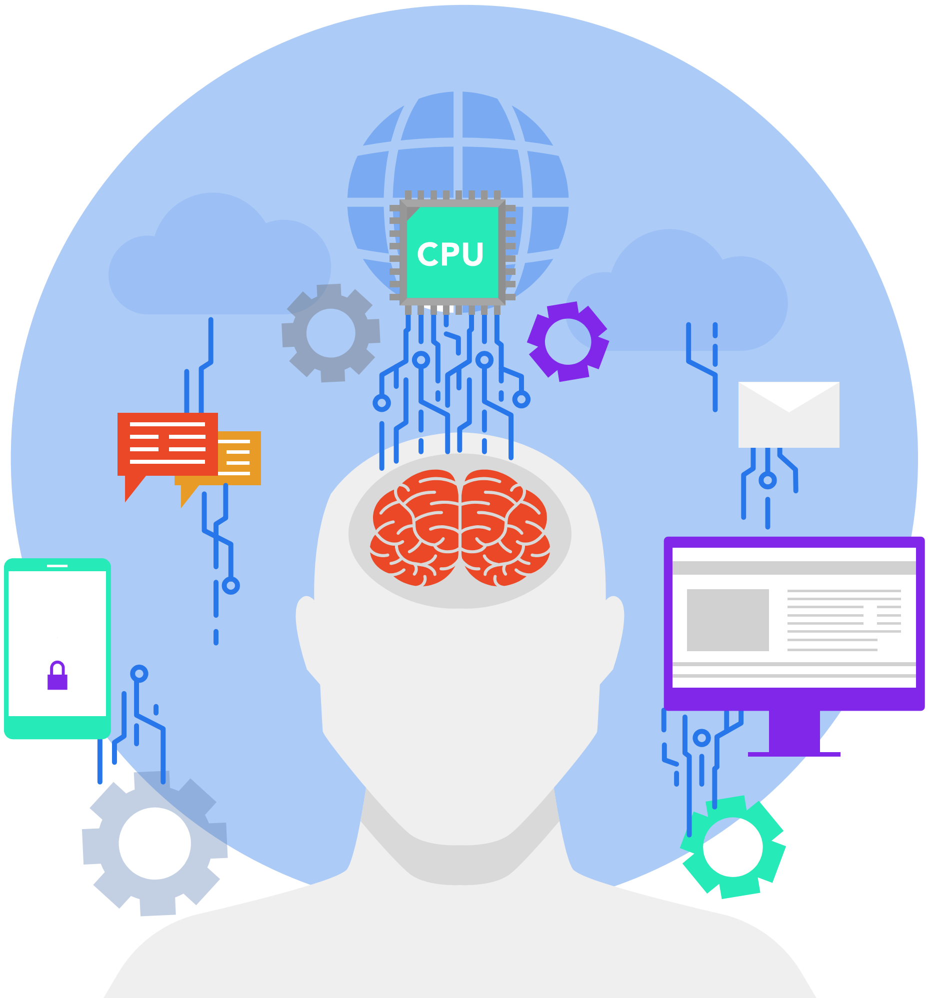


Classifying Propositional Content in annotated Argumentative Discourse Units

Natural Language Processing
M.EIC - FEUP
Assignment 1

André Nascimento, up201806461@up.pt
Gonçalo Alves, up201806451@up.pt

21st April, 2022

Problem definition

In the scope of the DARGMINTS project, an annotation project was carried out which consisted of annotating argumentation structures in opinion articles published in the Público newspaper:

1. Selecting text spans that are taken to have an argumentative role (ADUs);
2. Connecting such ADUs through support or attack relations;
3. Classifying the propositional content of ADUs as propositions of **Fact**, **Value**, or **Policy**. Within propositions of **value**, distinguish between those with a **positive** (+) or **negative** (-) connotation.

Our **goal** is to develop the **best classifiers** possible for this multi-class classification problem.

Problem definition

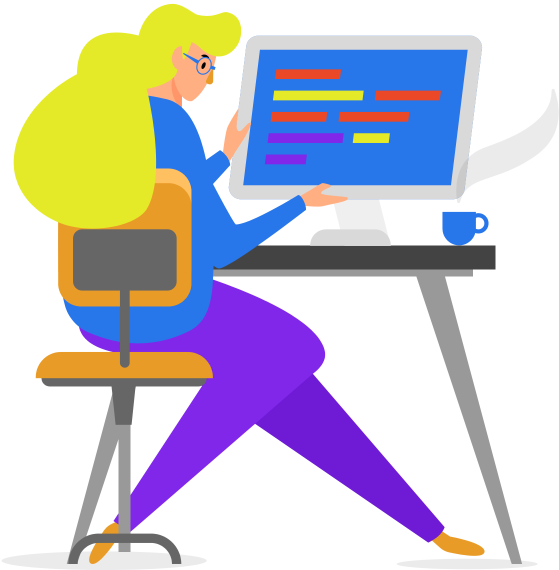
OpArticles

Content of each annotated ADU span, its 5-class classification, its annotator and the document from which it has been taken

OpArticles_ADUs

Details for each opinion article that has been annotated, including the full article content

