
        get_wordnet_pos(w)) for w in x.split() if
        w not in sw]))
    return text_sample
```

Stemming:

```
def clean_text_stemming(text_sample):
    ps = PorterStemmer()
    sw = set(stopwords.words('english'))
    text_sample = text_sample.apply(lambda x:
        ' '.join([ps.stem(w) for w in x.split() if w
        not in sw]))
    return text_sample
```

Tokenization:

```
def clean_text_tokenization(text_sample):
    sw = set(stopwords.words('english'))
    text_sample = text_sample.apply(lambda x:
        ' '.join([w for w in x.split() if w not in
        sw]))
    return text_sample
```

Apply Vectorizers

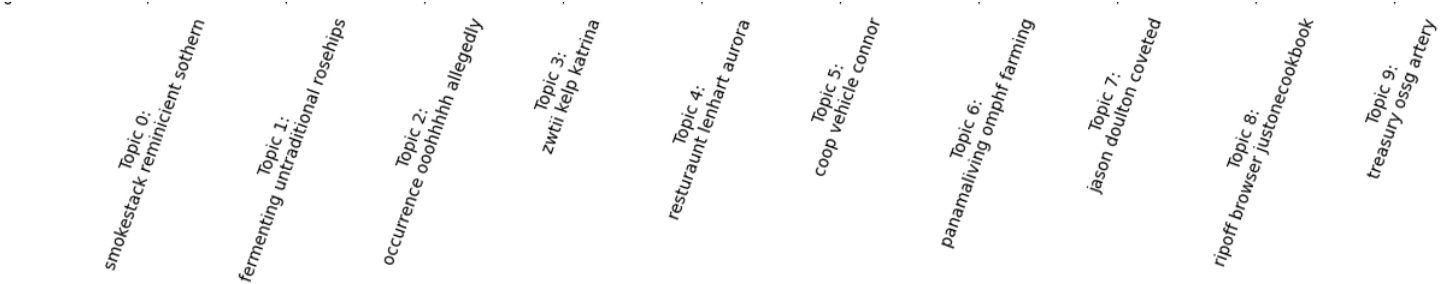
For the later step of showing the clustering of recipes, we first needed to build the sparse vectors; for this we applied 2 different types of vectorizers:

- Count Vectorizer
- Tf-Idf Vectorizer

Topic Modelling

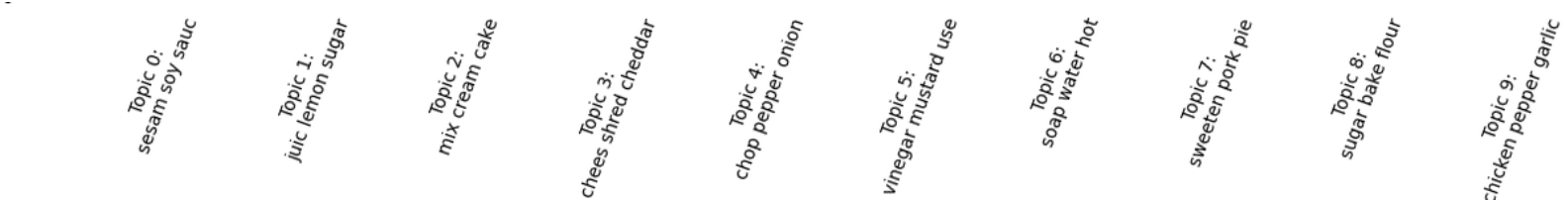
- In order to characterize each topic we decided to apply a function to get the top 3 words in the topic.

- Then we tried to apply TF-IDF per each topic to get the most relevant ones.



Topic Modelling

- We decided to apply the top frequent words, but with a TF-IDF transformer that would exclude non-relevant words from the topics.



LDA Model

For every attribute combination previously mentioned we applied LDA to the data, splitting it in a total of **10** topics:

