Latent text analysis in ML: Predicting what, which and when.

"Lord of the Strings" - a title given by Minstrel

Matias T, Emma F, Jonas J

June 9, 2026

University of Copenhagen, NBI

Outline

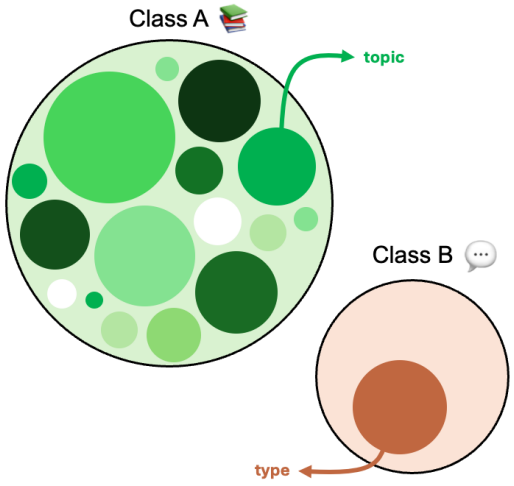
Data origin

Data Origin

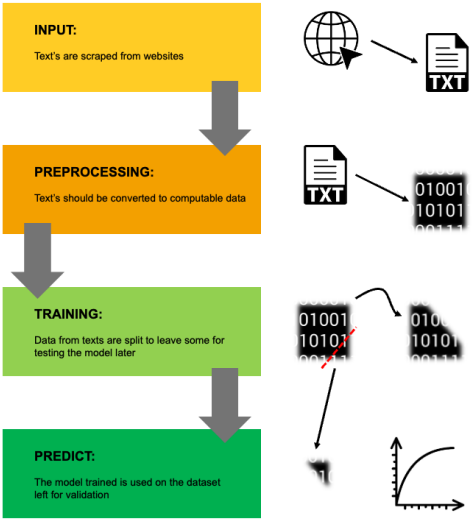
Two main classes of text's

- Class A: Written text
- Class B: Speeches

Class	A	B
Number of texts	780044	4753
Number of subclasses	> 50	2
avg. text length (words)	446 ± 749	3714 ± 3942



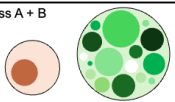
The problems to tackle



Main purpose of each process step

- Input: From webside to txt file
- Preprocessing: From txt file to data
- Training: Training models
- Predict: Predicting unknown data-sets

The problems to tackle

PROBLEM	CLASSES	MODEL	PREPROCESSING
Binary Classification	Class B 	BDT	Trigrams
			NLTK
		NN	Trigrams
			NLTK
Binary Classification	Class A + B 	BDT	Trigrams
			NLTK
		NN	Trigrams
			NLTK
Multi Classification	Class A 	BDT	Trigrams
			NLTK
		NN	Trigrams
			NLTK
Regression	Class A 	BDT	Trigrams
			NLTK
		NN	Trigrams
			NLTK

Specific problems to solve

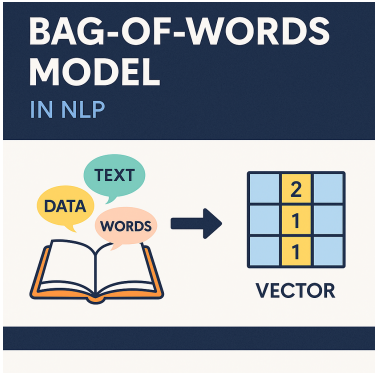
- Input: Finding enough texts, variying in both date and content type. (Class A and B)
- Preprocessing: Turning many unique strings (words) into a reasonable amount of features
- Training: Trying both NN (Pytorch) and BDT (Adaboost).
- Predict: Using the trained models on the unknown data

Tokenization and Preprocessing

Initial Tokenization

General method: Bag-of-words

- Make a list of one of the following
  - Characters ("a", "b", ..., "å", " ")
  - Trigrams of chars ("lys", "r d", "åeæ")
  - Full words ("dansk", "lars", "bedr")
- Count in each text, occurrences of each number
- For each text a "Bag of Words" is now obtained, this is a **vector**
- To analyze the entire dataset these are put together in sparse matrices (next slide)



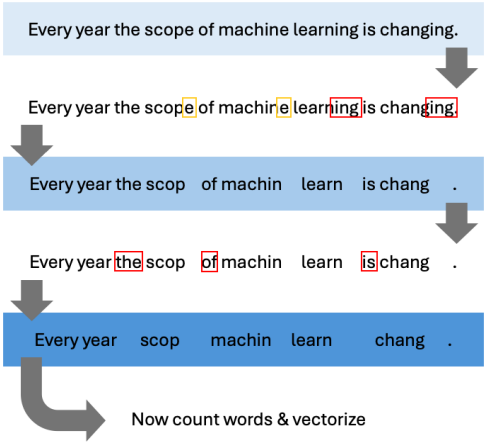
Source: https://aiml.com/what-is-bag-of-words-model/

To reduce sparsity further, redundant or similar words are preprocessed:

NLP Preprocessing methods

- Stemming (Remove conjugations)
- Stopwords (Remove common tokens)
- Embedding (Vectorize tokens)

[Ex 1:] One-hot encoding counts token occurance, but inefficient.
[Ex 2:] TF-IDF weighs by frequency and text similarity. SKlearn has a vectorizer for sparse TF-IDF matrices,



Sparsity

Data are extremely sparse ($> 95\%$)

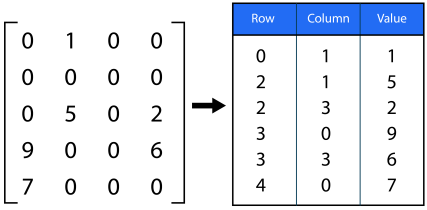
- Speeches sparsity 0.035
- Article sparsity 0.001

High memory usage $\rightarrow$ slower training

Usage of Compressed sparse row arrays

Sparsity-aware algorithms for efficiency

Sparse Matrix



Source: https://botpenguin.com/glossary/sparse-matrix

Dimensionality reduction

Data has extremely high dimensionality, though, with low information content in a single dimension.

- Trigram tokenization results in 27000 features.
- NLTK / word tokenization can be near unbounded in features. (Corpus)

Dimension reduced by UMAP/SVD embedding, fitted to part of the data or by excluding very rare or common features.

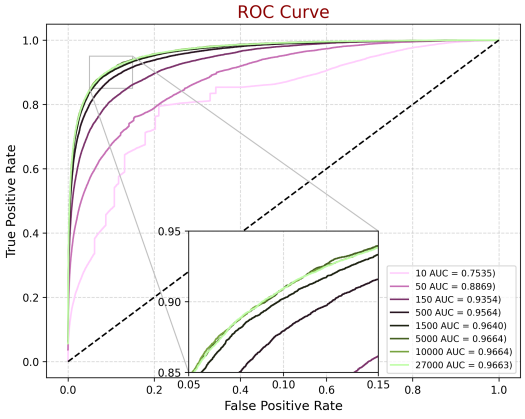


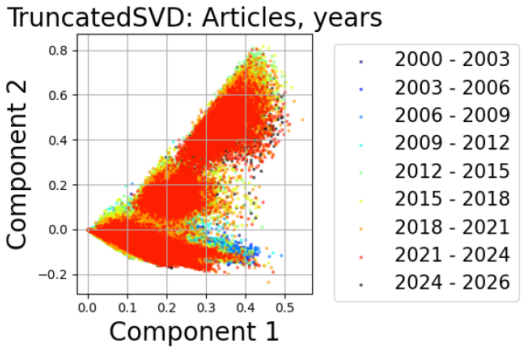
Figure of ROC curves with different amounts of variables.

Dimensionally reduced 2D plots

TruncatedSVD for TF-IDF or Sparse data. Essential for Latent Semantic Analysis.

Singular value decomposition is used to reduce the number of columns while preserving the similarity structure among rows.

Umap also used for dimensional reduction.



Training and Results

Hyper-Parameter Optimization

Optuna used for HPs with some trial and error to determine number of NN layers.

Not all data used for hyperparameter optimization to make sure training time was under 10 min per trial.

