Problem set 3: Feature expansion and word embeddings

NetID:

Name:

People with whom you discussed this problem set:

Did you consult AI concerning any aspect of this problem set (see the class AI policy)? Yes/No

---

Instructions for submission

1. Supply your name, NetID, and other information above.
2. Submit your completed work via CMS.
3. **Remember to execute every cell of code! Unexecuted code will receive zero credit.** The best way to make sure that everything is in order is to restart your kernel and run all cells immediately prior to submission.
4. Make sure to print all outputs with informative labels.
5. You will submit *both* this fully executed notebook *and* a PDF copy of it. TAs will grade the PDF copy. TAs may refer to or run the code as necessary, but they will not execute it to fill in missing outputs.
6. To generate a PDF version of the completed notebook, export the notebook to HTML, open the resulting HTML file in your browser, and print the rendered HTML to PDF from your browser. You may produce the PDF in other ways, but you are solely responsible for verfiying that it is correct and complete.

Part 1: Feature expansion

Summary and general instructions

Calculate and work with textual features beyond token unigram counts to predict volume publication dates.

In the first part of the assignment, we're going to work with **regression** rather than classification. This means that, rather than trying to predict a class label for each text, we'll try to estimate a continuous value for each text (in this case, the year when a given parliamentary debate occurred).

The general `sklearn` workflow is similar in regression and classification. You'll still create features using a vectorizer of some sort, you'll still set up a predictor object (now a regressor rather than a classifier), you'll still fit your predictor to your feature data, and you'll still produce a vector of predictions (now in the form of numbers rather than discrete labels). You can still score and cross-validate your results, but now by measuring the coefficient of determination, R^2, rather than F_1.

Imports and setup

Recall that you can install spaCy and the necessary models, if you haven't done so already, by running the following lines in a terminal window:

```
conda activate 3350
conda install spacy
python -m spacy download en_core_web_sm
python -m spacy download en_core_web_lg
```

In [1]:
```
# Imports. All here, none elsewhere.
import sqlite3
```

```
## Imports and setup
## All imports here, none in your code below
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from   sklearn.decomposition import LatentDirichletAllocation
import pyLDAvis
import pyLDAvis.lda_model
import warnings
from   sklearn.preprocessing import StandardScaler

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.cluster import KMeans, SpectralClustering, AgglomerativeClustering
from sklearn.datasets import make_blobs
from sklearn.decomposition import PCA, TruncatedSVD
from sklearn.metrics import accuracy_score, f1_score
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.feature_selection import SelectKBest,mutual_info_classif, f_classif,f_regression,mutua
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
```

1. Read data (5 points)

We will be working with the Canadian Hansard dataset, which includes transcripts of Canadian parliamentary debates from 1901 to 2019. In the `data/hansard/selected` folder in the course GitHub repo, there are 59 transcripts (one transcript from every second year in the indicated range). Each transcript has been truncated to the first 50k characters.

For this problem:

1. Read in the text of the selected transcripts.
2. Parse the file names to create gold-standard year labels.
3. Print the length of your data structures to confirm that they each contain 59 items.

In [2]:
```
import os
import pandas as pd

transcript_path = 'selected/'
data = []

for filename in os.listdir(transcript_path):
    if filename.endswith('.txt'):
        year = int(filename.split('_')[0])
        try:
            with open(os.path.join(transcript_path, filename), 'r', encoding='utf-8') as file:
                content = file.read()
        except UnicodeDecodeError:
            with open(os.path.join(transcript_path, filename), 'r', encoding='latin1') as file:
                content = file.read()

        data.append({'year': year, 'content': content})

df = pd.DataFrame(data)

print("Number of transcripts:", len(df))
print(df.head())
```

```
Number of transcripts: 59
   year                                              content
0  1901  I have the honour to inform the House that a v...
1  1903  Mr. LEONARD—by Mr. Ingram—asked :\n1.\tHow man...
2  1905  1.\tIs the government, or any of the ministers...
3  1909  Mr. HUGHES—by Mr. Geo. Taylor— asked:\n1.\tIs ...
4  1911  Bill (No. 214) respecting the Sault St. Louis ...
```

2. Simple vectorizer (10 points)

Begin by building a classifier that uses function-word frequencies as features.