- Highest frequency: 2883
- Lowest frequency: 60
- Total recipes: 9691

Similar performance models are on annexes

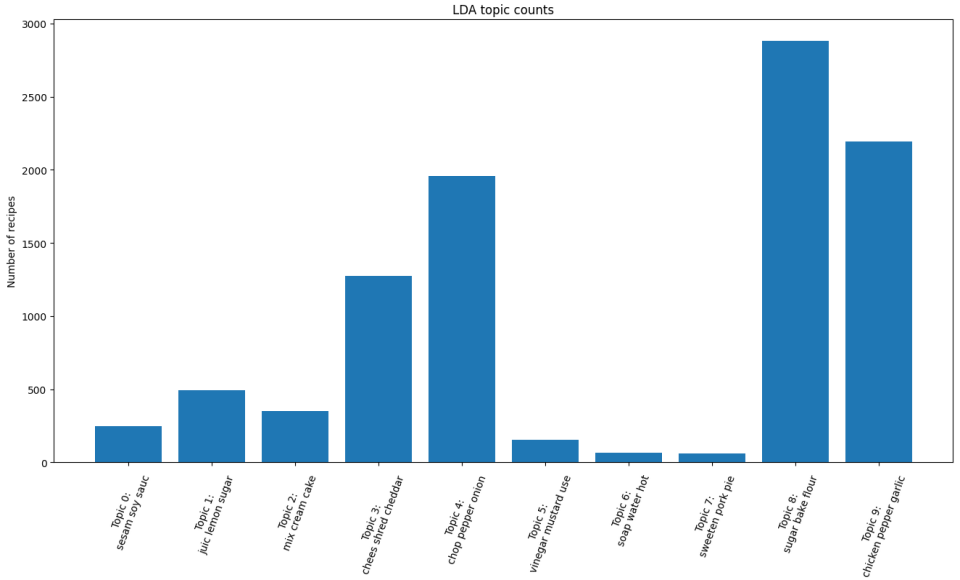


Fig. 1: Frequency of topics

LDA Topic clustering

- We then applied the clustering for the topics.
- We used t-SNE Clustering to confirm the distribution of the topics in a graphical way.
- Our best mode was the combination between the dataset with most amount of reviews, using ingredients only, with stemming applied, using count vectorizer

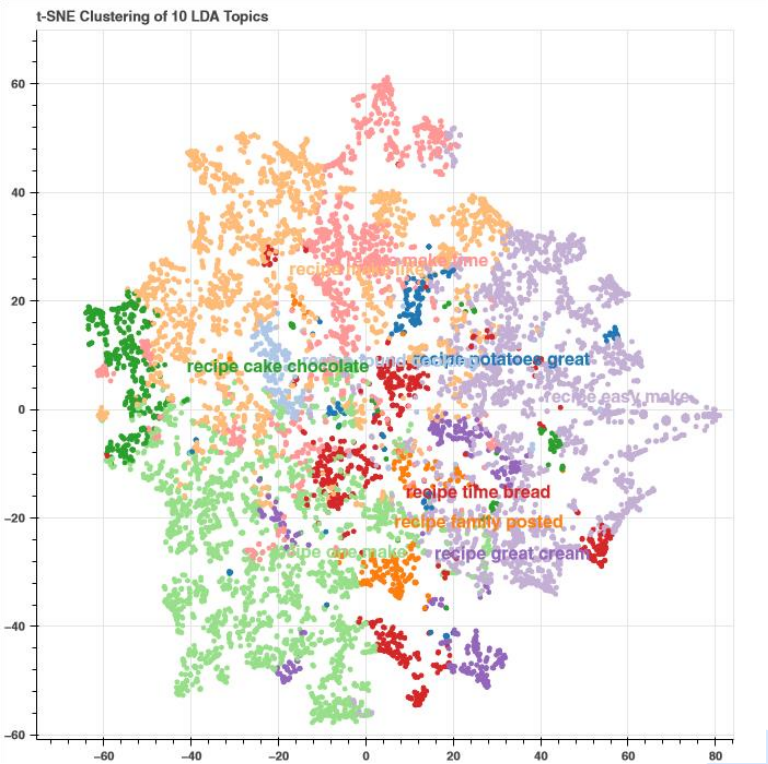
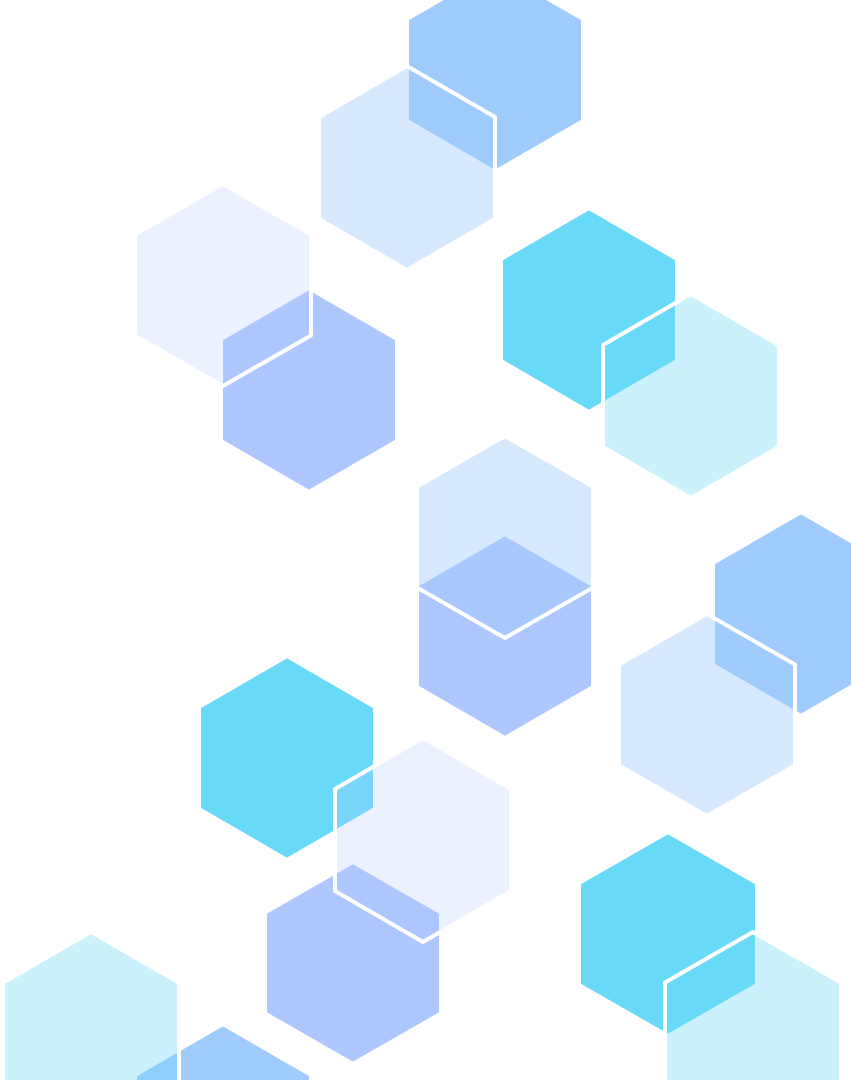


Fig. 2: Clustering of Top 10 LDA Topics

04

Time Series

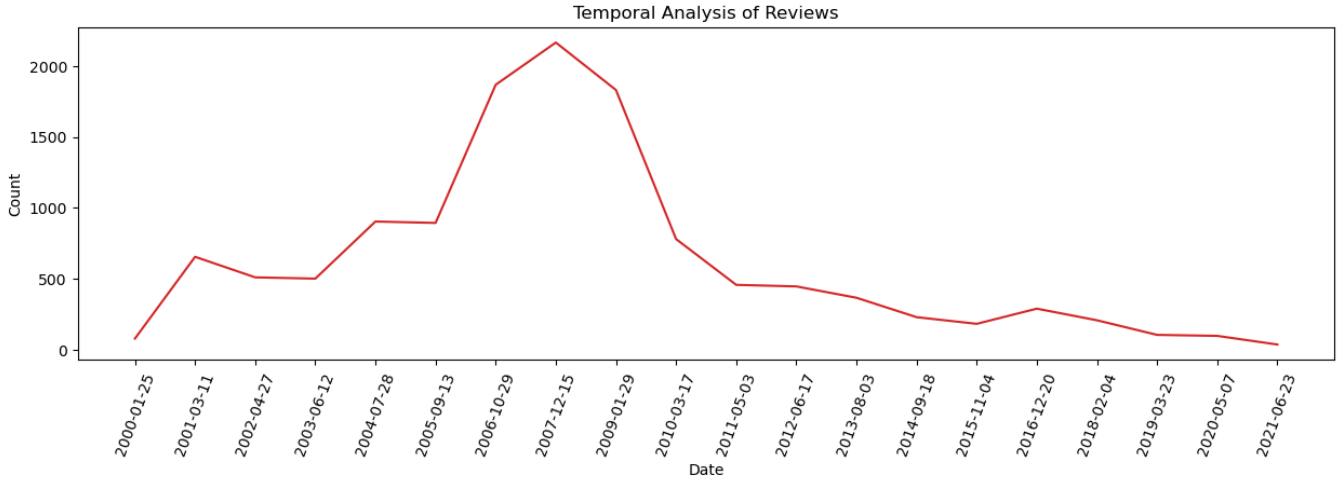


Topic Analysis

- Analyze topic to realize how we would model our problem
- Created some plots
- Needed to:
  - Sort reviews by date
  - Bin reviews by date

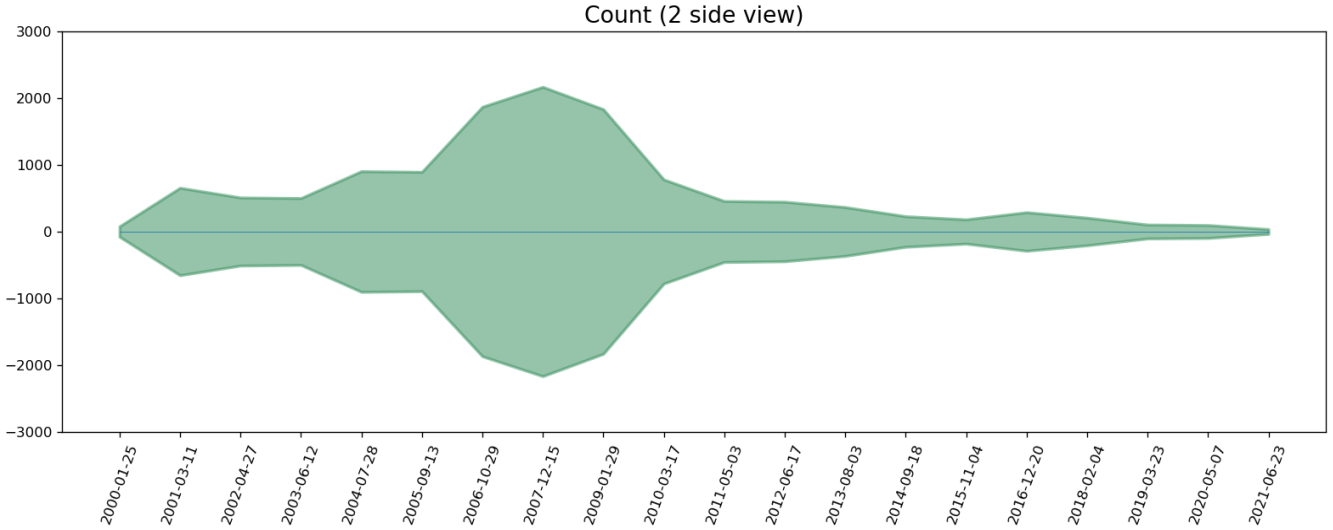
Temporal Analysis of reviews

- Plot displaying review count along the years.
- Positive review count trend up until 2007
- Peak review count in the year of 2007
- Negative review count trend from 2007 on

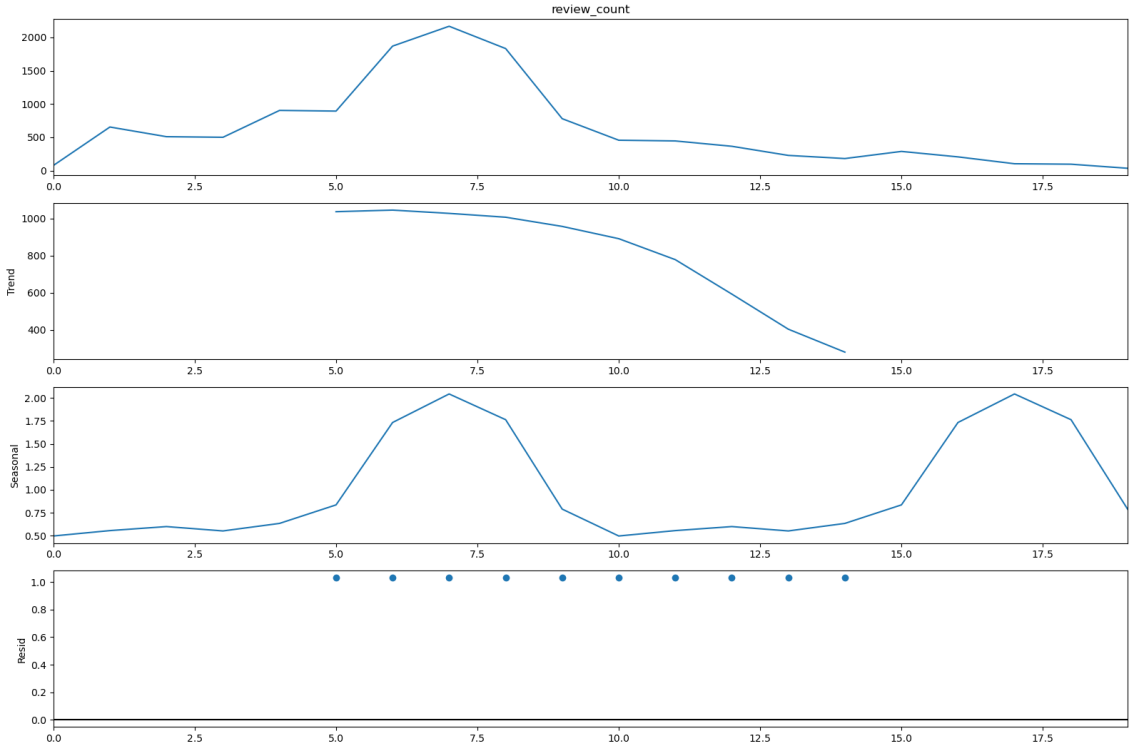


Temporal Analysis of reviews

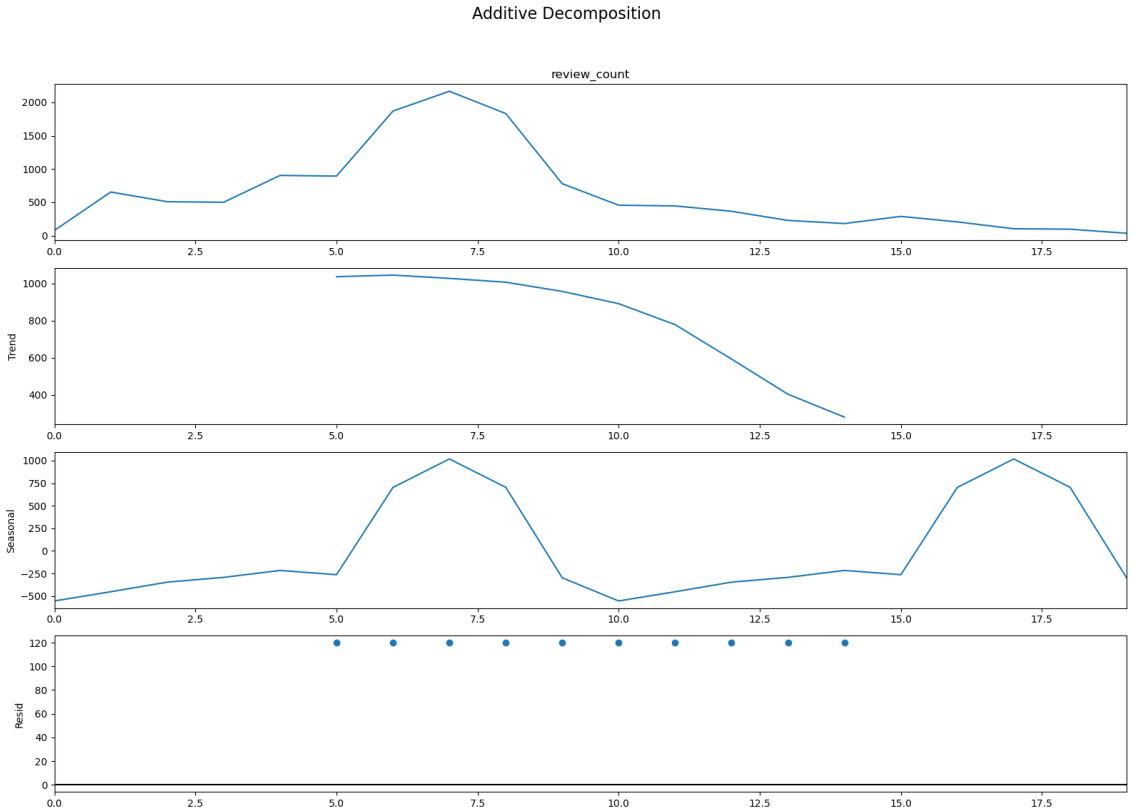
- Plot displaying review count along the years, now 2 side to get a better view of data.
