Automatic Scoring with Machine Learning

This notebook explores classic machine learning algorithms with vectorized features from the student essays.

In [1]:
```
# Import modules and setup notebook
%matplotlib inline

import numpy as np
import pandas as pd
import re
from datetime import datetime
```

In [2]:
```
# Plotting libraries
import matplotlib.pyplot as plt
from matplotlib import cm
import seaborn as sns
plt.rcParams['figure.dpi']= 100
```

In [3]:
```
# Text processing
import spacy
from spacy.lang.en.stop_words import STOP_WORDS
```

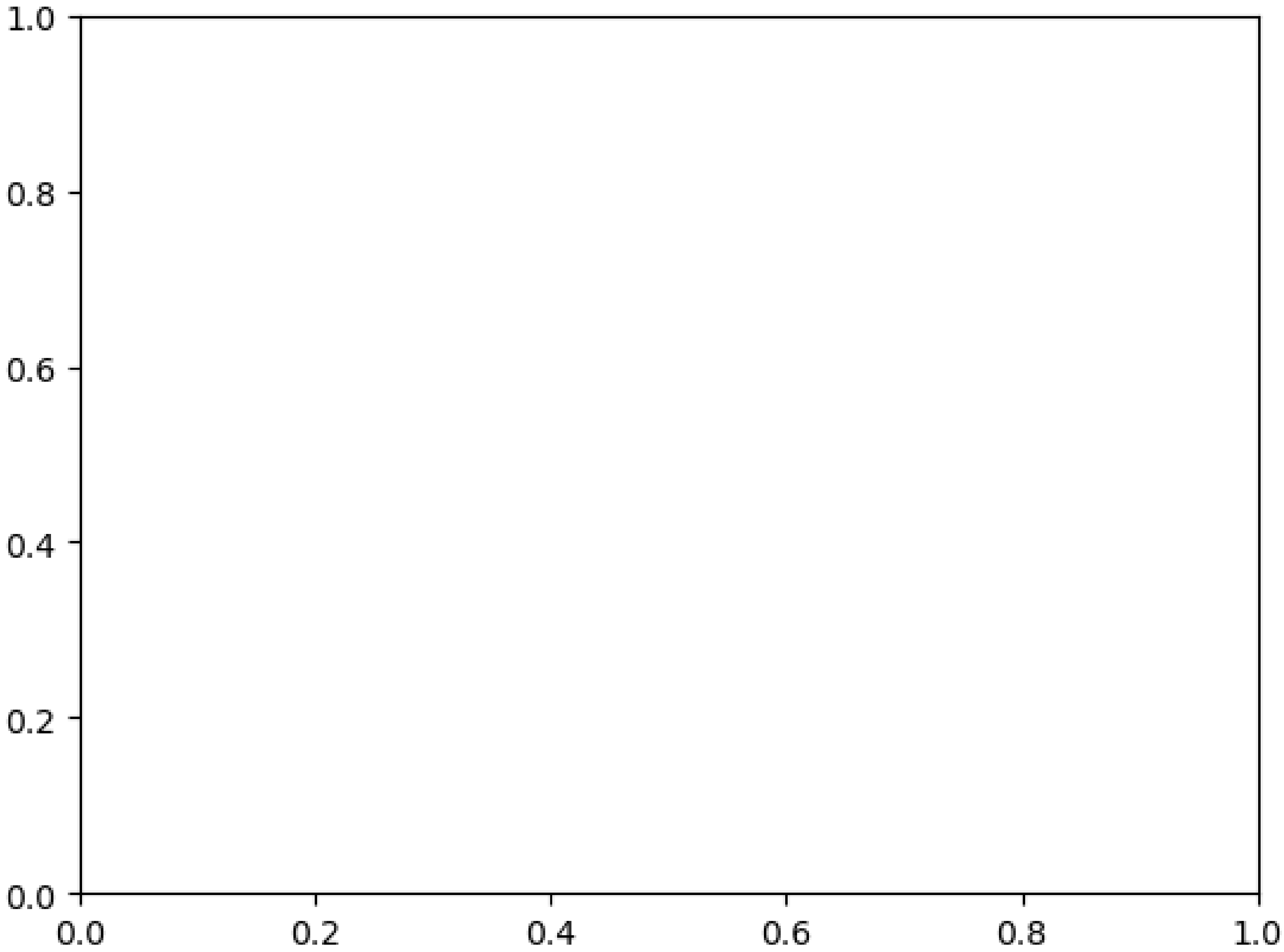
In [4]:
```
# Machine learning libraries
from sklearn import metrics
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import ElasticNet, LinearRegression
from sklearn.svm import LinearSVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.model_selection import GridSearchCV, train_test_split, cross_val_score
```

```
from sklearn.ensemble import ExtraTreesClassifier, RandomForestRegressor
from sklearn.feature_selection import SelectKBest, f_classif, f_regression
```