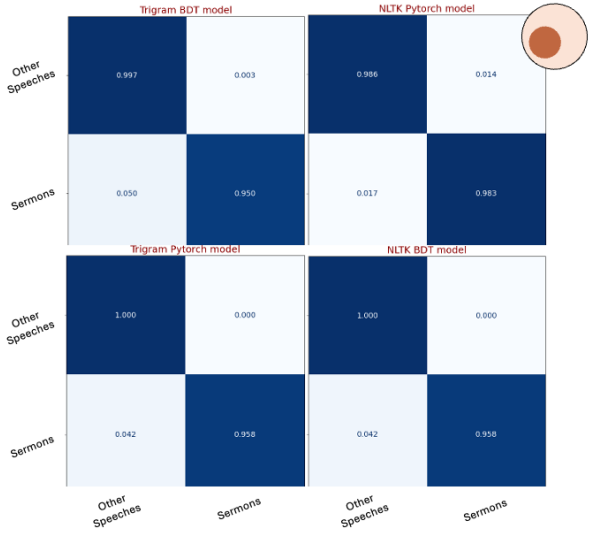
Alternatively use random search. Less precise, but simpler to implement, essential for quick tests of our algorithms like the experiments seen in appendix.

```
params_sermons = {
    "objective": "binary",
    "boosting_type": "gbdt",
    "num_leaves": 350,
    "min_data_in_leaf": 350,
    "learning_rate": 0.1,
    "feature_fraction": 0.75,
    "bagging_fraction": 0.6,
    "bagging_freq": 8,
    "verbose": -1
}
```

Binary classification of sermons

Data is the 4753 speeches.
Quick to train because of low data and low feature importance.
Expected features were important.
Method: [Binary LogLoss], [Accuracy]

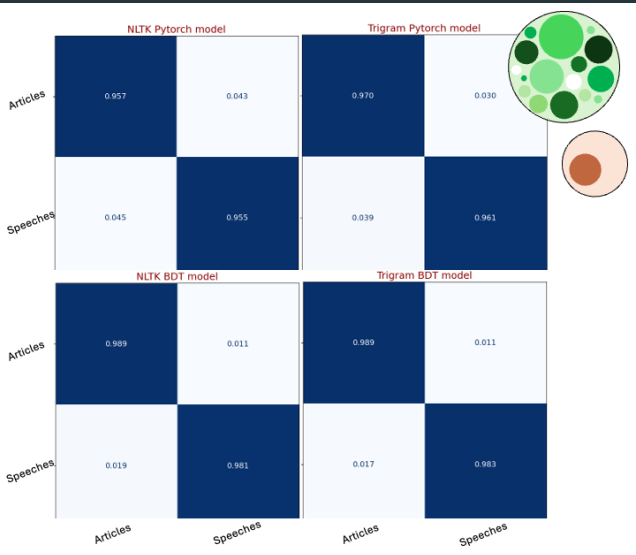
- NLTK Pytorch: 0.0541, 0.985
- NLTK BDT: 0.0284, 0.989
- Trigram Pytorch: 0.0375, 0.989
- Trigram BDT: 0.0357, 0.985



Binary Classification of Speeches

Data is all speeches and articles.
Predict which texts are speeches vs articles.
Weights are used because the data are extremely unbalanced 100:1.
Method: [Binary LogLoss], [Accuracy]

- NLTK Pytorch: 0.0119, 0.981
- NLTK BDT: 0.0074, 0.988
- Trigram Pytorch: 0.0141, 0.974
- Trigram BDT: 0.0065, 0.989



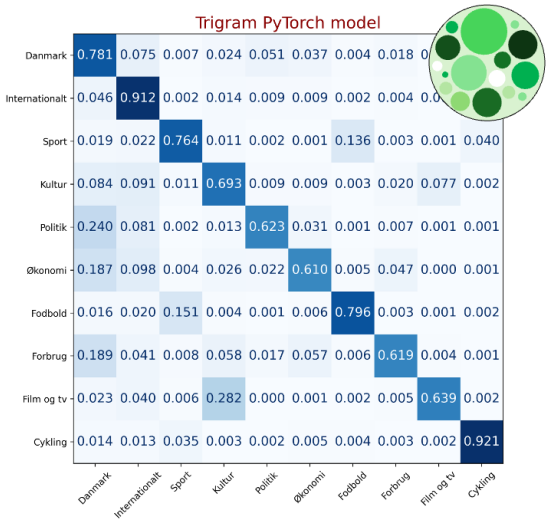
Multiclass Classification of top 10 article topics

Reduction from 780k to 500k texts, by limiting to top 10 most common topics. Needs weighted

Method: [CE LogLoss], [Accuracy]

- NLTK Pytorch: 1.28, 0.70
- NLTK BDT: 0.96, 0.68
- Trigram Pytorch: 0.559, 0.79
- Trigram BDT: 0.558, 0.802

Denmark are generally the most spread, and "cykling" (biking) is the least spread. Other topics show simlarities:
Film & tv with kultur, sport with fodbold.



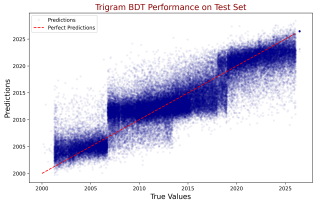
Regression of articles

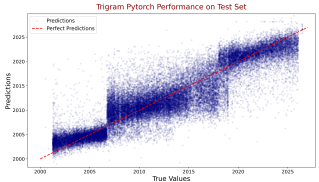
Regression of the article publication year.

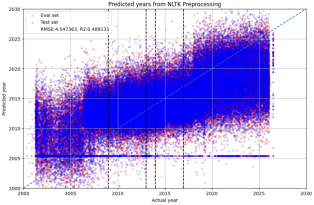
Method: Mean Squared Error (MSE), R^2

- NLTK PyTorch:
- NLTK BDT: 19.37, 0.451
- Trigram PyTorch: 7.75, 0.824
- Trigram BDT: 7.80, 0.808

Three distinct periods:
2000-2006 $\rightarrow$ 2006-2018 $\rightarrow$ 2018-2026
Long text $\rightarrow$ less text $\rightarrow$ Long text
In these periods the formt of texts where distinct, e.g. they would all end on similar sentence.







Very hard to get anything predictive with Pytorch NN + NLTK
Might update before presenting

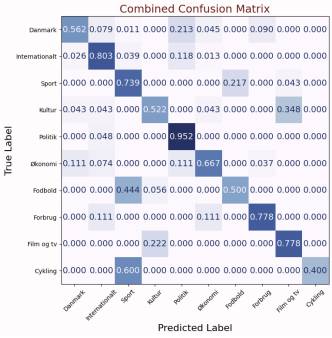
Generative AI Predictions

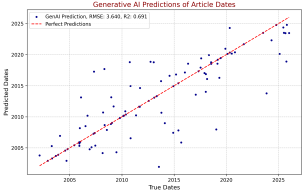
We compared our approach with Microsoft Copilot using 100 randomly selected articles for both classification and publication-date regression.

Task	Data	Result
Classification	Preprocessed text	accuracy = **0.62**
Classification	Raw text	accuracy = **0.68**
Classification	Raw text + priors	accuracy = **0.73**
Regression	Raw text	MSE = **13.25 years**
		R^2 = **0.691**

Conclusion

Copilot performed okay on, but doesnt consistently beat our models.





Conclusion

Conclusion: What did we learn?

Acquiring Data & Preprocessing

Scraping articles takes a long time!

Classification:
All acquired data works well
Trigrams & NLTK gives similar results

Regression:
Only data with uniform temporal spread works somewhat
Trigrams give better results than NLTK, an indication of the context that trigrams provide

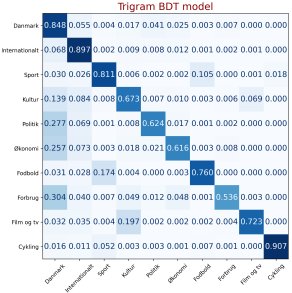
Hyperparameter Optimization

Running preemptive models to find important features is essential

NN:
Hard to determine number of layers.

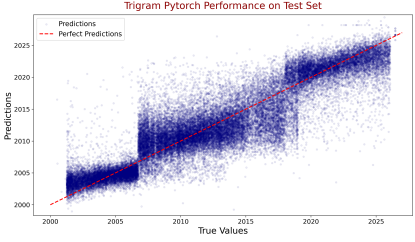
BDT: Simple to find good hyperparameters
DIfficult to know if some non-optimized HPs were important since many parameters exist.

Conclusion: What worked?



Classification models can realistically be used with our types of models and the amount of data!

Regression is harder to make work with our data and models. Might be due to slim range of data, and some artifacts in the way the text is written in the different intervals of time.



Thank You!

Questions?