- Also reveals a positive review count trend up until 2007
- Negative review count trend from 2007 on
- Peak review count in the year of 2007



Multiplicative decomposition of reviews data



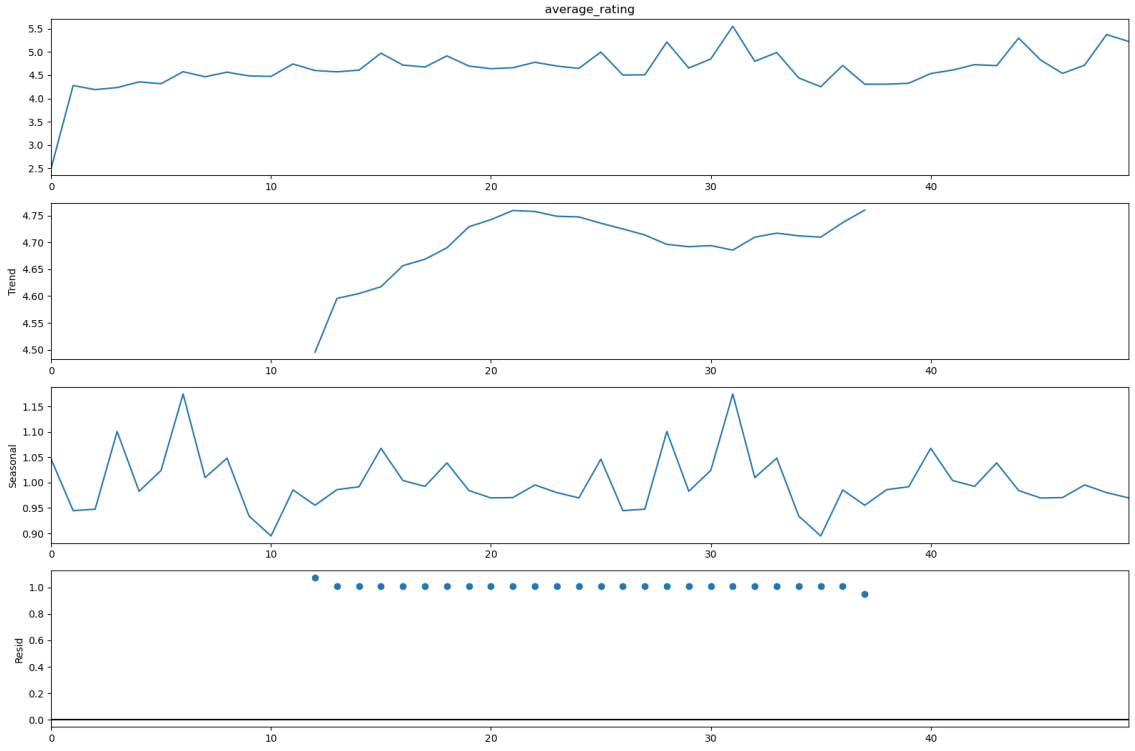
Additive decomposition of reviews data



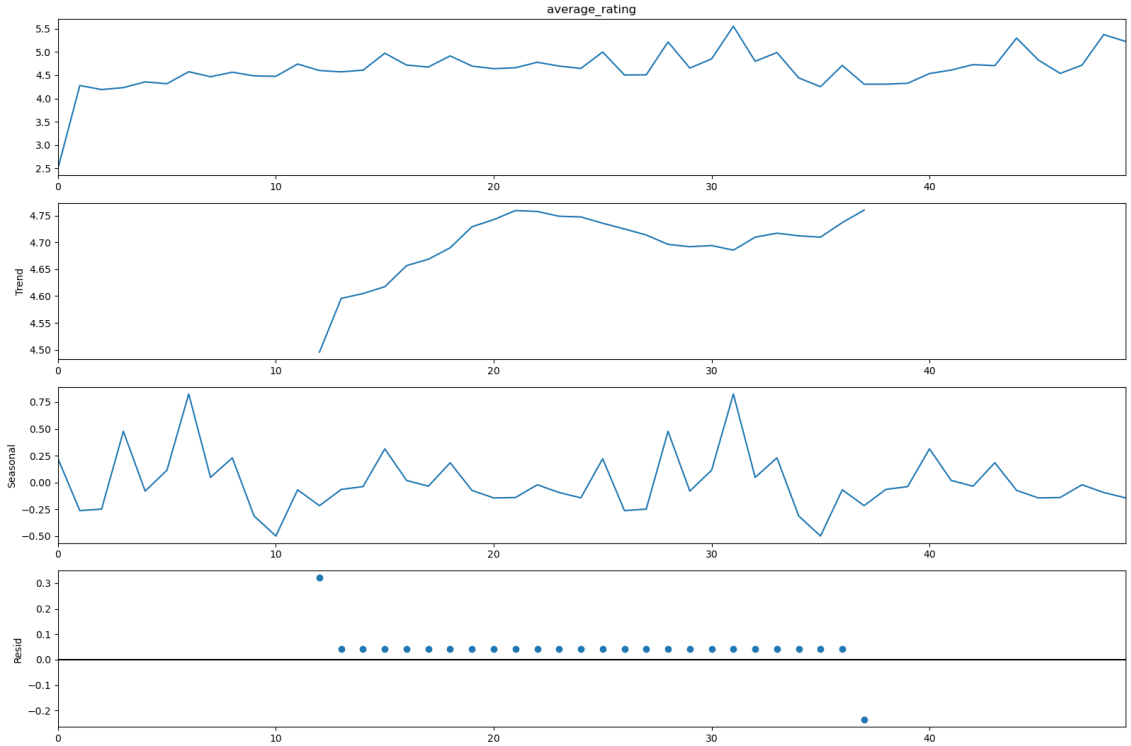
Additive and multiplicative decomposition analysis

- Divided in 20 bins
- Each bin represents approximately 1 year time period
- Very similar plots in additive and multiplicative decomposition
- Trend
  - Strong negative trend from bin 5 on
- Seasonality
  - One season every 10 years
- Residual
  - Not much variation
  - Multiplicative decomposition slightly better than additive one (120 outlier)

Multiplicative decomposition of average rating data



Additive decomposition of average rating data



Additive and multiplicative decomposition analysis

- Divided in 50 bins
- Each bin represents approximately 5 months period
- Very similar plots in additive and multiplicative decomposition
- Trend
  - Positive trend throughout the years
- Seasonality
  - One season every 25 bins
- Residual
  - Not much variation
  - Decomposition models were adequate

Approach

- After analyzing the data, we opted to predict the popularity of a topic
- Started by creating datasets with data divided by day, week and month
- Each table entry corresponds to all the data collected in a time period related to a topic

Daily

	last_modified_date	rating	likes	review_count
0	2000-01-25	5.0	5.0	1
1	2000-06-02	2.5	0.0	1
2	2000-09-12	5.0	10.0	2
3	2000-09-20	4.5	0.0	2
4	2000-10-18	5.0	0.0	1

Weekly

	last_modified_date	rating	likes	review_count
0	2000-01-31	5.0	5.0	1
1	2000-06-05	2.5	0.0	1
2	2000-09-18	5.0	10.0	2
3	2000-09-25	4.5	0.0	2
4	2000-10-23	5.0	0.0	1

Monthly

	last_modified_date	rating	likes	review_count
262	2022-08-31	6.000000	0.0	0
261	2022-07-31	5.666667	0.0	1
260	2022-06-30	5.500000	0.0	2
259	2022-04-30	5.116667	0.0	6
258	2022-03-31	5.000000	1.0	1
257	2022-02-28	4.250000	0.0	1

Analyzing feature correlation

- Available features
  - Date (last_modified_date)
  - Average rating (rating)
  - Sum of all the likes (likes)
  - Sum of all the reviews (review_count)

- Goal: analyze feature correlation to understand variable dependence/independence

Correlation matrix / Pearson correlation

Daily

	rating	likes	review_count
rating	1.000000	-0.053591	0.003739
likes	-0.053591	1.000000	0.203354
review_count	0.003739	0.203354	1.000000

Weekly

	rating	likes	review_count
rating	1.000000	-0.099157	0.073776
likes	-0.099157	1.000000	0.135990
review_count	0.073776	0.135990	1.000000

Monthly

	rating	likes	review_count
rating	1.000000	-0.193006	0.027433
likes	-0.193006	1.000000	0.146468
review_count	0.027433	0.146468	1.000000

Spearman Rank Coefficient

	Daily	Weekly	Monthly
Likes – Avg. Rating	-0.10	-0.18	-0.23
Likes - Review Count	0.19	0.38	0.54
Avg. Rating – Review Count	-0.22	-0.09	-0.12

Conclusions from correlation analysis

- Slightly negative correlation between review count and rating.
- Slightly negative correlation between like count and rating.