1. Create a vectorizer that produces a feature matrix of **normalized** token counts for the **12 most frequently occurring** words in the corpus. (3 points)
   - Set `min_df = 0.5`.
   - Set `use_idf=False`.
   - Set other parameters however you deem appropriate (you will be asked to justify later).
2. Standard-scale the feature matrix. (1 point)
3. Print the resulting feature matrix shape, mean value, and feature names. (1 point)
   - Your feature matrix should have the shape (59, 12). This is our **baseline** case.
   - The mean should be close to zero.
4. Justify your use of each parameter in the vectorizer, including `min_df` and `use_idf`. (5 points)
   - Why have we set `min_df = 0.5` when we are limiting our features to the 12 most frequently occurring tokens in the corpus?

In [3]:
```
corpus = df['content'].tolist()

vectorizer = TfidfVectorizer(
    max_features=12,
    min_df=0.5,
    use_idf=False,
    norm='l2'
)

X = vectorizer.fit_transform(corpus).toarray()

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

print("Feature matrix shape:", X_scaled.shape)
print("Mean Value:", X_scaled.mean())
print("Feature names:", vectorizer.get_feature_names_out())
```

```
Feature matrix shape: (59, 12)
Mean Value: 1.5116595985009194e-16
Feature names: ['and' 'be' 'for' 'in' 'is' 'it' 'not' 'of' 'that' 'the' 'this' 'to']
```

Discussion here. `max_features=12`: We want to limit to 12 most frequently occurring words in the corpus. `min_df=0.5`: We want to only include words appearing in at least 50% of documents `use_idf=False`: Disable IDF weighting for baseline case could help us focus on high-frequency words. `norm='l2'`:l2 norms will scale each element's vector length to 1 to reduce biased influence.

3. Simple linear regression (5 points)

Use a linear regressor to predict the date of each parliamentary debate in the corpus.

1. Train your regressor on your scaled input feature matrix and your gold labels, then use the fitted regressor to predict new *y* values given the same input. (3 points)

- This will provide a misleading sense of predictive performance, because training and testing on the same data encourages overfitting. We'll deal with this problem below.

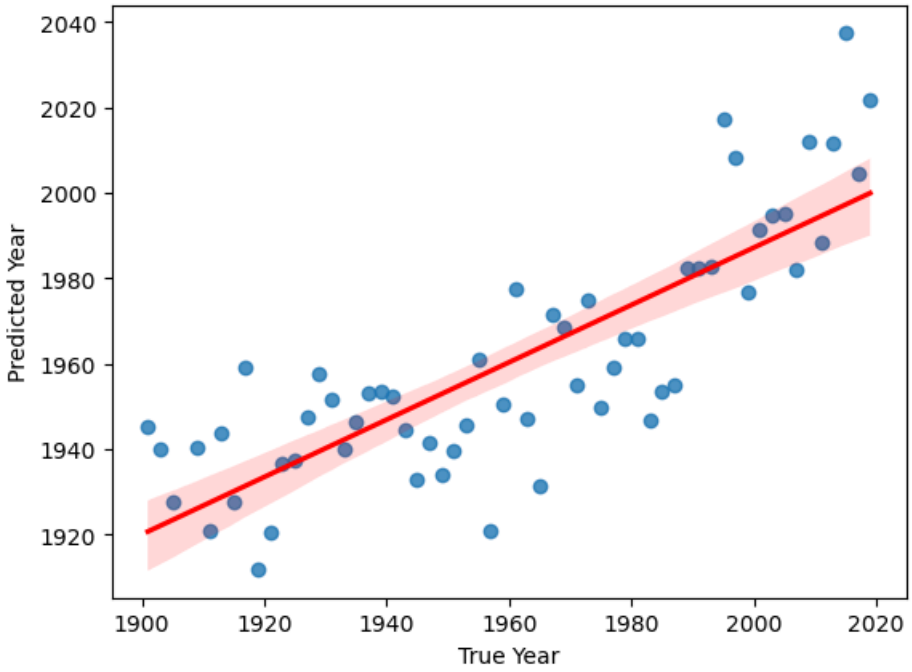
2. Plot your predicted publication dates as a function of the true publication dates, including a line of best fit and a visual confidence region. (2 points).
   - Hint: consult Seaborn.