In [13]:
```
# kappa metric for measuring agreement of automatic to human scores
from skll.metrics import kappa
#from bhkappa import mean_quadratic_weighted_kappa

#Ref: https://github.com/benhamner/Metrics/blob/master/Python/ml_metrics/quadratic_weighted_kapp
def mean_quadratic_weighted_kappa(kappas, weights=None):
    """
    Calculates the mean of the quadratic
    weighted kappas after applying Fisher's r-to-z transform, which is
    approximately a variance-stabilizing transformation.  This
    transformation is undefined if one of the kappas is 1.0, so all kappa
    values are capped in the range (-0.999, 0.999).  The reverse
    transformation is then applied before returning the result.

    mean_quadratic_weighted_kappa(kappas), where kappas is a vector of
    kappa values

    mean_quadratic_weighted_kappa(kappas, weights), where weights is a vector
    of weights that is the same size as kappas.  Weights are applied in the
    z-space
    """
    kappas = np.array(kappas, dtype=float)
    if weights is None:
        weights = np.ones(np.shape(kappas))
    else:
        weights = weights / np.mean(weights)

    # ensure that kappas are in the range [-.999, .999]
    kappas = np.array([min(x, .999) for x in kappas])
    kappas = np.array([max(x, -.999) for x in kappas])

    z = 0.5 * np.log((1 + kappas) / (1 - kappas)) * weights
```

```
    z = np.mean(z)
    return (np.exp(2 * z) - 1) / (np.exp(2 * z) + 1)
```

In [15]:
```
# Configure plotting style
palette = sns.color_palette("colorblind")
plt.gca().set_prop_cycle('color', palette)

# Setup Pandas
pd.set_option('display.width', 500)
pd.set_option('display.max_columns', 100)
pd.set_option('display.notebook_repr_html', True)
pd.set_option('display.max_colwidth', 100)
```



In [16]:
```
# Read essay data processed in previous notebook
training_set = pd.read_pickle('training_spacy.pkl')
```

In [17]:
```
training_set[['lemma', 'pos', 'ner']].sample(3)
```

Out[17]:

	lemma	pos	ner
12506	[the, @CAPS1, be, @CAPS2, @CAPS3, great, -, grandmother, once, tell, I, that, laughing, add, and...	[DET, PROPN, AUX, PROPN, PROPN, ADJ, PUNCT, NOUN, ADV, VERB, PRON, SCONJ, NOUN, VERB, CCONJ, ADJ...	[@CAPS1, five years, two, @CAPS3, first, one, two, one, eighteen, @CAPS7, @LOCATION1, about a we...
9947	[Marcia, @CAPS1, 's, article, ", the, Mooring, Mast, ", explain, the, obstacle, the, builder, of...	[PROPN, PROPN, PART, NOUN, PUNCT, DET, PROPN, PROPN, PUNCT, VERB, DET, NOUN, DET, NOUN, ADP, DET...	[Marcia @CAPS1's, The Mooring Mast, the Empire State Building, a thousand-foot, The Steel, sixty...
11865	[have, you, ever, babysitter, little, kid, ?, if, you, have, ,, then, you, know, how, much, pati...	[AUX, PRON, ADV, VERB, ADJ, NOUN, PUNCT, SCONJ, PRON, VERB, PUNCT, ADV, PRON, VERB, ADV, ADJ, NO...	[third, first, one]

Generate vectorized features from processed essays

A document similarity metric is available from *SpaCy*. In order to make use of it for essay scoring, we need to define a reference. Choosing an average, middle-scoring or aggregate essay would leave the sign of the difference undetermined: If the similarity is worse, does that mean the essay is better or worse? Choosing a low scoring essay would theoretically work, however many of the low scoring essays are very short and are full of spelling and grammatical errors. A high scoring essay would be best, though there is another point to consider. When an essay is written in a unique style, how will it compare? Since there are relatively few high scoring essays, the selection was performed manually and arbitrarily under consideration of a "representative style".

The selection process remains highly subjective.

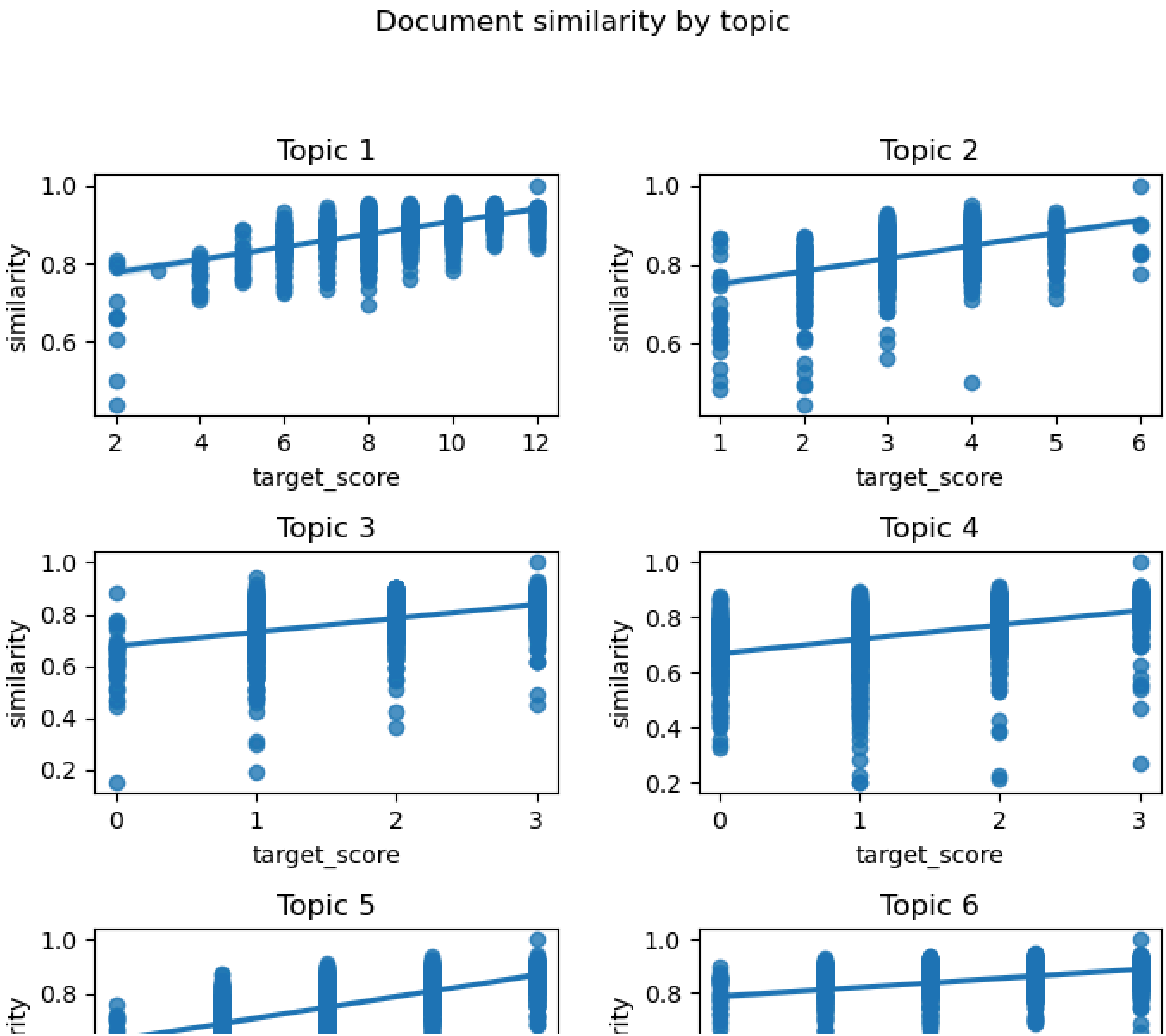
In [19]:

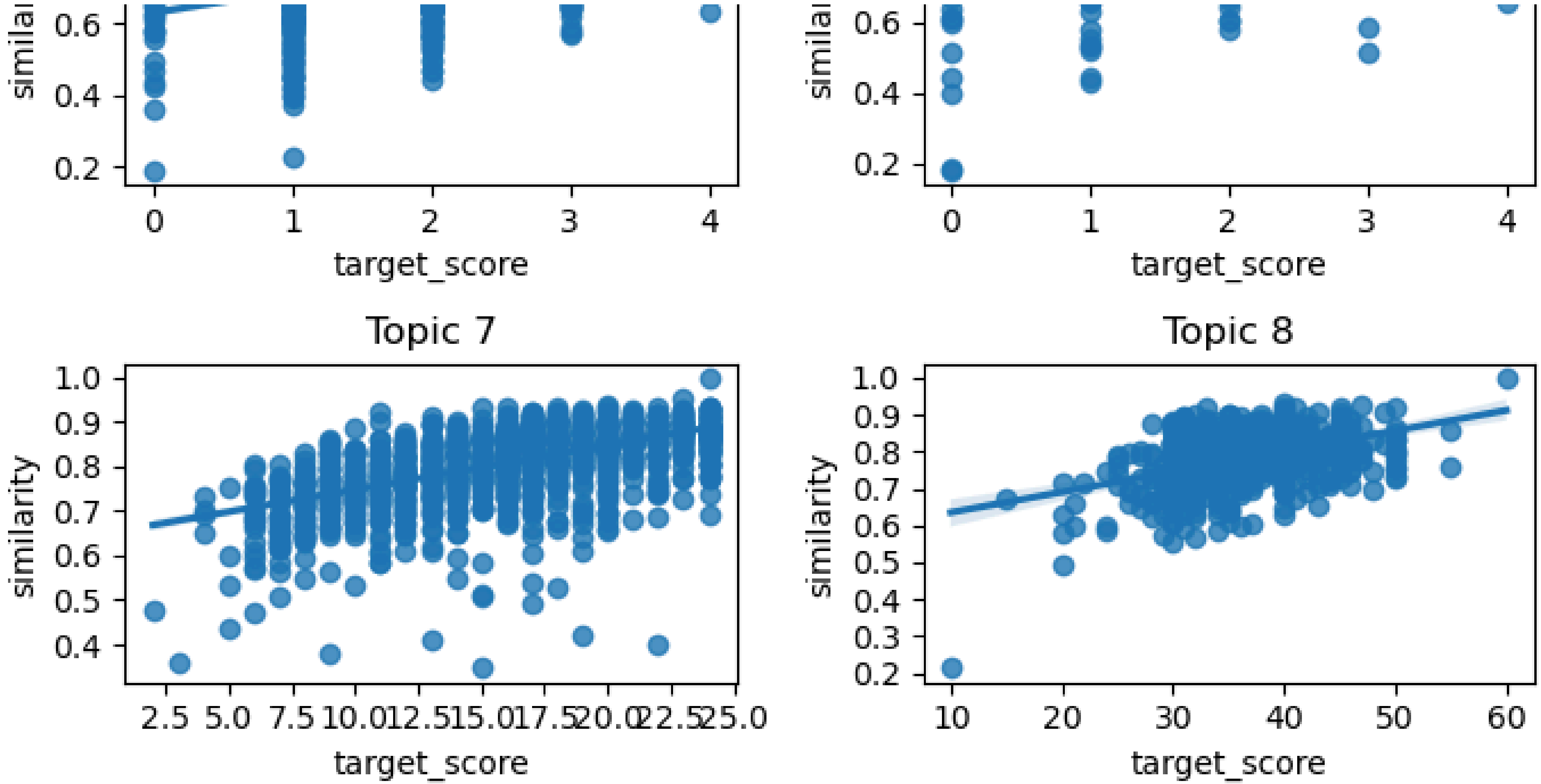
```
"""Choose arbitrary essay from highest available target_score for each topic.
all other essays will be compared to these.
The uncorrected essays will be used since the reference essays should have fewer errors.
```

```
"""
reference_essays = {1: 161, 2: 3022, 3: 5263, 4: 5341, 5: 7209, 6: 8896, 7: 11796, 8: 12340} # t

references = {}

t0 = datetime.now()

nlp = spacy.load('en_core_web_sm')
stop_words = set(STOP_WORDS)

# generate nlp object for reference essays:
for topic, index in reference_essays.items():
    references[topic] = nlp(training_set.iloc[index]['essay'])

# generate document similarity for each essay compared to topic reference
training_set['similarity'] = training_set.apply(lambda row: nlp(row['essay']).similarity(referen

t1 = datetime.now()
print('Processing time: {}'.format(t1 - t0))
```

```
/opt/anaconda3/lib/python3.9/site-packages/spacy/util.py:910: UserWarning: [W095] Model 'en_core_
web_sm' (3.0.0) was trained with spaCy v3.0.0 and may not be 100% compatible with the current ver
sion (3.7.4). If you see errors or degraded performance, download a newer compatible model or ret
rain your custom model with the current spaCy version. For more details and available updates, ru
n: python -m spacy validate
  warnings.warn(warn_msg)
/var/folders/f4/df90mp5j7bn59dwnfp302yq80000gn/T/ipykernel_59905/4158347446.py:19: UserWarning:
[W007] The model you're using has no word vectors loaded, so the result of the Doc.similarity met
hod will be based on the tagger, parser and NER, which may not give useful similarity judgements.
This may happen if you're using one of the small models, e.g. `en_core_web_sm`, which don't ship
with word vectors and only use context-sensitive tensors. You can always add your own word vector
s, or use one of the larger models instead if available.
  training_set['similarity'] = training_set.apply(lambda row: nlp(row['essay']).similarity(refere
nces[row['topic']]), axis=1)
Processing time: 0:05:56.401868
```