- Slightly positive correlation between like count and review count.
- These are slightly positive indicators that we can extract derived features from these features.

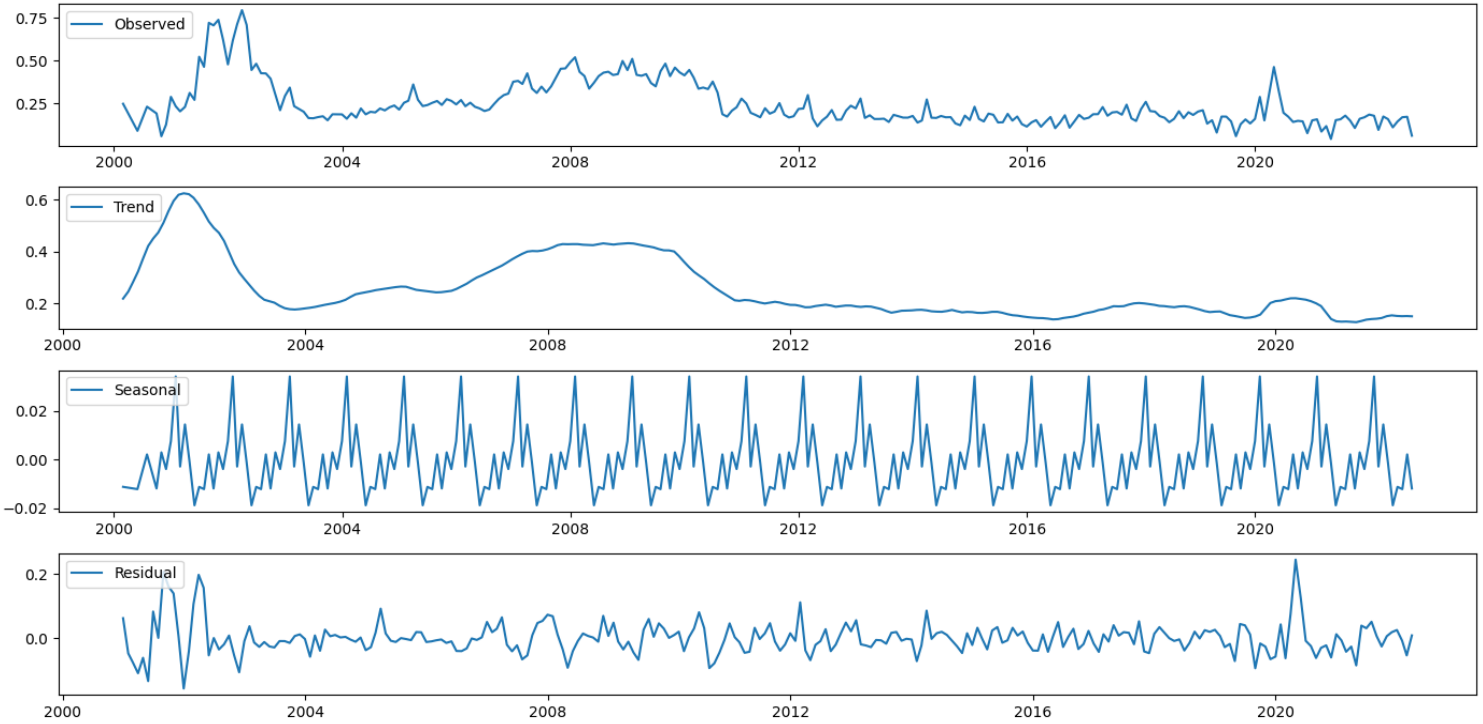
Feature Engineering

- Create more relevant features from the existing ones
- Created features
  - Likes to reviews ratio
  - Likes to rating ratio
  - **Popularity score**

Popularity score

Features	Weights
Average rating	0.2
Like count	0.4
Review count	0.4
Likes to reviews ratio	0.1
Likes to rating ratio	0.1

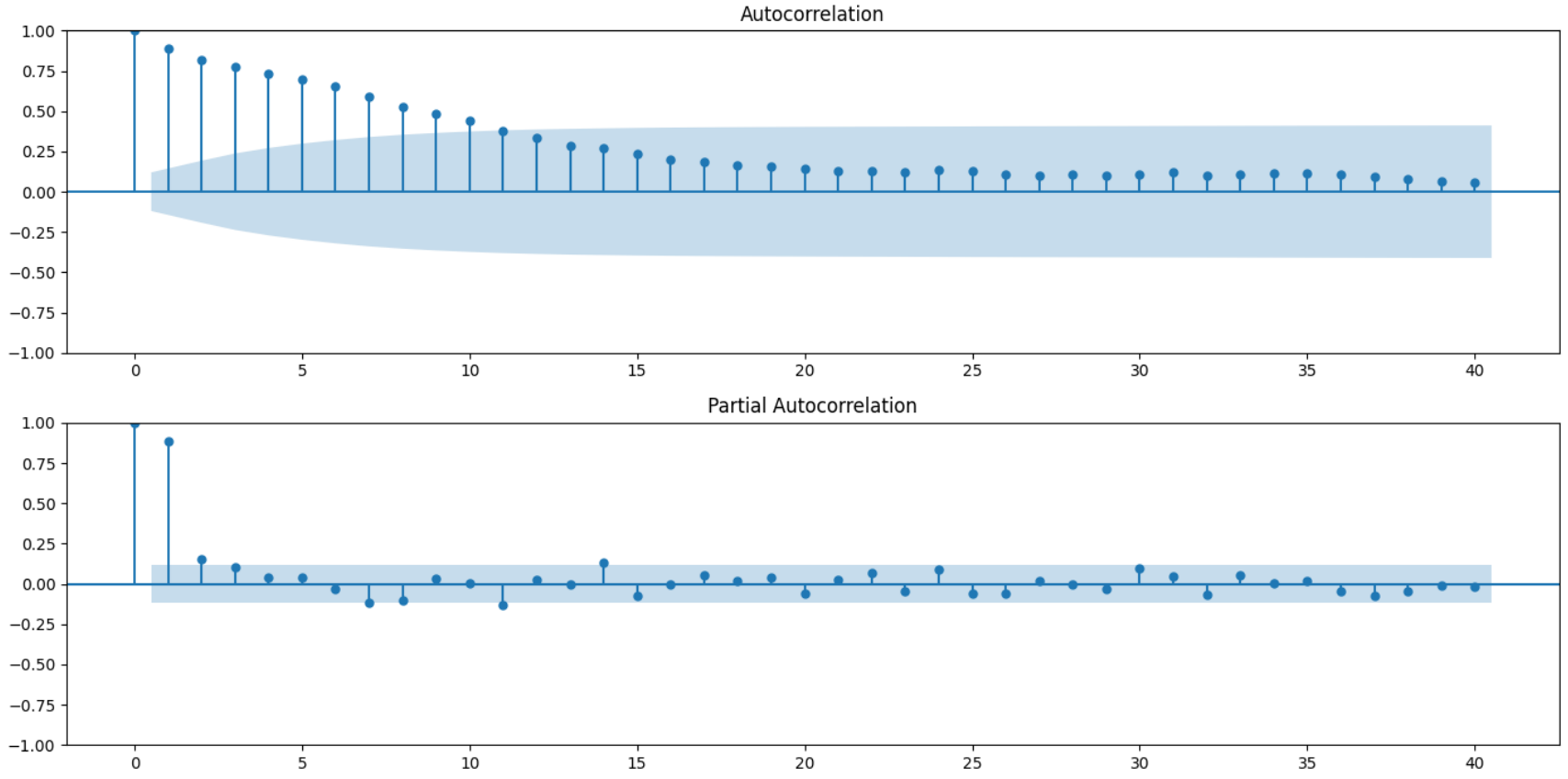
Seasonal Decompose of Popularity score of topic



Seasonal Decompose

- Observed values
  - Irregular and variable values
- Trend
  - First years are a bit irregular
  - Stabilization starting in 2012
- Seasonality
  - One season every year
- Residual
  - Instability
  - The trend and seasonality components might not have fully captured the patterns
  - Might need another decompose

ACF and PACF


Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30
35
40
Partial Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30
35
40

ACF and PACF

- ACF
  - Shape reveals strong trend component
  - Approximately one year trend
  - No evidence of seasonality
  - Non-stationary (significant autocorrelation between lags, data changes over time)

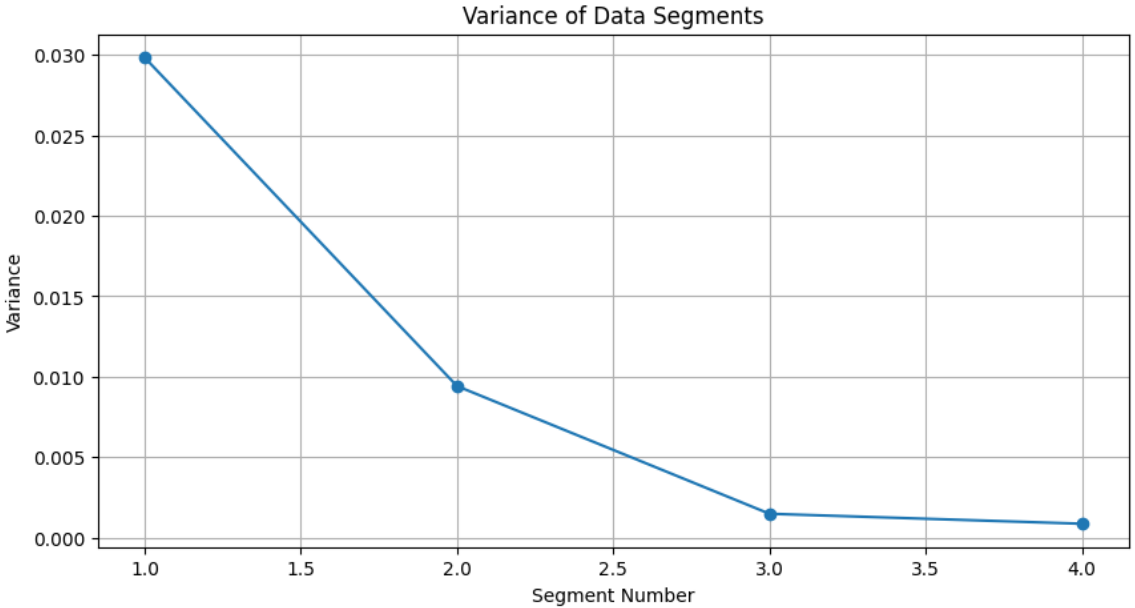
- PACF
  - Relevant values until 2

Handling Non-Stationary

- Creation of different datasets using methods to turn the *popularity_score* attribute into stationary.
- Application of differentiation, logarithmic reduction, box cox and a combination of logarithmic reduction with differentiation.
- Verification of stationarity with ADF and variation of segments applying *levene*.
- The right method to achieve stationarity within our whole dataset depends on the sample used.

Dataset	Sample	Logarithmic	Differenced	Diff-log	Box-Cox
P-value	0.13	$5.10 * 10^{-8}$	$7.13 * 10^{-8}$	$2.88 * 10^{-6}$	0.0028

Handling Non-Stationary



ARIMA

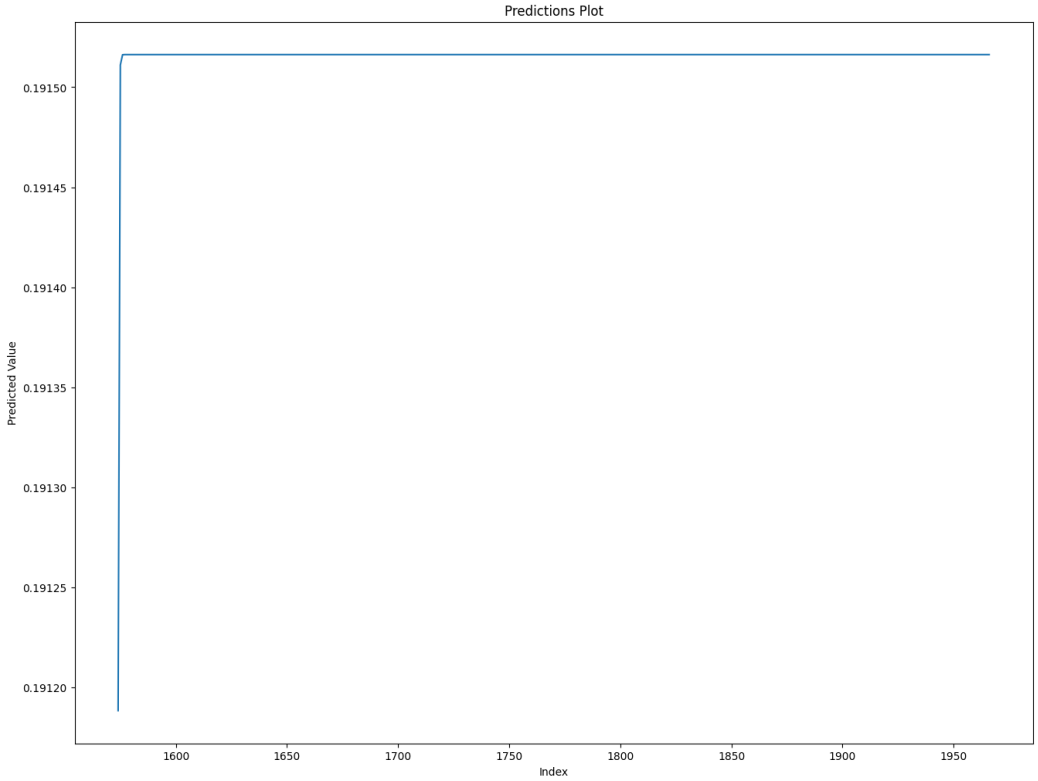
- We tried applying grid search to find the best values for parameters p, d and q

```
# Perform grid search
for param in param_grid:
    p, d, q = param
    try:
        # Train ARIMA model with current hyperparameters on the training set
        arima_model = ARIMA(train_data['popularity_score'], order=(p, d, q))
        arima_model_fit = arima_model.fit()

        # Make predictions on the validation set
        predictions_arima = arima_model_fit.forecast(steps=len(test_data))

        # Calculate Mean Squared Error (MSE) on the validation set
        mse = mean_squared_error(test_data['popularity_score'], predictions_arima)

        # Update best model if current model has lower MSE
        if mse < best_mse:
            best_model = arima_model_fit
            best_mse = mse

    except:
        continue
```

2373m 55.3s

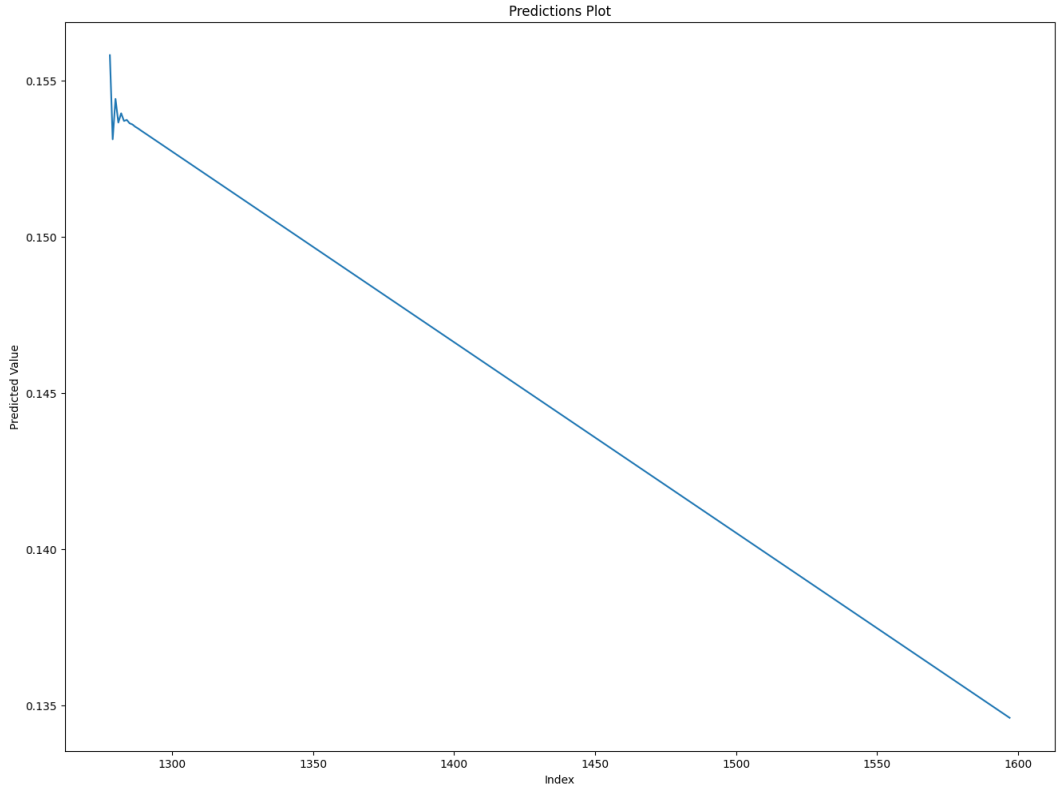
ARIMA

- p=1, d=1, q=1
- Horizontal line around 0.19150
- It follows the same values that existed between 2012 and 2020



ARIMA

- p=1, d=2, q=1
- Good starting variation with quick decay after 10 values.