In [4]:
```
# code here
y = df['year'].values

regressor = LinearRegression()
regressor.fit(X_scaled, y)

y_pred = regressor.predict(X_scaled)

sns.regplot(x=y, y=y_pred, ci=95, line_kws={"color": "red"})
plt.xlabel("True Year")
plt.ylabel("Predicted Year")
plt.show()
```



4. Score your baseline regressor (10 points)

1. Score your trained regressor on the scaled input feature matrix and gold standard values. This calculates R^2, the coefficient of determination, which is an appropriate scoring metric for a regression problem. **Print your score.** It will be somewhere near 0.7 (2 points)
2. Calculate a proper, non-overfitted **mean** R^2 across all folds with `cv=5`. **Print your result.** It will be lower than the previous score (3 points)
3. We will calculate these same two scores several more times (using new feature data) in subsequent questions. Wrap up the calculations as a function, `compare_scores`, that takes a feature matrix and a vector of gold values, fits a `LinearRegression` object, and **prints both versions of the score, naïve and mean five-fold cross validated**. Print your outputs with appropriately limited precision (that is, to a limited number of decimal places). (5 points)
   - It is expected that the `compare_scores` function prints both versions of the R^2 score; therefore, it is expected that both scores will be printed for subsequent questions where `compare_scores` is requested.
4. Call this function on your data to confirm that it works and that it produces the same results as the ones you've just calculated.

Note: We have received a small number of reports of large negative R^2 values from code that appears to be correct. This issue may affect the cross-validation scores in problems 4-8. We are investigating the issue and will take it into account during grading.

In [5]:
```
# code here
regressor = LinearRegression()
regressor.fit(X, y)
naive_score = regressor.score(X, y)

# Perform 5-fold cross-validation to get the mean cross-validated R^2 score
cv_scores = cross_val_score(regressor, X, y, cv=5, scoring='r2')
mean_cv_score = cv_scores.mean()
print(f"Test R^2 score: {naive_score:.3f}")
print(f"Test Mean 5-fold cross-validated R^2 score: {mean_cv_score:.3f}")

def compare_scores(X, y):
    regressor = LinearRegression()
    regressor.fit(X, y)
    naive_score = regressor.score(X, y)

    cv_scores = cross_val_score(regressor, X, y, cv=5, scoring='r2')
    mean_cv_score = cv_scores.mean()

    # It is expected that the compare_scores function prints both versions of thescore;
    # therefore, it is expected that both scores will be printed for subsequent questions where com
    print(f"Naïve R^2 score: {naive_score:.3f}")
    print(f"Mean 5-fold cross-validated R^2 score: {mean_cv_score:.3f}")

# Call this function on your data to confirm that it works and that it produces the same results as
compare_scores(X_scaled, y)
```

```
Test R^2 score: 0.672
Test Mean 5-fold cross-validated R^2 score: -25.417
Naïve R^2 score: 0.672
Mean 5-fold cross-validated R^2 score: -25.417
Cross-validated R^2 scores per fold: [-43.71837551 -10.33415542 -26.21039865 -22.54512803 -24.27674
386]
```

5. Lemmas (10 points)

1. Write a function, `lemmatizer`, that uses spaCy to lemmatize an input string. (4 points)
   - This function should take an arbitrary string of text and an initialized spaCy nlp object as input and return a list of lemmatized tokens. By default, this function should use the `en_core_web_sm` spacy nlp object.
   - Your lemmatizer should **also** remove punctuation and any "pure" whitespace tokens (look out in particular for `\n\n` pseudo-tokens).
2. Check if your lemmatizer function works by running it on the sample string below. **Print the output of your lemmatizer function when given the input string**. (1 point)
   - Test input string:
     `'''Her cats are \n\n   dancing faster, than the tallest dogs.'''`
   - Your output shoud be:
     `['her', 'cat', 'be', 'dance', 'fast', 'than', 'the', 'tall', 'dog']`
3. Use your lemmatizer function in a new version of your initial vectorizer to produce a feature matrix of the 12 most frequently occurring lemmas in the corpus. (3 points)
4. Scale the resulting features. (1 point)
5. Finally, use your `compare_scores` function to fit a linear regressor on the scaled lemma features and report both the "naïve" and the cross-validated R^2 scores, calculated as in the previous question. (1 point)
   - Your R^2 values should be lower than those calculated in the previous problem.

