DOPP_G18_Ex3_Final

January 24, 2021

Table of Contents

1 Introduction to Assignment

This is the third Exercise of **188.995 Data-Oriented Programming Paradigms**

We are group 18, and consist of: * Guillermo Alamán Requena, Matr. Nr: 11937906 * Michael Ferdinand Moser, Matr. Nr: 01123077 * Paul Joe Maliakel, Matr. Nr: 12012422 * Gunnar Sjúrðarson Knudsen, Matr. Nr: 12028205

In this task we were asked to choose one vaguely worded question, and then narrow the scope, figuring out how to get the data, before finally solving the question at hand. We chose **Question 21**, which contains: * How does the use of various communication languages in countries change over time? * Which languages grow and which disappear, and what are their characteristics? * Are there other factors that correlate with the appearance or disappearance of languages?

We soon realized that the question as stated is far too broad, and we therefore had to limit it.

After having discussed amoung our groups, we came to the following plan:

1.1 First draft

1.1.1 Topic and Questions to answer

We've selected question 21, which is regarding how communication languages in countries change over time.

After having discussed the data available, and planned a workflow, we've decided to try to answer the questions: * How has the English language changed in the past 100 years based on word frequencies, sentence length, …? * Can we find parallel developments between different genres of text? * Can the publication year of a movie/article/whatever be predicted based on the text and its characteristics?

1.1.2 Justification For Limit Of Scope

The sample questions stated in the task description are too broad, to be answered in a single 160 hour project. * Lot's of issues, such as: * Lack of census data; * other changes such as phonetic, semantic and syntactic meanings; * High correlation with e.g: * country population * age of speakers * … * What counts as a language? * dialect? * Mutually Intelligible? * Political dimensions * Multilingual people * How do we check accuracy of the available data? * …

Historical data for language use is likely not available for most languages, as it's topics for great research to estimate merely historical populations - especially before 1850 or so. The evolution of languages are much less documented. Lack of census data overall, but other changes are even harder to gauge, such as phonetic, semantic, and syntactic meanings. Highly correlated with population of countries, but also with "hidden" correlations, such as age of speakers, … Even dead languages can be revived.

What constitutes a language? Dialect? Mutually Intelligible? Also do not forget the political dimension, e.g. Croatian/Serbian really are just dialects of the same language but they want to keep separate. On the other end of this scheme the variant of Chinese spoken in Beijing may be drastically different from the Chinese spoken in other regions of the country, but still falls under the same "Chinese" umbrella to communicate unity.

How much is spoken? Should we consider people who studied a language as their second, third… language? If so, how well should be the command over the language for the person to count? A1/B1/C2 level? %How do we check accuracy of the available data?

1.1.3 Workflow plan & Project management

- Outline the plan
  - Get, understand and clean data: articles/movie scripts/video transcripts over the years (see next section)
  - Train-test split: keeping proportion of publication years within the splits.
  - Preprocessing: text feature extraction, feature selection, scaling, etc. (Come back here if necessary)
  - Visualization: evolution of words over the years, word-clouds and other relevant characteristics.
  - Define evaluation metrics, train different models/parameters using CV and select best one for predictions.
  - Predict, conclude, report and publish notebook in Kaggle Kernel.
- How the work will be divided up between group members
  - Acquisition, cleaning and prepossessing of the data will be done commonly.
  - Each member of the group will train a model and report results using same evaluation metrics.
  - Jointly choose the best model and conclude.
  - Presentation, report and publishing will be also split.

- Timeline: To be defined after review meeting

1.1.4 Data

Our goal is to get a dataset similar to:

Corpus	Year Published	Type	...
Text1	1976	News	...
Text2	1976	Movie Script	...
...	...	...	...
TextN	2009	Scientific Article	...

Feature extraction from texts will be performed to obtain appropriate features for modeling. To build a dataset like this one, we will rely on the following kind sources:

- https://www.kaggle.com/asad1m9a9h6mood/news-articles - News articles from 2015 until date.
- https://www.kaggle.com/snapcrack/all-the-news - 143000 articles from 15 American Publications.
- NLTK
- ...

1.2 Second Draft

After having a preliminary meeting with Univ.Prof. Dr. Hanbury and Dipl.-Ing. Dr. Piroi, who gave great input, we decided to further limit out goal to only use Project Gutenberg as a datasource, and setting our hypothesis to see wether it was possible to generate a model that predicted the publication year/decade for a set of books.

1.3 Pivoting Point

After having done a decent portion of work, we reached to the conclusion that our dataset was not suitable to solve the question we had original set out, and we were forced to pivot.

We discussed whether we wanted to change the goal from classifying, but as we were all quite interrested in a classification algorithm, and wanted to do proper NLP, we instead searched for another dataset.

1.4 Language change in Icelandic Parliamentary Speeches

We found the dataset with all icelandic parliamentary speeches going back a century. This is further described in section 3. With this great dataset, our goal was to develop a model that could try to predict which decade a speech is from

2 Estimating publication year from Project Gutenberg

This was the attempt at our first hypothesis. We import a large corpus of books from Project Gutenberg, and cleanse the data, so it's ready for machine learning

2.1 Setup

We start by setting up all packages needed for the project

2.1.1 Import packages

[1]:
```
from __future__ import absolute_import
from builtins import str
import os
from six import u

from os import listdir
from os.path import isfile, join

import nltk
import re
from operator import itemgetter
import pandas as pd
from functools import reduce

import random

pd.set_option('display.max_rows', None)

import math

from sklearn.feature_extraction.text import TfidfTransformer

from pprint import pprint
from time import time
import logging

from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.linear_model import SGDClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline

from sklearn.feature_selection import SelectKBest, chi2
```

2.1.2 Define Constants

Constant that are used in this part is also set

[2]:
```
file_path = "processedData"
```

```python
TEXT_START_MARKERS = frozenset((u(_) for _ in (
    "*END*THE SMALL PRINT",
    "*** START OF THE PROJECT GUTENBERG",
    "*** START OF THIS PROJECT GUTENBERG",
    "This etext was prepared by",
    "E-text prepared by",
    "Produced by",
    "Distributed Proofreading Team",
    "Proofreading Team at http://www.pgdp.net",
    "http://gallica.bnf.fr)",
    "      http://archive.org/details/",
    "http://www.pgdp.net",
    "by The Internet Archive)",
    "by The Internet Archive/Canadian Libraries",
    "by The Internet Archive/American Libraries",
    "public domain material from the Internet Archive",
    "Internet Archive)",
    "Internet Archive/Canadian Libraries",
    "Internet Archive/American Libraries",
    "material from the Google Print project",
    "*END THE SMALL PRINT",
    "***START OF THE PROJECT GUTENBERG",
    "This etext was produced by",
    "*** START OF THE COPYRIGHTED",
    "The Project Gutenberg",
    "http://gutenberg.spiegel.de/ erreichbar.",
    "Project Runeberg publishes",
    "Beginning of this Project Gutenberg",
    "Project Gutenberg Online Distributed",
    "Gutenberg Online Distributed",
    "the Project Gutenberg Online Distributed",
    "Project Gutenberg TEI",
    "This eBook was prepared by",
    "http://gutenberg2000.de erreichbar.",
    "This Etext was prepared by",
    "This Project Gutenberg Etext was prepared by",
    "Gutenberg Distributed Proofreaders",
    "Project Gutenberg Distributed Proofreaders",
    "the Project Gutenberg Online Distributed Proofreading Team",
    "**The Project Gutenberg",
    "*SMALL PRINT!",
    "More information about this book is at the top of this file.",
    "tells you about restrictions in how the file may be used.",
    "l'authorization à les utilizer pour preparer ce texte.",
    "of the etext through OCR.",
    "*****These eBooks Were Prepared By Thousands of Volunteers!*****",
    "We need your donations more than ever!",
```

```python
    " *** START OF THIS PROJECT GUTENBERG",
    "****     SMALL PRINT!",
    '["Small Print" V.',
    '      (http://www.ibiblio.org/gutenberg/',
    'and the Project Gutenberg Online Distributed Proofreading Team',
    'Mary Meehan, and the Project Gutenberg Online Distributed Proofreading',
    '                 this Project Gutenberg edition.',
)))


TEXT_END_MARKERS = frozenset((u(_) for _ in (
    "*** END OF THE PROJECT GUTENBERG",
    "*** END OF THIS PROJECT GUTENBERG",
    "***END OF THE PROJECT GUTENBERG",
    "End of the Project Gutenberg",
    "End of The Project Gutenberg",
    "Ende dieses Project Gutenberg",
    "by Project Gutenberg",
    "End of Project Gutenberg",
    "End of this Project Gutenberg",
    "Ende dieses Projekt Gutenberg",
    "        ***END OF THE PROJECT GUTENBERG",
    "*** END OF THE COPYRIGHTED",
    "End of this is COPYRIGHTED",
    "Ende dieses Etextes ",
    "Ende dieses Project Gutenber",
    "Ende diese Project Gutenberg",
    "**This is a COPYRIGHTED Project Gutenberg Etext, Details Above**",
    "Fin de Project Gutenberg",
    "The Project Gutenberg Etext of ",
    "Ce document fut presente en lecture",
    "Ce document fut présenté en lecture",
    "More information about this book is at the top of this file.",
    "We need your donations more than ever!",
    "END OF PROJECT GUTENBERG",
    " End of the Project Gutenberg",
    " *** END OF THIS PROJECT GUTENBERG",
)))


LEGALESE_START_MARKERS = frozenset((u(_) for _ in (
    "<<THIS ELECTRONIC VERSION OF",
)))


LEGALESE_END_MARKERS = frozenset((u(_) for _ in (
    "SERVICE THAT CHARGES FOR DOWNLOAD",
```

```
)))

TITLE_MARKERS = frozenset((u(_) for _ in (
    "Title:",
)))

AUTHOR_MARKERS = frozenset((u(_) for _ in (
    "Author:",
)))
DATE_MARKERS = frozenset((u(_) for _ in (
    "Release Date:","Release Date:"
)))
LANGUAGE_MARKERS = frozenset((u(_) for _ in (
    "Language:",
)))
ENCODING_MARKERS = frozenset((u(_) for _ in (
    "Character set encoding:",
)))
```

2.2 Importing the data

This is a very rough first draft at importing and cleansing the data. Solution is heavily inspired by https://gist.github.com/mbforbes/cee3fd5bb3a797b059524fe8c8ccdc2b

2.2.1 Getting the content

Start by downloading the repository of (english) books. This is done in bash. Only tested on Ubuntu, but mac should work the same

```
wget -m -H -nd "http://www.gutenberg.org/robot/harvest?filetypes[]=txt&langs[]=en"

                http://www.gutenberg.org/robot/harvest?offset=40532&filetypes[]=txt&langs[]=en
```

Takes a few hours to run, and is stored in a folder called rawContent. This is then copied to another folder, and we can start to clean up the mess

First we delete some dublications of the same books:

```
ls | grep "\-8.zip" | xargs rm
ls | grep "\-0.zip" | xargs rm
```

We can then unzip the files, and remove the zip files

```
unzip "*zip"
rm *.zip
```

Next we take care of some nested foldering

```
mv */*.txt ./
```

And finally, we remove all rubbish that isn't a real book:

```
ls | grep -v "\.txt" | xargs rm -rf
```

2.2.2 Data Cleansing

As the data is not given in a computer-friendly format, a lot of string operations are needed

Read a single file

[3]:
```
def read_file(file_name):
    file = open(file_name, encoding="ISO-8859-1")
    file_content = file.read()

    lines = file_content.splitlines()
    sep = str(os.linesep)

    # Initialize results for single book
    content_lines = []
    i = 0
    footer_found = False
    ignore_section = False

    title = ""
    author = ""
    date = ""
    language = ""
    encoding = ""
    year = 0

    # Reset flags for each book
    title_found = False
    author_found = False
    date_found = False
    language_found = False
    encoding_found = False

    for line in lines:
            reset = False

            #print(line)
            if i <= 600:
                # Shamelessly stolen
                if any(line.startswith(token) for token in TEXT_START_MARKERS):
                    reset = True

                # Extract Metadata
                if title_found == False:
                    if any(line.startswith(token) for token in TITLE_MARKERS):
                        title_found = True
                        title = line
                if author_found == False:
```

```
                    if any(line.startswith(token) for token in AUTHOR_MARKERS):
                        author_found = True
                        author = line
                if date_found == False:
                    if any(line.startswith(token) for token in DATE_MARKERS):
                        date_found = True
                        date = line
                        year = int(re.findall(r'\d{4}', date)[0])
                if language_found == False:
                    if any(line.startswith(token) for token in␣
↪LANGUAGE_MARKERS):
                        language_found = True
                        language = line
                if encoding_found == False:
                    if any(line.startswith(token) for token in␣
↪ENCODING_MARKERS):
                        encoding_found = True
                        encoding = line

                # More theft from above
                if reset:
                    content_lines = []
                    continue

            # I feel like a criminal by now. Guess what? Also stolen
            if i >= 100:
                if any(line.startswith(token) for token in TEXT_END_MARKERS):
                    footer_found = True

                if footer_found:
                    break

            if any(line.startswith(token) for token in LEGALESE_START_MARKERS):
                ignore_section = True
                continue
            elif any(line.startswith(token) for token in LEGALESE_END_MARKERS):
                ignore_section = False
                continue

            if not ignore_section:
                if line != "": # Screw the blank lines
                    content_lines.append(line.rstrip(sep))
                i += 1

            sep.join(content_lines)

    # Do more cleaning
```

```
    for token in TITLE_MARKERS:
        title = title.replace(token, '').lstrip().rstrip()
    for token in AUTHOR_MARKERS:
        author = author.replace(token, '').lstrip().rstrip()
    for token in LANGUAGE_MARKERS:
        language = language.replace(token, '').lstrip().rstrip()
    for token in DATE_MARKERS:
        date = date.replace(token, '').lstrip().rstrip()
    for token in ENCODING_MARKERS:
        encoding = encoding.replace(token, '').lstrip().rstrip()
    return title, author, date, year, language, encoding, content_lines
```

Return list of all words Currently quite an empty function. However, I assume that some cleaning of cases etc. will be done here

[4]:
```
def get_words(content_lines):
    all_text_lower = " ".join(content_lines).lower()
    words = re.findall(r'(\b[A-Za-z][a-z]{2,9}\b)', all_text_lower)

    # Do more cleansing. E.g. cases and stuff

    return words
```

2.3 Statistics

We start by doing some exploratory data analysis, to see how well our scraping works

2.3.1 First attempt

Trying a simple word frequency

[5]:
```
def get_word_frequencies(words):
    frequency = {}
    for word in words:
        count = frequency.get(word,0)
        frequency[word] = count + 1

    word_count = len(words)
    unique_word_count = 0
    word_list = []
    word_list_count = []
    for key, value in reversed(sorted(frequency.items(), key = itemgetter(1))):
        word_list.append(key)
        word_list_count.append(value)
        unique_word_count = unique_word_count + 1

    word_list_freq = [freq / word_count for freq in word_list_count]
```

```
    word_freq = pd.DataFrame(list(zip(word_list, word_list_count,␣
 ↪word_list_freq))
                                , columns = ['Word', 'count', 'freq'])

    word_freq['rank'] = word_freq['count'].rank(ascending = False,␣
 ↪method="dense")

    return(word_freq, unique_word_count)
```