In [20]:
```
# Plot document similarity vs target score for each topic
topic_number = 0
fig, ax = plt.subplots(4,2, figsize=(7,10))
for i in range(4):
    for j in range(2):
        topic_number += 1
        sns.regplot(x='target_score', y='similarity', data=training_set[training_set['topic'] ==
        ax[i,j].set_title('Topic %i' % topic_number)
ax[3,0].locator_params(nbins=10)
ax[3,1].locator_params(nbins=10)
plt.suptitle('Document similarity by topic')
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.savefig('image5.png', dpi=300)
plt.show()
```


Document similarity by topic
Topic 1
Topic 2
Topic 3
Topic 4
Topic 5
Topic 6
similarity
target_score



Document similarity may prove to be ineffective for persuasive/narritive essays. The example below shows the highest scored essay for topic 8. The author uses a unique creative style which is unlikely to be replicated.

In [21]:
```
training_set.iloc[12340]['essay']
```

Out[21]:
```
' Bell rings.  Shuffle, shuffle. @CAPS1. Snap. EEEE. Crack. Slam. Click, stomp, @CAPS1. Tap tap
tap. SLAM. Creak. Shoof, shoof.  Sigh. Seventh class of the day. Here we go. "@CAPS2! Tu va ou
pas? On a +¬tude cette class-l+á. Tu peux aller au bibliotheque si tu veux...." @CAPS3 all blin
ked at me, @PERSON1, @NUM1le and @ORGANIZATION1, chocolate-haired and mocha skinned, impatientl
y awaiting my answer. The truth was, I knew @CAPS3 didn\'t really care if I came or not. It mad
e no difference to them if I trailed a few feet behind like some pathetic puppy. I was silent b
ut adorable, loved only because I was an @CAPS4. Because I spoke fidgety @CAPS5. Because I was
the exchange student, because my translator and colorful clothes were so shocking for ten secon
ds, and were then forgotten about.  I was a flock of seagulls haircut. So why are you here? I t
hought. Why did you go on exchange at all? You are the complete opposite of everyone here. No o
ne wants you. Just go home.  But my ego had a ready answer. You begged for this remember? For m
onths and months, it was all you wanted, all you thought about, all you dreamt about. So I went
with the girls. As expected, @CAPS3 walked down the three-person wide staircase side-by-side, a
nd I shuffled awkwardly behind them. Finally arriving at @NUM2scalier, we sat at a table, the t
hree girls talking. I glazed my eyes over, attempting to look lost in thought, as if I didn\'t
care I wasn\'t included. Selfish thoughts buzzed in my head; if @CAPS3 weren\'t talking to me,
why should I make the effort to talk to them?   I really had no idea how @CAPS3 felt about me.
How does someone feel about their shadow? @CAPS3 notice it, sure, but it never offers up insigh
t, it never makes you laugh. It\'s all in the confidence, said my mother\'s voice, all how you
carry yourself. But I knew it wasn\'t that simple. I was just too alien. These girls would neve
r understand me, as I would never understand them. In frustration, I started to flick peas acro
ss the room with my spoon. Pat, flick, sproing.  This caught the interest of @PERSON1, as @NUM1
le and @ORGANIZATION1 were discussing something very emotional. Tears began to pour out of @ORG
ANIZATION1\'s eyes. Sniffling, she and @NUM1le went to the bathroom, leaving me all alone with
@PERSON1. Only @PERSON2 could have felt my felt my same emotion as he stared up at @CAPS6. Sile
ntly, I continued shooting peas. @PERSON1 just stared at them as @CAPS3 darted around the room.
Suddenly, with a horrible miscalculation, a pea hit a boy in the face. And then, he turned arou
nd and swore. And then, @PERSON1 and I looked at each other from across the table.  And then, w
e laughed.  We laughed so hard I cried. So hard that huge, alien tears flooded from my eyes. Pe
ople around us were laughing too, even though @CAPS3 had no idea what was so funny. I didn\'t e
ven know what was so funny. But it didn\'t matter, because we were dripping tears and snot, rea
ching for each other, reenacting the pea hitting the boy\'s face. It was as if we had been frie
nds for years, and laughing happened all the time. It was saturated with all the angst and lonl
iness and despair I had felt the past four weeks. The connection we felt was instantaneous, lik
e lightening, the kind of connection I felt with my best friends back home. I felt that huge sw
```

```
elling sensation in my chest, like a balloon was stuck inside. My stomach was aching and my che
eks were so sore I felt them seizing up. My heart felt whole even for that second. My soul was
open. It was the best laugh of my life.  Sniffle sniffle. GASP. Laughter. GASP. Swipe of tears.
Sniffle sniffle. Laughter. GASP. This is why. I thought. This is why you came.  Bell rings.'
```