Assignment Steps



01 Exploratory Analysis

02 Preprocessing

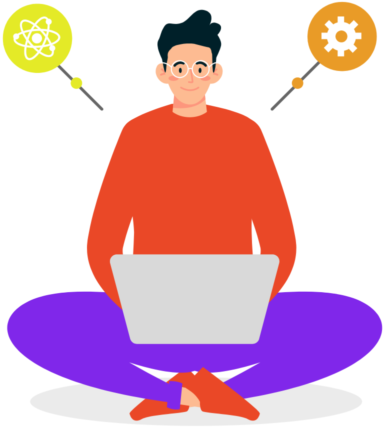
03 Representation Techniques

04 Classification & Results

05 Results Discussion

06 Conclusions

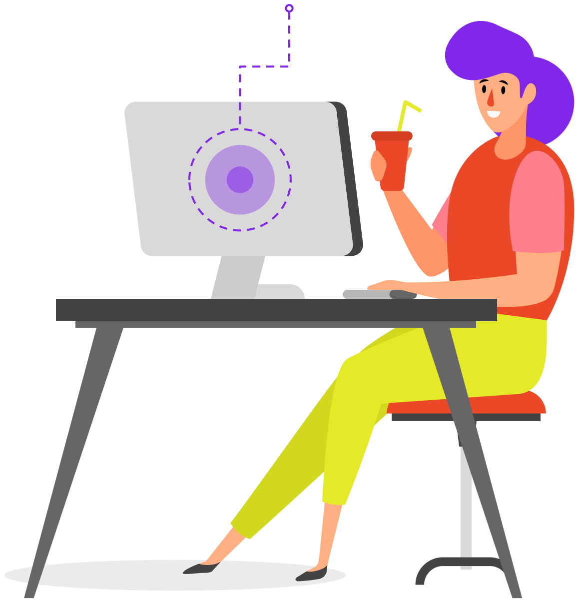
Exploratory Analysis



Initial Analysis

01 OpArticles

- 373 rows
- 8 columns (*title, authors, body, meta_description, topics, keywords, publish_date and url_canonical*)
- 8 unique topics
- 0 missing values



OpArticles_ADUs 02

- 16743 rows
- 5 columns (*annotator, node, ranges, tokens and label*)
- 12008 unique ADUs
- 0 missing values