2.3.2 Read all files, and do preprocessing

Well… Only ten files currently

[6]:
```
# Get all filenames
files = [f for f in listdir(file_path) if isfile(join(file_path, f))]
files = list(filter(lambda file: file[0].isdigit(), files))
random.shuffle(files)



# Do only subset
files = files[0:10]

list_of_file = []
list_of_title = []
list_of_author = []
list_of_date = []
list_of_year = []
list_of_language = []
list_of_encoding = []
list_of_word_count = []
list_of_unique_word_count = []
list_of_word_frequencies = []
iter_ = 0

for file in files:
    # Read in basic information from file
    title, author, date, year, language, encoding, content_lines =␣
 ↪read_file(file_path + "/" + file)
    line_count = len(content_lines)

    # Not sure if we want this for later:
    #content_all = " ".join(content_lines)

    # Split into words (and do various cleaning)
    words = get_words(content_lines)
    word_count = len(words)
```

```
    # First analysis, but should do something proper
    word_frequencies_table, unique_word_count = get_word_frequencies(words)

    # Append to results
    list_of_file.append(file)
    list_of_title.append(title)
    list_of_author.append(author)
    list_of_date.append(date)
    list_of_year.append(year)
    list_of_language.append(language)
    list_of_encoding.append(encoding)
    list_of_word_count.append(word_count)
    list_of_unique_word_count.append(unique_word_count)
    list_of_word_frequencies.append(word_frequencies_table)


    # Show basic information
    #print(iter_)
    iter_ = iter_ + 1
    #print("################################")
    #print("################################")
    #print("Filename: " + str(file))
    #print("Title: " + str(title))
    #print("Author(s): " + str(author))
    #print("Date: " + str(date))
    #print("Year: " + str(year))
    #print("Language: " + str(language))
    #print("Encoding: " + str(encoding))
    #print("################################")
    #print("Words in book: " + str(word_count))
    #print("Unique words in book: " + str(unique_word_count))
    #print("################################")
    #print(word_frequencies_table)

# Feel free to change to dict? list? separate files?
## nested dataframes works, but looks super ungly when printing
### Fuck it - This is tooo useless killing it again
#all_res = pd.DataFrame(list(zip(list_of_file
#                                 , list_of_title
#                                 , list_of_author
#                                 , list_of_date
#                                 , list_of_language
#                                 , list_of_encoding
#                                 , list_of_word_count
#                                 , list_of_unique_word_count
#                                 , list_of_word_frequencies
```

```
#                               ))
#                           , columns = ['file'
#                                        , 'title'
#                                        , 'author'
#                                        , 'date'
#                                        , 'language'
#                                        , 'encoding'
#                                        , 'word_count'
#                                        , 'unique_word_count'
#                                        , 'word_frequencies'
#                                       ]
#                     )
```

2.3.3 Compare Word ranking between titles

This is our first attemt at seeing how the ranking of words change between titles. Idea is to see that the zipf-distribution changes as time passes buy

[7]:
```
list_count= []
list_freq = []
list_rank = []



col_names = list_of_title.copy()
col_names.insert(0,'Word')

for df in list_of_word_frequencies:
    list_count.append(df[['Word', 'count']])
    list_freq.append(df[['Word', 'freq']])
    list_rank.append(df[['Word', 'rank']])

df_count = reduce(lambda left, right: pd.merge(left, right, on="Word",␣
 ↪how='outer'), list_count)
df_count.columns = col_names
df_count['Sum'] = df_count.drop('Word', axis=1).apply(lambda x: x.sum(), axis=1)
df_count = df_count.sort_values(ascending = False, by=['Sum'])

df_freq = reduce(lambda left, right: pd.merge(left, right, on="Word",␣
 ↪how='outer'), list_freq)
df_freq.columns = col_names
df_freq['Avg'] = df_freq.drop('Word', axis=1).apply(lambda x: x.mean(), axis=1)
df_freq = df_freq.sort_values(ascending = False, by=['Avg'])

df_rank = reduce(lambda left, right: pd.merge(left, right, on="Word",␣
 ↪how='outer'), list_rank)
df_rank.columns = col_names
```

```python
df_rank['Avg'] = df_rank.drop('Word', axis=1).apply(lambda x: x.mean(), axis=1)
df_rank = df_rank.sort_values(by=['Avg'])
```

[8]:
```python
df_rank.head(30)
```

[8]:
```
            Word  Hal Kenyon Disappears  \
0            the                    1.0
1            and                    2.0
15028        ltd                    NaN
3           that                    4.0
15030      warne                    NaN
15029     carrot                    NaN
17110  tubercles                    NaN
15040     potter                    NaN
15032    skinner                    NaN
15035      hides                    NaN
15031  carmelite                    NaN
15037  scratches                    NaN
15038      berne                    NaN
15039  signatory                    NaN
15034     shoots                    NaN
15033      peeps                    NaN
15041    beatrix                    NaN
17111      genus                    NaN
17112  maxillary                    NaN
17113      frogs                    NaN
2            was                    3.0
11           for                   12.0
10          with                   11.0
16277   borghesi                    NaN
17114   inguinal                    NaN
4            his                    5.0
17116     genera                    NaN
17115   terminal                    NaN
16278   bertolla                    NaN
17             not                   18.0

       The Great Events by Famous Historians, v. 13  Mystic Christianity  \
0                                               1.0                  1.0
1                                               2.0                  2.0
15028                                           NaN                  NaN
3                                               5.0                  3.0
15030                                           NaN                  NaN
15029                                           NaN                  NaN
17110                                           NaN                  NaN
15040                                           NaN                  NaN
15032                                           NaN                  NaN
```

```
15035                                        NaN                 NaN
15031                                        NaN                 NaN
15037                                        NaN                 NaN
15038                                        NaN                 NaN
15039                                        NaN                 NaN
15034                                        NaN                 NaN
15033                                        NaN                 NaN
15041                                        NaN                 NaN
17111                                        NaN                 NaN
17112                                        NaN                 NaN
17113                                        NaN                 NaN
2                                            3.0                 5.0
11                                          10.0                 7.0
10                                           6.0                11.0
16277                                        NaN                 NaN
17114                                        NaN                 NaN
4                                            4.0                 4.0
17116                                        NaN                 NaN
17115                                        NaN                 NaN
16278                                        NaN                 NaN
17                                          15.0                12.0

       Memoir  The Story of a Fierce Bad Rabbit  The Stranger  \
0         1.0                               1.0           1.0
1         2.0                               3.0           2.0
15028     NaN                               5.0           NaN
3         4.0                               NaN           5.0
15030     NaN                               6.0           NaN
15029     NaN                               6.0           NaN
17110     NaN                               NaN           NaN
15040     NaN                               7.0           NaN
15032     NaN                               7.0           NaN
15035     NaN                               7.0           NaN
15031     NaN                               7.0           NaN
15037     NaN                               7.0           NaN
15038     NaN                               7.0           NaN
15039     NaN                               7.0           NaN
15034     NaN                               7.0           NaN
15033     NaN                               7.0           NaN
15041     NaN                               7.0           NaN
17111     NaN                               NaN           NaN
17112     NaN                               NaN           NaN
17113     NaN                               NaN           NaN
2         5.0                               NaN          27.0
11        6.0                               7.0           9.0
10       10.0                               6.0          13.0
16277     NaN                               NaN           NaN
```

```
17114     NaN                                NaN           NaN
4        15.0                                4.0          11.0
17116     NaN                                NaN           NaN
17115     NaN                                NaN           NaN
16278     NaN                                NaN           NaN
17       20.0                                NaN           4.0

       Harper's Young People, July 20, 1880  \
0                                       1.0
1                                       2.0
15028                                   NaN
3                                       4.0
15030                                   NaN
15029                                   NaN
17110                                   NaN
15040                                   NaN
15032                                   NaN
15035                                   NaN
15031                                   NaN
15037                                   NaN
15038                                   NaN
15039                                   NaN
15034                                   NaN
15033                                   NaN
15041                                   NaN
17111                                   NaN
17112                                   NaN
17113                                   NaN
2                                       3.0
11                                      7.0
10                                      9.0
16277                                   NaN
17114                                   NaN
4                                      14.0
17116                                   NaN
17115                                   NaN
16278                                   NaN
17                                     11.0

       The Borghesi Astronomical Clock in the Museum of History and Technology
\
0                                                    1.0
1                                                    2.0
15028                                                NaN
3                                                    4.0
15030                                                NaN
15029                                                NaN
```

```
17110                                                      NaN
15040                                                      NaN
15032                                                      NaN
15035                                                      NaN
15031                                                      NaN
15037                                                      NaN
15038                                                      NaN
15039                                                      NaN
15034                                                      NaN
15033                                                      NaN
15041                                                      NaN
17111                                                      NaN
17112                                                      NaN
17113                                                      NaN
2                                                          8.0
11                                                         8.0
10                                                         6.0
16277                                                     10.0
17114                                                      NaN
4                                                          9.0
17116                                                      NaN
17115                                                      NaN
16278                                                     12.0
17                                                        23.0

       Genera of Leptodactylid Frogs in Mexico  \
0                                          1.0
1                                          2.0
15028                                      NaN
3                                         15.0
15030                                      NaN
15029                                      NaN
17110                                      6.0
15040                                      NaN
15032                                      NaN
15035                                      NaN
15031                                      NaN
15037                                      NaN
15038                                      NaN
15039                                      NaN
15034                                      NaN
15033                                      NaN
15041                                      NaN
17111                                      7.0
17112                                      8.0
17113                                      8.0
2                                         23.0
```

```
11                                          25.0
10                                          14.0
16277                                        NaN
17114                                       10.0
4                                           27.0
17116                                       11.0
17115                                       11.0
16278                                        NaN
17                                           4.0

       The Supplies for the Confederate Army        Avg
0                                         1.0   1.000000
1                                         2.0   2.100000
15028                                     NaN   5.000000
3                                         5.0   5.444444
15030                                     NaN   6.000000
15029                                     NaN   6.000000
17110                                     NaN   6.000000
15040                                     NaN   7.000000
15032                                     NaN   7.000000
15035                                     NaN   7.000000
15031                                     NaN   7.000000
15037                                     NaN   7.000000
15038                                     NaN   7.000000
15039                                     NaN   7.000000
15034                                     NaN   7.000000
15033                                     NaN   7.000000
15041                                     NaN   7.000000
17111                                     NaN   7.000000
17112                                     NaN   8.000000
17113                                     NaN   8.000000
2                                         3.0   8.888889
11                                        4.0   9.500000
10                                       11.0   9.700000
16277                                     NaN  10.000000
17114                                     NaN  10.000000
4                                        15.0  10.800000
17116                                     NaN  11.000000
17115                                     NaN  11.000000
16278                                     NaN  12.000000
17                                        9.0  12.888889
```

[9]:

```
#df_freq['Avg'] = df_freq.drop('Word', axis=1).apply(lambda x: x.mean(), axis=1)
df_freq = df_freq.sort_values(ascending = False, by=['Avg'])

df_freq.head(20)
```

[9]:
```
            Word  Hal Kenyon Disappears  \
0            the               0.095039
1            and               0.043636
15027     rabbit                    NaN
15028        ltd                    NaN
2            was               0.022507
3           that               0.014155
15030      warne                    NaN
15029     carrot                    NaN
17110  tubercles                    NaN
4            his               0.013539
17111      genus                    NaN
17113      frogs                    NaN
17112  maxillary                    NaN
11           for               0.008528
12          this               0.007385
10          with               0.008968
17           not               0.005392
17114   inguinal                    NaN
17116     genera                    NaN
17115   terminal                    NaN

       The Great Events by Famous Historians, v. 13  Mystic Christianity  \
0                                          0.109884             0.121139
1                                          0.043128             0.051943
15027                                           NaN                  NaN
15028                                           NaN                  NaN
2                                          0.018942             0.012008
3                                          0.013064             0.019161
15030                                           NaN                  NaN
15029                                           NaN                  NaN
17110                                           NaN                  NaN
4                                          0.016020             0.016826
17111                                           NaN                  NaN
17113                                           NaN                  NaN
17112                                           NaN                  NaN
11                                         0.008495             0.008636
12                                         0.007406             0.008580
10                                         0.010666             0.007487
17                                         0.005996             0.007375
17114                                           NaN                  NaN
17116                                           NaN                  NaN
17115                                           NaN                  NaN

         Memoir  The Story of a Fierce Bad Rabbit  The Stranger  \
0      0.079243                          0.078431      0.048657
1      0.045662                          0.045752      0.032568
```

15027	NaN	0.065359	0.000078
15028	NaN	0.019608	NaN
2	0.014935	NaN	0.004139
3	0.017314	NaN	0.012496
15030	NaN	0.013072	NaN
15029	NaN	0.013072	NaN
17110	NaN	NaN	NaN
4	0.008181	0.032680	0.009763
17111	NaN	NaN	NaN
17113	NaN	NaN	NaN
17112	NaN	NaN	NaN
11	0.013318	0.006536	0.011325
12	0.009989	0.032680	0.010778
10	0.010369	0.013072	0.008747
17	0.005993	NaN	0.013824
17114	NaN	NaN	NaN
17116	NaN	NaN	NaN
17115	NaN	NaN	NaN

	Harper's Young People, July 20, 1880 \
0	0.076009
1	0.049199
15027	NaN
15028	NaN
2	0.014202
3	0.011593
15030	NaN
15029	NaN
17110	NaN
4	0.006087
17111	NaN
17113	NaN
17112	NaN
11	0.009347
12	0.002536
10	0.008188
17	0.007826
17114	NaN
17116	NaN
17115	NaN

	The Borghesi Astronomical Clock in the Museum of History and Technology \
0	0.141561
1	0.037406
15027	NaN
15028	NaN

```
2                                                0.008240
3                                                0.010730
15030                                                 NaN
15029                                                 NaN
17110                                                 NaN
4                                                0.008003
17111                                                 NaN
17113                                                 NaN
17112                                                 NaN
11                                               0.008240
12                                               0.008299
10                                               0.009722
17                                               0.003913
17114                                                 NaN
17116                                                 NaN
17115                                                 NaN

       Genera of Leptodactylid Frogs in Mexico  \
0                                     0.067406
1                                     0.061197
15027                                      NaN
15028                                      NaN
2                                     0.002217
3                                     0.005765
15030                                      NaN
15029                                      NaN
17110                                 0.012417
4                                     0.000443
17111                                 0.010643
17113                                 0.010200
17112                                 0.010200
11                                    0.001330
12                                    0.001330
10                                    0.006652
17                                    0.018182
17114                                 0.008426
17116                                 0.007982
17115                                 0.007982

       The Supplies for the Confederate Army       Avg
0                                   0.095608  0.091298
1                                   0.038884  0.044937
15027                                    NaN  0.032719
15028                                    NaN  0.019608
2                                   0.032136  0.014370
3                                   0.014296  0.013175
15030                                    NaN  0.013072
```

```
15029                                          NaN  0.013072
17110                                          NaN  0.012417
4                                         0.005833  0.011737
17111                                          NaN  0.010643
17113                                          NaN  0.010200
17112                                          NaN  0.010200
11                                        0.018641  0.009440
12                                        0.004346  0.009333
10                                        0.007434  0.009130
17                                        0.008920  0.008602
17114                                          NaN  0.008426
17116                                          NaN  0.007982
17115                                          NaN  0.007982
```

[10]:
```
df_count.head(20)
```

[10]:
```
       Word  Hal Kenyon Disappears  \
0       the                 3243.0
1       and                 1489.0
2       was                  768.0
3      that                  483.0
4       his                  462.0
10     with                  306.0
11      for                  291.0
5       had                  364.0
112   which                   45.0
12     this                  252.0
9       but                  307.0
16     from                  186.0
13     were                  241.0
17      not                  184.0
6      they                  360.0
21    their                  164.0
24     have                  150.0
18      all                  184.0
14      him                  216.0
7       you                  352.0

     The Great Events by Famous Historians, v. 13  Mystic Christianity  \
0                                         13012.0               6537.0
1                                          5107.0               2803.0
2                                          2243.0                648.0
3                                          1547.0               1034.0
4                                          1897.0                908.0
10                                         1263.0                404.0
11                                         1006.0                466.0
5                                          1171.0                360.0
```

```
112                                        1081.0             510.0
12                                          877.0             463.0
9                                           721.0             412.0
16                                          829.0             409.0
13                                         1041.0             253.0
17                                          710.0             398.0
6                                           638.0             314.0
21                                          865.0             306.0
24                                          431.0             247.0
18                                          520.0             276.0
14                                          436.0             354.0
7                                           127.0             121.0

     Memoir  The Story of a Fierce Bad Rabbit  The Stranger  \
0     833.0                              12.0         623.0
1     480.0                               7.0         417.0
2     157.0                               NaN          53.0
3     182.0                               NaN         160.0
4      86.0                               5.0         125.0
10    109.0                               2.0         112.0
11    140.0                               1.0         145.0
5      93.0                               NaN          26.0
112   126.0                               NaN          35.0
12    105.0                               5.0         138.0
9      46.0                               1.0         151.0
16     60.0                               NaN          80.0
13     92.0                               NaN          30.0
17     63.0                               NaN         177.0
6     202.0                               NaN          41.0
21    123.0                               NaN          32.0
24    101.0                               NaN         157.0
18     62.0                               2.0          61.0
14     33.0                               1.0          86.0
7       3.0                               NaN         385.0

     Harper's Young People, July 20, 1880  \
0                                  1049.0
1                                   679.0
2                                   196.0
3                                   160.0
4                                    84.0
10                                  113.0
11                                  129.0
5                                   111.0
112                                  62.0
12                                   35.0
9                                   141.0
```