FYI, vectorization with the lemmatizer takes about a minute on my computer.

In [6]:
```
# code here

nlp = spacy.load("en_core_web_sm")
```

```
def lemmatizer(text, nlp=nlp):
    doc = nlp(text)
    lemmas = [
        token.lemma_.lower()
        for token in doc
        if not token.is_punct and not token.is_space
    ]
    return lemmas
def custom_tokenizer(text):
    return lemmatizer(text, nlp=nlp)

# Test input string
test_string = '''Her cats are \n\n   dancing faster, than the tallest dogs.'''
print("Lemmatized output:", lemmatizer(test_string))

#Use your lemmatizer function in a new version of your initial vectorizer
vectorizer = TfidfVectorizer(
    max_features=12,
    min_df=0.5,
    use_idf=False,
    norm='l2',
    tokenizer=custom_tokenizer
)

X_lemma = vectorizer.fit_transform(df['content']).toarray()
scaler = StandardScaler()
X_lemma_scaled = scaler.fit_transform(X_lemma)

# Run compare_scores on the scaled lemma-based features
compare_scores(X_lemma_scaled, y)
```

```
Lemmatized output: ['her', 'cat', 'be', 'dance', 'fast', 'than', 'the', 'tall', 'dog']
C:\Users\kewei\anaconda3\envs\3350\Lib\site-packages\sklearn\feature_extraction\text.py:521: UserWa
rning: The parameter 'token_pattern' will not be used since 'tokenizer' is not None'
  warnings.warn(
Naïve R^2 score: 0.553
Mean 5-fold cross-validated R^2 score: -28.806
Cross-validated R^2 scores per fold: [-47.95413062  -6.95258126 -23.29386831 -34.04897712 -31.78017
57 ]
```

6. Entities and POS tags (10 points)

1. Use spaCy to count the number of entities and the number of tokens tagged with parts of speech of the types indicated in the stub code below. (5 points)
   - Use the `en_core_web_sm` model.
   - This will require about the same amount of runtime as did the previous, lemmatized vectorization
2. Transform your counts into a feature matrix. **Print the shape of this matrix** (it should be `(59, 12)`). (3 points)
3. Standard-scale the resulting matrix (1 point)
4. Calculate and print both a naïve and a cross-validated R^2 value for the linear regression prediction of transcript date from these features. (1 point)

In [7]:
```
#code here
valid_ent = ['PERSON', 'MONEY', 'TIME']
valid_pos = ['ADJ', 'ADV', 'AUX', 'CCONJ', 'DET', 'NOUN', 'PRON', 'PROPN', 'VERB']

nlp = spacy.load("en_core_web_sm")

feature_counts = []
for doc_text in df['content']:
    doc = nlp(doc_text)
    counts = []

    for entity_type in valid_ent:
        counts.append(sum(1 for ent in doc.ents if ent.label_ == entity_type))
```

```
    for pos_tag in valid_pos:
        counts.append(sum(1 for token in doc if token.pos_ == pos_tag))

    feature_counts.append(counts)

X_entity_pos = np.array(feature_counts)
print("Feature matrix shape:", X_entity_pos.shape)

#Standard-scale the resulting matrix
scaler = StandardScaler()
X_entity_pos_scaled = scaler.fit_transform(X_entity_pos)

compare_scores(X_entity_pos_scaled, y)
```

```
Feature matrix shape: (59, 12)
Naïve R^2 score: 0.826
Mean 5-fold cross-validated R^2 score: -14.092
Cross-validated R^2 scores per fold: [-25.19229848  -4.53438366 -16.35293391 -10.88355557 -13.49771
685]
```