In [22]:

```python
# count various features

t0 = datetime.now()

training_set['token_count'] = training_set.apply(lambda x: len(x['tokens']), axis=1)
training_set['unique_token_count'] = training_set.apply(lambda x: len(set(x['tokens'])), axis=1)
training_set['nostop_count'] = training_set \
            .apply(lambda x: len([token for token in x['tokens'] if token not in stop_words]), a
training_set['sent_count'] = training_set.apply(lambda x: len(x['sents']), axis=1)
training_set['ner_count'] = training_set.apply(lambda x: len(x['ner']), axis=1)
training_set['comma'] = training_set.apply(lambda x: x['corrected'].count(','), axis=1)
training_set['question'] = training_set.apply(lambda x: x['corrected'].count('?'), axis=1)
training_set['exclamation'] = training_set.apply(lambda x: x['corrected'].count('!'), axis=1)
training_set['quotation'] = training_set.apply(lambda x: x['corrected'].count('"') + x['correcte
training_set['organization'] = training_set.apply(lambda x: x['corrected'].count(r'@ORGANIZATION
training_set['caps'] = training_set.apply(lambda x: x['corrected'].count(r'@CAPS'), axis=1)
training_set['person'] = training_set.apply(lambda x: x['corrected'].count(r'@PERSON'), axis=1)
training_set['location'] = training_set.apply(lambda x: x['corrected'].count(r'@LOCATION'), axis
training_set['money'] = training_set.apply(lambda x: x['corrected'].count(r'@MONEY'), axis=1)
training_set['time'] = training_set.apply(lambda x: x['corrected'].count(r'@TIME'), axis=1)
training_set['date'] = training_set.apply(lambda x: x['corrected'].count(r'@DATE'), axis=1)
training_set['percent'] = training_set.apply(lambda x: x['corrected'].count(r'@PERCENT'), axis=1
training_set['noun'] = training_set.apply(lambda x: x['pos'].count('NOUN'), axis=1)
training_set['adj'] = training_set.apply(lambda x: x['pos'].count('ADJ'), axis=1)
training_set['pron'] = training_set.apply(lambda x: x['pos'].count('PRON'), axis=1)
training_set['verb'] = training_set.apply(lambda x: x['pos'].count('VERB'), axis=1)
training_set['noun'] = training_set.apply(lambda x: x['pos'].count('NOUN'), axis=1)
training_set['cconj'] = training_set.apply(lambda x: x['pos'].count('CCONJ'), axis=1)
training_set['adv'] = training_set.apply(lambda x: x['pos'].count('ADV'), axis=1)
training_set['det'] = training_set.apply(lambda x: x['pos'].count('DET'), axis=1)
```

```
training_set['propn'] = training_set.apply(lambda x: x['pos'].count('PROPN'), axis=1)
training_set['num'] = training_set.apply(lambda x: x['pos'].count('NUM'), axis=1)
training_set['part'] = training_set.apply(lambda x: x['pos'].count('PART'), axis=1)
training_set['intj'] = training_set.apply(lambda x: x['pos'].count('INTJ'), axis=1)

t1 = datetime.now()
print('Processing time: {}'.format(t1 - t0))
```

```
Processing time: 0:00:03.027787
```