16	56.0
13	74.0
17	108.0
6	135.0
21	37.0
24	81.0
18	76.0
14	72.0
7	98.0

	The Borghesi Astronomical Clock in the Museum of History and Technology \
0	2388.0
1	631.0
2	139.0
3	181.0
4	135.0
10	164.0
11	139.0
5	68.0
112	172.0
12	140.0
9	54.0
16	195.0
13	37.0
17	66.0
6	38.0
21	50.0
24	77.0
18	53.0
14	16.0
7	64.0

	Genera of Leptodactylid Frogs in Mexico \
0	152.0
1	138.0
2	5.0
3	13.0
4	1.0
10	15.0
11	3.0
5	NaN
112	4.0
12	3.0
9	10.0
16	18.0
13	2.0
17	41.0

```
6                                             NaN
21                                            5.0
24                                           16.0
18                                            8.0
14                                            NaN
7                                             NaN

     The Supplies for the Confederate Army      Sum
0                                    836.0  28685.0
1                                    340.0  12091.0
2                                    281.0   4490.0
3                                    125.0   3885.0
4                                     51.0   3754.0
10                                    65.0   2553.0
11                                   163.0   2483.0
5                                    114.0   2307.0
112                                   66.0   2101.0
12                                    38.0   2056.0
9                                     85.0   1928.0
16                                    66.0   1899.0
13                                   122.0   1892.0
17                                    78.0   1825.0
6                                     36.0   1764.0
21                                    30.0   1612.0
24                                    47.0   1307.0
18                                    49.0   1291.0
14                                    33.0   1247.0
7                                     32.0   1182.0
```

2.4 Second testing

This definately needs some proper refactoring, but Was curious whether we get anything decent from reading a bunch of random books in

Requires an additional folder "decades" in the root directory

[11]:
```
# Get all filenames
files = [f for f in listdir(file_path) if isfile(join(file_path, f))]

# Do only subset
## Is done for 5000 files already, so set down to 20 to increase performance.␣
↪5000 books are currently stored in the file
files = files[0:20]


counter = 0
for file in files:
    counter = counter + 1
```

```
    # Read in basic information from file
    title, author, date, year, language, encoding, content_lines =␣
↪read_file(file_path + "/" + file)
    #line_count = len(content_lines)
    decade = math.floor(year / 10) * 10
    decade_file = "decades/" + str(decade) + ".txt"
    content_all = " ".join(content_lines)

    if os.path.exists(decade_file):
        append_write = 'a' # append if already exists
    else:
        append_write = 'w' # make a new file if not

    fileWriter = open(decade_file,append_write)
    fileWriter.write(content_all + '\n')
    fileWriter.close()
```

2.4.1 Read in from the decades files, and see the distributions

[12]:
```
# Get all filenames
files = [f for f in listdir("decades") if isfile(join("decades", f))]
print(files)
files.sort(reverse=True)


col_names = []
col_names.append("Word")

tables = []

for file_name in files:
    print(file_name)

    file = open("decades/" + file_name, encoding="ISO-8859-1")
    file_content = file.read()

    # Split into words (and do various cleaning)
    all_text_lower = file_content.lower()
    words = re.findall(r'(\b[A-Za-z][a-z]{2,9}\b)', all_text_lower)

    # First analysis, but should do something proper
    word_frequencies_table, unique_word_count = get_word_frequencies(words)
    tables.append(word_frequencies_table)
    col_names.append(file_name)
```

```
['00.txt', '0.txt', '2010.txt', '2000.txt', '2020.txt', '1990.txt']
2020.txt
```

```
2010.txt
2000.txt
1990.txt
00.txt
0.txt
```

2.4.2 Preliminary Conclusion

We see that even though the books are quite old, no decade prior to 1990s is found.

This is when we found out that the "year" that's registered in the dataset is the upload-date.

Haven gotten this far, we however decided to see if we could find a pattern in this

2.4.3 Compare ranking between upload-decades

[13]:
```
list_count= []
list_freq = []
list_rank = []

for df in tables:
    #list_count.append(df[['Word', 'count']])
    #list_freq.append(df[['Word', 'freq']])
    list_rank.append(df[['Word', 'rank']])

#df_count = reduce(lambda left, right: pd.merge(left, right, on="Word",␣
 ↪how='outer'), list_count)
#df_count.columns = col_names

#df_freq = reduce(lambda left, right: pd.merge(left, right, on="Word",␣
 ↪how='outer'), list_freq)
#df_freq.columns = col_names

df_rank = reduce(lambda left, right: pd.merge(left, right, on="Word",␣
 ↪how='outer'), list_rank)
df_rank.columns = col_names
```

[14]:
```
df_rank.head(100)
```

[14]:

	Word	2020.txt	2010.txt	2000.txt	1990.txt	00.txt	0.txt
0	the	1.0	1.0	1.0	1.0	1.0	1.0
1	and	2.0	2.0	2.0	2.0	2.0	2.0
2	that	3.0	3.0	3.0	3.0	4.0	3.0
3	was	4.0	4.0	4.0	4.0	23.0	5.0
4	you	5.0	9.0	8.0	5.0	163.0	19.0
5	with	6.0	6.0	6.0	7.0	3.0	6.0
6	for	7.0	7.0	7.0	11.0	13.0	8.0
7	his	8.0	5.0	5.0	6.0	5.0	4.0

8	not	9.0	10.0	11.0	12.0	11.0	9.0
9	had	10.0	8.0	9.0	10.0	40.0	14.0
10	but	11.0	11.0	10.0	13.0	72.0	10.0
11	which	12.0	12.0	14.0	22.0	82.0	7.0
12	they	13.0	17.0	15.0	16.0	58.0	21.0
13	from	14.0	15.0	18.0	21.0	25.0	15.0
14	were	15.0	21.0	21.0	20.0	81.0	22.0
15	have	16.0	16.0	16.0	18.0	22.0	11.0
16	this	17.0	14.0	17.0	15.0	8.0	13.0
17	are	18.0	20.0	23.0	27.0	37.0	16.0
18	she	19.0	18.0	13.0	9.0	65.0	47.0
19	all	20.0	19.0	20.0	17.0	15.0	12.0
20	their	21.0	24.0	25.0	26.0	21.0	30.0
21	him	22.0	22.0	19.0	14.0	28.0	20.0
22	her	23.0	13.0	12.0	8.0	20.0	33.0
23	its	24.0	40.0	52.0	98.0	68.0	67.0
24	one	25.0	23.0	22.0	25.0	17.0	28.0
25	there	26.0	25.0	24.0	23.0	37.0	32.0
26	them	27.0	28.0	28.0	32.0	55.0	38.0
27	what	28.0	33.0	32.0	24.0	56.0	24.0
28	has	29.0	38.0	44.0	46.0	176.0	40.0
29	been	30.0	29.0	31.0	33.0	30.0	23.0
30	will	31.0	32.0	33.0	37.0	85.0	27.0
31	would	32.0	30.0	29.0	31.0	194.0	25.0
32	said	33.0	31.0	26.0	19.0	61.0	141.0
33	when	34.0	27.0	27.0	28.0	105.0	34.0
34	more	35.0	34.0	36.0	42.0	190.0	26.0
35	who	36.0	26.0	30.0	30.0	27.0	18.0
36	into	37.0	37.0	37.0	39.0	97.0	63.0
37	out	38.0	35.0	34.0	29.0	62.0	77.0
38	then	39.0	36.0	35.0	35.0	47.0	50.0
39	other	40.0	46.0	56.0	58.0	88.0	66.0
40	men	41.0	77.0	79.0	81.0	108.0	53.0
41	only	42.0	48.0	54.0	60.0	192.0	71.0
42	can	43.0	58.0	50.0	45.0	98.0	54.0
43	upon	44.0	50.0	57.0	80.0	121.0	89.0
44	our	45.0	57.0	49.0	91.0	149.0	43.0
45	than	46.0	47.0	48.0	64.0	201.0	31.0
46	now	47.0	43.0	40.0	38.0	134.0	42.0
47	time	48.0	42.0	43.0	49.0	96.0	57.0
48	power	49.0	256.0	246.0	327.0	204.0	111.0
49	great	50.0	63.0	60.0	79.0	64.0	37.0
50	these	51.0	52.0	65.0	87.0	48.0	60.0
51	government	52.0	419.0	438.0	534.0	NaN	146.0
52	man	53.0	41.0	41.0	34.0	222.0	29.0
53	over	54.0	68.0	64.0	62.0	92.0	139.0
54	could	55.0	45.0	39.0	41.0	199.0	64.0

55	very	56.0	44.0	42.0	50.0	159.0	65.0
56	your	57.0	56.0	46.0	36.0	201.0	55.0
57	first	58.0	66.0	77.0	77.0	219.0	81.0
58	society	59.0	582.0	595.0	535.0	NaN	231.0
59	two	60.0	53.0	58.0	70.0	124.0	100.0
60	made	61.0	62.0	62.0	83.0	174.0	82.0
61	such	62.0	64.0	74.0	78.0	146.0	45.0
62	about	63.0	51.0	45.0	40.0	99.0	113.0
63	some	64.0	39.0	38.0	44.0	70.0	49.0
64	any	65.0	55.0	55.0	55.0	130.0	49.0
65	did	66.0	60.0	53.0	54.0	154.0	99.0
66	know	67.0	82.0	76.0	48.0	159.0	98.0
67	pendleton	68.0	3305.0	3741.0	773.0	NaN	NaN
68	same	69.0	99.0	122.0	118.0	203.0	112.0
69	well	70.0	67.0	59.0	52.0	54.0	58.0
70	under	71.0	103.0	121.0	153.0	129.0	103.0
71	may	72.0	49.0	69.0	86.0	162.0	36.0
72	general	73.0	196.0	230.0	374.0	254.0	195.0
73	before	74.0	65.0	66.0	59.0	101.0	84.0
74	most	75.0	79.0	86.0	120.0	243.0	46.0
75	even	76.0	91.0	98.0	125.0	77.0	78.0
76	much	77.0	75.0	75.0	76.0	228.0	73.0
77	like	78.0	61.0	51.0	47.0	10.0	74.0
78	stephanie	79.0	NaN	3720.0	NaN	NaN	NaN
79	lorraine	80.0	3197.0	3608.0	766.0	NaN	383.0
80	those	81.0	83.0	89.0	138.0	26.0	44.0
81	down	82.0	76.0	70.0	63.0	63.0	126.0
82	back	83.0	95.0	88.0	85.0	165.0	217.0
83	came	84.0	93.0	85.0	74.0	172.0	236.0
84	see	85.0	69.0	63.0	51.0	150.0	88.0
85	how	86.0	81.0	73.0	57.0	152.0	56.0
86	way	87.0	89.0	84.0	69.0	118.0	145.0
87	think	88.0	125.0	103.0	82.0	227.0	114.0
88	little	89.0	54.0	47.0	43.0	229.0	80.0
89	without	90.0	104.0	106.0	107.0	105.0	79.0
90	here	91.0	84.0	82.0	68.0	89.0	93.0
91	against	92.0	143.0	142.0	171.0	220.0	92.0
92	people	93.0	112.0	116.0	121.0	203.0	124.0
93	after	94.0	59.0	61.0	53.0	103.0	95.0
94	must	95.0	71.0	78.0	88.0	213.0	69.0
95	don	95.0	141.0	108.0	56.0	NaN	274.0
96	where	96.0	74.0	72.0	84.0	151.0	114.0
97	never	97.0	86.0	81.0	75.0	172.0	83.0
98	own	98.0	94.0	90.0	96.0	111.0	68.0
99	right	99.0	134.0	136.0	117.0	236.0	161.0

2.5 Trying to fit models to predict

2.5.1 Read in files

[15]:
```
file_contents = []
targets = []

files = [f for f in listdir(file_path) if isfile(join(file_path, f))]
files = list(filter(lambda file: file[0].isdigit(), files))
random.shuffle(files)

targets_=['70','80','90','00','10']
iter_ = 0

for f in files[:120]:
    file = open("processedData/" + f, encoding="ISO-8859-1")
    file_contents.append(file.read())
    iter_ = iter_+1
    targets.append(targets_[iter_%5])
```

2.5.2 Train models

[16]:
```
pipeline = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('kbest', SelectKBest(chi2, k=100)),
    ('nb', MultinomialNB()),
])

parameters = {
    #'vect__max_df': [1.0),
    # 'vect__max_features': (None, 5000, 10000, 50000),
    #'vect__ngram_range': ((1, 1), (1, 2)),  # unigrams or bigrams
    # 'tfidf__use_idf': (True, False),
    # 'tfidf__norm': ('l1', 'l2'),
    #'clf__max_iter': (20),
    #'clf__alpha': (0.00001),
    #'clf__penalty': ('l2'),
    # 'clf__max_iter': (10, 50, 80),
}

grid_search = GridSearchCV(pipeline, parameters, verbose=1)

grid_search.fit(file_contents, targets)
best_parameters = grid_search.best_estimator_.get_params()

for param_name in sorted(parameters.keys()):
    print("\t%s: %r" % (param_name, best_parameters[param_name]))
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   17.1s finished
```

2.6 Realisation and conclusion

At this point, we came to the conclusion that "year" in the Gutenberg dataset shows when the data **was published** to the project, and not the release date of the book.

We searched for possible solutions to get the years for book publications, but were unable to find any free API that we could link to our current dataset.

We therefore went on a search for other datasets, and to remake our hypothesis entirely. Thus, this part ended in a blind spot. However science is not only about the results, but also about the discoveries along the way, and therefore it is added into this file.

3 Studying language change in Icelandic parliamentary speeches

Our task involves research into language change over the past 100 years. Additionally we have been tasked with working out factors that influence language change.

Another proposed research question involves figuring out which languages are going extinct. This particular task has been found out to be near impossible to answer given the available data. It is estimated to be very hard to come up with data that capture the amount of speakers for a large enough ranges of combinations of language and year. Furthermore, any data that are available are likely to apply a different definition of "speaker" (sometimes including second/third… language speakers, sometimes not) and is also likely to contain politically motivated noise.

3.1 Introduction

Therefore we decided to search for English language corpora containing a wide array of text documents collected over the past century for predefined dialects of English and genre of text (movie, articles, books, …). This surprisingly turned out to be a complex endeavour as all high quality corpora were available only for a big price tag.

We also looked into the material provided by the Guttenberg Project Link. This turned out to be promising at first sight as it appears that there is a lot of recently published material. However release date of these documents does not match the year when the documents were actually written and soon enough we figured out that all material is from before 1923. This obviously did not allow us to look much into language change of the 20th and 21st century.

Gerlach, M., & Font-Clos, F. (2020). A standardized Project Gutenberg corpus for statistical analysis of natural language and quantitative linguistics. Entropy, 22(1), 126.

Theoretically one could obtain books from after 1923 and include them into the analysis. But one would quickly run into copyright/licensing issues here.

Obtaining the content of these books and preprocessing them for the purposes of data analysis turned out to be quite cumbersome as well. Look at Gunnar's notebooks (first draft here, second draft here) for the details.

Finally we turned to looking for non-English corpora and found an annotated corpus including pre-factured lemmatization of Icelandic parlimentary speeches from 1911 until 2018:

Steingrímsson, Steinþór, Sigrún Helgadóttir, Eiríkur Rögnvaldsson, Starkaður Barkarson and Jón Guðnason. 2018. Risamálheild: A Very Large Icelandic Text Corpus. Proceedings of LREC 2018, pp. 4361-4366. Myazaki, Japan.