Exploratory Analysis

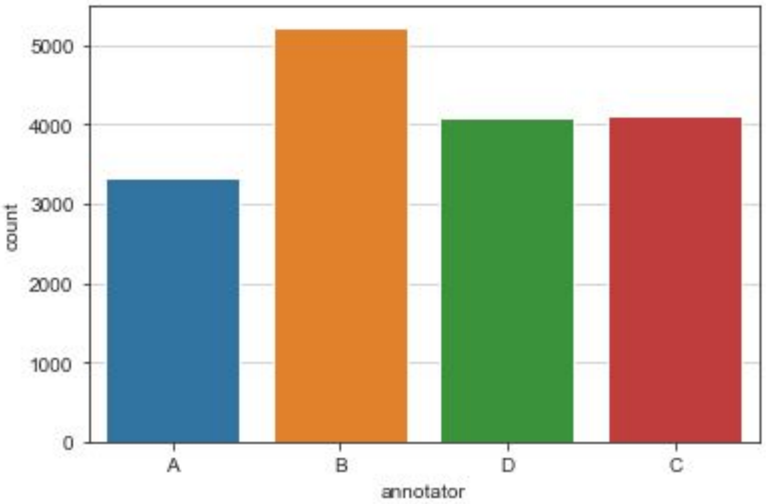


Fig. 1 Distribution of articles per annotator

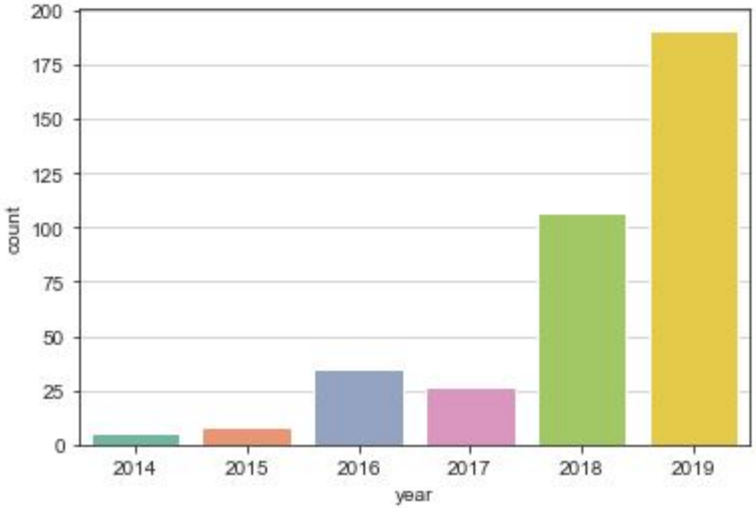


Fig. 2 Distribution of articles per year

Exploratory Analysis

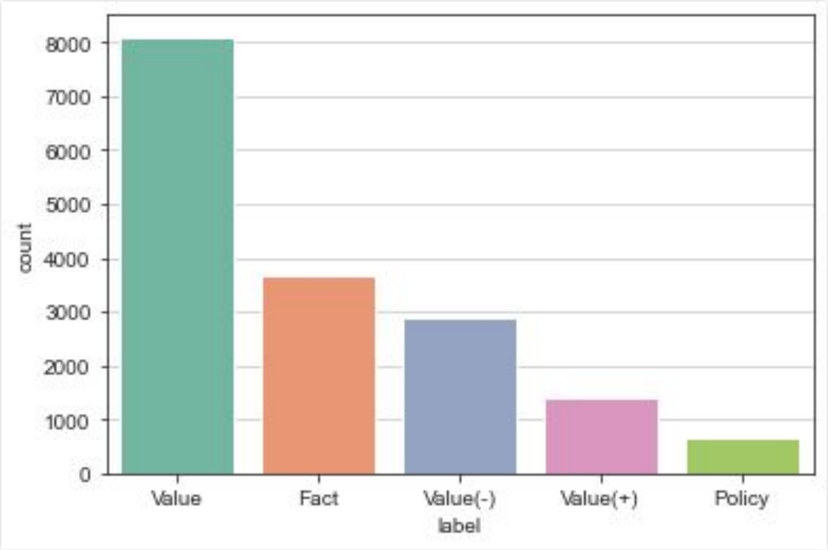


Fig. 3 Distribution of Labels

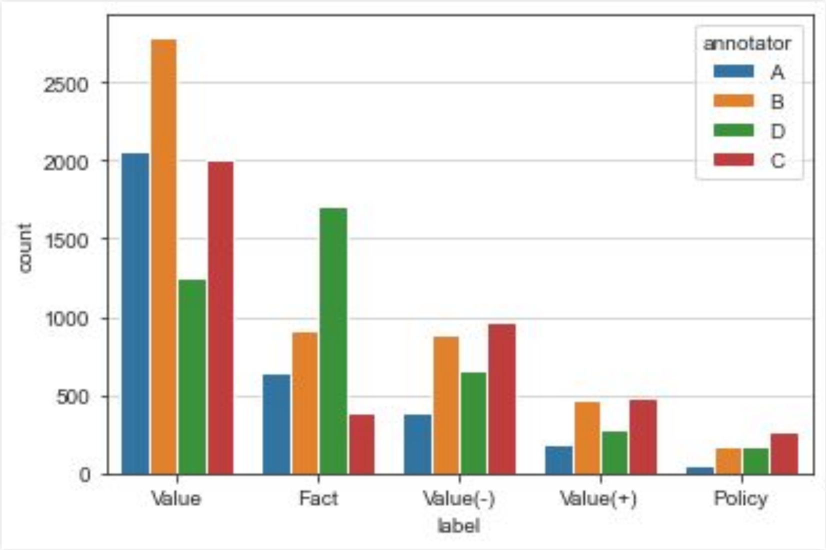


Fig. 4 Distribution of Labels regarding Annotators

Exploratory Analysis

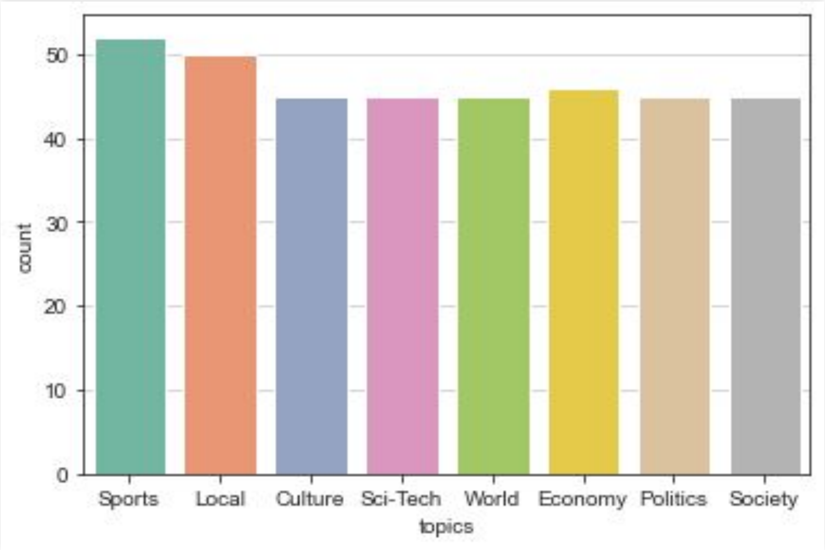


Fig. 5 Distribution of Topics

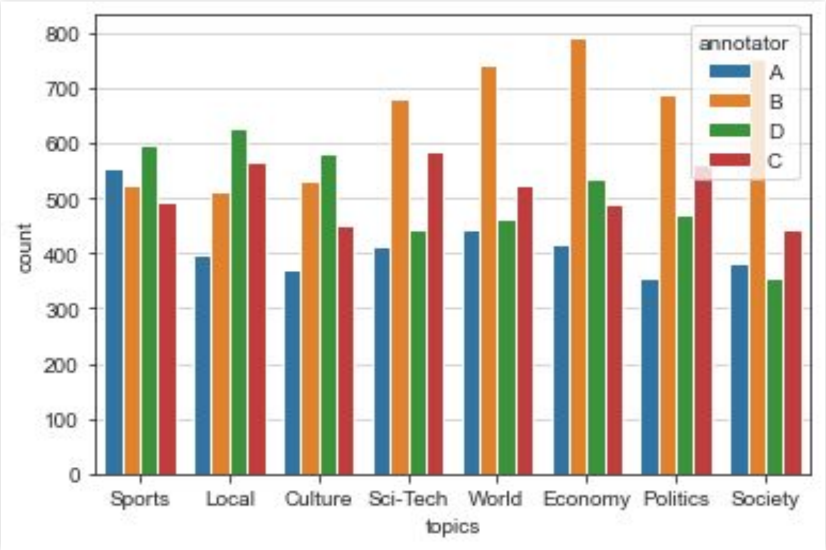


Fig. 6 Distribution of Topics regarding Annotator

Exploratory Analysis

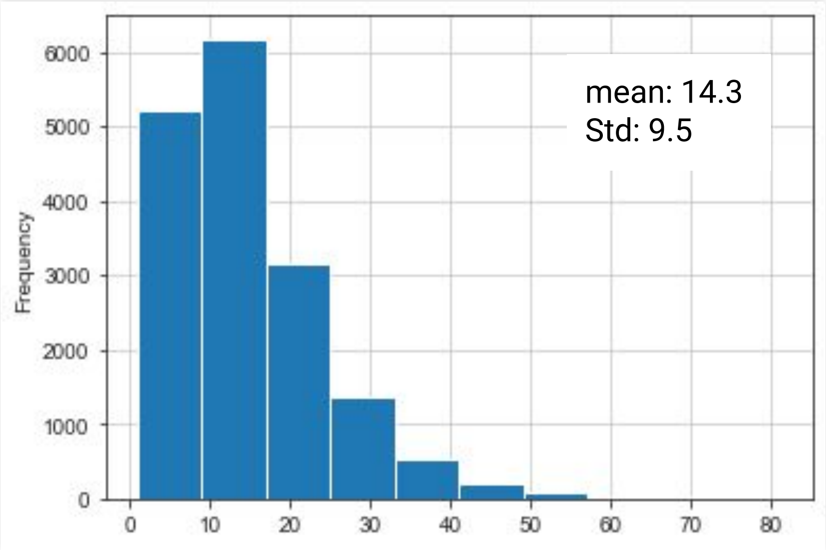