In [23]:
```
# save to file
training_set.to_pickle('training_features.pkl')
```

In [24]:
```
training_set = pd.read_pickle('training_features.pkl')
```

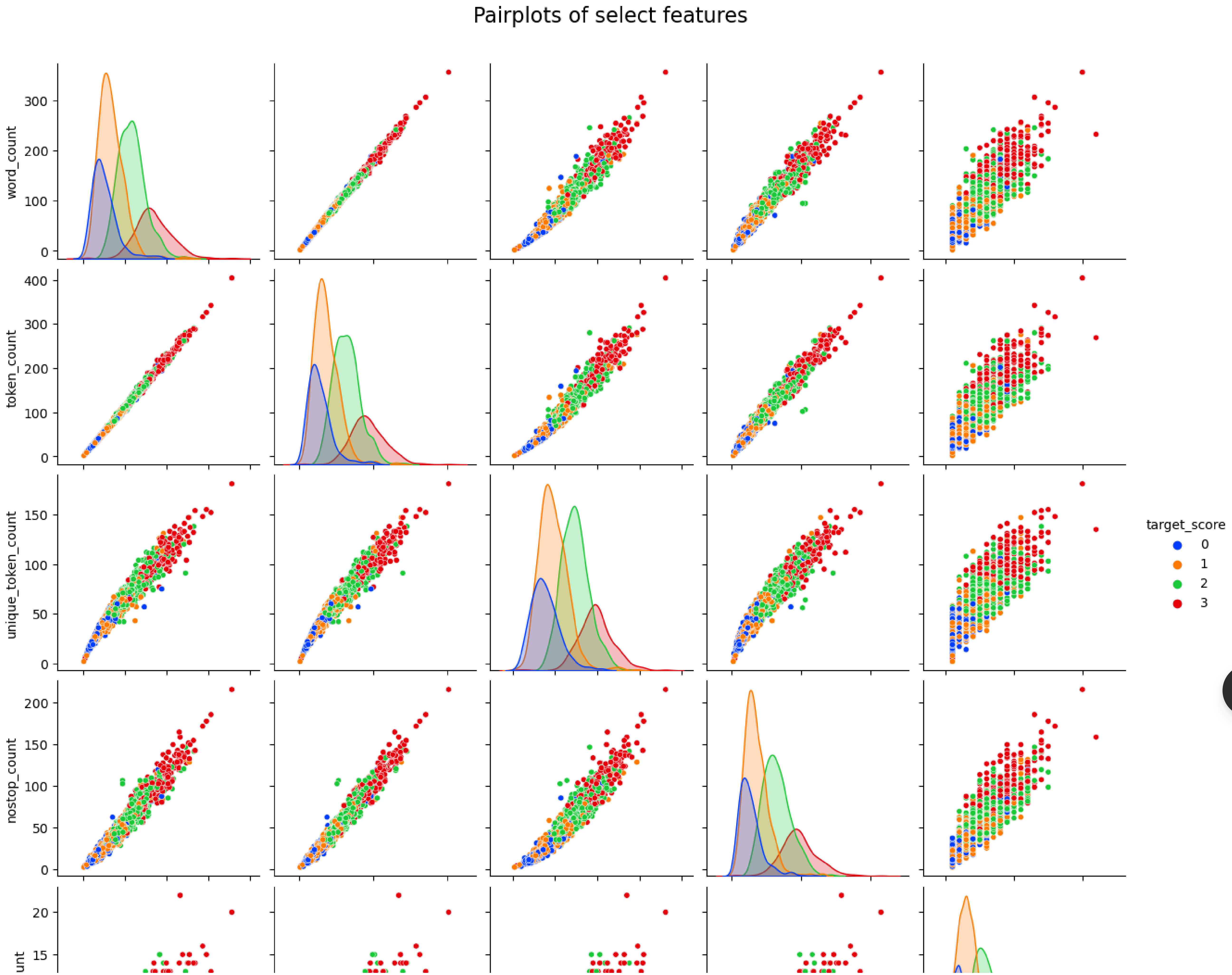
Feature Selection

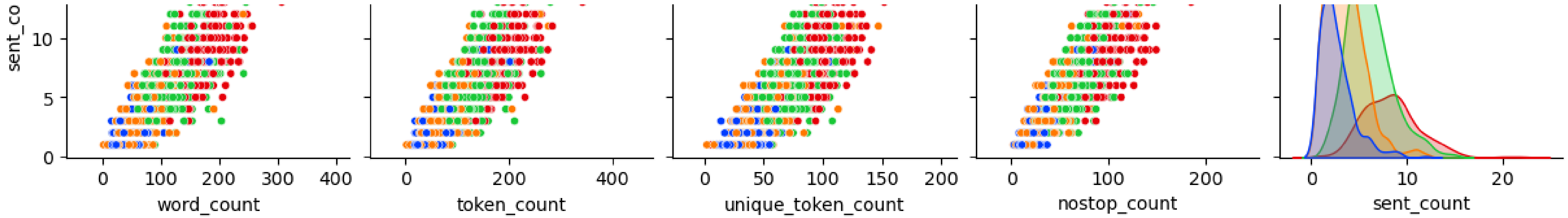
Many of the generated features are correlated with essay length. Collinearity is potentially an issue here.

In [27]:
```
# Plot correlation of essay-length related features
usecols = ['word_count', 'token_count', 'unique_token_count', 'nostop_count', 'sent_count']
g = sns.pairplot(training_set[training_set.topic == 4], hue='target_score', vars=usecols, plot_k
g.fig.subplots_adjust(top=.93)
g.fig.suptitle('Pairplots of select features', fontsize=16)
plt.show()
```

```
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na
option is deprecated and will be removed in a future version. Convert inf values to NaN before op
erating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_oldcore.py:1075: FutureWarning: When grouping
with a length-1 list-like, you will need to pass a length-1 tuple to get_group in a future versio
n of pandas. Pass `(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_oldcore.py:1075: FutureWarning: When grouping
with a length-1 list-like, you will need to pass a length-1 tuple to get_group in a future versio
n of pandas. Pass `(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na
option is deprecated and will be removed in a future version. Convert inf values to NaN before op
erating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_oldcore.py:1075: FutureWarning: When grouping
with a length-1 list-like, you will need to pass a length-1 tuple to get_group in a future versio
n of pandas. Pass `(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_oldcore.py:1075: FutureWarning: When grouping
with a length-1 list-like, you will need to pass a length-1 tuple to get_group in a future versio
n of pandas. Pass `(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na
option is deprecated and will be removed in a future version. Convert inf values to NaN before op
erating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_oldcore.py:1075: FutureWarning: When grouping
with a length-1 list-like, you will need to pass a length-1 tuple to get_group in a future versio
n of pandas. Pass `(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_oldcore.py:1075: FutureWarning: When grouping
with a length-1 list-like, you will need to pass a length-1 tuple to get_group in a future versio
n of pandas. Pass `(name,)` instead of `name` to silence this warning.
```

```
  data_subset = grouped_data.get_group(pd_key)
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na
option is deprecated and will be removed in a future version. Convert inf values to NaN before op
erating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_oldcore.py:1075: FutureWarning: When grouping
with a length-1 list-like, you will need to pass a length-1 tuple to get_group in a future versio
n of pandas. Pass `(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_oldcore.py:1075: FutureWarning: When grouping
with a length-1 list-like, you will need to pass a length-1 tuple to get_group in a future versio
n of pandas. Pass `(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na
option is deprecated and will be removed in a future version. Convert inf values to NaN before op
erating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_oldcore.py:1075: FutureWarning: When grouping
with a length-1 list-like, you will need to pass a length-1 tuple to get_group in a future versio
n of pandas. Pass `(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/opt/anaconda3/lib/python3.9/site-packages/seaborn/_oldcore.py:1075: FutureWarning: When grouping
with a length-1 list-like, you will need to pass a length-1 tuple to get_group in a future versio
n of pandas. Pass `(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
```