3.2 Setup

3.2.1 Load required libraries

[17]:
```
import pandas as pd
import numpy as np
import xml.etree.ElementTree as ET

import glob

from nltk.probability import FreqDist
import random

from functools import reduce

from nltk import ngrams

# Used for building models for classifying:
from pprint import pprint
from time import time
import logging
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import SGDClassifier, LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import make_scorer, accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.base import BaseEstimator
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import confusion_matrix
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
```

```python
import matplotlib.pyplot as plt

from gensim.models import Word2Vec

from gensim.models import Doc2Vec
from gensim.models.doc2vec import TaggedDocument

from sklearn.metrics import plot_confusion_matrix
```

[18]:
```python
namespace = "{http://www.tei-c.org/ns/1.0}"
```

3.2.2 Get the data

Download data from here: http://www.malfong.is/index.php?dlid=81&lang=en

Then extract zip folder such that a folder labelled `CC_BY` shows up in the parent folder of this notebook. *Test*: `ls ../CC_BY/althingi` should work when run from `.../IcelandicParliamentSpeeches.ipynb`.

3.2.3 Preprocessing helpers

The data are available as XML. The text has already been preprocessed to be separated into paragraphs, sentences and words. Furthermore each word tag also includes a `lemma` attribute relating inflected/declensed forms of words to its lemma. This has been done by the authors of the original paper using Machine Learning approaches.

Given a relative path to a file, pull out a list with all the words. This can be achieved by looking for all tags of type `w`, additionally also retrieve the lemma for each word.

We will discard all sentences of length 3 or smaller to remove noise and to avoid that our models are able to detect year of speech just based on some short introductory/outro phrases. Furthermore the raw data appear to contain plenty of elements tagged as words that comprise of just a single letter followed by a dot. These will be removed here as well.

Pitfall: The namespace from above must be included when parsing out content from these XML files based on tag names.

In this kind of preprocessing we lose information about sentence boundaries as all punctuation items from the raw data are dropped.

[19]:
```python
def extract_words(path):
    xml_tree = ET.parse(open(path, 'r', encoding="utf8"))
    words = []
    lemmata = []

    for sentence in xml_tree.getroot().iter('{}s'.format(namespace)):
        words_in_sent = sentence.findall('{}w'.format(namespace))
        if len(words_in_sent) > 2:
            for word in sentence.findall('{}w'.format(namespace)):
                if not word.text.endswith('.'):
```

```
                    words.append(word.text)
                    lemmata.append(word.attrib['lemma'])

    return words, lemmata
```

Extract content of files separated into sentences, note that all stop items are wrapped in a `p` tag in the original documents and are not included here.

Also note that some further pre-processing could be done here to exclude items such as numbers, percentages, names, abbreviations, etc. In the original documents these are also assigned to be words:

```
[20]: def extract_sentences(path, lemma=False):
          xml_tree = ET.parse(open(path, 'r', encoding="utf8"))
          sentences = []

          for sentence in xml_tree.getroot().iter('{}s'.format(namespace)):
              sentence_cur = []
              words_in_sent = sentence.findall('{}w'.format(namespace))

              if len(words_in_sent) > 2:
                  for word in sentence.findall('{}w'.format(namespace)):
                      if not word.text.endswith('.'):
                          if lemma:
                              sentence_cur.append(word.attrib['lemma'])
                          else:
                              sentence_cur.append(word.text)

                  sentences.append(sentence_cur)

          return sentences
```

Retrieve a random selection of `k` file names from the entire corpus. The files must be of type `xml`. This method does not load the entire corpus into memory and allows you to work with smaller selections for test purposes. This method samples only from the `althingi` folder so far:

```
[21]: def get_random_sample(k):
          files = [filename for filename in glob.iglob('../CC_BY/althingi/**/*.xml',
                                                        recursive=True)]
          return random.sample(files, k)
```

Do the same as above but choose `k` files only from a given year (range: 1911-2017)

```
[22]: def get_files_for_year(year, k):
          files = [filename for filename in glob.iglob('../CC_BY/althingi/{}/'.
       ↪format(year) + '**/*.xml',
                                                        recursive=True)]
          return random.sample(files, min(len(files), k))
```

3.3 Preliminary Data Analysis

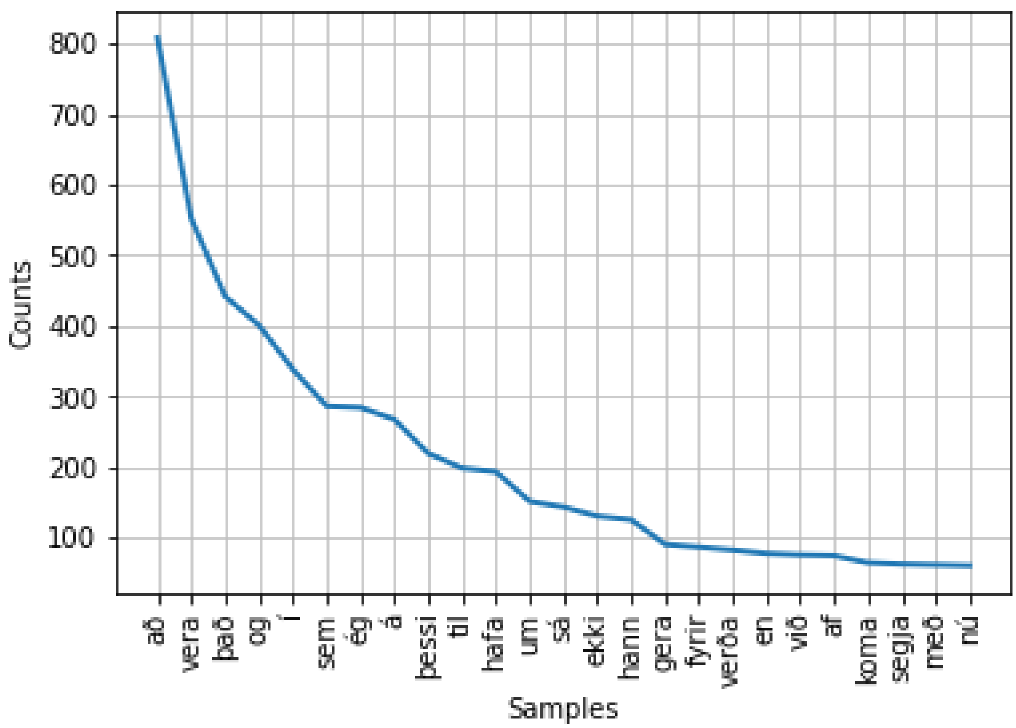
Doing some basic analysis on the documents in this section.

3.3.1 Zipf's Law

First using frequency distributions of the Natural Language ToolKit (`NLTK`) to look into whether or not we can confirm Zipf's Law based on the data we have.

Note that the analysis is done based on 15 randomly selected files from the entire corpus at this point:

[23]:
```
words = []

for file in get_random_sample(15):
    words.extend(extract_words(file)[1])

fq = FreqDist(word.lower() for word in words)
fq.plot(25, cumulative=False)
```

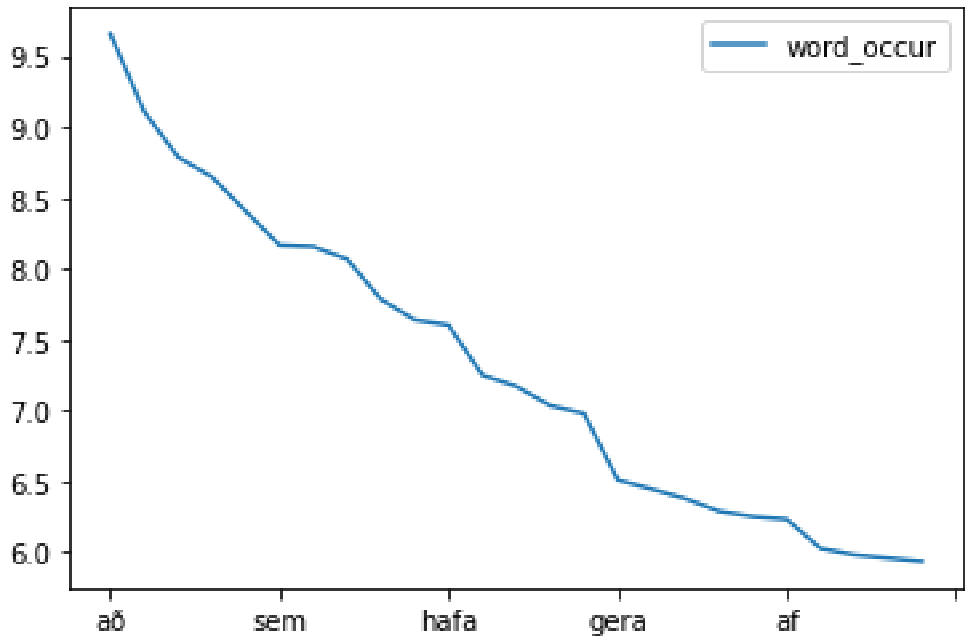


[23]: `<AxesSubplot:xlabel='Samples', ylabel='Counts'>`

Visualizing the same data but with using the logarithm of the occurrences, this should ideally obtain a straight line:

[24]:
```
freq_df = pd.DataFrame.from_dict(fq, orient='index', columns=['word_occur'])
freq_df.sort_values(by='word_occur', inplace=True, ascending=False)
freq_df.word_occur = np.log2(freq_df['word_occur'])
freq_df.head(25).plot(kind='line')
```

[24]:
```
<AxesSubplot:>
```



3.3.2 Disappearing words / new words

Here is a description

[25]:
```
words_1914 = []
words_2014 = []

for file in get_files_for_year(1914, 25):
    words_1914.extend(extract_words(file)[1])

for file in get_files_for_year(2014, 25):
    words_2014.extend(extract_words(file)[1])
```

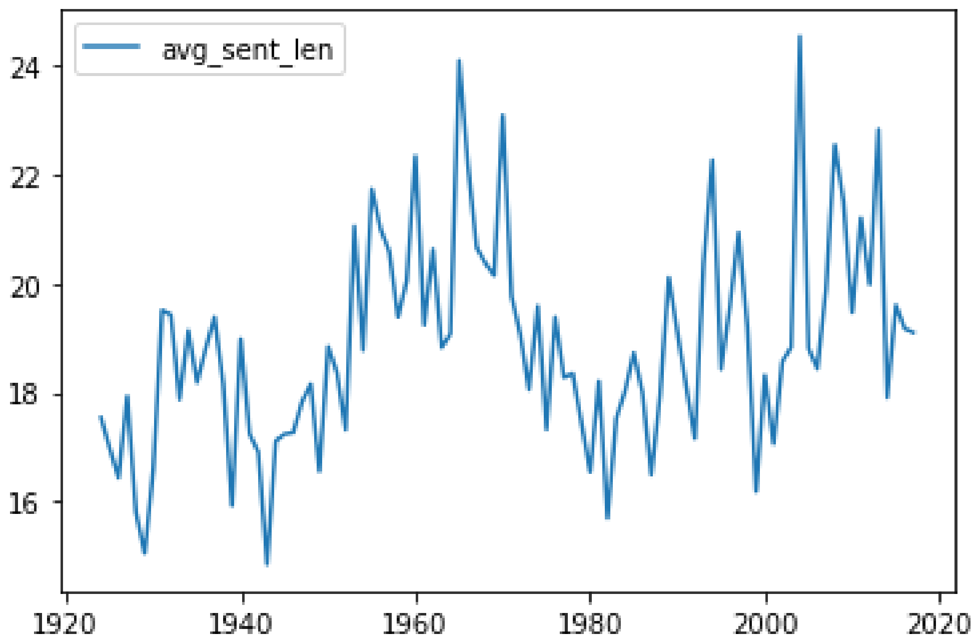
3.3.3 Development of average sentence length

This is just one possible metric for the development/analysis of language complexity. There is so much more you could come up with here.

Obviously our choice to discard very short sentences in the preprocessing step has an impact on the values here:

[26]:
```
def avg_sentence_length_year(year, k):
    sentence_len = []
    for file in get_files_for_year(year, k):
        sentences = extract_sentences(file)
        sentence_len.extend([len(s) for s in sentences])

    return reduce(lambda a, b: a + b, sentence_len) / len(sentence_len)

sentence_len_years = []

for year in range(1924, 2018):
    sentence_len_years.append(avg_sentence_length_year(year, 20))

avg_df = pd.DataFrame(sentence_len_years, index=range(1924, 2018),␣
 ↪columns=['avg_sent_len'])
avg_df.plot(kind='line')
```

[26]:
```
<AxesSubplot:>
```

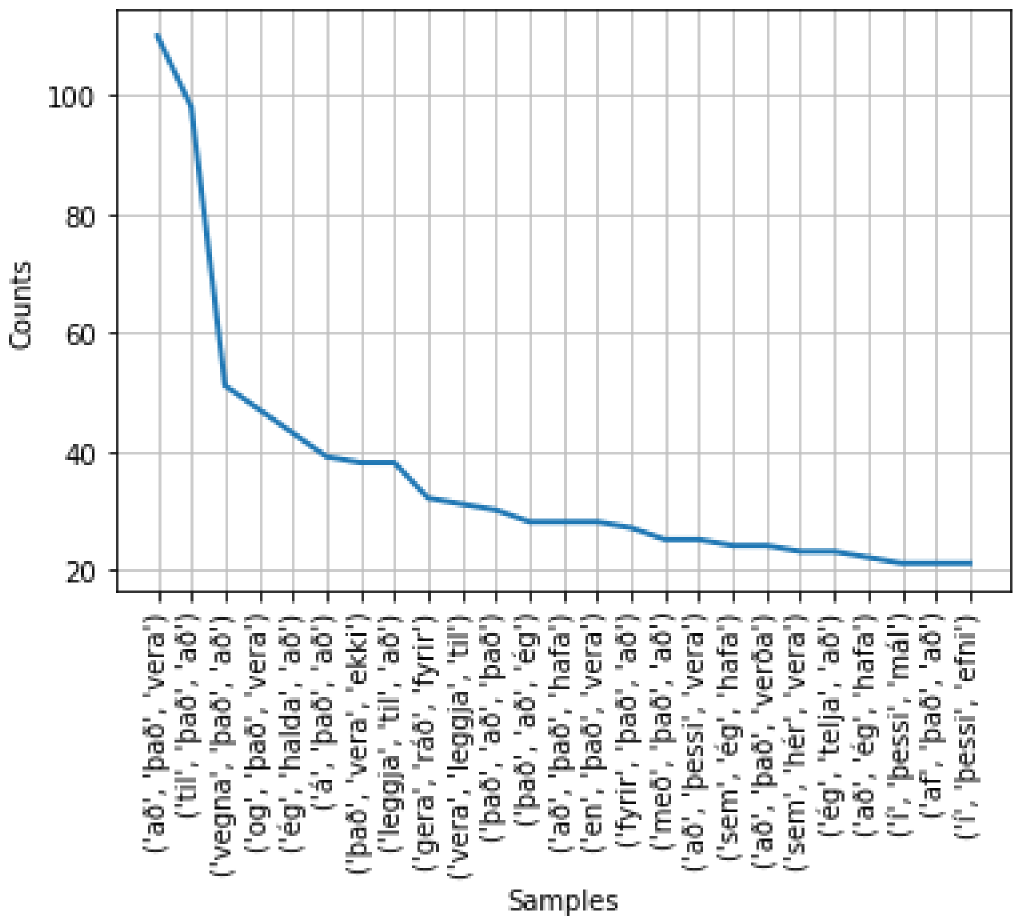


3.3.4 n-grams

Here is a description

[27]:
```
def most_common_ngrams(n, top_k, sample):
    file_contents = []
```

```
    for file in get_random_sample(sample):
        file_contents.extend(extract_words(file)[1])

    fq_ngr = FreqDist(ngrams(file_contents, n))
    fq_ngr.plot(top_k, cumulative=False)

most_common_ngrams(n=3, top_k=25, sample=100)
```



3.4 Building model for classifying speeches

The idea of this section is as follows:

- Import necessary libraries for the implementation of the next steps.
- Split the data in two sets: training and test.
- Select feature extraction methods to be used.
- Select classifiers to be trained.
- Train the classifiers using different feature extraction methods and different combination of hyperparameters within the classifiers. Perform a cross-validated grid search for that purpose.
- Compare the cross validation results within the trained models and select the best ones.