Fig. 7 Tokens length

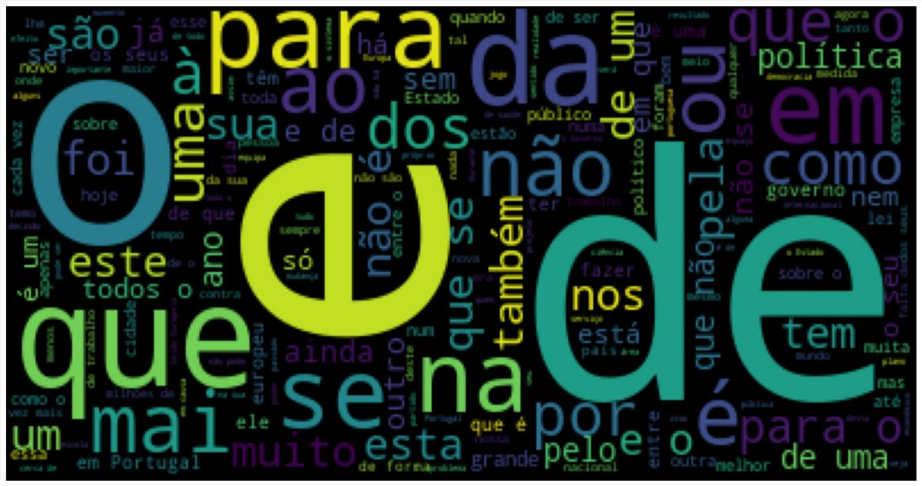


Fig. 8 Tokens WordCloud

Relevant fields used

Dataset

OpArticles

01

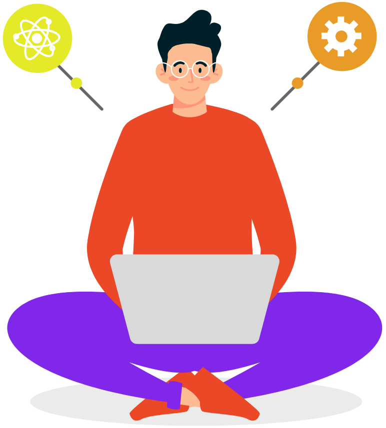
- ***body*** to train a model for Word Embeddings

OpArticles_ADUs

02

- ***tokens*** to build the corpus for the classifiers
- ***label*** to use as the class to predict

Preprocessing



Preprocessing

01 **Remove non alphabetic characters**

re.sub('[^a-zA-Z\u00C0-\u00ff]', ' ', ...)

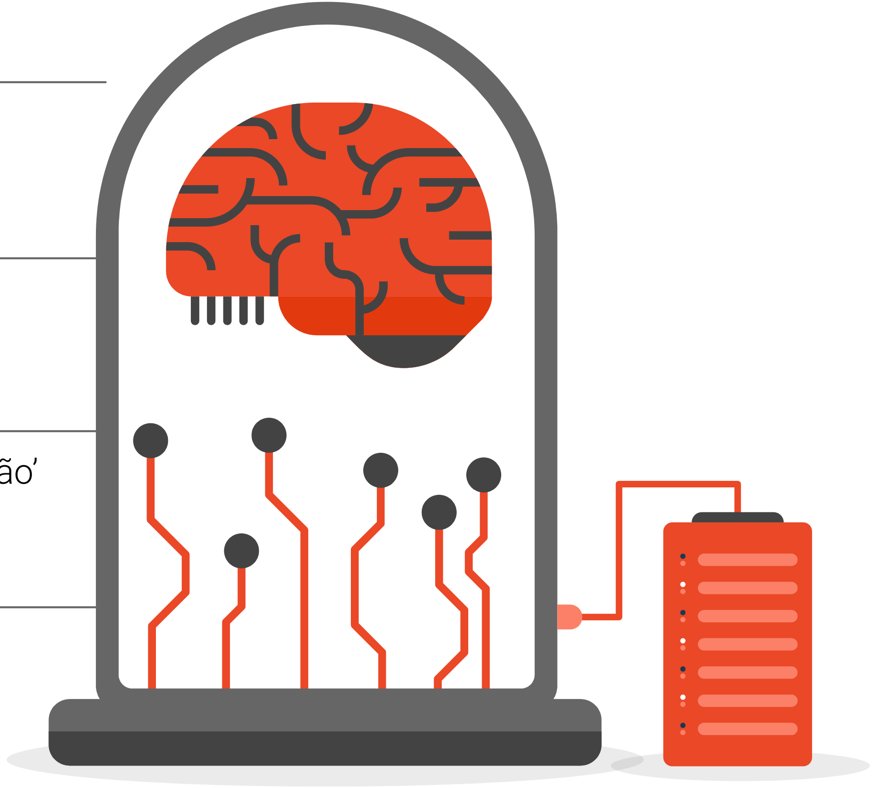
02 **Lowercase tokens**

03 **Remove stopwords**

Portuguese stopwords from *nltk.corpus* except 'não'