Pairplots of select features
word_count
token_count
unique_token_count
nostop_count
unt
target_score
0
1
2
3



In [26]:
```
training_set.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12976 entries, 0 to 12975
Data columns (total 66 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   essay_id            12976 non-null  int64  
 1   topic               12976 non-null  int64  
 2   essay               12976 non-null  object 
 3   rater1_domain1      12976 non-null  int64  
 4   rater2_domain1      12976 non-null  int64  
 5   rater3_domain1      128 non-null    float64
 6   target_score        12976 non-null  int64  
 7   rater1_domain2      1800 non-null   float64
 8   rater2_domain2      1800 non-null   float64
 9   topic2_target       1800 non-null   float64
 10  rater1_trait1       2292 non-null   float64
 11  rater1_trait2       2292 non-null   float64
 12  rater1_trait3       2292 non-null   float64
 13  rater1_trait4       2292 non-null   float64
 14  rater1_trait5       723 non-null    float64
 15  rater1_trait6       723 non-null    float64
 16  rater2_trait1       2292 non-null   float64
 17  rater2_trait2       2292 non-null   float64
 18  rater2_trait3       2292 non-null   float64
 19  rater2_trait4       2292 non-null   float64
 20  rater2_trait5       723 non-null    float64
 21  rater2_trait6       723 non-null    float64
 22  rater3_trait1       128 non-null    float64
 23  rater3_trait2       128 non-null    float64
 24  rater3_trait3       128 non-null    float64
 25  rater3_trait4       128 non-null    float64
 26  rater3_trait5       128 non-null    float64
 27  rater3_trait6       128 non-null    float64
 28  word_count          12976 non-null  int64  
 29  matches             12976 non-null  object 
```

```
 30  corrections        12976 non-null  int64
 31  corrected          12976 non-null  object
 32  tokens             12976 non-null  object
 33  lemma              12976 non-null  object
 34  pos                12976 non-null  object
 35  sents              12976 non-null  object
 36  ner                12976 non-null  object
 37  similarity         12976 non-null  float64
 38  token_count        12976 non-null  int64
 39  unique_token_count 12976 non-null  int64
 40  nostop_count       12976 non-null  int64
 41  sent_count         12976 non-null  int64
 42  ner_count          12976 non-null  int64
 43  comma              12976 non-null  int64
 44  question           12976 non-null  int64
 45  exclamation        12976 non-null  int64
 46  quotation          12976 non-null  int64
 47  organization       12976 non-null  int64
 48  caps               12976 non-null  int64
 49  person             12976 non-null  int64
 50  location           12976 non-null  int64
 51  money              12976 non-null  int64
 52  time               12976 non-null  int64
 53  date               12976 non-null  int64
 54  percent            12976 non-null  int64
 55  noun               12976 non-null  int64
 56  adj                12976 non-null  int64
 57  pron               12976 non-null  int64
 58  verb               12976 non-null  int64
 59  cconj              12976 non-null  int64
 60  adv                12976 non-null  int64
 61  det                12976 non-null  int64
 62  propn              12976 non-null  int64
 63  num                12976 non-null  int64
 64  part               12976 non-null  int64
```

```
 65  intj                12976 non-null  int64
dtypes: float64(23), int64(35), object(8)
memory usage: 6.5+ MB
```

Incomplete columns are not used for modeling and can be ignored.

Univariate feature selection performed on the vectorized data shows few differences in the 10 best features by topic number. It is not surprising that `similarity` has little influence on `target_score` in topic 4 since there are only four unique scores and the the similarity by score plot above shows high variance.

In [28]:
```
# Selecting k best features: Some features omitted due to high correlation

predictors = [
#                'word_count',
                'corrections',
                'similarity',
#                'token_count',
                'unique_token_count',
#                'nostop_count',
                'sent_count',
                'ner_count',
                'comma',
                'question',
                'exclamation',
                'quotation',
                'organization',
                'caps',
                'person',
                'location',
                'money',
                'time',
                'date',
                'percent',
                'noun',
```

```
                 'adj',
                 'pron',
                 'verb',
                 'cconj',
                 'adv',
                 'det',
                 'propn',
                 'num',
                 'part',
                 'intj'
                 ]

# Create and fit selector
selector = SelectKBest(f_regression, k=10) # f_classif, chi2, f_regression, mutual_info_classif,

# Create empty dataframe
df = pd.DataFrame()

for topic in range(1, 9):
    kpredictors = []

    # test for division by zero errors due to insufficient data:
    for p in predictors:
        if np.std(training_set[training_set.topic == topic][p], axis=0) != 0:
            kpredictors.append(p)

    # select k best for each topic:
    X = training_set[training_set.topic == topic][kpredictors]
    y = training_set[training_set.topic == topic].target_score

    selector.fit(X, y)

    # Get idxs of columns to keep
    mask = selector.get_support(indices=True)
```

```
    selected_features = training_set[training_set.topic == topic][predictors].columns[mask]
    df["Topic " + str(topic)] = selected_features
df
```