7. Combined features (3 points)

1. Combine your scaled, lemmatized features with your scaled, entity/POS features to produce a single feature matrix with shape `(59, 24)`. Print the shape of the resulting matrix. (2 points)
2. Calculate naïve and cross-validated R^2 scores for the same predition task as in the previous problems, using the combined features as input. (1 point)

In [8]:
```
# code here
X_combined = np.hstack((X_lemma_scaled, X_entity_pos_scaled))
print("shape of the resulting matrix:", X_combined.shape)
compare_scores(X_combined, y)
```

```
Combined feature matrix shape: (59, 24)
Naïve R^2 score: 0.893
Mean 5-fold cross-validated R^2 score: -17.677
Cross-validated R^2 scores per fold: [-31.3680591   -8.44316445  -7.74330253 -10.63128761 -30.20134
187]
```

8. Once more, from the top (10 points)

Let's reset and see if we can get better performance.

1. Vectorize again, this time lemmatizing (keep all lemmas that occur in 2 or more documents, without a max on the number of retained features), and also using IDF weighting. (1 point)
2. Scale your features. (1 point)
3. Print the shape of the resulting feature matrix. (1 point)
4. Select the 20 most-informative features as scored by `mutual_info_regression`. (1 point)
5. Use `compare_scores` (twice) to print the naïve and cross-validated R^2 scores for a linear regressor on **first** the full set of features and **then** on the selected features. (1 point)
6. Explain the results of the naïve and cross-validated scoring on the full features set and on the selected features (5 points)

In [19]:
```
# code here
def lemmatizer_8(text, nlp=nlp):
    doc = nlp(text)
    lemmas = [
        token.lemma_.lower()
        for token in doc
        if not token.is_punct and not token.is_space
    ]
    return lemmas

def custom_tokenizer(text):
    return lemmatizer_8(text, nlp=nlp)
```

```
#Vectorize again, this time lemmatizing (keep all lemmas that occur in 2 or more documents, without
vectorizer = TfidfVectorizer(
    min_df=2,
    use_idf=True,
    tokenizer=custom_tokenizer

)
X_lemmatized = vectorizer.fit_transform(df['content']).toarray()

#Scale your features.
scaler = StandardScaler()
X_lemmatized_scaled = scaler.fit_transform(X_lemmatized)
#Print the shape of the resulting feature matrix.
print("Feature matrix shape after vectorization:", X_lemmatized.shape)

#Select the 20 most-informative features as scored by mutual_info_regression
mi_scores = mutual_info_regression(X_lemmatized_scaled, y)
top_20_indices = np.argsort(mi_scores)[-20:]
X_lemmatized_top20 = X_lemmatized_scaled[:, top_20_indices]

print("Scores for first full feature set:")
compare_scores(X_lemmatized_scaled, y)
print("\nScores for top 20 features set:")
compare_scores(X_lemmatized_top20, y)
```

```
C:\Users\kewei\anaconda3\envs\3350\Lib\site-packages\sklearn\feature_extraction\text.py:521: UserWa
rning: The parameter 'token_pattern' will not be used since 'tokenizer' is not None'
  warnings.warn(
Feature matrix shape after vectorization: (59, 6863)
Scores for first full feature set:
Naïve R^2 score: 1.000
Mean 5-fold cross-validated R^2 score: -17.334
Cross-validated R^2 scores per fold: [-35.8646536   -2.39295862  -0.4010737   -4.52018942 -43.49321
666]

Scores for top 20 features set:
Naïve R^2 score: 0.906
Mean 5-fold cross-validated R^2 score: -11.234
Cross-validated R^2 scores per fold: [ -6.20674799  -7.72237138  -6.1019817   -4.83495169 -31.30471
197]
```

Your explanation of the results here.

Extra credit (3 points, optional)

Note that we've been working with just 59 transcripts from a total of 586. The full set of transcripts can be found in `data/hansard/all` on GitHub.

Will you be able to get better performance with more transcripts? For optional extra credit, try playing around with different input sizes to improve performance. If the computing times are getting too long, you may truncate the transcripts into even smaller excerpts. Display and discuss your results.