- Values much higher than last testing results, closer to initial testing results (2002, 2008)

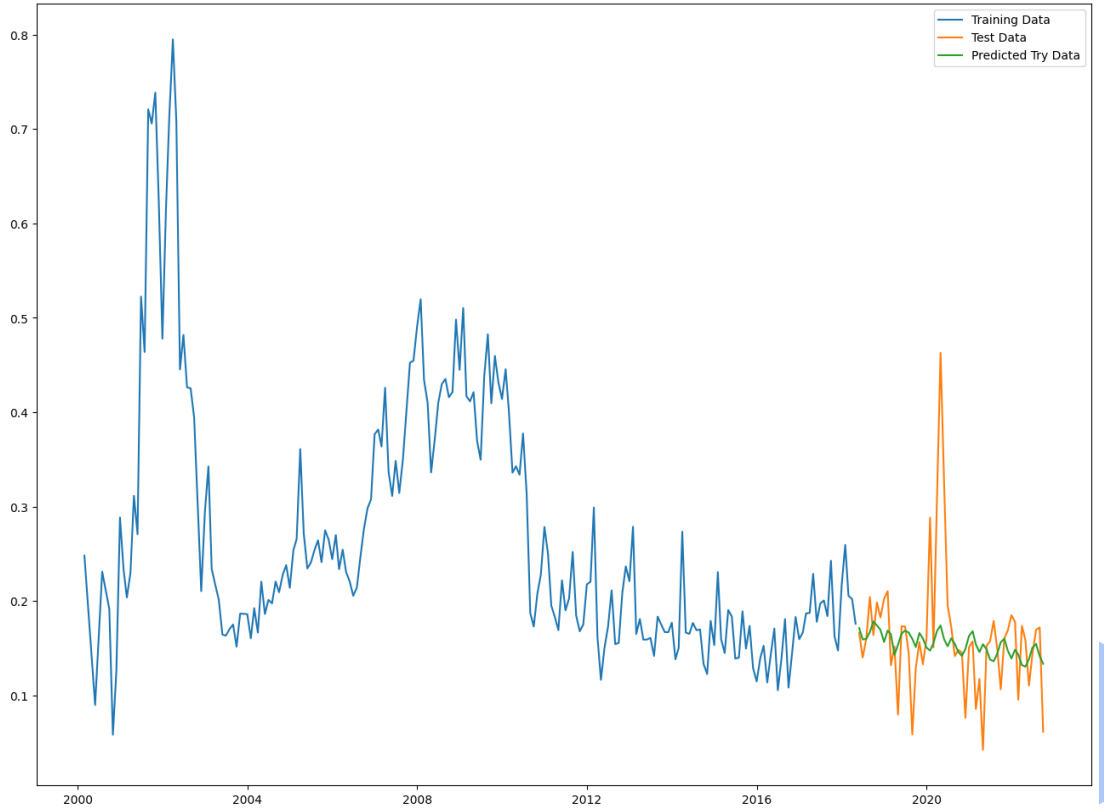


AUTO-ARIMA

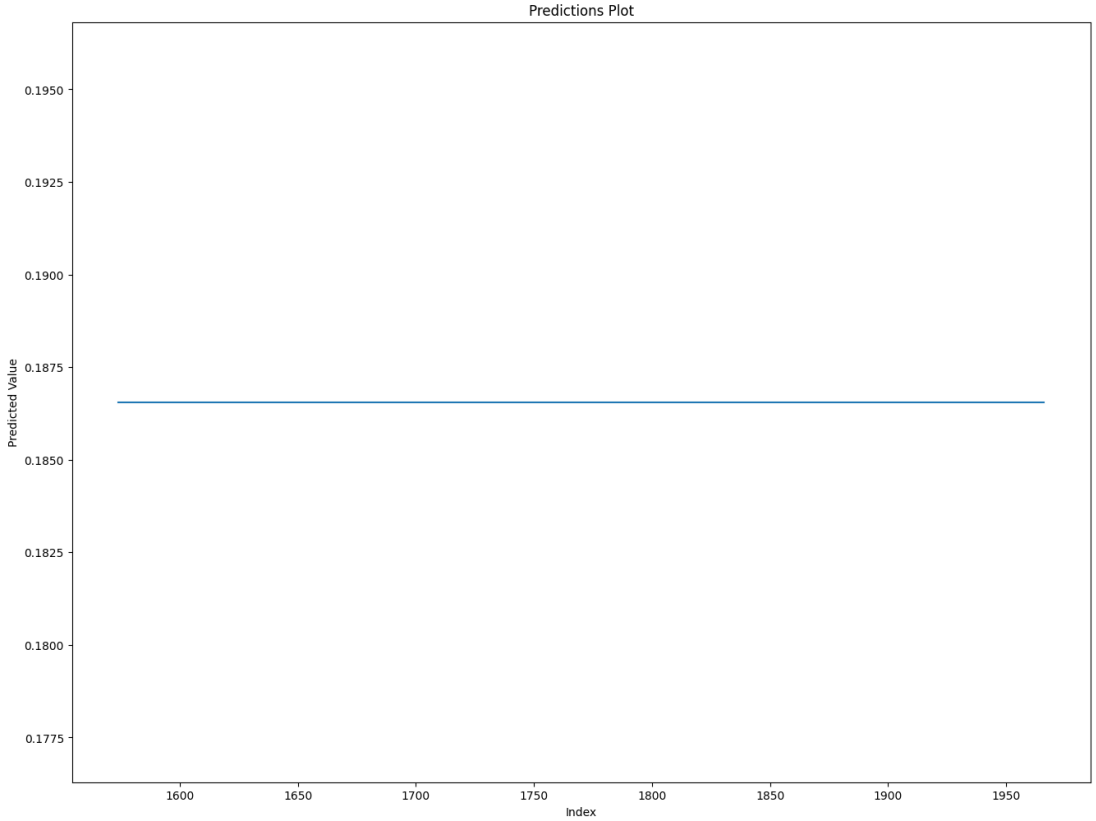
- Auto-Arima is a tool to fine tune the parameters of the ARIMA model.
- We combined some of the previously retrieved datasets (sample, differenced and logarithmic, discarding the other two due to lack of relevance) with some fixed parameters on the auto_arima (max_p = 5, max_q = 5) and some variable ones.
- We defined as variable parameters m (value of 9, 10 and 12, due to revealing relevant lags on the original ACF), d (varying differencing between 0 and 1), seasonal (varying between True and False, to take advantage of seasonality sometimes present on samples) and D (varying between 0 and 1, similar to d but for seasonality).

AUTO-ARIMA

- Our best result came from using the sample dataset with fields d=1, D=1, m=9 and seasonal=True.
- Best model: ARIMA(0,1,1)(2, 1,1)[9]
- Similar performance models are on annexes



Exponential Smoothing

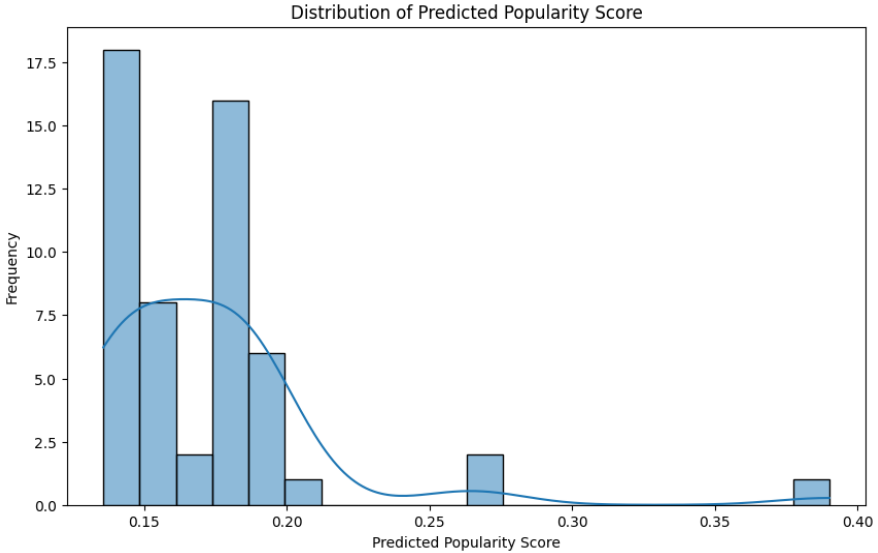


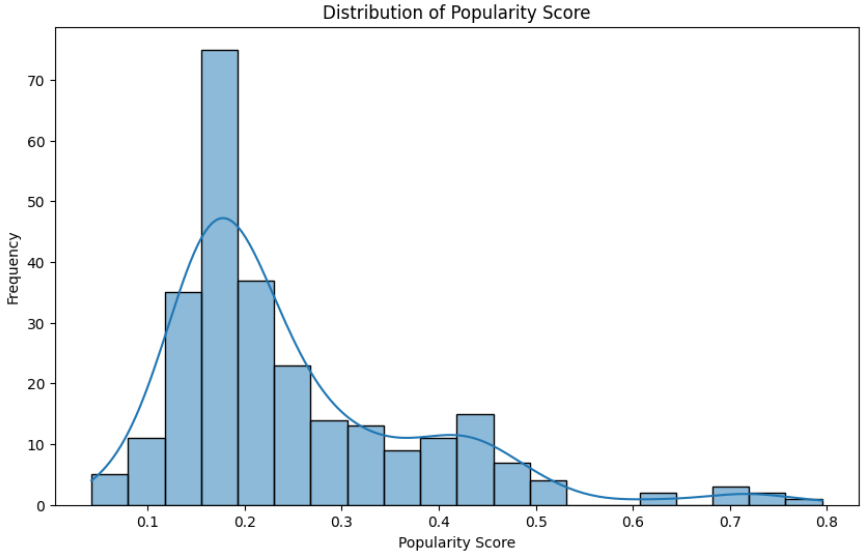
Random Forest

- We wanted to experiment using RF to compare performance with time series models.