Appendix

Appendix: Classification of sermons, Hyper parameters

All optimized by Optuna, though, the amount of layers was some trial and error.

```python
params_sermons = {
    "objective": "binary",
    "boosting_type": "gbdt",
    "num_leaves": 350,
    "min_data_in_leaf": 350,
    "learning_rate": 0.1,
    "feature_fraction": 0.75,
    "bagging_fraction": 0.6,
    "bagging_freq": 8,
    "verbose": -1
}
```

```python
learning_rate = 7e-05
batch_size    = 64
n_epochs      = 20
class SermonModel(nn.Module):
    def __init__(self):
        super(SermonModel, self).__init__()
        self.input_layer = nn.Linear(251, 512)
        self.hidden_layer1 = nn.Linear(512, 128)
        self.hidden_layer2 = nn.Linear(128, 12)
        self.output_layer = nn.Linear(12, 2)
        self.relu = nn.ReLU()

    def forward(self, inputs):
        x = self.input_layer(inputs)
        x = self.relu(x)
        x = self.hidden_layer1(x)
        x = self.relu(x)
        x = self.hidden_layer2(x)
        x = self.relu(x)
        x = self.output_layer(x)
        return x
```

```python
params_sermons_nltk = {
    "objective": "binary",
    "boosting_type": "gbdt",
    "num_leaves": 450,
    "min_data_in_leaf": 300,
    "learning_rate": 0.2,
    "feature_fraction": 0.65,
    "bagging_fraction": 0.5,
    "bagging_freq": 3,
    "verbose": -1
}
```

```python
learning_rate = 7e-05
batch_size    = 64
n_epochs      = 20
class SermonModelNLTK(nn.Module):
    def __init__(self):
        super(SermonModelNLTK, self).__init__()
        self.input_layer = nn.Linear(7328, 256)
        self.hidden_layer1 = nn.Linear(256, 128)
        self.hidden_layer2 = nn.Linear(128, 8)
        self.output_layer = nn.Linear(8, 2)
        self.relu = nn.ReLU()

    def forward(self, inputs):
        x = self.input_layer(inputs)
        x = self.relu(x)
        x = self.hidden_layer1(x)
        x = self.relu(x)
        x = self.hidden_layer2(x)
        x = self.relu(x)
        x = self.output_layer(x)
        return x
```

Appendix: Classification of speeches, Hyper parameters

All optimized by Optuna, though, the amount of layers was some trial and error.

```python
params_speeches = {
    "objective": "binary",
    "boosting_type": "gbdt",
    "scale_pos_weight": 100.0,
    "num_leaves": 250,
    "min_data_in_leaf": 375,
    "learning_rate": 0.02,
    "feature_fraction": 0.95,
    "bagging_fraction": 0.65,
    "bagging_freq": 5,
    "verbose": -1
}
```

```python
learning_rate = 0.00044983189913331916
batch_size    = 16
n_epochs      = 4
class SpeechModel(nn.Module):
    def __init__(self):
        super(SpeechModel, self).__init__()
        self.input_layer = nn.Linear(251, 392)
        self.hidden_layer1 = nn.Linear(392, 156)
        self.hidden_layer2 = nn.Linear(156, 128)
        self.output_layer = nn.Linear(128, 2)
        self.relu = nn.ReLU()

    def forward(self, inputs):
        x = self.input_layer(inputs)
        x = self.relu(x)
        x = self.hidden_layer1(x)
        x = self.relu(x)
        x = self.hidden_layer2(x)
        x = self.relu(x)
        x = self.output_layer(x)
        return x
```

```python
params_speeches_nltk = {
    "objective": "binary",
    "boosting_type": "gbdt",
    "scale_pos_weight": 100,
    "num_leaves": 900,
    "min_data_in_leaf": 900,
    "learning_rate": 0.25,
    "feature_fraction": 0.7,
    "bagging_fraction": 0.6,
    "bagging_freq": 7,
    "verbose": -1
}
```

```python
learning_rate = 3e-6
batch_size    = 16
n_epochs      = 4
class SpeechModelNLTK(nn.Module):
    def __init__(self):
        super(SpeechModelNLTK, self).__init__()
        self.input_layer = nn.Linear(2001, 512)
        self.hidden_layer1 = nn.Linear(512, 256)
        self.hidden_layer2 = nn.Linear(256, 32)
        self.output_layer = nn.Linear(32, 2)
        self.relu = nn.ReLU()

    def forward(self, inputs):
        x = self.input_layer(inputs)
        x = self.relu(x)
        x = self.hidden_layer1(x)
        x = self.relu(x)
        x = self.hidden_layer2(x)
        x = self.relu(x)
        x = self.output_layer(x)
        return x
```

Appendix: Hyper parameters for multiclass and regression BDTs

```
params_multiclass = {
    "objective": "multiclass",
    "num_class": 10,
    "boosting_type": "gbdt",
    "num_leaves": 100,
    "min_data_in_leaf": 300,
    "learning_rate": 0.03,
    "feature_fraction": 0.8,
    "bagging_fraction": 0.8,
    "bagging_freq": 2,
    "device": "cpu",
    "verbose": -1
}
```

Multiclassification BDT (NLTK)

HP	Value
N estimators	539
Learning rate	4.58×10
Max depth	8
Min child weight	4
Gamma	0.196
Reg alpha	0.299
Reg lambda	0.0.127
Subsample	0.702

```
params_regress = {
    "objective": "regression",
    "boosting_type": "gbdt",
    "num_leaves": 275,
    "min_data_in_leaf": 25,
    "learning_rate": 0.03,
    "feature_fraction": 0.75,
    "bagging_fraction": 0.65,
    "bagging_freq": 3,
    "verbose": -1
}
```

Article Regression BDT (NLTK)

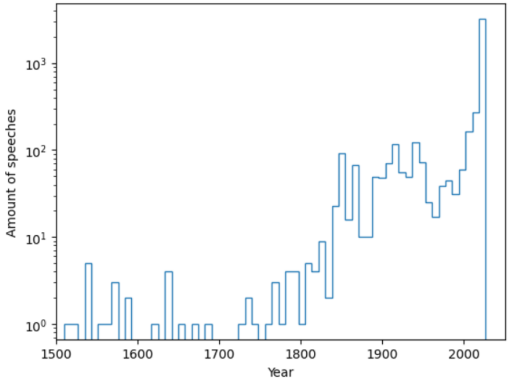
HP	Value
N estimator	629
Learning rate	4.89×10
Max depth	8
Min child weight	7
Reg alpha	0.413
Reg lambda	0.508
Subsample	0.777
colsample bytree	0.836

Appendix: Hyperparameter (HP) Optimization of PyTorch Models (Articles)

Multiclassification PyTorch (Trigrams)		**Multiclassification PyTorch (NLTK)**		**Regression PyTorch (Trigrams)**		**Regression PyTorch (NLTK)**	
HP	Value	HP	Value	HP	Value	HP	Value
Learning rate	1.76×10^{-4}	Learning rate	6.30×10^{-5}	Learning rate	1.32×10^{-5}	Learning rate	1.68×10^{-4}
Batch size	64	Batch size	64	Weight decay	6.15×10^{-6}	Weight decay	4.85×10^{-6}
Layer 1	245	Layer 1	463	Batch size	16	Batch size	256
Layer 2	81	Layer 2	134	Layer 1	937	Layer 1	890
Layer 3	60	Layer 3	56	Layer 2	163	Layer 2	548
Dropout	0.196	Dropout	0.214	Layer 3	121	Layer 3	164
				Dropout	0.130	Dropout	0.144

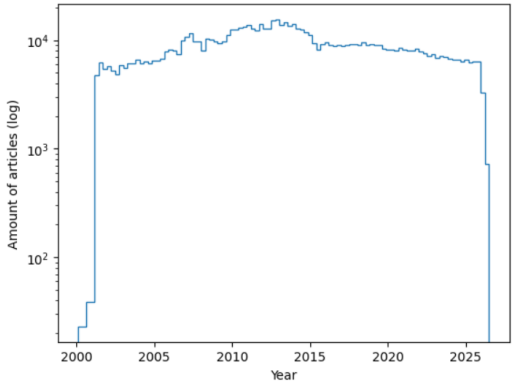
Hyperparameters optimized with Optuna. Number of hidden layers selected through a combination of Optuna optimization and manual experimentation.

Appendix: Distribution of speech dates



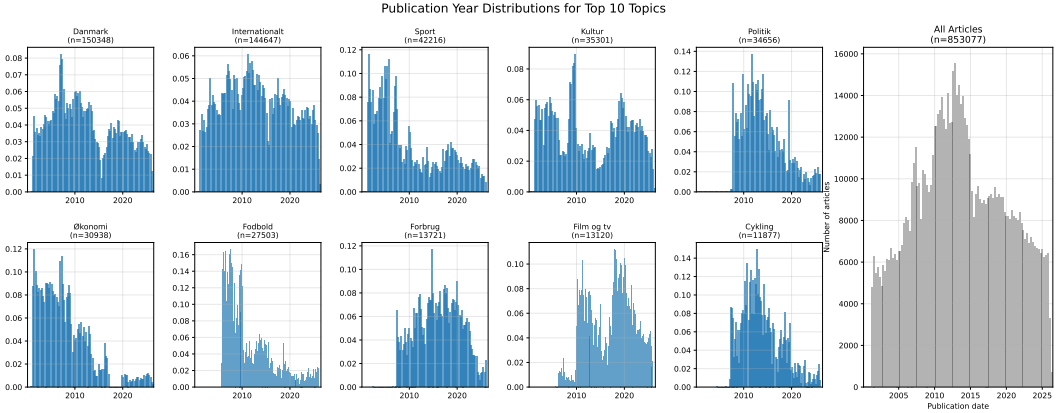
Logarithmic scale distribution of speech dates. The extremely uneven distribution makes date regression very difficult.