In [10]:
```
# optional extra credit code here
```

Part 2: Word embeddings and sentiment

Summary and general instructions

Predict the polarity of album reviews from *Pitchfork*, 1999-2017, using token features and word embeddings.

In part 2, we're back to classification, though the problem might easily be transformed into a regression task. We'll examine a corpus of more than 18,000 album reviews published between 1999 and 2017 on the music site *Pitchfork*. Our dataset includes the full text of each review, along with a numerical rating of the album in question (between 0

and 10). We'll classify a review as positive if its score is above the corpus mean and negative if the score is below the corpus mean.

Data

Our data are stored in a sqlite database that you can download from the class GitHub repo (in `data/reviews`). We'll use the `sqlite3` and `pandas` packages to read the data and get it ready for use.

NOTE: SQLite is a lightweight way to manage and retrieve data from a database. This course won't cover SQL or SQLite, but if you'd like to learn more you might review the Python documentation for executing SQLite in Python and an intro to SQL syntax.

In [11]:
```
# Data location
db_file = os.path.join( 'database.sqlite')

# Set up sqlite connection
conn = sqlite3.connect(db_file)

# Read score data
scores =  pd.read_sql_query("""SELECT * FROM reviews""", conn)

# Read review content
reviews = pd.read_sql_query("""SELECT * FROM content""", conn)

# Close the connection
conn.close()

# Examine our data
display(scores.head())
display(reviews.head())
```

	reviewid	title	artist	url	score	best_new_music	author	auth
0	22703	mezzanine	massive attack	http://pitchfork.com/reviews/albums/22703-mezz...	9.3	0	nate patrin	co
1	22721	prelapsarian	krallice	http://pitchfork.com/reviews/albums/22721-prel...	7.9	0	zoe camp	co
2	22659	all of them naturals	uranium club	http://pitchfork.com/reviews/albums/22659-all-...	7.3	0	david glickman	co
3	22661	first songs	kleenex, liliput	http://pitchfork.com/reviews/albums/22661-firs...	9.0	1	jenn pelly	a
4	22725	new start	taso	http://pitchfork.com/reviews/albums/22725-new-...	8.1	0	kevin lozano	co

	reviewid	content
0	22703	“Trip-hop” eventually became a ’90s punchline,...
1	22721	Eight years, five albums, and two EPs in, the ...
2	22659	Minneapolis’ Uranium Club seem to revel in bei...
3	22661	Kleenex began with a crash. It transpired one ...
4	22725	It is impossible to consider a given release b...

There are other tables in the database. See the dataset documentation (linked above) for more information. But these two tables are the only ones you need in order to complete the problem set.

9. Build a minimal dataframe and remove bad data (5 points)

We only want two of the columns in the data (review text and album score), and we know that there are some empty fields than can cause downstream problems.

1. Merge the two dataframes (`scores` and `reviews`) by the shared `reviewid` , retaining only the `score` and `content` columns. Set `reviewid` as index. Eliminate duplicate entries, null values, and any review that is less than 1,000 characters (not words!) long. (3 points)
2. Retain only a 10% sample of the resulting data, being careful not to duplicate any rows. Set the random state of your sampler to `40` . You will work with this data for the remaining problems. (1 point)
3. Call the `.describe()` method on the resulting sampled dataframe. (1 point)

In [12]:
```
# code here
merged_data = pd.merge(scores, reviews, on='reviewid')
minimal_df = merged_data.set_index('reviewid')[['score', 'content']]

# Eliminate duplicate entries, null values, and any review that is less than 1,000 characters (not
minimal_df = minimal_df.drop_duplicates()
minimal_df = minimal_df.dropna()
minimal_df = minimal_df[minimal_df['content'].str.len() >= 1000]

# Retain only a 10% sample of the resulting data
sample_df = minimal_df.sample(frac=0.1, random_state=40)

# Display
print(sample_df.describe())
```

```
             score
count  1836.000000
mean      7.014651
std       1.271322
min       1.000000
25%       6.400000
50%       7.200000
75%       7.800000
max      10.000000
```