- Some transformations were needed
  - Substitute date timestamp into categorical variables day, month and year (new features)
  - Shift target variable (popularity score)

Random Forest

- These were our values distribution when comparing predicted vs actual test values for the popularity score for Random Forest classifier





Random Forest

- After analyzing the values provided from the distribution, we wanted to define a threshold to better measure model statistics by binarizing the target variable (popularity score).
- The model predicted a continuous variable and it is hard for a model to predict the exact continuous variable value, so binarization of the target variable seems to be the best option to understand and analyze performance.
- It might be a bit biased approach, but we defined a threshold of 0.2 and above for considering a topic as popular and validated that value with metric analysis.
- We tried different thresholds, but from trade-off analysis it looked to be the value that produced the most balanced results.

Random Forest (metrics)

MSE	0.0031
Accuracy	0.89
Precision	0.75
Recall	0.38
F1 Score	0.5

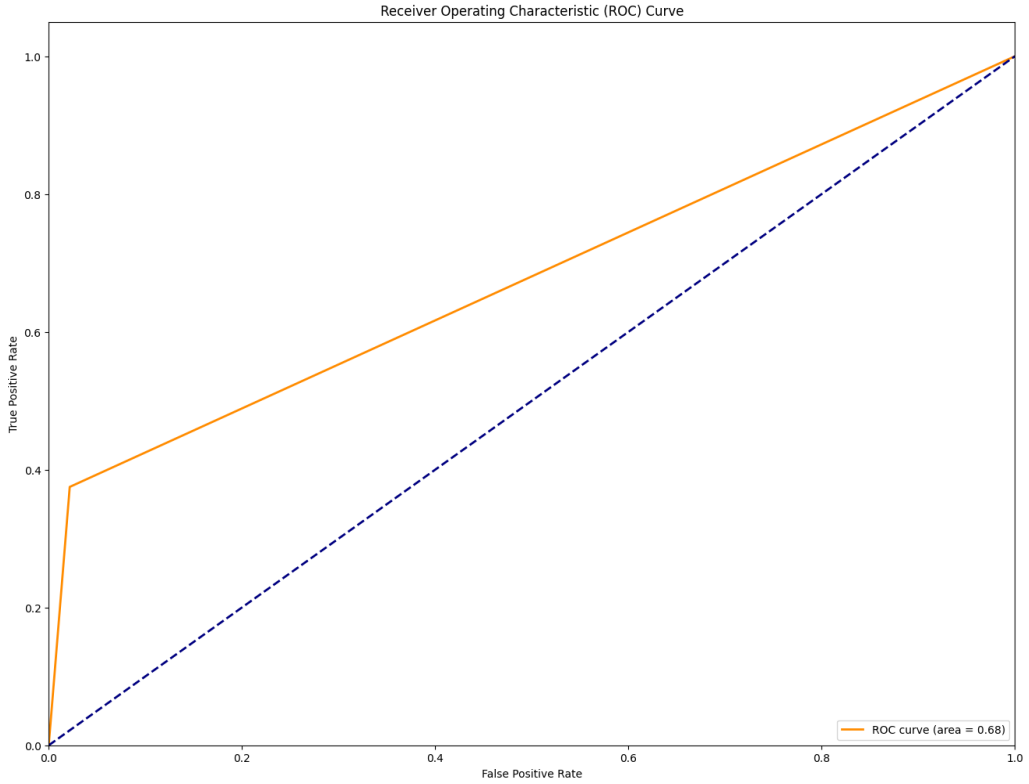
Confusion Matrix

45	1
5	3

Random Forest (metrics)

- **Accuracy:** suggests that the model correctly classified 88.8% of the instances in the test set, but we should analyze other metrics to understand the model's behavior better.
- **Precision:** out of all instances predicted as positive, 75% are true positives, while the remaining 25% are false positives.
- **Recall:** it missed 62.5% of positive instances, indicating a relatively low sensitivity.
- **F1 Score:** indicates moderate performance in terms of both precision and recall.

Random Forest (metrics)



AUC score: 0.68

Random Forest (metrics)

- **AUC and ROC curve analysis:**
  - 68% chance that the model will be able to distinguish between positive and negative instances.
  - AUC score is above 0.5, which indicates that the model is better than random chance, and points to the same conclusion.
  - Moderate capability of distinguishing between the positive and negative class, due to the ROC curve always being above the diagonal line.

Random Forest (conclusions)

- Better at making predictions than a random model
- Guideline for understanding the performance when comparing with time series forecast models
- When comparing with ARIMA, has a lower value distribution but overall better forecasting capabilities due to it being more moderate in making predictions
- Predicted values are slightly lower than actual values from the test set.

05

Conclusion

Conclusion

- Use of ingredients allows for better results when creating LDA topics compared to description
- Stemming was the best way of splitting strings into tokens
- AUTO-ARIMA was a good tool to find parameters to create good forecast models.
- Despite the lower quality of the data, there were some models of ARIMA that had an interesting performance with similar forecast results to expected.

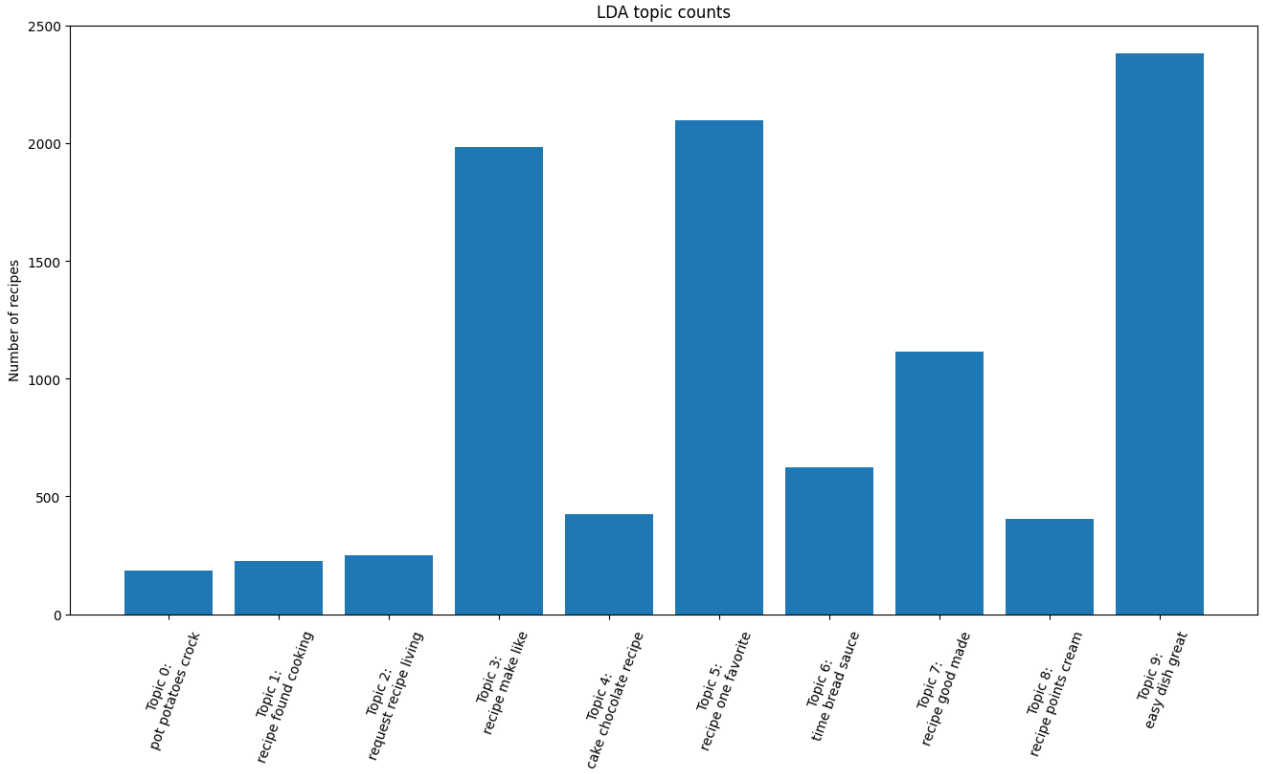
Thanks!