3.4.1 Constructing training and test data

So far, we don't have a defined strategy for train-test split. However, we extract some documents to be able to try the models.

We will have to design an appropiate strategy for a train-test split in which we shpuld keep the proportions within each class and so on.

Train data

[28]:
```
#set seed for reproducibility
random.seed(123)

file_contents = []
targets = []

#for year in [1914, 1933,1959,1968,1971,1984,1997,2005,  2016]:
#    for file in get_files_for_year(year, 20):
for year in [1914, 1912,
             1933, 1936, 1939, 1938, 1934,
             1954, 1955, 1957, 1959, 1956,
             1975, 1978, 1972, 1971, 1977,
             1992, 1995, 1999, 1997, 1993,
             2014, 2015, 2013, 2016, 2012]:
    for file in get_files_for_year(year, 200):
        file_contents.append(extract_words(file)[1])
        targets.append(year - year%10)
```

Let's randomly choose a fixed number of documents (here currently: 5) from various different decades. Then passing (document, decade) pairs to the model below. The decade is computed by subtracting `mod(<year>, 10)` from `<year>`.

Test data

[29]:
```
#seed for reproducibility
random.seed(123)

file_contents_test = []
targets_test = []

#for year in [1914, 1936, 1955, 1975, 1995, 2015]:
for year in [1911, 1937, 1958, 1973, 1994, 2017]:
    for file in get_files_for_year(year, 50):
        file_contents_test.append(extract_words(file)[1])
        targets_test.append(year - year%10)
```

3.4.2 Text feature extraction

We have considered 3 different methods for text feature extraction: Tf-idf, word2vec and doc2vec. All of them will be implemented through the corresponding functions from *sklearn* library.

TF-IDF Helper function to transform the data so that it is in the right format for the tfidfVectorizer() function that will be used later on:

[30]:
```
class JoinElement(object):
    def fit(self, X, y):
        return self

    def transform(self, X):
        #joins the elements of a list (which represents a document) into a␣
↪single string
        #with a blank space separation between each word
        return [' '.join(X[i]) for i in range(len(X))]
```

More information about it: sklearn documentation.

Word2Vec Original paper

: Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. Advances in neural information processing systems, 26, 3111-3119.

With this model every word is assigned a unique vector of configurable cardinality such that the dot product of two randomly chosen vectors should be proportional to the semantic similarity for the associated words. This happens during the training step using logistic regression and sliding windows. Personally I found that this video delivers a solid explanation of the concepts: https://www.youtube.com/watch?v=QyrUentbkvw

However, since we are working with entire documents as training items we have to somehow aggregate the vectors for every word in a given document. This can be done e.g. by taking the mean and/or summing up the vectors (see `MeanEmbeddingVectorizer`), optionally weighted by TF-IDF (see `MeanEmbeddingVectorizerTfidf`).

[31]:
```
class MeanEmbeddingVectorizer(BaseEstimator):
    def fit(self, X, y):
        self.word2vec = Word2Vec(X)
        return self

    def transform(self, X):
        return np.array([
            np.mean([self.word2vec.wv[w] for w in words if w in self.word2vec.
↪wv.vocab]
                    or [np.zeros(self.word2vec.vector_size)], axis=0)
            for words in X
        ])

    def fit_transform(self, X, y):
        self.fit(X, y)
        return self.transform(X)
```

[32]:
```python
class MeanEmbeddingVectorizerTfidf(BaseEstimator):
    def fit(self, X, y):
        self.word2vec = Word2Vec(X)
        self.X_joined = [' '.join(X[i]) for i in range(len(X))]
        self.vectorizer = TfidfVectorizer()
        self.transformed = self.vectorizer.fit_transform(self.X_joined)
        self.transformed = pd.DataFrame.sparse.from_spmatrix(self.transformed)
        return self

    def tfidf(self, w, docid):
        if w in self.vectorizer.vocabulary_:
            return self.transformed[self.vectorizer.vocabulary_[w]][docid]
        else:
            return 0

    def transform(self, X):
        return np.array([
            np.mean([self.word2vec.wv[w] * self.tfidf(w, i) for w in words if w␣
↪in self.word2vec.wv.vocab]
                    or [np.zeros(self.word2vec.vector_size)], axis=0)
            for i, words in enumerate(X)
        ])

    def fit_transform(self, X, y):
        self = self.fit(X, y)
        return self.transform(X)
```

Doc2Vec Finally we are attempting to build a model using *Doc2Vec*. After training this model with our training corpus we receive a vector of configurable cardinality for each document.

Original paper

: Le, Quoc, and Tomas Mikolov. "Distributed representations of sentences and documents." International conference on machine learning. 2014.

[33]:
```python
class Doc2Vectorizer(BaseEstimator):
    def __init__(self, window=2, vector_size=100):
        self.window = window
        self.vector_size = vector_size

    def fit(self, X, y):
        docs = [TaggedDocument(X[i], [y[i]]) for i in range(len(X))]
        self.doc_vec = Doc2Vec(docs, vector_size=self.vector_size, window=self.
↪window, min_count=1, workers=4)
        return self

    def transform(self, X):
        return [self.doc_vec.infer_vector(X[i]) for i in range(len(X))]
```

BERT (*Bidirectional Encoder Representations from Transformers*) is also interesting to look at, but we'll skip this here because we predict training a model from scratch would use up too many resources. Given more time however you could search for pretrained networks that roughly serve the purpose of classification of documents according to publication year.

Paper

: Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

3.4.3 Classifiers

3 different classifiers are going to be trained: Multinomial Naive Bayes, Support Vector Machines and Random Forest Classifier. All of them will be implemented using sklearn library.

Multinominal Naive Bayes Source: sklearn documentation.

Support Vector Machines Source: sklearn documentation.

Random Forest Classifier Source: sklearn documentation.

3.4.4 Train models

Since there are 3 methods for feature extraction and 3 classifiers, we should train 9 kind of models with their different combinations of hyperparameters. However, multinomial naive bayes does not take negative values produced by Word2Vec and Doc2Vec. Therefore, we have 7.

For each model, a grid search is performed with different combinations of hyperparameters for the classifiers and the text extraction methods. Afterwards, the most relevant results of each of the models are stored in a pandas data frame.

Model 1: TFIDF vectorizer, select K best and Multinomial Naive Bayes

[34]:
```
#choose parameters for the different steps in the pipeline
parameters_model_1 = {

    #select KBest
    "k_best__k": [1000],
    "k_best__score_func": [chi2],

    #MultinomialNaiveBayes
    "MNB__alpha" : [0,0.05,0.1,0.5,1],
    "MNB__fit_prior": [True,False]

}


#build a pipeline
model_1_pipeline = Pipeline([
        #joins list into a single string
```

```
        ('join', JoinElement()),
        #tfidf vectorizer
        ('tfidf', TfidfVectorizer()),
        #select 1000 best from word vectors
        ('k_best', SelectKBest()),
        #apply naive bayes
        ('MNB', MultinomialNB())
    ])

#design grid search
grid_search_model_1 = GridSearchCV(
    #pipeline to be followed
    model_1_pipeline,
    #parameters
    param_grid=parameters_model_1,
    #number of folds for CV
    cv=5,
    #scoring to be considered for the cv
    scoring = "accuracy",
    #parallelize if possible
    n_jobs=-1
)

#fit the grid search for training data
grid_search_model_1.fit(file_contents, targets)

#save results of cross validation
cv_results_model_1 = pd.DataFrame(grid_search_model_1.cv_results_)

#filter columns to be kept in the dataframe
filter_col = [col for col in cv_results_model_1 if (col.startswith("param_") or col.startswith("mean_") or col.startswith("rank"))]

#save results with only filtered columns
cv_results_model_1 = cv_results_model_1[filter_col]

#save name of the model for later comparison
cv_results_model_1.insert(loc=0, column="Model", value= "1")

#round mean_test_score
cv_results_model_1["mean_test_score"] = cv_results_model_1["mean_test_score"].round(2)

#show best 5 sorted by mean_test_score
display(cv_results_model_1.sort_values(by="mean_test_score", ascending=False).head(5))
```

```
/home/waxfactor2nd/anaconda3/lib/python3.8/site-
packages/sklearn/naive_bayes.py:511: UserWarning: alpha too small will result in
numeric errors, setting alpha = 1.0e-10
  warnings.warn('alpha too small will result in numeric errors, '
```

```
  Model  mean_fit_time  mean_score_time param_MNB__alpha param_MNB__fit_prior  \
1     1       1.524927         0.331172                0                False
3     1       1.391712         0.330401             0.05                False
5     1       1.407895         0.342288              0.1                False
0     1       1.444507         0.329906                0                 True
2     1       1.401343         0.336928             0.05                 True

  param_k_best__k           param_k_best__score_func  mean_test_score  \
1            1000  <function chi2 at 0x7fbbdbf5dc10>             0.82
3            1000  <function chi2 at 0x7fbbdbf5dc10>             0.82
5            1000  <function chi2 at 0x7fbbdbf5dc10>             0.81
0            1000  <function chi2 at 0x7fbbdbf5dc10>             0.80
2            1000  <function chi2 at 0x7fbbdbf5dc10>             0.78

   rank_test_score
1                1
3                2
5                3
0                4
2                5
```

Model 2: TFIDF vectorizer, select K best and SVC

[35]:
```
#choose parameters for the different steps in the pipeline
parameters_model_2 = {

    #select KBest
    "k_best__k": [1000],
    "k_best__score_func": [chi2],

    #SVC
    "SVC__kernel" : ["linear", "poly", "sigmoid"],
    "SVC__degree": [2,3]

}


#build a pipeline
model_2_pipeline = Pipeline([
        #joins list into a single string
        ('join', JoinElement()),
        #tfidf vectorizer
        ('tfidf', TfidfVectorizer()),
```

```
        #select 1000 best from word vectors
        ('k_best', SelectKBest()),
        #apply naive bayes
        ('SVC', SVC())
    ])

#design grid search
grid_search_model_2 = GridSearchCV(
    #pipeline to be followed
    model_2_pipeline,
    #parameters
    param_grid=parameters_model_2,
    #number of folds for CV
    cv=5,
    #scoring to be considered for the cv
    scoring = "accuracy",
    #parallelize if possible
    n_jobs=-1
)

#fit the grid search for training data
grid_search_model_2.fit(file_contents, targets)

#save results of cross validation
cv_results_model_2 = pd.DataFrame(grid_search_model_2.cv_results_)

#filter columns to be kept in the dataframe
filter_col = [col for col in cv_results_model_2 if (col.startswith("param_") or␣
 ↪col.startswith("mean_") or col.startswith("rank"))]

#save results with only filtered columns
cv_results_model_2 = cv_results_model_2[filter_col]

#save name of the model for later comparison
cv_results_model_2.insert(loc=0, column="Model", value= "2")

#round mean_test_score
cv_results_model_2["mean_test_score"] = cv_results_model_2["mean_test_score"].
 ↪round(2)

#show best 5 sorted by mean_test_score
display(cv_results_model_2.sort_values(by="mean_test_score", ascending=False).
 ↪head(5))
```

```
  Model  mean_fit_time  mean_score_time param_SVC__degree param_SVC__kernel  \
1     2       5.597282         1.432608                 2              poly
0     2       5.205752         1.409004                 2            linear
```

```
3     2       6.459674        1.811362                3            linear
4     2       6.378722        1.368371                3              poly
2     2       5.524270        1.345796                2           sigmoid

  param_k_best__k           param_k_best__score_func  mean_test_score  \
1            1000  <function chi2 at 0x7fbbdbf5dc10>             0.78
0            1000  <function chi2 at 0x7fbbdbf5dc10>             0.75
3            1000  <function chi2 at 0x7fbbdbf5dc10>             0.75
4            1000  <function chi2 at 0x7fbbdbf5dc10>             0.73
2            1000  <function chi2 at 0x7fbbdbf5dc10>             0.55

   rank_test_score
1                1
0                2
3                2
4                4
2                5
```

Model 3: TFIDF vectorizer, select K best and Random Forest Classifier

[36]:
```
#choose parameters for the different steps in the pipeline
parameters_model_3 = {

    #select KBest
    "k_best__k": [1000],
    "k_best__score_func": [chi2],

    #RF classifier
    "clf__n_estimators" : [10,100,200,300]

}


#build a pipeline
model_3_pipeline = Pipeline([
        #joins list into a single string
        ('join', JoinElement()),
        #tfidf vectorizer
        ('tfidf', TfidfVectorizer()),
        #select 1000 best from word vectors
        ('k_best', SelectKBest()),
        #apply naive bayes
        ('clf', RandomForestClassifier())
    ])

#design grid search
grid_search_model_3 = GridSearchCV(
    #pipeline to be followed
```

```
    model_3_pipeline,
    #parameters
    param_grid=parameters_model_3,
    #number of folds for CV
    cv=5,
    #scoring to be considered for the cv
    scoring = "accuracy",
    #parallelize if possible
    n_jobs=-1
)

#fit the grid search for training data
grid_search_model_3.fit(file_contents, targets)

#save results of cross validation
cv_results_model_3 = pd.DataFrame(grid_search_model_3.cv_results_)

#filter columns to be kept in the dataframe
filter_col = [col for col in cv_results_model_3 if (col.startswith("param_") or␣
↪col.startswith("mean_") or col.startswith("rank"))]

#save results with only filtered columns
cv_results_model_3 = cv_results_model_3[filter_col]

#save name of the model for later comparison
cv_results_model_3.insert(loc=0, column="Model", value= "3")

#round mean_test_score
cv_results_model_3["mean_test_score"] = cv_results_model_3["mean_test_score"].
↪round(2)

#show best 5 sorted by mean_test_score
display(cv_results_model_3.sort_values(by="mean_test_score", ascending=False).
↪head(5))
```

```
  Model  mean_fit_time  mean_score_time param_clf__n_estimators  \
2     3       6.932103         0.480441                     200
3     3       8.926858         0.383013                     300
1     3       3.773559         0.393197                     100
0     3       1.563078         0.324162                      10

  param_k_best__k           param_k_best__score_func  mean_test_score  \
2            1000  <function chi2 at 0x7fbbdbf5dc10>             0.73
3            1000  <function chi2 at 0x7fbbdbf5dc10>             0.73
1            1000  <function chi2 at 0x7fbbdbf5dc10>             0.72
0            1000  <function chi2 at 0x7fbbdbf5dc10>             0.62
```

```
   rank_test_score
2                1
3                2
1                3
0                4
```

Model 4: Word2Vec and SVC

[37]:
```
#choose parameters for the different steps in the pipeline
parameters_model_4 = {

    #SVC
    "SVC__kernel" : ["linear", "poly", "sigmoid"],
    "SVC__degree": [2,3]

}



#build a pipeline
model_4_pipeline = Pipeline([
        #tfidf vectorizer
        ('word2vec', MeanEmbeddingVectorizer()),
        #apply naive bayes
        ('SVC', SVC())
    ])

#design grid search
grid_search_model_4 = GridSearchCV(
    #pipeline to be followed
    model_4_pipeline,
    #parameters
    param_grid=parameters_model_4,
    #number of folds for CV
    cv=5,
    #scoring to be considered for the cv
    scoring = "accuracy",
    #parallelize if possible
    n_jobs=-1
)

#fit the grid search for training data
grid_search_model_4.fit(file_contents, targets)

#save results of cross validation
cv_results_model_4 = pd.DataFrame(grid_search_model_4.cv_results_)

#filter columns to be kept in the dataframe
```

```
filter_col = [col for col in cv_results_model_4 if (col.startswith("param_") or␣
↪col.startswith("mean_") or col.startswith("rank"))]

#save results with only filtered columns
cv_results_model_4 = cv_results_model_4[filter_col]

#save name of the model for later comparison
cv_results_model_4.insert(loc=0, column="Model", value= "4")

#round mean_test_score
cv_results_model_4["mean_test_score"] = cv_results_model_4["mean_test_score"].
↪round(2)

#show best 5 sorted by mean_test_score
display(cv_results_model_4.sort_values(by="mean_test_score", ascending=False).
↪head(5))
```

```
  Model  mean_fit_time  mean_score_time param_SVC__degree param_SVC__kernel  \
0     4      33.631611         2.571135                 2            linear
3     4      38.590638         2.533163                 3            linear
4     4      36.673138         2.196457                 3              poly
1     4      38.521074         2.761024                 2              poly
2     4      40.036771         2.953432                 2           sigmoid

   mean_test_score  rank_test_score
0             0.54                1
3             0.53                2
4             0.49                3
1             0.46                4
2             0.39                5
```