10. Create gold labels (5 points)

1. Create a vector of gold labels for your album scores. Your label should be `1` when the score is above the mean score in the data set, `0` otherwise. **Print the length of your vector.** (2 points)
2. Calculate the fraction of all albums in the data set that scored above the mean and **print your result**. (1 point)
   - This is your naïve baseline accuracy. Any useful classifier must achieve an accuracy score higher than this value.
3. Explain what the fraction of albums that scored above the mean implies about the distribution of review scores in the data set (2 points)

In [13]:
```
# code here
mean = sample_df['score'].mean()
gold_labels = sample_df['score'].apply(lambda x: 1 if x > mean else 0)
print("Length of gold labels vector:", len(gold_labels))
above_mean =  gold_labels.sum()/len(sample_df)
print("naïve baseline accuracy:", above_mean)
```

```
Length of gold labels vector: 1836
naïve baseline accuracy: 0.5599128540305011
```

Discussion here

It means that in this dataset, more than half of review have scores larger than mean, which might contribute to a left-skewed distribution, and median should be higher than mean.

11. Build a token-based classifier (6 points)

1. Vectorize the review texts, then **print the shape of the resulting feature matrix**. (1 point)

- Set `min_df = 0.01`.
- Set `max_df = 0.9`.
- Set `use_idf=True`.
- Set other parameters however you deem appropriate.

2. Standard-scale your features to mean 0 and variance 1. (1 point)
3. Select the 300 most informative features using the `mutual_info_classif` criterion. (1 point)
4. Calculate a 10-fold cross-validated accuracy score using a logistic regression classifier on your selected feature data. **Print the (averaged) cross-validated accuracy score.** (3 points)

In [17]:
```
# code here

# Suppress warnings temporarily if necessary
warnings.filterwarnings("ignore", category=UserWarning, module='sklearn.metrics.cluster._supervised

vectorizer = TfidfVectorizer(
    min_df=0.01,
    max_df=0.9,
    use_idf=True
)

X_token = vectorizer.fit_transform(sample_df['content'])
print("Feature matrix shape after vectorization:", X_token.shape)

scaler = StandardScaler(with_mean=False)
X_token_scaled = scaler.fit_transform(X_token)

selector = SelectKBest(mutual_info_classif, k=300)
X_token_selected = selector.fit_transform(X_token_scaled, gold_labels)

log_reg = LogisticRegression(max_iter=10000)
cross_val_scores = cross_val_score(log_reg, X_token_selected, gold_labels, cv=10, scoring='accuracy
average_accuracy = cross_val_scores.mean()

print("10-fold cross-validated accuracy:", average_accuracy)
```

```
Feature matrix shape after vectorization: (1836, 5641)
10-fold cross-validated accuracy: 0.6312574245664054
```

12. Build a word-embedding-based classifier (15 points)

1. Write a function, `get_doc_embedding`, that takes as arguments a text string and a spaCy NLP factory object (use `en_core_web_lg`) and returns a word embedding vector that is the average of the input text's token vectors. (10 points)

   - Your function should remove any token that spaCy tags as `is_stop`, `is_punct`, or `is_space`, as well as any token that does not have an embedding vector (`has_vector` is false).
   - Verify that your function performs as expected. Given the input string `"The bad cat is drinking my coffee."` your function should return a vector of length 300 that begins:

   ```
   array([ 0.16729254, -0.35514995, -2.967725  , -3.2744699 ,  0.55169547, ...
   ```

   This vector is the mean of the embeddings for the tokens `[bad, cat, drinking, coffee]`.

2. Pass the text of each review to your `get_doc_embedding` function and store the output as a row in a numpy array of appropriate shape. This is your feature matrix. Calculating it may be slow (3 minutes on my computer using a non-optimized approach). If necessary, you may want to do your development on a subset of the data, but you must eventually use the full 10% sample of reviews. **Print the shape of the resulting matrix.** (3 points)

3. Compute a 10-fold cross-validated logistic regression classification accuracy using the new features as input. **Print this score.** (2 points)

   - If you encounter a `ConvergenceWarning` during cross validation, increase `max_iters` in your classifier.