Appendix: Distribution of article dates



Logarithmic scale distribution of article dates. Evenly spread dates make regression easier, but the small timeframe does not allow a large variance like for speeches.

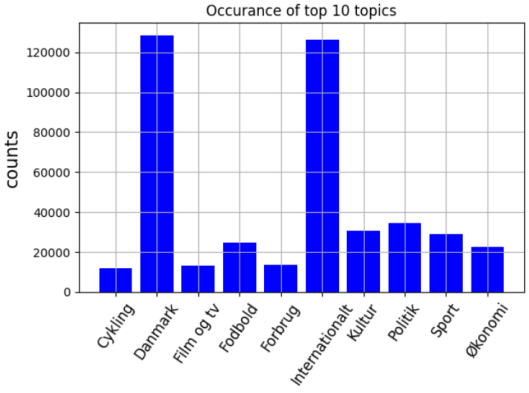
Appendix: Distribution of article dates, for seperated top 10 topics

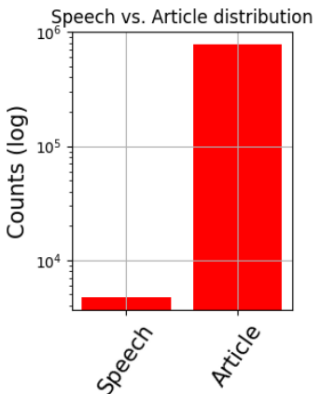


Looking at the distribution of article dates while separating topics, the individual distributions a significantly different. If this reflects a change in which articles are written, then a regressor might have learned this underlying structure instead of a more representative model for which times speeches happen.

Appendix: Unbalanced data

The ratio of Speeches to Articles is very large. As well as the distribution of the top 10 topics of the articles. We must then take care to weigh the to classes when training.





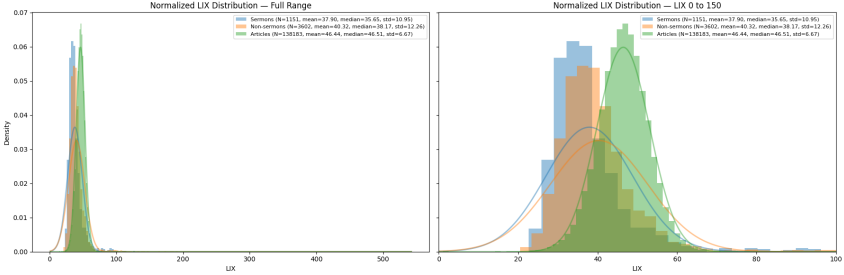
Whoops, switched labels :(

Appendix: Meta Data & LIX

We attempted to train using meta data of the texts. The LIX (Readability of text).

$$\text{LIX} = \text{avg. sentence length} + \% \text{ of words with } +6 \text{ letters}$$

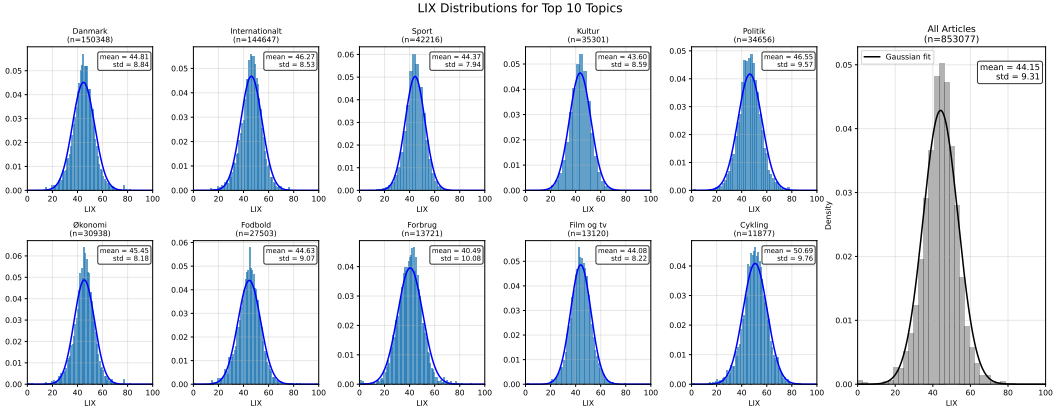
Ranging from 20 ("very easy") to +60 ("very difficult"). Though it didn't (OR DID IT?) change much.



Distribution of LIX for speeches, Only a sample of articles are used.

Appendix: LIX of articles, wrt. topics

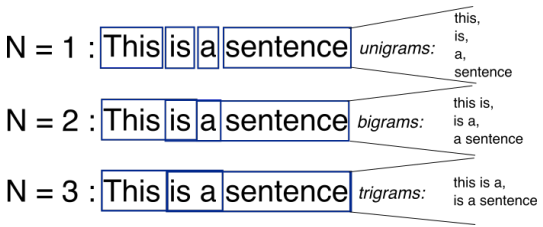
There was not a huge variation of LIX between article topics



Distribution of LIX for articles of top 10 most occuring topics.

Appendix: Trigrams & context

Trigrams retain some context. Either letter-grams or word-grams, depends on use. Letter-grams are less limited by the models corpus (dictionary) and "retains" the order of letters.



src: https://nlp-iiith.vlabs.ac.in/exp/n-grams-smoothing/theory.html

Appendix: Tri-lettergrams example

"ML is nice"

↓

(ML_), (L_i) (_is) (is_) (s_n) (_ni) (nic) (ice)

Many tokens, but they easily overlap between texts, so a few total tokens. This is also useful for grammar / spelling correction.

Appendix: Stemming

Pre-trained models. Recognising conjugations and returning "root" form.

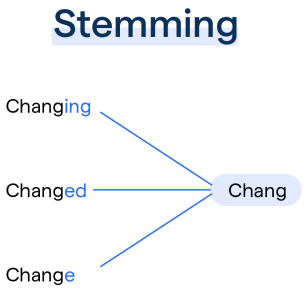
Snowball / Porter

- Rule-based
- Multiple languages
- Aggressive

Lancaster

- Iterative ruling
- more precise, but slow

Alternative is lemmatization.



Source:https://medium.com/thedeephub/nlp-text-preprocessing-part-3-stemming-lemmatization-a3362789d3a7

Appendix: Stopwords

Overly common words can be bad for training

I	me	my	myself	we
our	ours	ourselves	you	his

But sometimes, they are important because of differences in medium. Political speeches use "I" and "our" much more than articles or sermons.

Danish stopwords repository
English stopwords repository

Appendix: Processing example

```
landsmænd dansk student fædreland skyld udret ...
udpræg centralt skøn evangelium kristus saml f...
syndsforlad naad fortælling jesu mød synderind...
ord indehold her forjæt jesus gav discipl foru...
god livsråd grønland fremtid bær ansvar ældr g...
                        ...
thank much dont know say heart jumping uhørbar...
excellenci distinguis delegat partn three deca...
toward green reading european chart local self...
ret uddan børn sær pig vigt ret verdenssamfund...
eupolitik vigt engag sund demokrati valg juni ...
ext, Length: 4753, dtype: object
```

Preprocessed speeches

```
landsmænd og danske studenter sit fædreland sk...
der er noget udpræget centralt over dette skøn...
syndsforladelsens naade med fortællingen om je...
disse ord indeholder en herlig forjættelse som...
ti gode livsråd du er grønlands fremtid men du...
                        ...
oh thank you so much i dont know what to say m...
excellencies distinguished delegates and partn...
towards a green reading of the european charte...
retten til uddannelse for alle børn særligt pi...
hvorfor er eupolitikken vigtig at engagere sig...
```

Unprocessed speeches

Appendix: TF-IDF

How to count words, and which "rare" words are important?

Term Frequency: Common words in a text tell about its themes.

$$TF = \frac{\text{Times word t appears in text d}}{\text{Number of words in text d}}$$

Inverse Document Frequency: Groups texts that share a rare word.

$$IDF = \log\left(\frac{\#\text{ texts}}{\#\text{ texts containing word t}}\right)$$

So common words across texts are weighted lower. Finally, tokens weighted by weight $= \text{TF} \cdot \text{IDF}$.