04 **Apply stemming**

PorterStemmer, *Snowball PortugueseStemmer*, *RSLPStemmer*

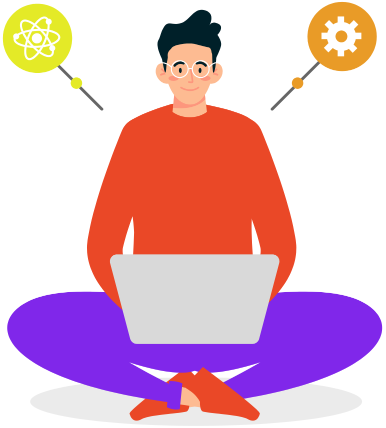


Stemmer Comparison

Tab. 1 Different Stemmers Metrics

	Preprocess time (s)	Corpus Set length	Number of features
PorterStemmer	4.14	11761	15170
Snowball PortugueseStemmer	4.63	11753	9148
RSLPStemmer	7.98	11753	8256

Representation Techniques



Vectorizers

Bag of Words

Ignores word sequence

CountVectorizer - converts a collection of text documents to a matrix of token counts.

One hot Encoding

1-hot vector

CountVectorizer with binary parameter - represent each review as a 1-hot vector with a 0 or a 1 for each of the features.

Tf-idf

TF of a word is multiplied by its IDF

TfidfVectorizer - provides a way to directly obtain TF-IDF weighted features.

Tab. 2 Different Vectorizers Metrics		Vectorizer fit_transform time (s)	Number of features	Model fit-predict time (s)	Accuracy	F1-score
Bow	Unigram ngram_range=(1,1)	0.38	8256	6.57	0.48	0.49
1-hot		0.30		8.15	0.49	0.49
TF-IDF		0.37		0.37	0.50	0.50
Bow	Bigram ngram_range=(2,2)	0.51	61558	57.79	0.39	0.42
1-hot		0.67		56.99	0.39	0.42
TF-IDF		0.71		3.57	0.39	0.43
Bow	UniBigram ngram_range=(1,2)	0.83	**69814**	66.67	0.46	0.47
1-hot		0.83		67.39	0.46	0.47
TF-IDF		**0.67**		**5.01**	**0.52**	**0.52**

Test suite with the before mentioned preprocessing, 80/20 split, and ComplementNB classifier

Vectorizer Parameters

TfidfVectorizer with ngrams=(1,2) will be the vectorizer used in the pipeline since it leads to considerably **faster classification times** and **better metrics**. The following parameters were also explored:

Tab. 3 Addition Parameters to *TfidfVectorizer*

Additional Parameters	Vectorizer fit_transform time (s)	Number of features	Model fit-predict time (s)	Accuracy	F1-score
strip_accents='unicode'	1.30	69263	5.05	0.52	0.52
min_df=3 max_df=0.8	1.64	19262	1.03	0.52	0.52

Tuning the document frequency parameters **reduced drastically** the **number of features** and consequently the **classification times**, which will be useful for the classification tasks.

Word Embeddings

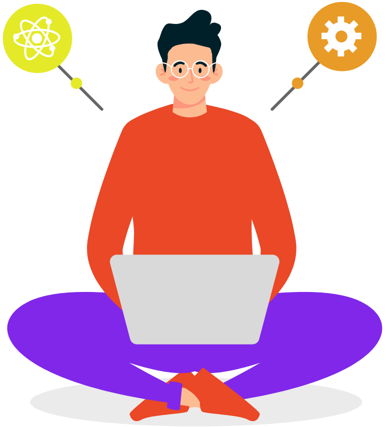
Besides representation with **Vectorizers**, we also tested **text classification** with a ***Word2Vec* model** trained on the ***body*** of the articles, and some **pre-trained models from NILC**.

Tab. 4 Word Embeddings Models

Model	Number of features	Accuracy	F1-score
Article's body	2250	0.49	0.35
Word2Vec CBOW 100	1500	0.48	0.47
Word2Vec Skip-gram 100		0.48	0.46
FastText CBOW 100		0.40	0.41
FastText Skip-gram 100		0.48	0.48
FastText Skip-gram 1000	15000	0.51	0.51

Test suite with the before mentioned preprocessing, 80/20 split, and ComplementNB classifier

Classification & Results



Baseline Classification

Tested a wide range of Classifiers with default parameters in order to assess the base performance.