Model 5: Word2Vec and Random Forest Classifier

[38]:
```
#choose parameters for the different steps in the pipeline
parameters_model_5 = {

    #RF classifier
    "clf__n_estimators" : [10,100,200,300]

}


#build a pipeline
model_5_pipeline = Pipeline([
        #tfidf vectorizer
        ('word2vec', MeanEmbeddingVectorizer()),
        #apply naive bayes
        ('clf', RandomForestClassifier())
```

```
    ])

#design grid search
grid_search_model_5 = GridSearchCV(
    #pipeline to be followed
    model_5_pipeline,
    #parameters
    param_grid=parameters_model_5,
    #number of folds for CV
    cv=5,
    #scoring to be considered for the cv
    scoring = "accuracy",
    #parallelize if possible
    n_jobs=-1
)

#fit the grid search for training data
grid_search_model_5.fit(file_contents, targets)

#save results of cross validation
cv_results_model_5 = pd.DataFrame(grid_search_model_5.cv_results_)

#filter columns to be kept in the dataframe
filter_col = [col for col in cv_results_model_5 if (col.startswith("param_") or␣
 ↪col.startswith("mean_") or col.startswith("rank"))]

#save results with only filtered columns
cv_results_model_5 = cv_results_model_5[filter_col]

#save name of the model for later comparison
cv_results_model_5.insert(loc=0, column="Model", value= "5")

#round mean_test_score
cv_results_model_5["mean_test_score"] = cv_results_model_5["mean_test_score"].
 ↪round(2)

#show best 5 sorted by mean_test_score
display(cv_results_model_5.sort_values(by="mean_test_score", ascending=False).
 ↪head(5))
```

```
  Model  mean_fit_time  mean_score_time param_clf__n_estimators  \
2     5      41.454146         1.613599                     200
3     5      34.221691         0.899461                     300
1     5      41.179541         2.397368                     100
0     5      33.133284         2.269344                      10

   mean_test_score  rank_test_score
```

2	0.47	2
3	0.47	1
1	0.46	3
0	0.37	4

Model 6: Doc2Vec and Support Vector Machines

[39]:
```
#choose parameters for the different steps in the pipeline
parameters_model_6 = {

    #doc2Vec
    'doc2vec__window': [4],
    'doc2vec__vector_size': [300],

    #SVC
    "SVC__kernel" : ["linear", "poly", "sigmoid"],
    "SVC__degree": [2,3]

}



#build a pipeline
model_6_pipeline = Pipeline([
        #tfidf vectorizer
        ('doc2vec', Doc2Vectorizer()),
        #apply naive bayes
        ('SVC', SVC())
    ])

#design grid search
grid_search_model_6 = GridSearchCV(
    #pipeline to be followed
    model_6_pipeline,
    #parameters
    param_grid=parameters_model_6,
    #number of folds for CV
    cv=5,
    #scoring to be considered for the cv
    scoring = "accuracy",
    #parallelize if possible
    n_jobs=-1
)

#fit the grid search for training data
grid_search_model_6.fit(file_contents, targets)

#save results of cross validation
cv_results_model_6 = pd.DataFrame(grid_search_model_6.cv_results_)
```

```
#filter columns to be kept in the dataframe
filter_col = [col for col in cv_results_model_6 if (col.startswith("param_") or␣
↪col.startswith("mean_") or col.startswith("rank"))]

#save results with only filtered columns
cv_results_model_6 = cv_results_model_6[filter_col]

#save name of the model for later comparison
cv_results_model_6.insert(loc=0, column="Model", value= "6")

#round mean_test_score
cv_results_model_6["mean_test_score"] = cv_results_model_6["mean_test_score"].
↪round(2)

#show best 5 sorted by mean_test_score
display(cv_results_model_6.sort_values(by="mean_test_score", ascending=False).
↪head(5))
```

```
  Model  mean_fit_time  mean_score_time param_SVC__degree param_SVC__kernel  \
0     6     101.292369        10.695421                 2            linear
3     6     107.363069         9.940223                 3            linear
1     6     111.739065        10.010586                 2              poly
4     6     103.355583         8.901331                 3              poly
2     6     106.517593        11.002855                 2           sigmoid

  param_doc2vec__vector_size param_doc2vec__window  mean_test_score  \
0                        300                     4             0.68
3                        300                     4             0.67
1                        300                     4             0.49
4                        300                     4             0.47
2                        300                     4             0.22

   rank_test_score
0                1
3                2
1                3
4                4
2                6
```

Model 7: Doc2Vec and Random Forest Classifier

[40]:
```
#choose parameters for the different steps in the pipeline
parameters_model_7 = {

    #doc2Vec
    'doc2vec__window': [4],
    'doc2vec__vector_size': [300],
```

```
    #RF classifier
    "clf__n_estimators" : [10,100,200,300]

}


#build a pipeline
model_7_pipeline = Pipeline([
        #tfidf vectorizer
        ('doc2vec', Doc2Vectorizer()),
        #apply naive bayes
        ('clf', RandomForestClassifier())
    ])

#design grid search
grid_search_model_7 = GridSearchCV(
    #pipeline to be followed
    model_7_pipeline,
    #parameters
    param_grid=parameters_model_7,
    #number of folds for CV
    cv=5,
    #scoring to be considered for the cv
    scoring = "accuracy",
    #parallelize if possible
    n_jobs=-1
)

#fit the grid search for training data
grid_search_model_7.fit(file_contents, targets)

#save results of cross validation
cv_results_model_7 = pd.DataFrame(grid_search_model_7.cv_results_)

#filter columns to be kept in the dataframe
filter_col = [col for col in cv_results_model_7 if (col.startswith("param_") or col.startswith("mean_") or col.startswith("rank"))]

#save results with only filtered columns
cv_results_model_7 = cv_results_model_7[filter_col]

#save name of the model for later comparison
cv_results_model_7.insert(loc=0, column="Model", value= "7")

#round mean_test_score
```

```
cv_results_model_7["mean_test_score"] = cv_results_model_7["mean_test_score"].
↪round(2)

#show best 5 sorted by mean_test_score
display(cv_results_model_7.sort_values(by="mean_test_score", ascending=False).
↪head(5))
```

```
  Model  mean_fit_time  mean_score_time param_clf__n_estimators  \
1     7     108.986529         7.831541                     100
2     7      99.640191         6.571433                     200
3     7      87.066936         4.432079                     300
0     7      97.149339         9.613009                      10

  param_doc2vec__vector_size param_doc2vec__window  mean_test_score  \
1                        300                     4             0.59
2                        300                     4             0.59
3                        300                     4             0.59
0                        300                     4             0.49

   rank_test_score
1                3
2                1
3                2
0                4
```

3.4.5 Compare CV results from trained models

In this section, the results from CV are compared within the trained models.

Raw results A dataframe showing the best models according to the mean accuracy within the test folds used for cross validation.

[41]:
```
#merge cv results into 1 that keeps the relevant information

#empty dataframe that will keep all the results
cv_results = pd.DataFrame()

#loop over cv results
for i in [cv_results_model_1, cv_results_model_2, cv_results_model_3,␣
↪cv_results_model_4,
          cv_results_model_5, cv_results_model_6, cv_results_model_7]:

    #select relevant columns
    selected = i[["Model","mean_fit_time","mean_score_time","mean_test_score"]]

    #append to cv results
    cv_results = cv_results.append(selected)
```

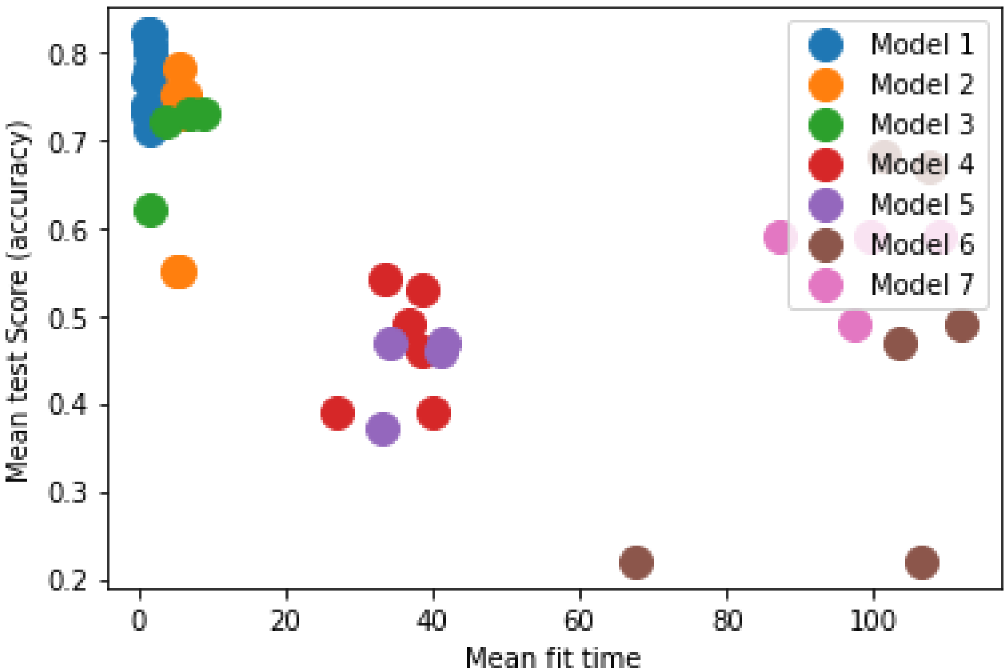
```
#show models with best scores
display(cv_results.sort_values(by="mean_test_score", ascending=False).head(10))
```

	Model	mean_fit_time	mean_score_time	mean_test_score
1	1	1.524927	0.331172	0.82
3	1	1.391712	0.330401	0.82
5	1	1.407895	0.342288	0.81
0	1	1.444507	0.329906	0.80
1	2	5.597282	1.432608	0.78
2	1	1.401343	0.336928	0.78
4	1	1.385342	0.340789	0.77
7	1	1.431214	0.349296	0.77
0	2	5.205752	1.409004	0.75
3	2	6.459674	1.811362	0.75

This table could be improved by also indicating the parameters used in each model but I thought it would be a bit overwhelming

Tradeoff score vs mean fit time A plot to check if there is some kind of tradeoff between accuracy and runtime of the algorithms.

[42]:
```
#group by model
groups = cv_results.groupby("Model")

# Plot
fig, ax = plt.subplots()
ax.margins(0.05) # Optional, just adds 5% padding to the autoscaling
for name, group in groups:
    ax.plot(group.mean_fit_time, group.mean_test_score, marker='o',␣
 ↪linestyle='', ms=12, label="Model %s" %name)
ax.legend(loc = 1)
plt.xlabel("Mean fit time")
plt.ylabel("Mean test Score (accuracy)")
plt.show()
```



We can add as much as we want here...

Best estimator from each model

[43]:
```
#plot to add
```

Best 5 models

[44]:
```
 #plot to add
```

3.5 Evaluation and model selection

Predict on "unseen" data using the best models obtained in the training phase and evaluate using different metrics.

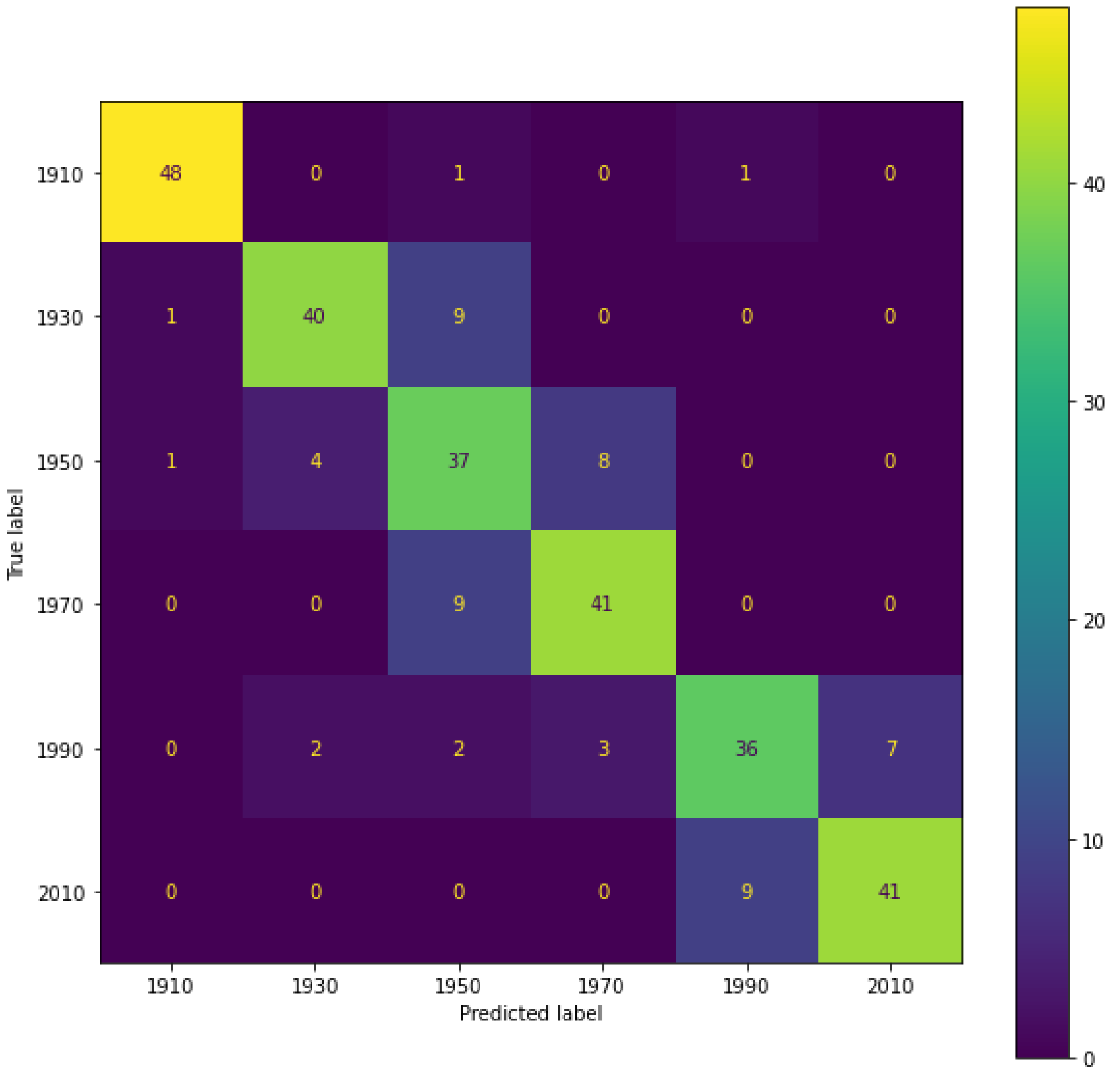
Best models will be selected when training the models with the actual train data