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Annexes

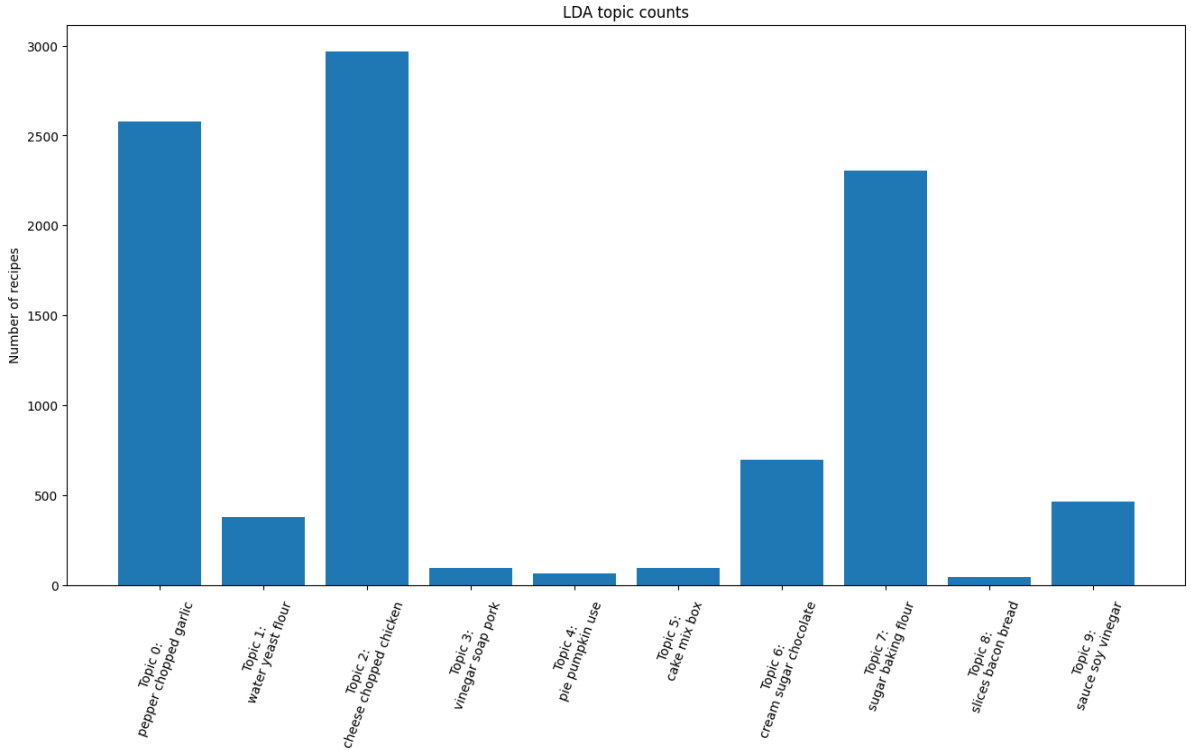
Topic Distribution

- Sample: most reviews
- Attributes: description
- Text process: tokenization
- Vectorizer: count



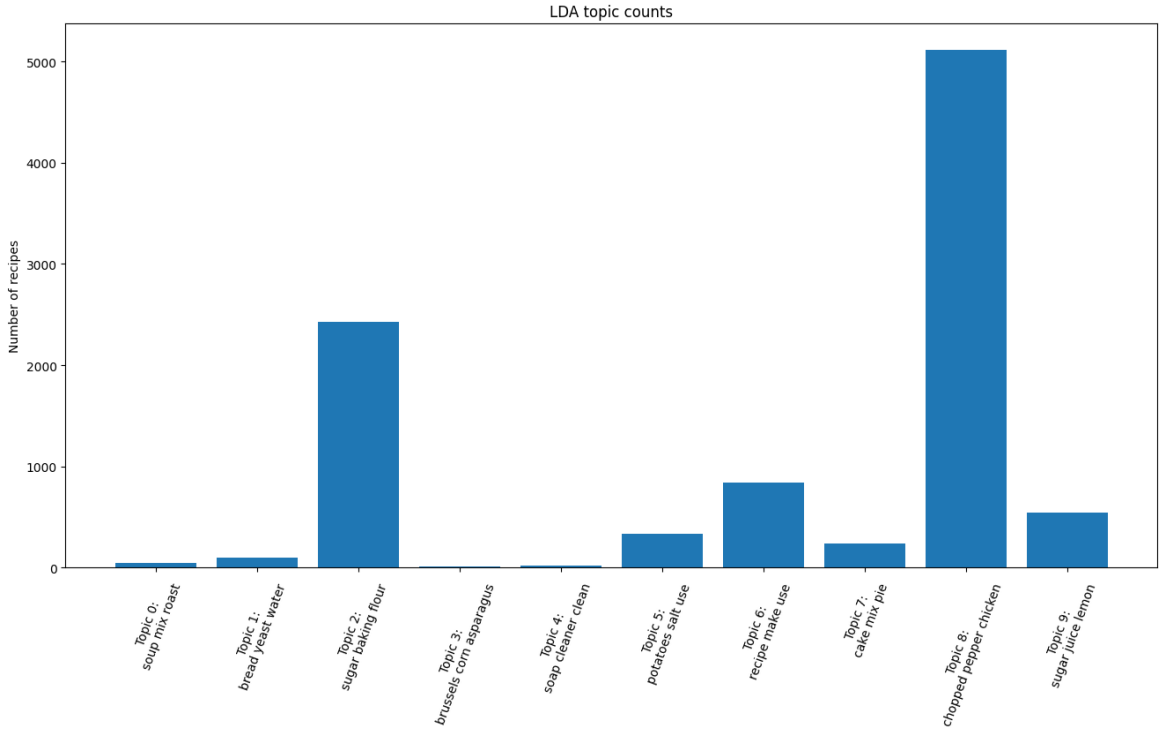
Topic Distribution

- Sample: most reviews
- Attributes: ingredients
- Text process: tokenization
- Vectorizer: count



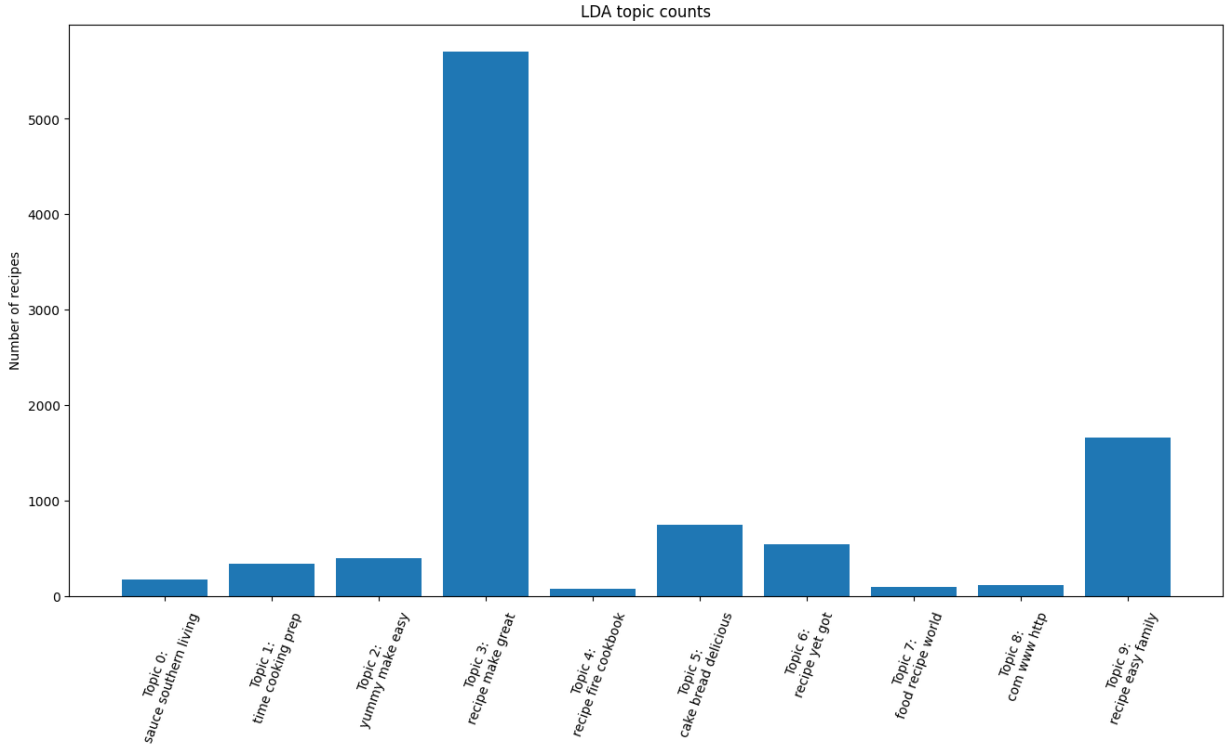
Topic Distribution

- Sample: most reviews
- Attributes: both
- Text process: tokenization
- Vectorizer: count



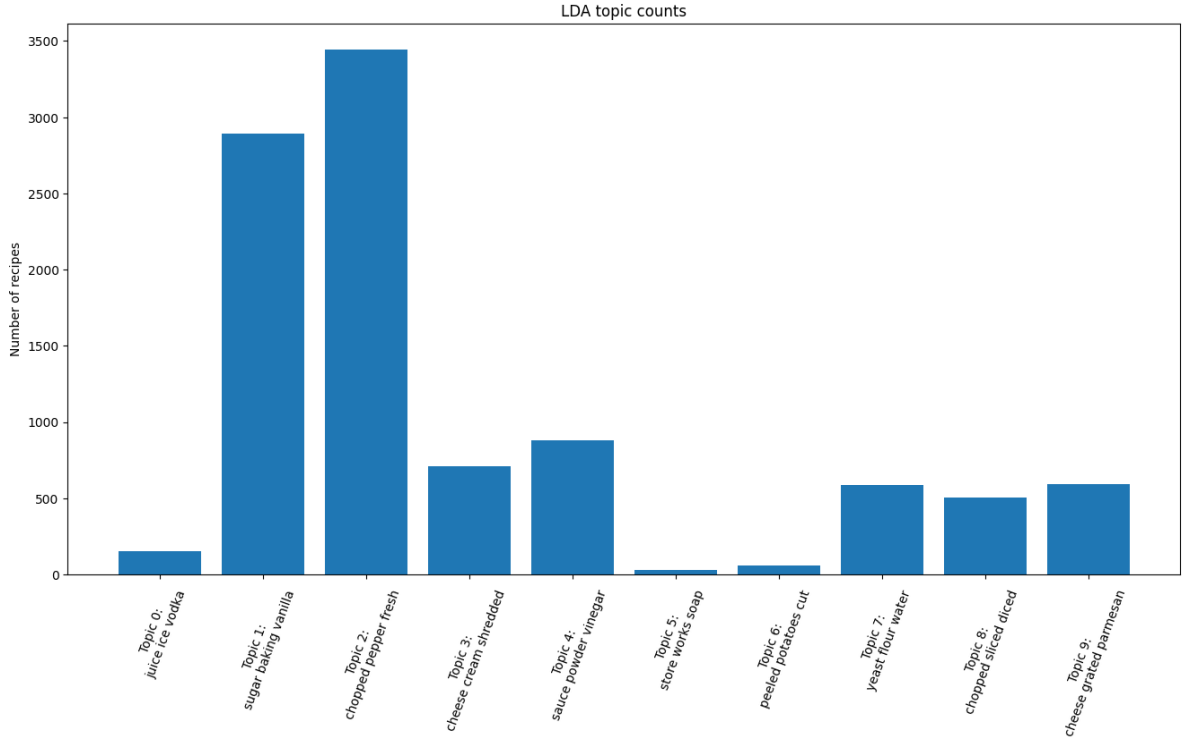
Topic Distribution

- Sample: recent dates
- Attributes: description
- Text process: tokenization
- Vectorizer: count



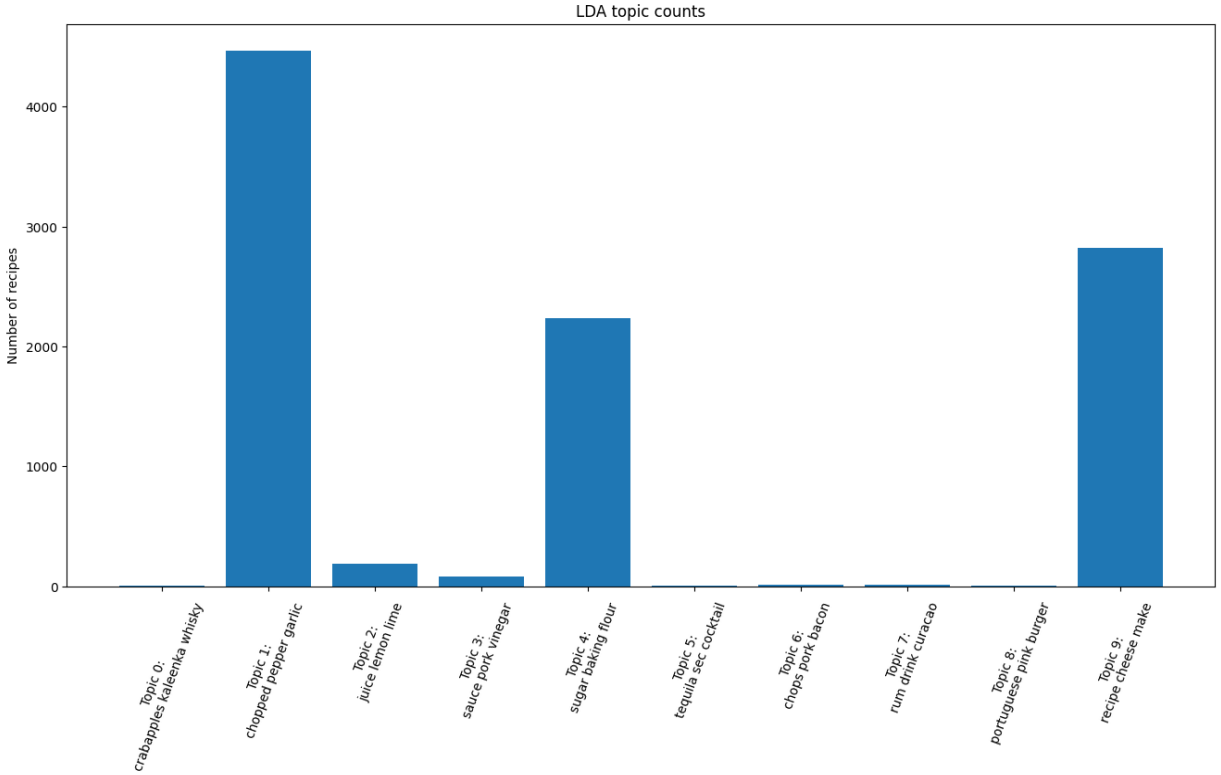
Topic Distribution

- Sample: recent dates
- Attributes: ingredients
- Text process: tokenization
- Vectorizer: count