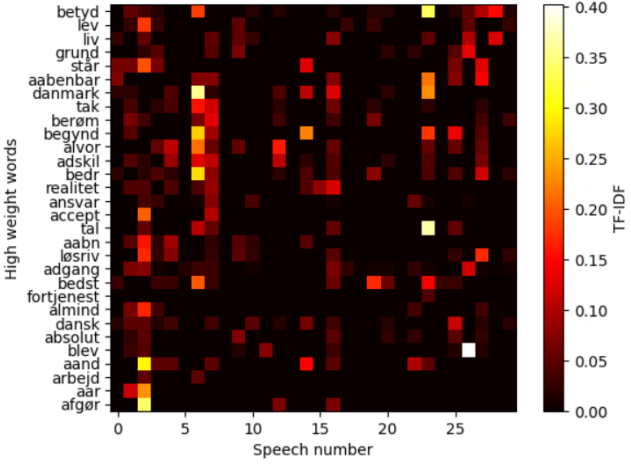
Appendix: Occurence Matrix

Texts sharing a word / token with high TF-IDF value are similar. This can be seen in a co-occurance matrix of tokens and texts.
Dotting the matrix with its transverse gives either, which texts are similar, or which tokens are similar:

$$M^T \cdot M = \textbf{Similarity matrix}$$

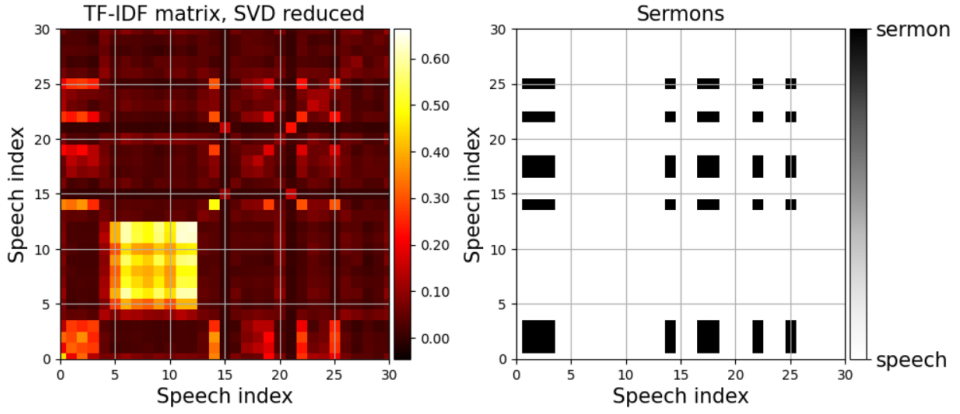
Dimensionality reduction helps with reducing the size of the co-occurance.

Appendix: Occurence Matrix



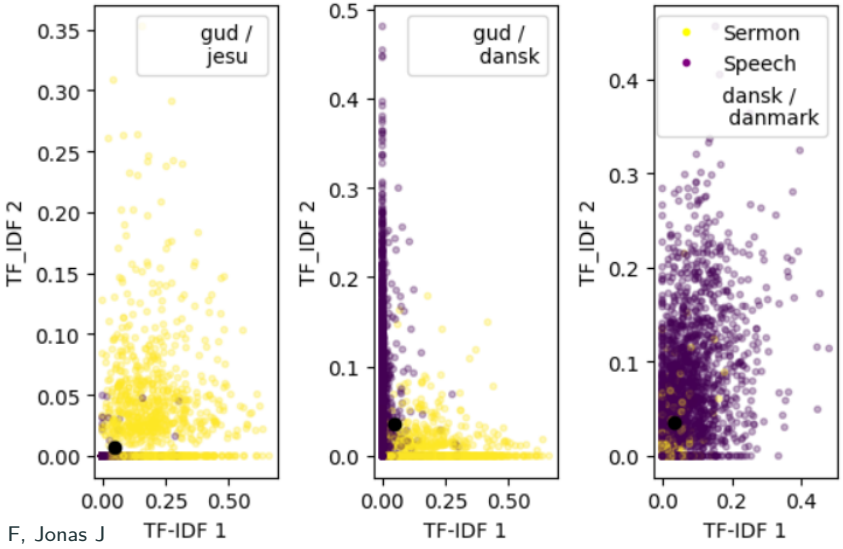
Occurence matrix of the 30 most important features of the regression model and the 30 first texts.

Appendix: Similarity from occurance



TF-IDF similarity matrix and class matrix of 30 first speeches. Texts, that are similar, have a high value in the similarity matrix.

Appendix: Word similarity from TF-IDF, speeches



Regression of Speech dates

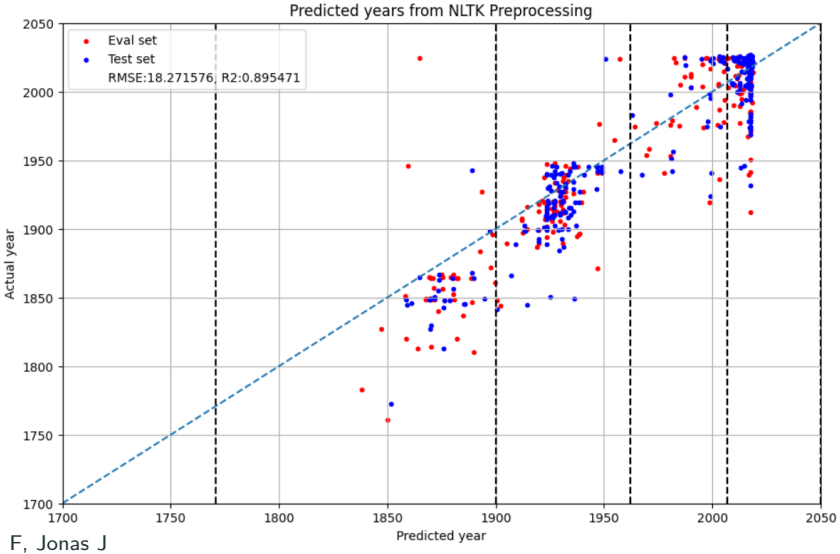
XGBoost model, as dataset has become "small" after preprocessing.
HP optimized by random search (Maybe I should try optuna, though it is slow)
Histogram trees as tf-idf data is continuous. Thus more efficient without major losses.
Bad R^2 score for test set suggests overfitting.

15 most important tokens for training by weight, though all meaning lost by preprocessing:

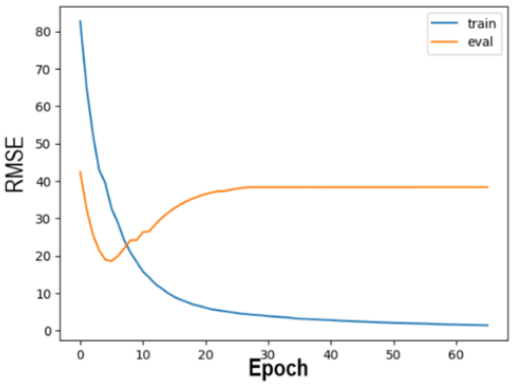
'accept'	'tal'	'aabn'	'løsriv'	'adgang'
'bedst'	'fortjenest'	'almind'	'dansk'	'absolut'
'blev'	'aand'	'arbejd'	'aar'	'afgør'

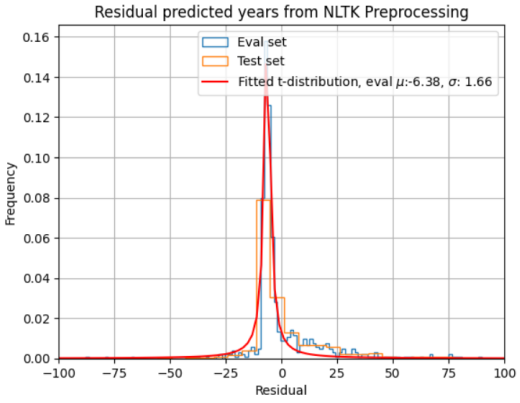
HP	Value
Metric	RMSE
Method	hist
Eval RMSE	18.539
Eval χ^2	0.892
Test RMSE	96.718
Test χ^2	0.201

Appendix: Reggresion of speeches (Eval set)



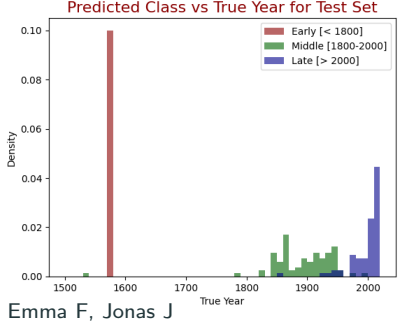
Appendix: Reggresion of speeches, other

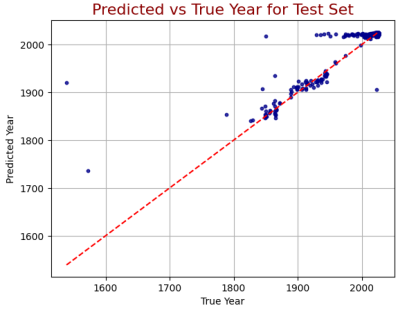




Appendix: Multiclassification + sub-regression of speeches

Because speeches were split in 3 date regions, the date region could be classified with regression for each of the regions. The lack of data in all but the late region as well as lopsidedness in the late region for specifically post 2020 years still made regression problematic, albeit it did actually successfully place many of the middle and later points. The HPs for this experiment was not optimized.