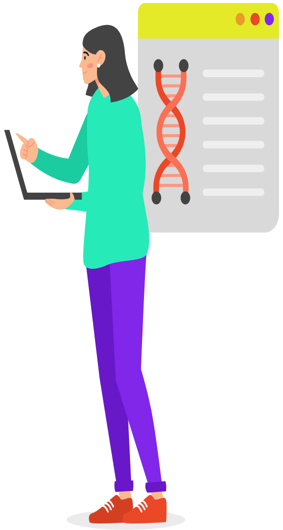
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0, stratify=y)
```

Fig. 9 Train/Test split Function

A stratified train/test split was previously applied in order to have the same label proportion in the train and test sets

Tab. 5 Tested Classifiers with default parameters

Classifier	Elapsed Time (s)	Accuracy	Precision	Recall	F1-score
MultinominalNB	0.95	0.51	0.53	0.51	0.41
ComplementNB	1.21	0.52	0.52	0.52	**0.52**
SGD	29.15	**0.54**	0.53	0.54	**0.52**
LogisticRegression	194.34	**0.53**	0.54	0.53	0.48
SVC (max_iter=100)	225.65	0.41	0.38	0.41	0.35
Perceptron	20.72	0.49	0.50	0.49	0.49
DecisionTree	793.28	0.48	0.48	0.48	0.48
RandomForest	242.21	**0.54**	0.53	0.54	**0.52**
KNeighbors	14.88	0.41	0.41	0.41	0.40
XGBoost	605.67	0.52	0.51	0.52	0.44



SGD - Best base algorithm

Elapsed time: 29.15s

Classification report:

	precision	recall	f1-score	support
Fact	0.46	0.31	0.37	733
Policy	0.55	0.29	0.38	133
Value	0.57	0.77	0.66	1621
Value(+)	0.47	0.30	0.36	282
Value(-)	0.52	0.37	0.43	580
accuracy			0.54	3349
macro avg	0.51	0.41	0.44	3349
weighted avg	0.53	0.54	0.52	3349

Fig. 10 Classification Report

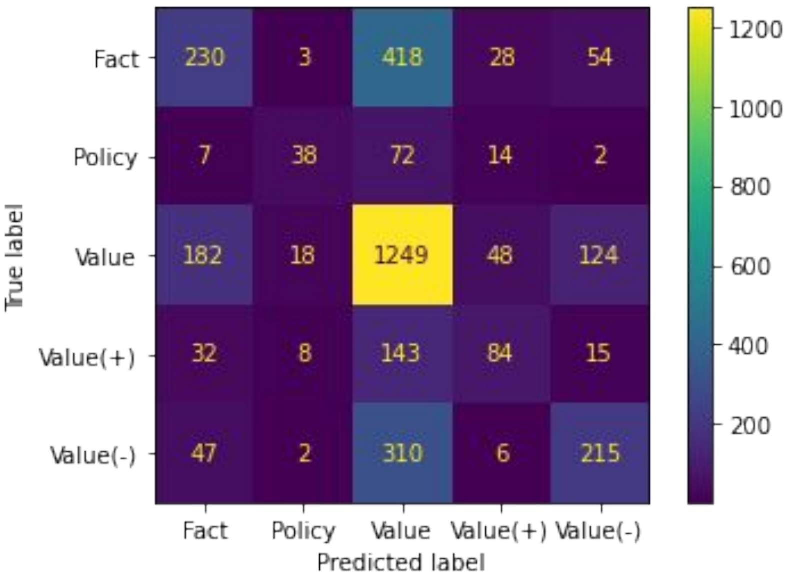


Fig. 11 SGD Confusion Matrix

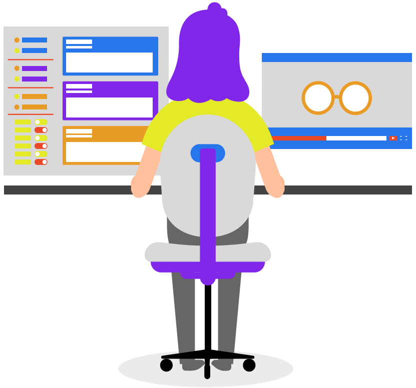
Hyperparameter Tuning

```
grid_search = GridSearchCV(clf,
                           param_grid=parameter_grid,
                           scoring='f1_weighted',
                           cv=StratifiedKFold(n_splits=5),
                           verbose=4,
                           n_jobs=2,
                           refit=True)
```

Fig. 12 GridSearch Function

Perform *GridSearch* through the parameters to obtain the best model regarding f1-score.

StratifiedKFold is used in order to have the same label proportion in the several train and validation sets.



Applied only to the top 2 base algorithms, ***SGD*** and ***RandomForest*** due to time constraints.

Hyperparameter Tuning

SGD

```
{
    'loss': ['log', 'hinge', 'perceptron'],
    'penalty': ['elasticnet', 'l1', 'l2'],
    'class_weight': [None, 'balanced']
}
```

```
Fit time: 6288.64s

Best score: 0.4924921335782078
Best parameters: {'class_weight': None, 'loss': 'hinge', 'penalty': 'l2'}
Best estimator: SGDClassifier(early_stopping=True, n_jobs=-1, random_state=0)

Classification report:
              precision    recall  f1-score   support

        Fact       0.52      0.20      0.29       733
      Policy       0.67      0.25      0.36       133
       Value       0.54      0.85      0.66      1621
    Value(+)       0.59      0.15      0.23       282
    Value(-)       0.47      0.34      0.39       580

    accuracy                           0.53      3349
   macro avg       0.56      0.36      0.39      3349
weighted avg       0.53      0.53      0.49      3349
```

Fig. 13 SGD Classification Report and Parameter List

Random Forest

```
{
    'criterion': ['gini','entropy'],
    'max_features': ['sqrt', 'log2'],
    'max_depth' : [30, 50, 100],
    'class_weight': [None, 'balanced']
}
```