In [18]:
```
# code here
nlp = spacy.load("en_core_web_lg")

# Write a function, get_doc_embedding, that takes as arguments a text string and a spaCy NLP factor
# and returns a word embedding vector that is the average of the input text's token vectors.
def get_doc_embedding(text, nlp=nlp):
    doc = nlp(text)
    valid_vectors = [
        token.vector for token in doc
        if not (token.is_stop or token.is_punct or token.is_space) and token.has_vector
    ]
    if valid_vectors:
        return np.mean(valid_vectors, axis=0)
    else:
        return np.zeros(nlp.vocab.vectors_length)

# Verify that your function performs as expected
test_sentence = "The bad cat is drinking my coffee."
test_embedding = get_doc_embedding(test_sentence)
print(" the mean of the embeddings for the tokens:", test_embedding)
print("Embedding length:", len(test_embedding))

#Pass the text of each review to your get_doc_embedding function and store the output as a row in a
review_embeddings = np.array([get_doc_embedding(text) for text in sample_df['content']])
print(" shape of the resulting matrix:", review_embeddings.shape)

# Compute a 10-fold cross-validated logistic regression classification accuracy using the new featu
log_reg = LogisticRegression(max_iter=10000)
cross_val_scores = cross_val_score(log_reg, review_embeddings, gold_labels, cv=10, scoring='accurac
average_accuracy = cross_val_scores.mean()
print("10-fold cross-validated accuracy:", average_accuracy)
```

```
Test embedding: [ 0.16729254 -0.35514995 -2.967725   -3.2744699   0.55169547]
Embedding length: 300
Feature matrix shape: (1836, 300)
10-fold cross-validated accuracy: 0.6601063197909243
```

13. Improve classification performance and comment on your results (20 points)

Note that our classification accuracy isn't very good relative to baseline!

- Experiment with both the token-based and the embedding-based classification tasks in order to improve classification accuracy. Anything is fair game. **Show your work and clearly display the cross-validated score of your best-performing classifer for each feature type.** (5 points for improved accuracy)
- Comment on your experimental process and results. What did you try? What improved accuracy? What didn't? By how much were you able to improve? Why do you think your best approach out-performed the others? (15 points)

Note that iterating on the full dataset may be slow. It's fine to do your development work on a subset of the data, but be sure to present your best cross-validated scores as run on the full dataset. It's possible, of course, that a classifier that performs well on a random subset of the full data may not perform as well on the full dataset.

Suggested approaches

There are three basic areas to examine when you're trying to improve classification accuracy:

1. **Input data and features.** Can you engineer or select better features?
2. **Classification algorithm.** Have you chosen the classifier best suited to your problem?
3. **Classifier parameters.** Have you found the parameters that produce the highest accuracy for your classifier and data?

Logistic regression with default parameters run on TF-IDF weighted token features is a non-crazy baseline for many document classification tasks. But it is unlikely to produce the highest possible accuracy in most cases. Here are a few ideas to explore:

- *N*-gram features
- Metadata and non-text features
- Stylistic and NLP-based features
- Recursive feature elimination or exploration of different *k* values for `SelectKBest`
  - Pay particular attention to whether you are underfit or overfit. If you're underfit, adding more features will improve cross-validated accuracy. If you're overfit, it won't (and may lower cross-validated accuracy).
- Scikit-learn makes it very easy to swap classifiers. Consult the documentation for options.
- Use `GridSearchCV` to explore parameter space
- ...

Two notes:

1. It is not a valid aproach to try different *scoring* methods or different numbers of cross-validation folds. These amount to moving the goalposts rather than improving your classification system. Yes, you might want to use a scoring metric that's most appropriate to your problem (accuracy might not be best), but you need to use the same scoring metric when you compare baseline performance to your revised system.
2. The classifier that performs best "out of the box" (using default parameters and features) may not be the one that performs best after feature engineering and tuning. Improving classificatin performance is an iterative process. Do not select just one classifier with which to work from the off.

In [ ]:
```
# your code here
```

Comment on your experimental process and results here.