Appendix: Reggresion of speeches, Hyper parameters

HP-optimized by random search.

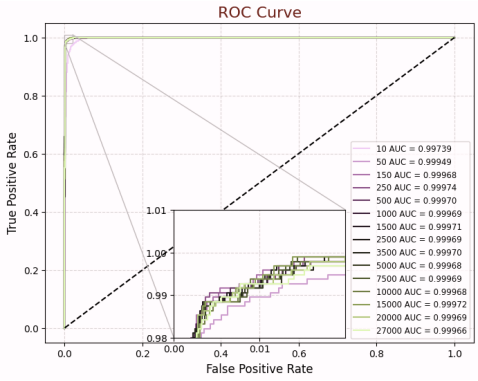
```
model = xgb.XGBRegressor(objective='reg:squarederror',
                         eval_metric = 'rmse',
                         importance_type="weight",
                         n_estimators=200,
                         learning_rate=0.29,
                         max_depth = 11,
                         reg_alpha = 0.7,
                         reg_lambda = 0.82,
                         gamma = 0.63,
                         subsample = 0.93,
                         n_jobs = 1,
                         tree_method = 'hist',
                         early_stopping_rounds = 60,
                         grow_policy = 'lossguide',
                         max_bin = 128,
                         random_state=42)
```

Appendix: Speech dimensionality reduction

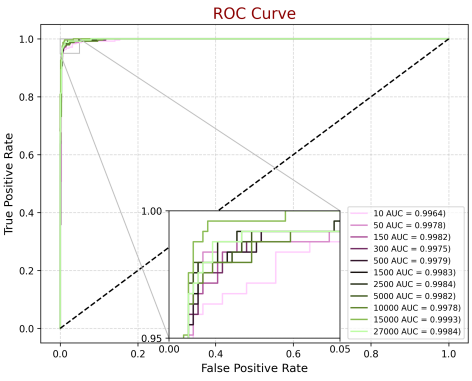
Depending on the preprocessing method, token importance vary, so it should only be necessary to train with the most important, while speeding up the training.

It is apparent in the ROC-curve, as the final AUC-score differes only slightly, when changing the number of features / tokens used.

Appendix: Speech and Sermon ROC curve truncation

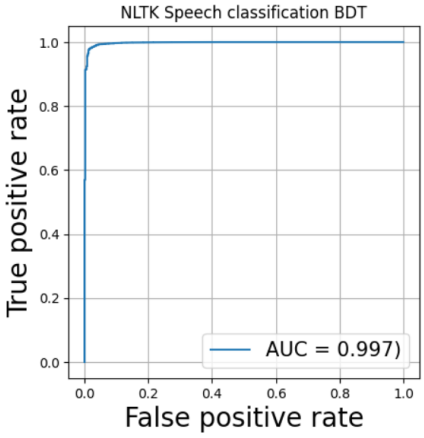


ROC Curve of different feature use for speech classification.



ROC Curve of different feature use for Sermon classification.

Appendix: Speech ROC curve NLTK bdt



ROC Curve of NLTK BDT use for speech classification. Pre-training, data dimensionality reduced to 100 components

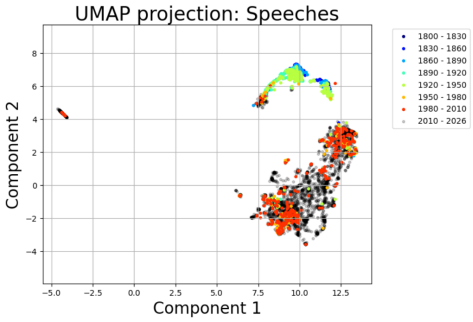
Regression: High weight features

Broad data / high cardinality $\rightarrow$ feature importance by weight rather than gain.

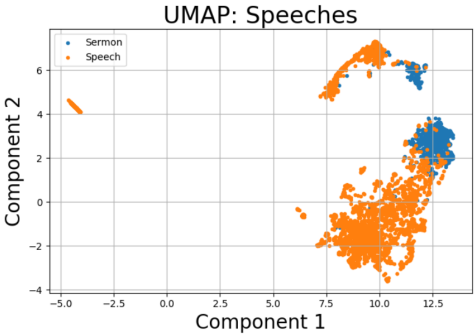
Though this gives insight into which features / tokens are used the most when splitting in a BDT, we don't see the gain of that feature.

Appendix: Latent Semantic Analysis (LSA)

Words with similar meaning should appear in similar texts. Thus they cluster together. UMAP clusters differently than SVD.



Speeches dataset



Speeches dataset

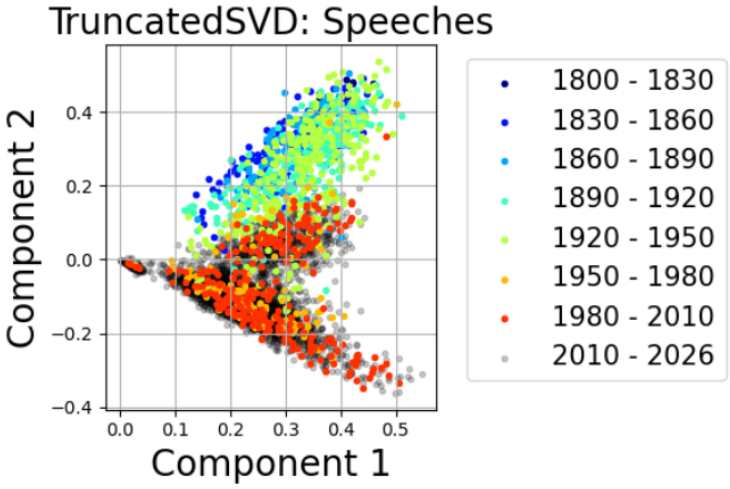
Appendix: Singular Value Decomposition (SVD)

Decompose DataFrame / Matrix to smaller vectors with singular values.
Reduces dimensionality, while preserving data relations (importance, similarity & generality).
Works well with sparse data matrices. Truncated SVD keeps only the top k singular values (k = dimension to reduce to).
Due to the way SVD decomposes, using k-means becomes very straight forward.
Compare the year labelling and class labelling (Next slide); SVD has clustered according to year and class!

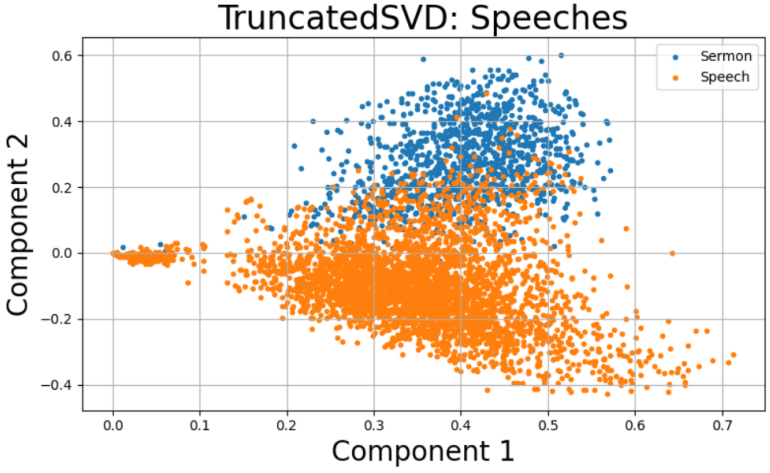
Appendix: Truncated SVD Dimensionality Reduction

Comment

For speeches, the years seem somewhat well seperated within a 30-year interval. Using the sermon / speech labels, it also becomes clear it has seperated them both on the label and year. Weighing with IDF also clusters them differently.