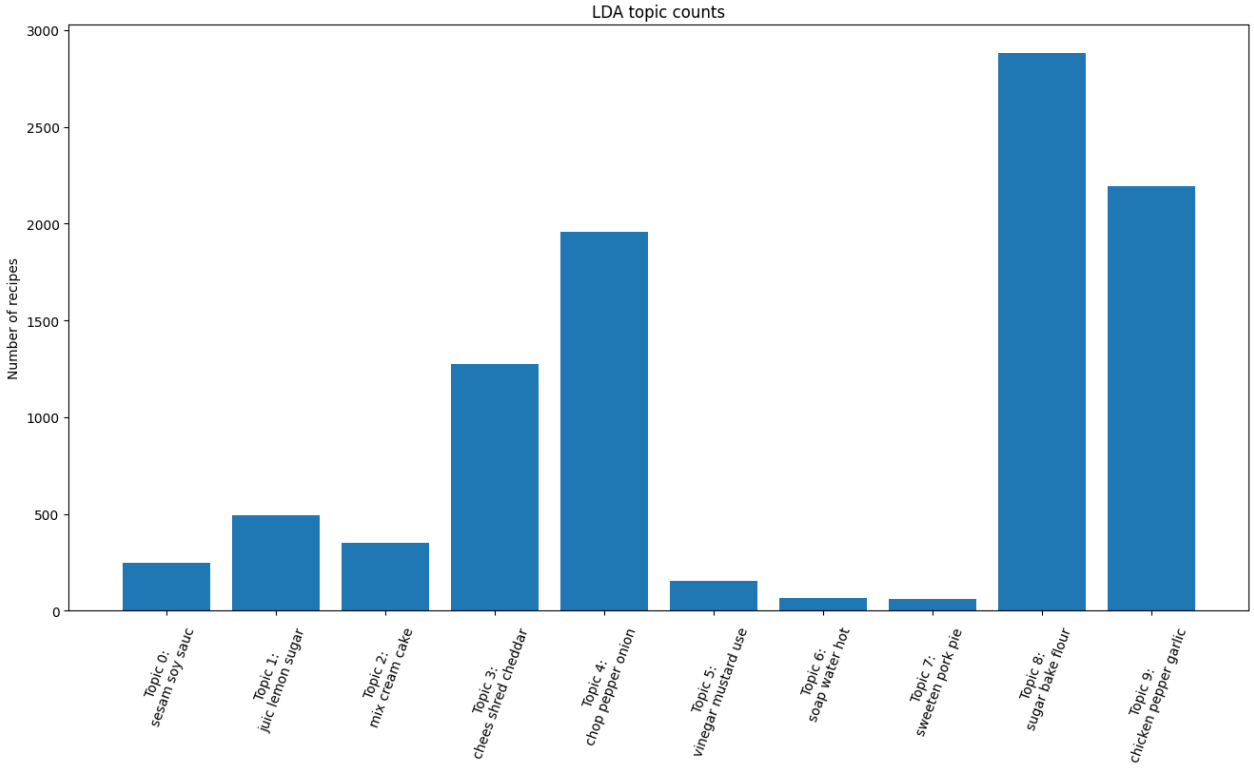
Topic Distribution

- Sample: recent dates
- Attributes: both
- Text process: tokenization
- Vectorizer: count



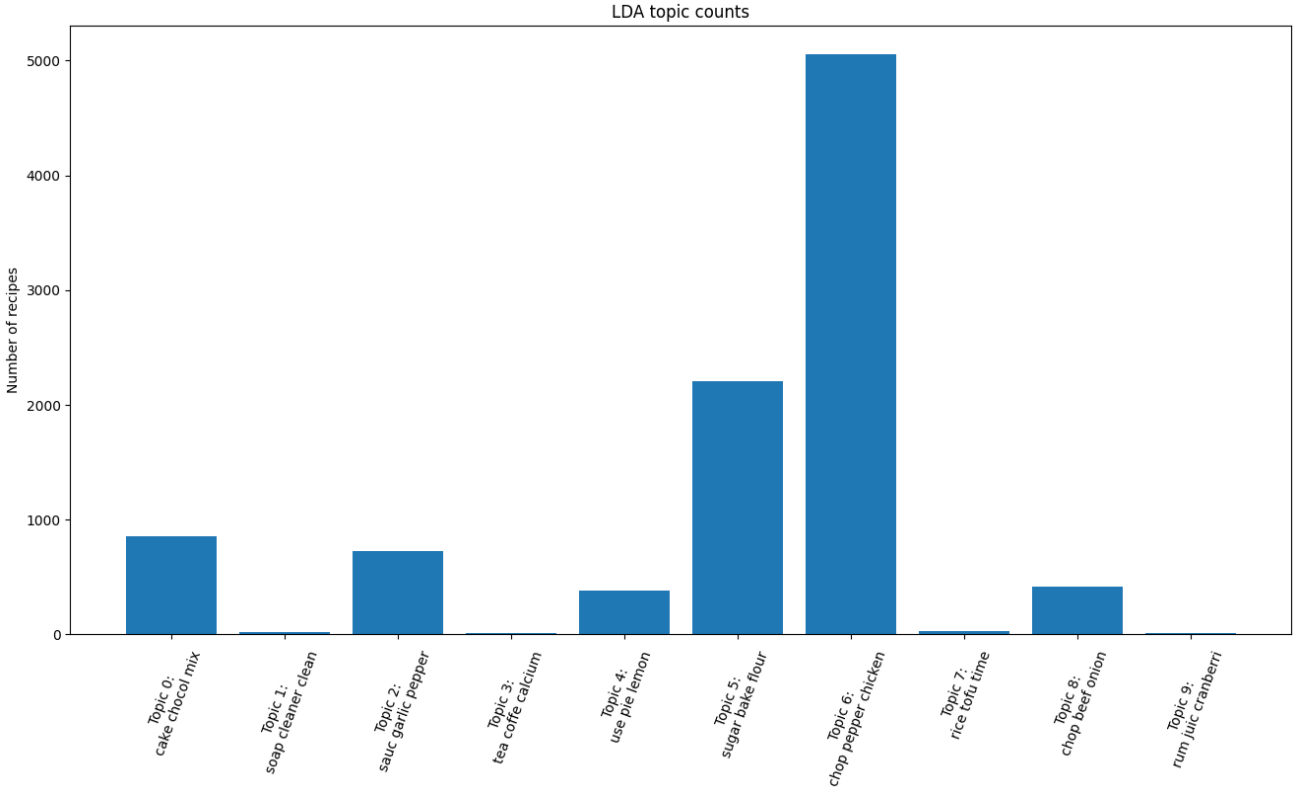
Topic Distribution

- Sample: most reviews
- Attributes: ingredients
- Text process: stemming
- Vectorizer: count



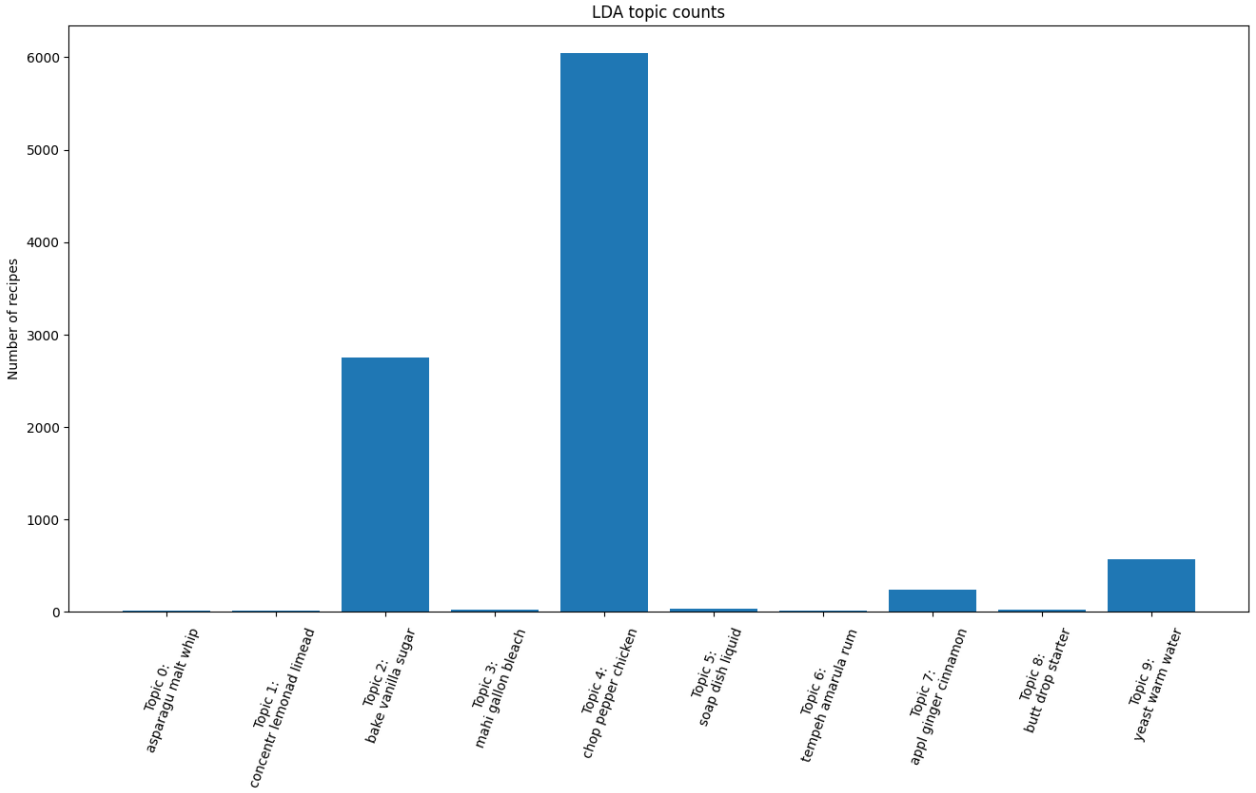
Topic Distribution

- Sample: most reviews
- Attributes: both
- Text process: stemming
- Vectorizer: count



Topic Distribution

- Sample: most reviews
- Attributes: ingredients
- Text process: stemming
- Vectorizer: tf-idf



AUTO-ARIMA

- Our best result came from using the differenced dataset with fields d=1, D=1, m=10 and seasonal=True
- Best model: ARIMA(0, 1,1)(0,1,1)[10]

AUTO-ARIMA

- Our best result came from using the differenced dataset with fields d=0, D=1, m=12 and seasonal=True
- Best model: ARIMA(0, 0,1)(1,1,1)[12]