```
Fit time: 2798.36s

Best score: 0.49320507397648533
Best parameters: {'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 100,
'max_features': 'sqrt'}
Best estimator: RandomForestClassifier(class_weight='balanced', criterion='entropy',
                       max_depth=100, max_features='sqrt', n_jobs=-1,
                       random_state=0)

Classification report:
              precision    recall  f1-score   support

        Fact       0.38      0.45      0.41       733
      Policy       0.58      0.49      0.53       133
       Value       0.59      0.61      0.60      1621
    Value(+)       0.42      0.35      0.38       282
    Value(-)       0.48      0.37      0.42       580

    accuracy                           0.51      3349
   macro avg       0.49      0.45      0.47      3349
weighted avg       0.51      0.51      0.51      3349
```

Fig. 14 Random Forest Classification Report and Parameter List

Additional Preprocessing - Resampling

Since the dataset is very **imbalanced**, we initially tried some basic resampling techniques, but all yield **worst** results.

- **Undersampling**
  - *RandomUnderSampler* (f1-score: 0.36)
  - *NearMiss* (f1-score: 0.24)
- **Oversampling**
  - *RandomOverSampler* (f1-score: 0.46)
  - *SMOTE* (f1-score: 0.48)

Another strategy to balance the dataset is to perform **data augmentation** through **translations**: translate the tokens from the minority classes to a random language and then translate it back to portuguese.

```
Original sentence:
Estou a estudar para engenheiro, o Porto berra quando eu passo.
Gosto muito de processamento de linguagem natural

Iter 0 with lang te
['Estou a estudar engenharia, grita Porto quando passo. Eu amo
o processamento de linguagem natural']

Iter 1 with lang de
['Estou a estudar engenharia, grita Porto ao passar. Eu
realmente gosto de processamento de linguagem natural']

Iter 2 with lang nl
['Estou estudando para ser engenheiro, grita Porto quando
passo. Eu realmente gosto de processamento de linguagem
natural']
```

Fig. 15 Example of back translation in three different languages

Additional Preprocessing - Data Augmentation

A **train/test split** was executed in the original dataset and the **data augmentation** was only performed in the train set in order to avoid data leakage to the test set.

A base ***SGDClassifier*** was used in these new sets to evaluate the performance:

```
Elapsed time: 53.67s

Classification report:
              precision    recall  f1-score   support

        Fact       0.42      0.45      0.43       733
      Policy       0.35      0.65      0.46       133
       Value       0.65      0.39      0.49      1621
    Value(+)       0.27      0.57      0.37       282
    Value(-)       0.43      0.57      0.49       580