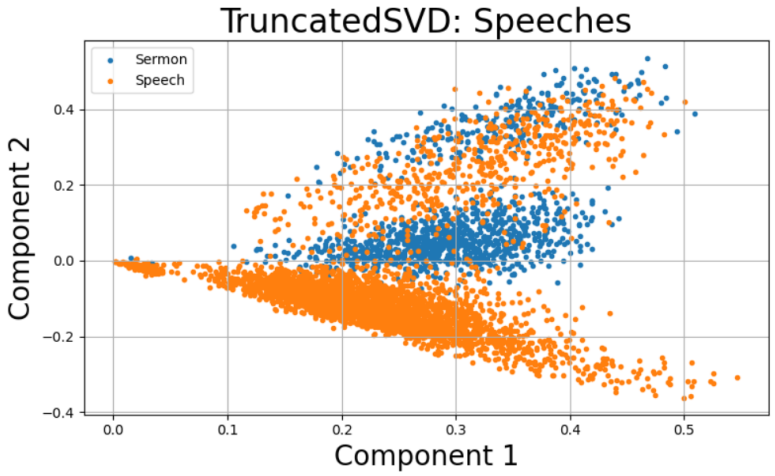


Appendix: Truncated SVD Dimensionality reduction

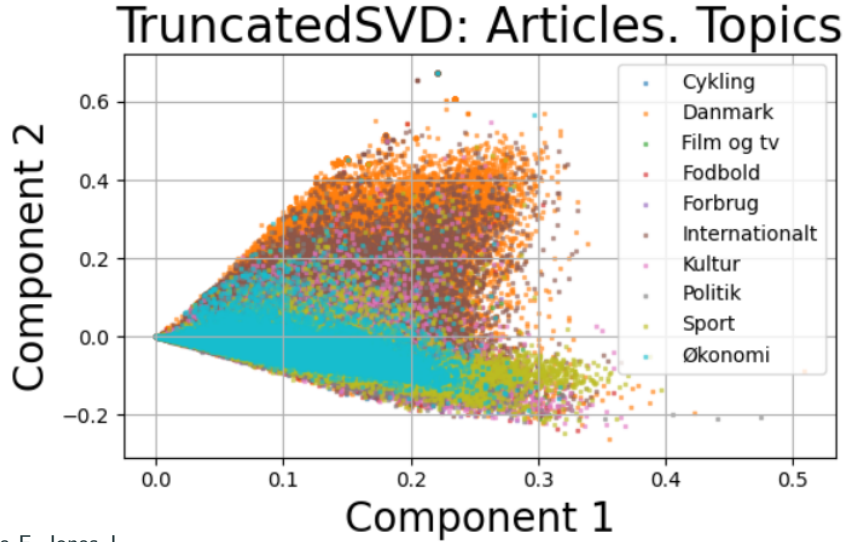


Speeches dataset. Labeling of sermons and speeches. Without IDF weighting

Appendix: Truncated SVD Dimensionality reduction

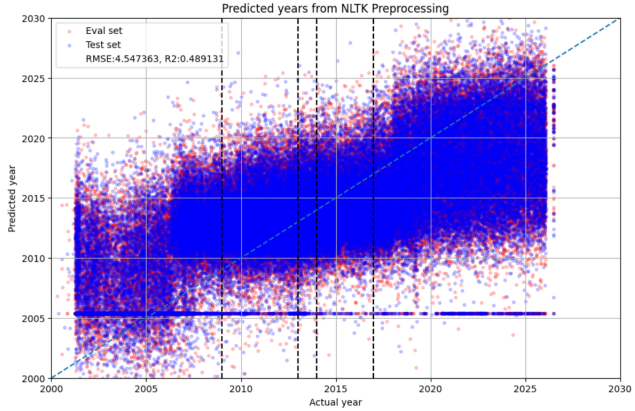


Speeches dataset. Labeling of sermons and speeches. With IDF weighting.


TruncatedSVD: Articles. Topics
Component 2
Component 1
0.6
0.4
0.2
0.0
−0.2
0.0
0.1
0.2
0.3
0.4
0.5
Cykling
Danmark
Film og tv
Fodbold
Forbrug
Internationalt
Kultur
Politik
Sport
Økonomi

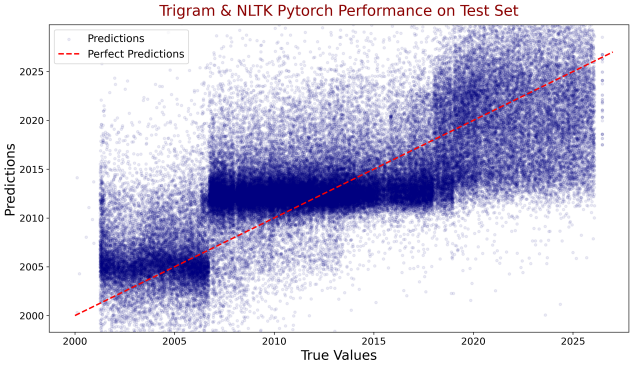
Appendix: Evaluation of NLTK Reggresion

NLTK Preprocessed data for article date regression. It could definetly be better.

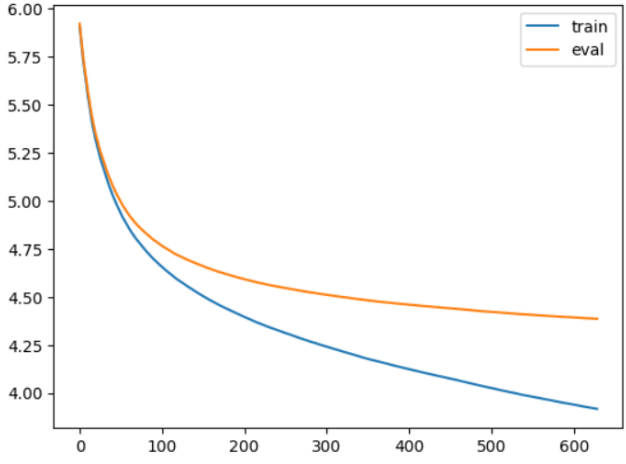


Appendix: Trying to mix NLTK & Trigrams

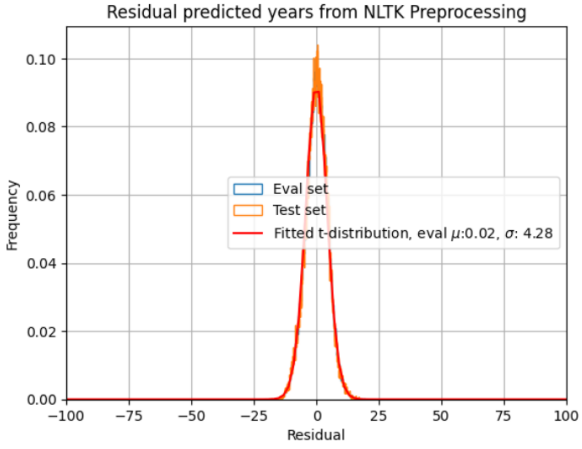
We tryed with 1k most important trigrams and 5k most important NLTK tokens, and obtained a worse result than just trigrams. NLTK seems generally a bad determiner for date. The regression model achieved an R^2 score of 0.242 and a mean squared error (MSE) of 31.56.



The loss curve shows the training could be set for longer, but the evaluation has plateaued, so it has trained enough.

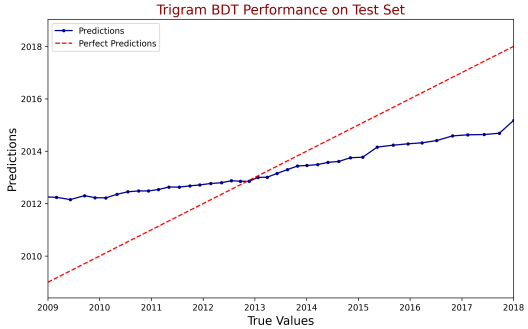


Showing the residual between true and predicted date reveals the test set is centered around 0 and with an error close to 4 years.



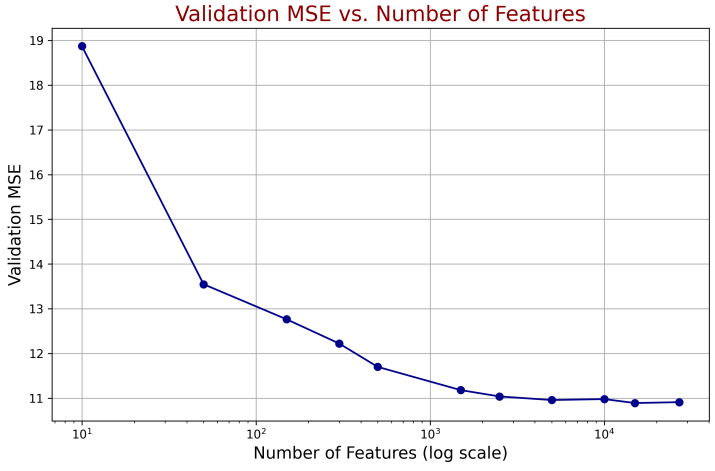
Appendix: Regression on only the middle years.

You can see a small incline in the average prediction for a model trained only on the middle years, though it is not a very strong regression. It makes the regression even tougher, because these articles were the ones with very little text.