[45]:
```
#add models to be evaluated
models = [
    grid_search_model_1,
    grid_search_model_2,
    grid_search_model_3,
    grid_search_model_4,
    grid_search_model_5,
    grid_search_model_6,
    grid_search_model_7
]
```

```
evaluation = pd.DataFrame(columns=["model"
                                   , "mean_fit_time", "accuracy"
                                   , "recall_macro", "recall_micro"
                                   , "precision_macro", "precision_micro"
                                   , "f1_macro", "f1_micro"
                                   , "model_definition"
                                  ])

i = -1 # Ensure that first item is index 0 in the loop
for model_ in models:
    # Yucky method of finding mean fit times:
    i = i +1
    mean_fit_time = cv_results.groupby("Model")["mean_fit_time"].mean()[i]

    # Predict
    preds = model_.best_estimator_.predict(file_contents_test)
    model = cv_results.iloc[model_.best_index_,0]

    # Calculate metrix
    to_append = [
            "Model " + str(i+1),
            mean_fit_time,
            accuracy_score(y_true=targets_test,y_pred=preds),
            #choose micro or macro according to criteria
            recall_score(y_true=targets_test,y_pred=preds, average="macro"),
            recall_score(y_true=targets_test,y_pred=preds, average="micro"),
            precision_score(y_true=targets_test,y_pred=preds, average="macro"),
            precision_score(y_true=targets_test,y_pred=preds, average="micro"),
            f1_score(y_true=targets_test,y_pred=preds, average="macro"),
            f1_score(y_true=targets_test,y_pred=preds, average="micro"),
            model_
            ]

    #Append Metrics
    evaluation_length = len(evaluation)
    evaluation.loc[evaluation_length] = to_append

    # Print results and Confusion Matrix for each model
   ␣
 ↪print("####################################################################")
   ␣
 ↪print("####################################################################")
    print("                        Model "+ str(i+1) + ":")
   ␣
 ↪print("--------------------------------------------------------------------")
```

```
    print(evaluation.loc[i, 'mean_fit_time':'f1_macro'])
    print("\n")
    print("Pipeline: ")
    print(model_.best_estimator_)

    print("\n")
    print("Confusion Matrix: ")
    fig, ax = plt.subplots(figsize=(10, 10))
    plot_confusion_matrix(estimator=model_.best_estimator_
                          , X=file_contents_test
                          , y_true=targets_test
                          , ax=ax
                         )
    plt.show()

# Print table of the models compared and sorted:
evaluation.sort_values(by="accuracy", ascending = False)
```

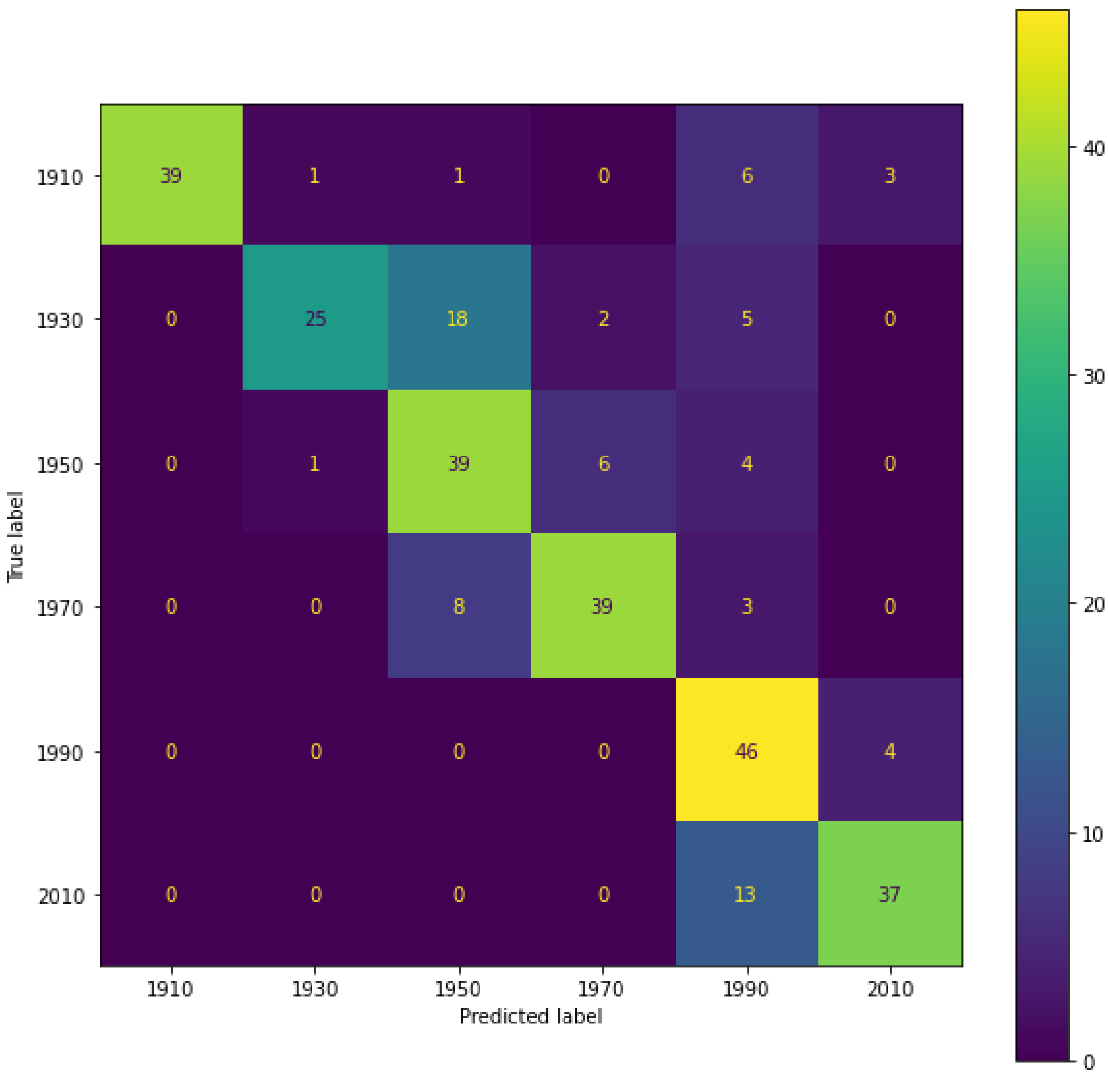
```
####################################################################
####################################################################
                        Model 1:
--------------------------------------------------------------------
mean_fit_time      1.415787
accuracy               0.81
recall_macro           0.81
recall_micro           0.81
precision_macro    0.815456
precision_micro        0.81
f1_macro           0.811529
Name: 0, dtype: object


Pipeline:
Pipeline(steps=[('join', <__main__.JoinElement object at 0x7fbabd0fbf70>),
                ('tfidf', TfidfVectorizer()),
                ('k_best',
                 SelectKBest(k=1000,
                             score_func=<function chi2 at 0x7fbbdbf5dc10>)),
                ('MNB', MultinomialNB(alpha=0, fit_prior=False))])


Confusion Matrix:
```




```
####################################################################
####################################################################
                        Model 2:
--------------------------------------------------------------------
mean_fit_time      5.701224
accuracy               0.75
recall_macro           0.75
recall_micro           0.75
precision_macro    0.797489
precision_micro        0.75
f1_macro           0.752323
Name: 1, dtype: object
```

```
Pipeline:
Pipeline(steps=[('join', <__main__.JoinElement object at 0x7fbbbbf05ee0>),
                ('tfidf', TfidfVectorizer()),
                ('k_best',
                 SelectKBest(k=1000,
                             score_func=<function chi2 at 0x7fbbdbf5dc10>)),
                ('SVC', SVC(degree=2, kernel='poly'))])
```

```
Confusion Matrix:
```

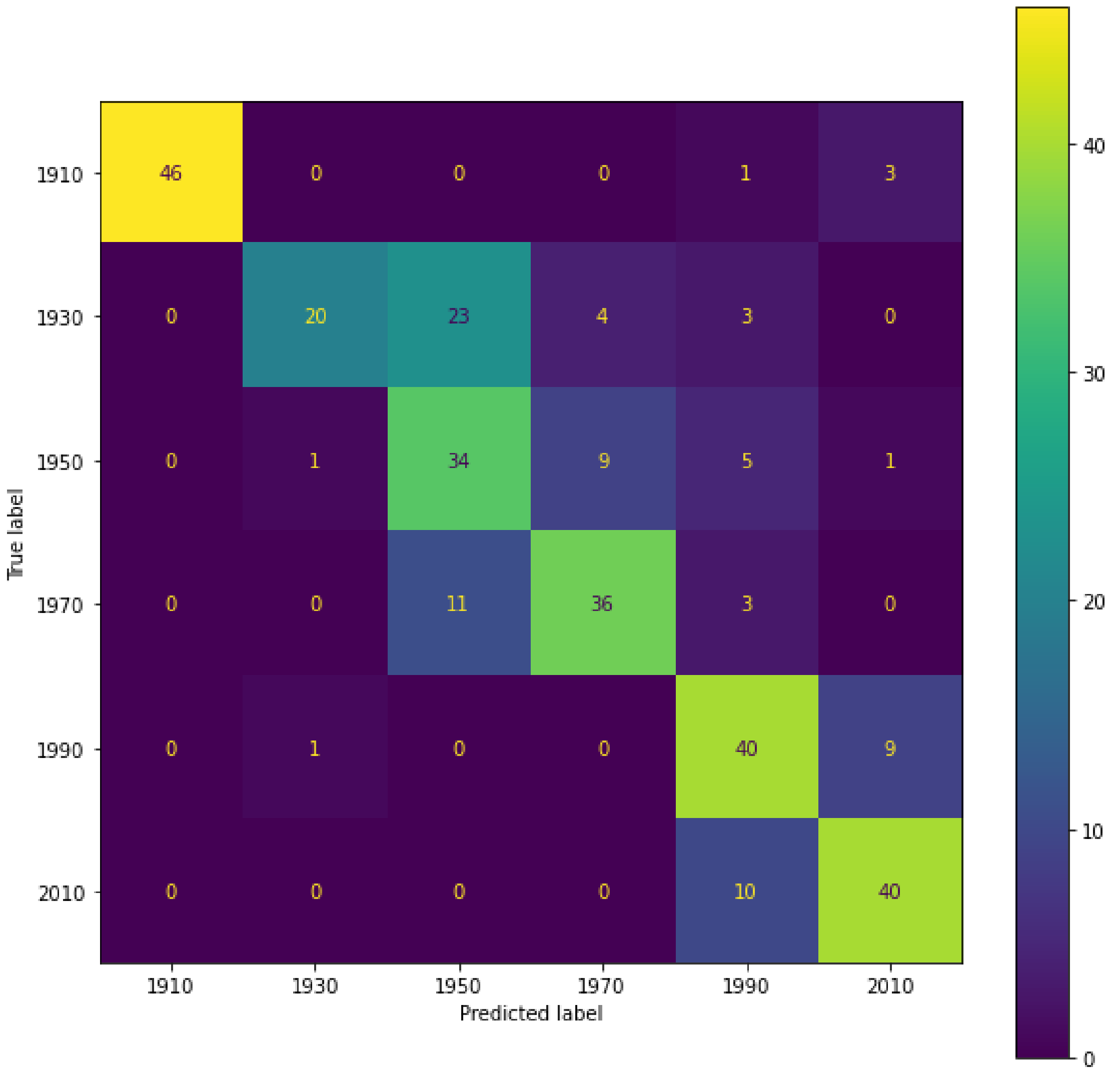



```
####################################################################
####################################################################
                        Model 3:
--------------------------------------------------------------------
```

```
mean_fit_time        5.2989
accuracy               0.72
recall_macro           0.72
recall_micro           0.72
precision_macro    0.757277
precision_micro        0.72
f1_macro            0.71807
Name: 2, dtype: object
```

```
Pipeline:
Pipeline(steps=[('join', <__main__.JoinElement object at 0x7fbbb97ba0d0>),
                ('tfidf', TfidfVectorizer()),
                ('k_best',
                 SelectKBest(k=1000,
                             score_func=<function chi2 at 0x7fbbdbf5dc10>)),
                ('clf', RandomForestClassifier(n_estimators=200))])
```

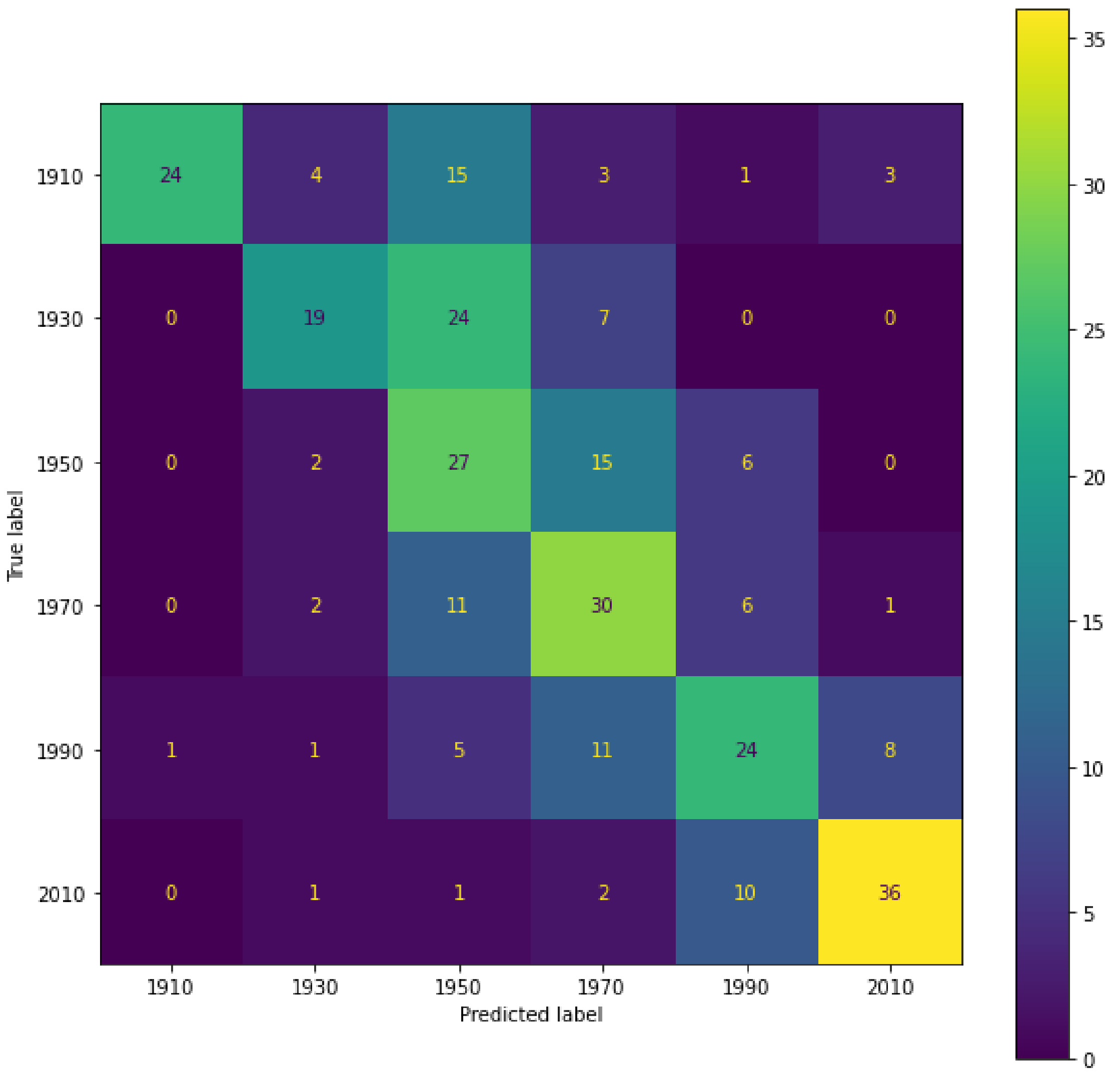
```
Confusion Matrix:
```




```
##################################################################
##################################################################
                       Model 4:
------------------------------------------------------------------
mean_fit_time      35.71164
accuracy           0.533333
recall_macro       0.533333
recall_micro       0.533333
precision_macro    0.607048
precision_micro    0.533333
f1_macro           0.544174
Name: 3, dtype: object
```