    accuracy                           0.46      3349
   macro avg       0.43      0.53      0.45      3349
weighted avg       0.52      0.46      0.46      3349
```

Fig. 16 SGD Classification Report after data augmentation

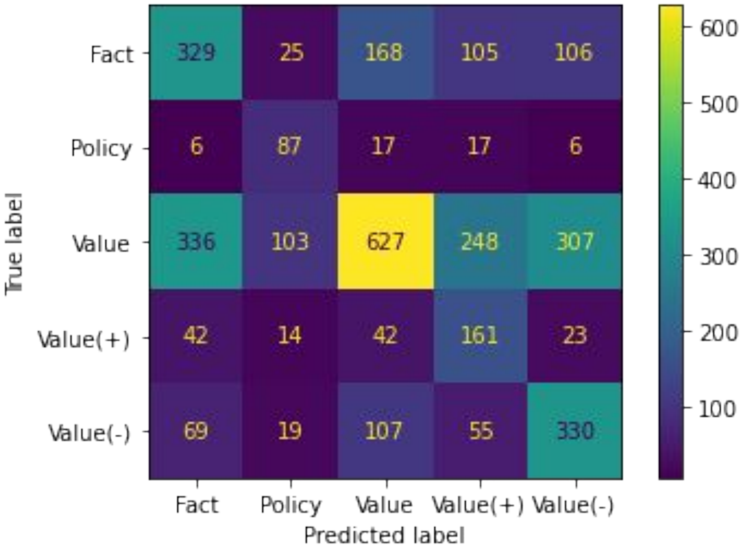


Fig. 17 SGD Confusion Matrix after data augmentation

Additional Preprocessing - Majority Voting

The last experiment consisted in **removing duplicate** annotated ADUs, only **keeping one** with the **label that most annotators agreed** on.

Elapsed time: 9.05s

Classification report:

	precision	recall	f1-score	support
Fact	0.49	0.39	0.43	637
Policy	0.34	0.21	0.26	107
Value	0.53	0.74	0.62	1135
Value(+)	0.34	0.13	0.19	170
Value(-)	0.42	0.23	0.29	353
accuracy			0.50	2402
macro avg	0.42	0.34	0.36	2402
weighted avg	0.48	0.50	0.48	2402

Fig. 18 SGD Classification Report after majority voting

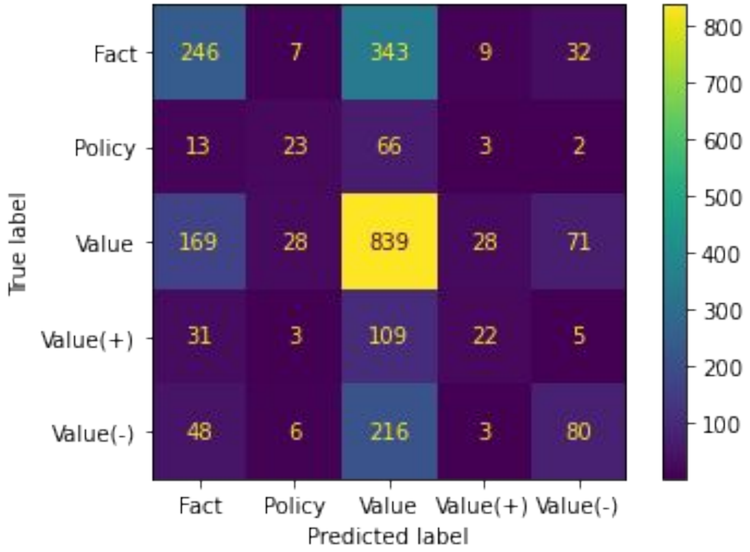
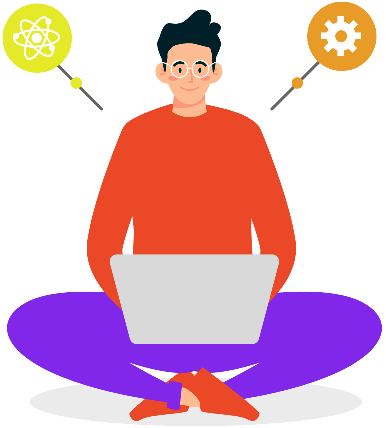


Fig. 19 SGD Confusion Matrix after majority voting

Results Discussion



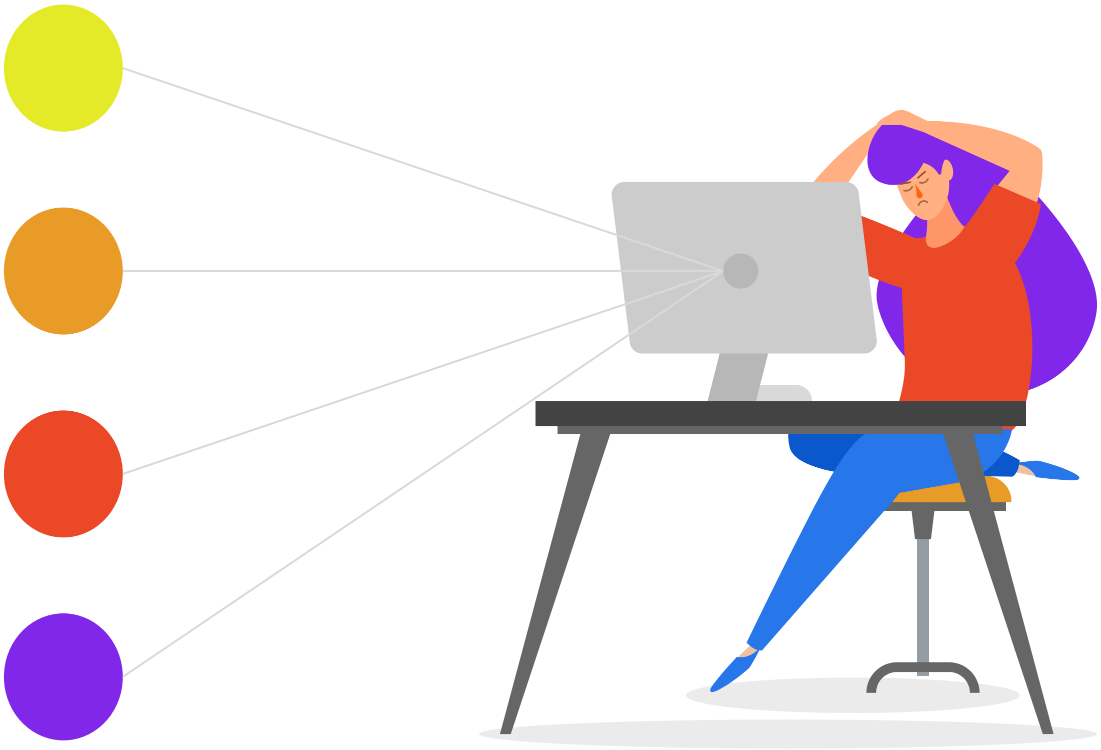
Results Discussion

Vectorizer parameter exploration proved to be helpful to **reduce classification times** and achieved the **same** scores with **fewer** features.

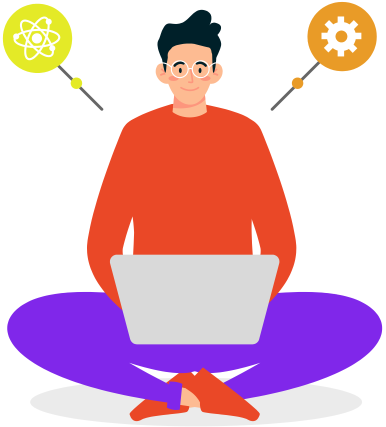
Simple resampling techniques and even **data augmentation** did **not enhance** performance.

Unable to explore **parameters** on all models since *GridSearch* would take **too long**.

After all the experimentations, **baseline SGD** was still the classifier with the **best** scores



Conclusions



Conclusions

- This project was hampered by poor data quality;
- Large amount of features impacted the classification times, and therefore the time spent in this project;
- Results below expectation.

Future Work

- Gather new data, related to ADUs classifications, to balance the dataset.
- Explore more preprocessing techniques;
- Delve into POS tagging and NER techniques;

Thank you!

Do you have any questions?

References

- Sklearn documentation
- Common pitfalls and recommended practices
- Classificação de Textos em Python
- Machine Learning, NLP: Text Classification using scikit-learn, python and NLTK.
- Multi-Class Text Classification Model Comparison and Selection
- Word2Vec and FastText Word Embedding with Gensim
- Data Augmentation in NLP: Best Practices From a Kaggle Master
- How I handled imbalanced text data