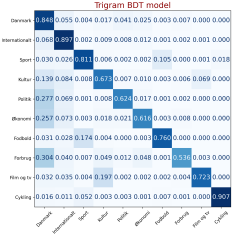
Regression on only the middle data with each dot the average of a thousand predictions.

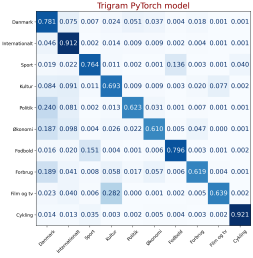
Appendix

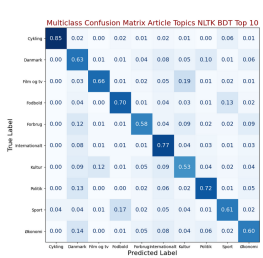

Validation MSE vs. Number of Features
Validation MSE
Number of Features (log scale)
19
18
17
16
15
14
13
12
11
10^1
10^2
10^3
10^4

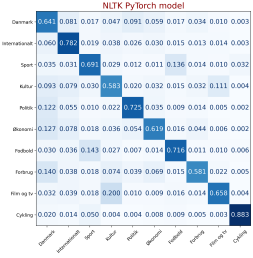
Appendix: Multiclass Classification of Article Topics

All multiclass classification models show the largest spread when predicting articles belonging to the topic **Denmark**. This may be related to the fact that some topic labels were only introduced around 2006, leading to greater overlap with other classes. Some classes show unique similarities. **Sport** and **Fodbold**. **Film og tv** and **Kultur**. Though correctly labelling can also be a human error from the author and some articles may fit multiple topics.



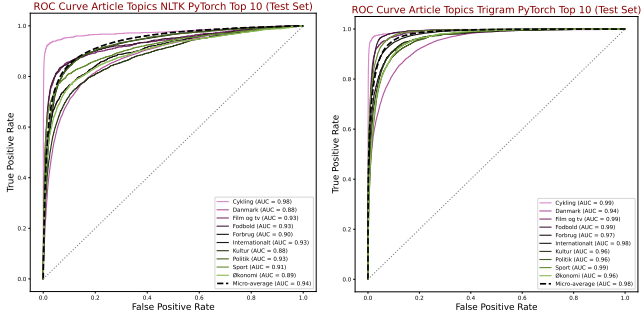






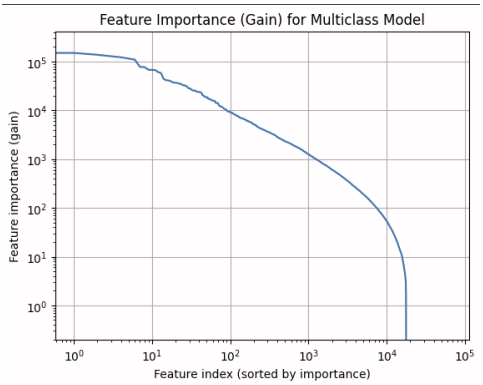
ROC-curves of Pytorch Multiclass classification

Using ROC-curver for individual topics we can more easily compare across two model, such at her where we se the trigram base pytorch model perform generally better than the NLTK based one. For both of them "Cykling" stands out as a topic which is most easy to determine, while "Denmark" is hardest to determine .

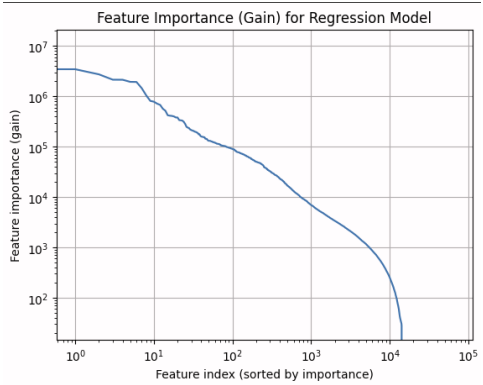


Appendix: Examples of feature importance plots

Plots show sorted features on the x-axis according to feature gain.



Caption



Caption

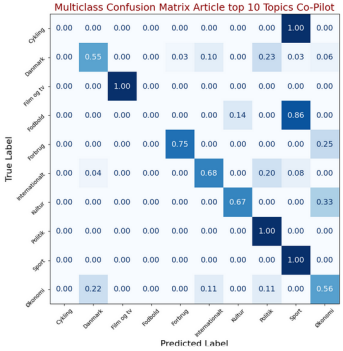
Appendix: Number of important features for each model

The NLTK preprocessing generally needed more features than the trigram version, but took well to reduction by SVD. **Model: Number of needed features**

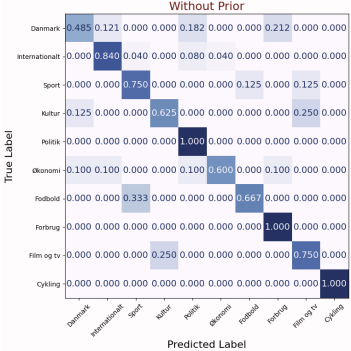
- Trigram sermon classification: 250
- NLTK sermon classification: 2500
- Trigram speech classification: 200
- NLTK speech classification: 2000 $\rightarrow$ 100 by svd
- Trigram multiclass classification: 5000
- NLTK multiclass classification: 20000 $\rightarrow$ 500 by svd
- Trigram regression: 3500
- NLTK regression: 15000 $\rightarrow$ 500 by svd

Appendix: Copilot multiclass CMs

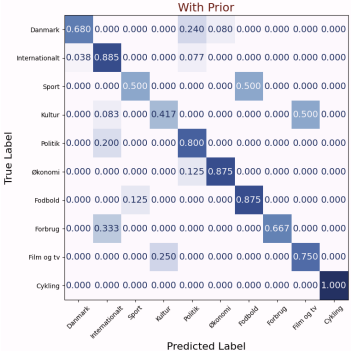
The 3 matrices for copilot multiclass classification.



With preprocessing

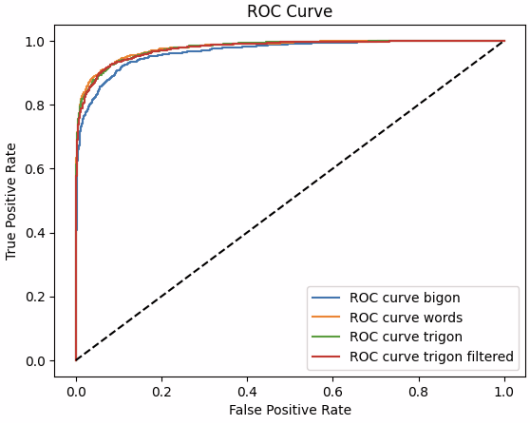


Raw text



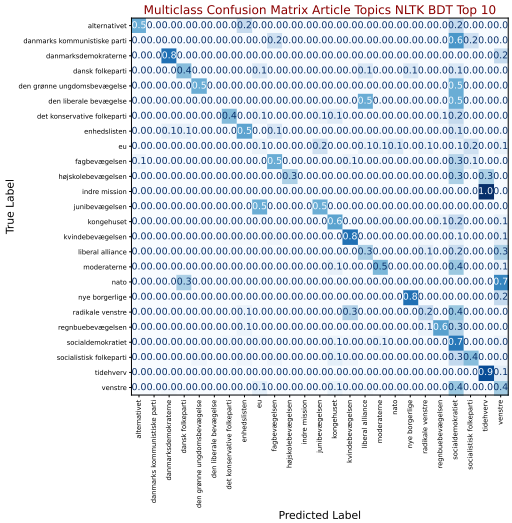
Raw text with prior

Appendix: Ngrams

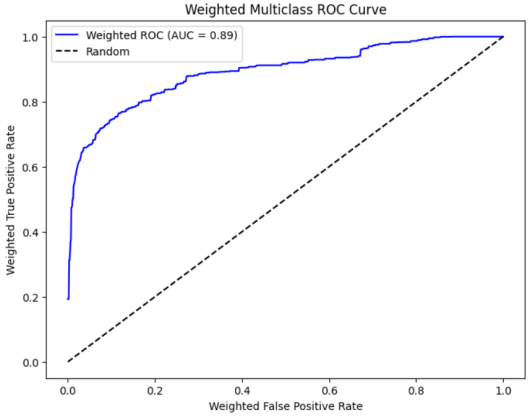


Looking at trigrams vs bigrams vs words (Written as n-gon in error). Single characters were much worse than even bigrams, but trigrams were already as good as words.

Appendix: Classification of political speeches.



Appendix: Classification of political speeches, ROC-curve



Appendix: Github URLs

Raw text files with title, date and topic found on Files. All code and processed data files found on Project.

https://github.com/Le-Jonas/ML2026_Files

https://github.com/Le-Jonas/ML2026_Project