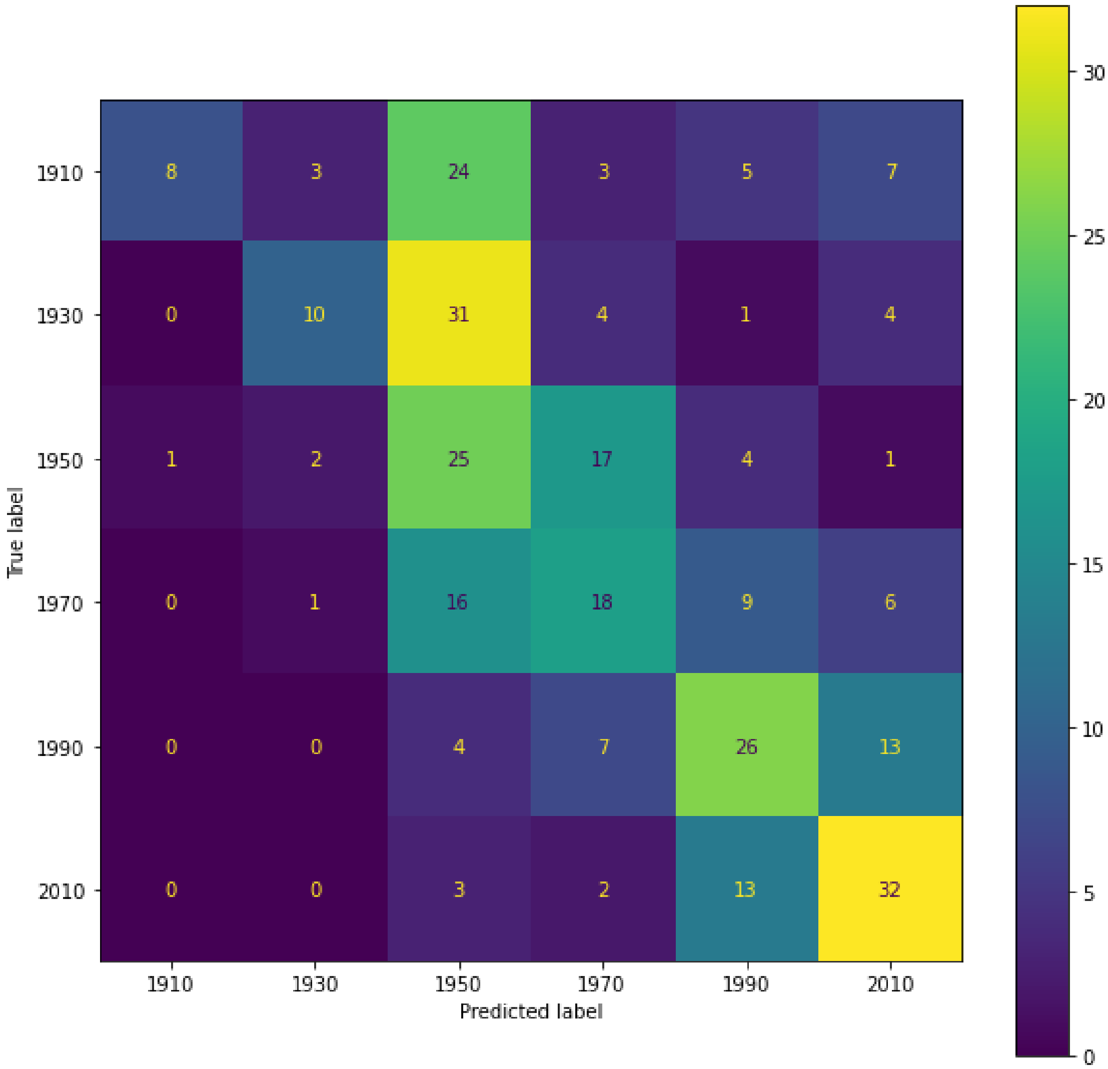
```
Pipeline:
Pipeline(steps=[('word2vec', MeanEmbeddingVectorizer()),
                ('SVC', SVC(degree=2, kernel='linear'))])
```

```
Confusion Matrix:
```



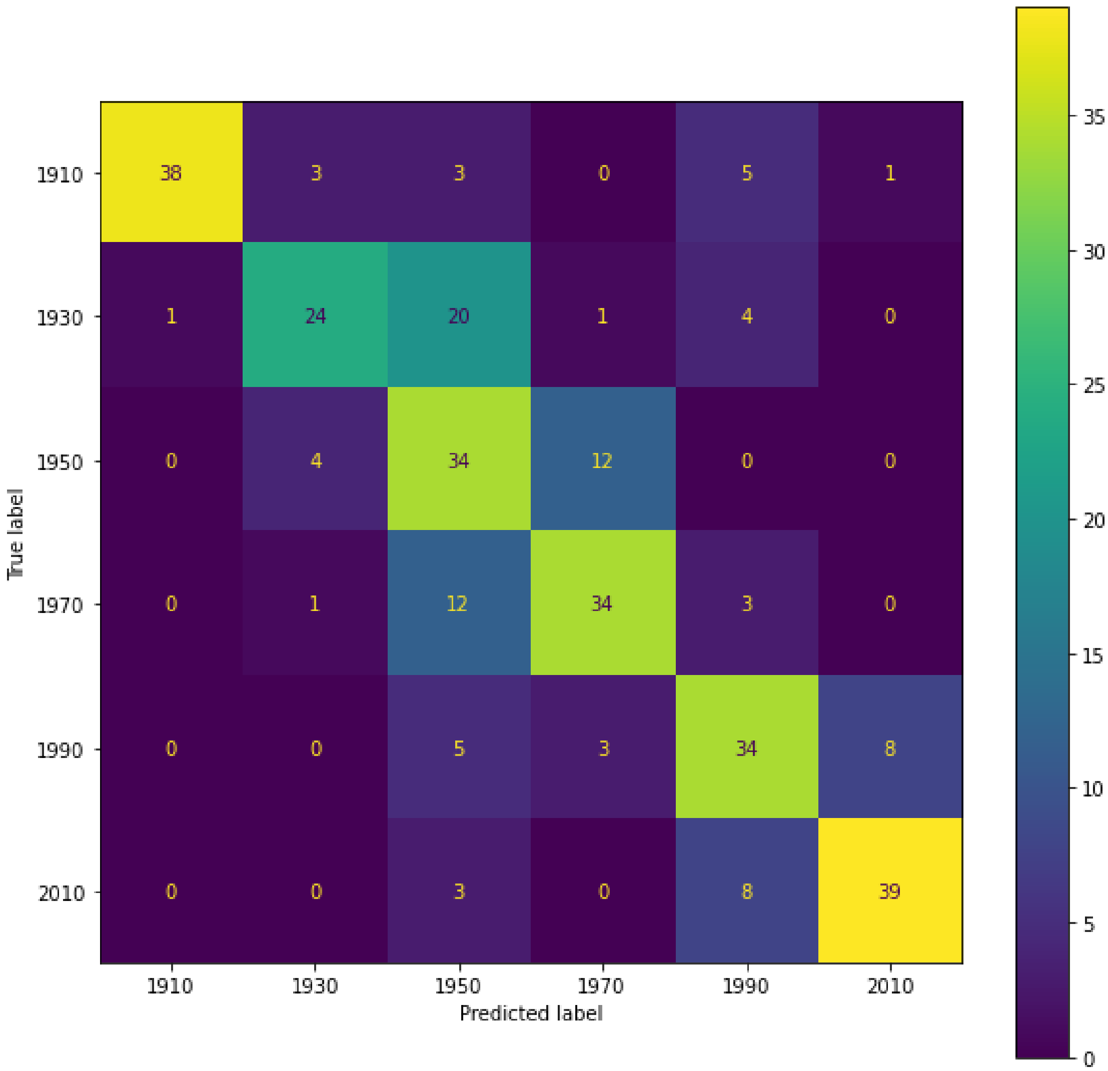

```
################################################################
################################################################
                        Model 5:
----------------------------------------------------------------
mean_fit_time       37.497166
accuracy             0.396667
recall_macro         0.396667
recall_micro         0.396667
```

```
precision_macro      0.51096
precision_micro     0.396667
f1_macro            0.384217
Name: 4, dtype: object


Pipeline:
Pipeline(steps=[('word2vec', MeanEmbeddingVectorizer()),
                ('clf', RandomForestClassifier(n_estimators=300))])


Confusion Matrix:
```



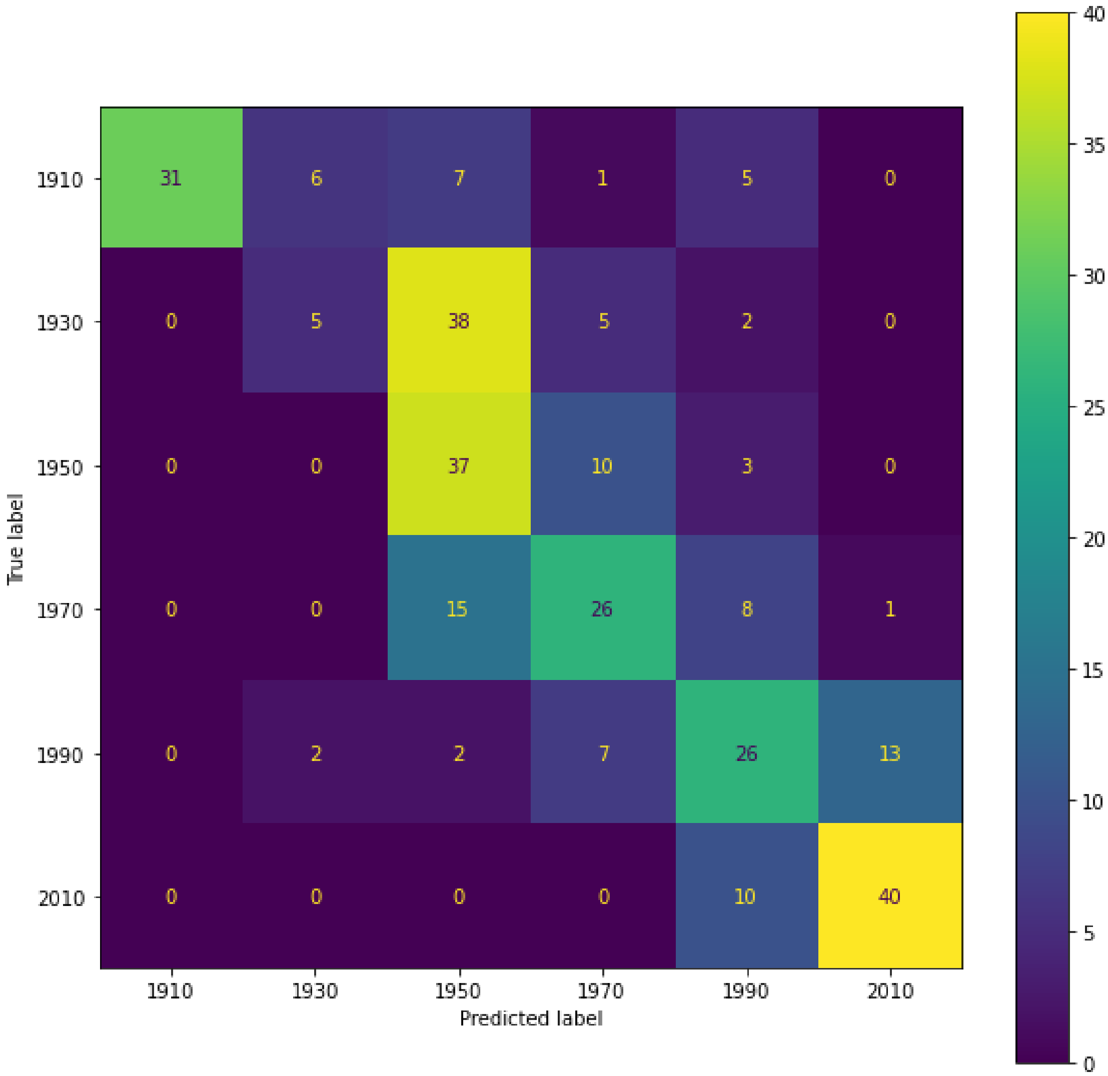

```
######################################################################
######################################################################
```

```
                        Model 6:
--------------------------------------------------------------------
mean_fit_time      99.642594
accuracy            0.693333
recall_macro        0.693333
recall_micro        0.693333
precision_macro     0.733468
precision_micro     0.693333
f1_macro            0.698787
Name: 5, dtype: object


Pipeline:
Pipeline(steps=[('doc2vec', Doc2Vectorizer(vector_size=300, window=4)),
                ('SVC', SVC(degree=2, kernel='linear'))])


Confusion Matrix:
```




```
####################################################################
####################################################################
                        Model 7:
--------------------------------------------------------------------
mean_fit_time      98.210749
accuracy            0.523333
recall_macro        0.523333
recall_micro        0.523333
precision_macro     0.562907
precision_micro     0.523333
f1_macro             0.51323
Name: 6, dtype: object
```

```
Pipeline:
Pipeline(steps=[('doc2vec', Doc2Vectorizer(vector_size=300, window=4)),
                ('clf', RandomForestClassifier(n_estimators=200))])


Confusion Matrix:
```



[45]:

	model	mean_fit_time	accuracy	recall_macro	recall_micro	\
0	Model 1	1.415787	0.810000	0.810000	0.810000	
1	Model 2	5.701224	0.750000	0.750000	0.750000	
2	Model 3	5.298900	0.720000	0.720000	0.720000	
5	Model 6	99.642594	0.693333	0.693333	0.693333	
3	Model 4	35.711640	0.533333	0.533333	0.533333	
6	Model 7	98.210749	0.523333	0.523333	0.523333	

```
4  Model 5      37.497166  0.396667      0.396667      0.396667

   precision_macro  precision_micro  f1_macro  f1_micro  \
0         0.815456         0.810000  0.811529  0.810000
1         0.797489         0.750000  0.752323  0.750000
2         0.757277         0.720000  0.718070  0.720000
5         0.733468         0.693333  0.698787  0.693333
3         0.607048         0.533333  0.544174  0.533333
6         0.562907         0.523333  0.513230  0.523333
4         0.510960         0.396667  0.384217  0.396667

                                   model_definition
0  GridSearchCV(cv=5,\n            estimator=Pip…
1  GridSearchCV(cv=5,\n            estimator=Pip…
2  GridSearchCV(cv=5,\n            estimator=Pip…
5  GridSearchCV(cv=5,\n            estimator=Pip…
3  GridSearchCV(cv=5,\n            estimator=Pip…
6  GridSearchCV(cv=5,\n            estimator=Pip…
4  GridSearchCV(cv=5,\n            estimator=Pip…
```

The **best estimator we found is:**

Best models will be selected when training the models with the actual train data

Predict using this estimator and show **confusion matrix** on test data:

Main **conclusion**: for unseen data, we would choose to use the estimator from above.

4 Conclusion

Draw conclusions from above. but add more evaulations and stuff.... Should probably be done after proper train/test is defined, and full model is tried

Some observations: * Model 1,2,3 (TFIDF) seems to work best overall, no matter training size * Model 5,6,7 (Doc2Vec) takes by far the most time, but their accuracy greatly increased when increasing training set

As often is the case in the fields of science, not all research leads to useable results. We ended up having to remodel our plans several times during this project, including a complete pivot of the datasets.

This did however give us some insight into how larger projects are managed. This also lead us to an interesting path of looking at a relatively obscure language.

Although further works is possible, we reached the conclusion that there is a change in the Icelandic spoken language throughout time, and it is therefore possible to train models that estimates which decade a given speech is from.

Overall we did work with Data-Oriented Programming best practices. We were able to develop a scientific workflow. From the given data, we managed to train a model for prediction with decent results.

5 Further Works

As we drilled down this dataset, we kept getting new ideas that we would like to experiment with, and try to gain better insight. Specifically, our next steps would be:

5.1 Predict Different Sources

As it currently stands, we are trying to estimate a decade of speeches from “Althingi”. However, the dataset has several other sources of Icelandic; both written and spoken.

We would like to see if it was possible to extend our model to be able to classify the source.

5.2 Treating years as Contious Variables

We are currently treating decades as a class. By discretizing results from a regression algorithm, we think it should be possible to keep some nominal knowledge of the ordering of the years, and thus improving our predictions

5.3 Gaining insight into Explanatory Variables

From our results, it is clear that it is somewhat possible to predict the decades. However, we are still treating the algorithms as “Black Boxes”. We would like to dive deeper into the decision treas/boundaries, to see if we can locate what it is that makes the predictions possible. It might be new words introduced, semantic changes, or something entirely different.

5.4 Additional Feature Extraction and Classifiers

We would like to extend the list to include more classifiers, as well as trying to develop some additional feature extractions. E.g. “Glove Embedding”