Out[28]:

	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5	
0	similarity	similarity	similarity	unique_token_count	similarity	
1	unique_token_count	unique_token_count	unique_token_count	sent_count	unique_token_count	unique_
2	sent_count	sent_count	sent_count	ner_count	sent_count	
3	comma	comma	comma	comma	date	
4	noun	noun	date	percent	percent	
5	adj	adj	percent	noun	noun	
6	verb	verb	noun	adj	adj	
7	cconj	adv	adj	pron	pron	
8	adv	det	verb	verb	verb	
9	det	part	cconj	adv	cconj	

Define the regression pipeline:

In [29]:
```
def evaluate(df, topic, features, model):
    """Regression pipeline with kappa evaluation"""

    X = df[df['topic'] == topic][features]
    y = df[df['topic'] == topic]['target_score'].astype(np.float64)
    # token_ct = X.token_count
    # X = X.div(token_ct, axis=0)
    # X['token_count'] = X['token_count'].mul(token_ct, axis=0)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=26)
```

```
    pipeline = Pipeline(model)
    pipeline.fit(X_train, y_train)


    y_pred = pipeline.predict(X_test)

    return kappa(y_pred, y_test, weights='quadratic')
```

An alternative feature selection strategy is to use **L1** regularization to limit the influence of less important features. This is implemented below in the ElasticNet regressor.

In [30]:
```
predictors = [
                'word_count',
                'corrections',
                'similarity',
                'token_count',
                'unique_token_count',
                'nostop_count',
                'sent_count',
                'ner_count',
                'comma',
                'question',
                'exclamation',
                'quotation',
                'organization',
                'caps',
                'person',
                'location',
                'money',
                'time',
                'date',
                'percent',
                'noun',
```

```
                'adj',
                'pron',
                'verb',
                'cconj',
                'adv',
                'det',
                'propn',
                'num',
                'part',
                'intj'
                ]

# feature selection
# fvalue_selector = SelectKBest(score_func=f_regression, k=10)

# for use in pipeline
models = [
            [('scaler', StandardScaler()),('linearSVC', LinearSVC(C=0.01))] ,
            [('scaler', StandardScaler()),('lm', LinearRegression())],
            [('rf', RandomForestRegressor(random_state=26))],
            [('en', ElasticNet(l1_ratio=0.01, alpha=0.1, max_iter=100000, random_state=26))]
        ]

for steps in models:
    kappas = []
    weights = []
    for topic in range(1,9):
        kappas.append(evaluate(training_set, topic, predictors, steps))
        weights.append(len(training_set[training_set.topic==topic]))

    mqwk = mean_quadratic_weighted_kappa(kappas, weights=weights)
    print(steps)
    print('Weighted by topic Kappa score: {:.4f}'.format(mqwk))
    print('')
```

```
[('scaler', StandardScaler()), ('linearSVC', LinearSVC(C=0.01))]
Weighted by topic Kappa score: 0.5852

[('scaler', StandardScaler()), ('lm', LinearRegression())]
Weighted by topic Kappa score: 0.7128

[('rf', RandomForestRegressor(random_state=26))]
Weighted by topic Kappa score: 0.7147

Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(R) SSE4.2) enabled on
ly processors has been deprecated. Intel oneAPI Math Kernel Library 2025.0 will require Intel(R)
Advanced Vector Extensions (Intel(R) AVX) instructions.
[('en', ElasticNet(alpha=0.1, l1_ratio=0.01, max_iter=100000, random_state=26))]
Weighted by topic Kappa score: 0.7072
```

Of the four models on which the data was evaluated, the three regression models returned very similar mean weighted kappa scores and the simple linear regression model slightly outperformed the others. The support vector classifier performed poorly.

Can we improve on the hyperparameters for ElasticNet by running GridSearchCV on each topic?

In [31]:
```
# ElasticNet with GridSearchCV for each individual topic

def en_evaluate(df, topic, features):
    # Regression pipeline with kappa evaluation
    paramgrid = {'l1_ratio': [.01, .1, .3, .5, .7, .99], 'alpha': [0.001, 0.01, 0.1, 1]}
    X = df[df['topic'] == topic][features]
    y = df[df['topic'] == topic]['target_score'].astype(np.float64)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=26)

    gs = GridSearchCV(ElasticNet(max_iter=100000, random_state=26),
                      param_grid=paramgrid,
                      cv=5)
```

```
    gs.fit(X_train, y_train)
    print('Topic', topic, 'best parameters:', gs.best_params_)
    y_pred = gs.predict(X_test)

    return kappa(y_pred, y_test, weights='quadratic')
```

In [32]:

```
kappas = []
weights = []
for topic in range(1,9):
    kappas.append(en_evaluate(training_set, topic, predictors))
    weights.append(len(training_set[training_set.topic==topic]))

mqwk = mean_quadratic_weighted_kappa(kappas, weights=weights)
print('Weighted by topic Kappa score: {:.4f}'.format(mqwk))
```

```
Topic 1 best parameters: {'alpha': 0.001, 'l1_ratio': 0.99}
Topic 2 best parameters: {'alpha': 0.001, 'l1_ratio': 0.99}
Topic 3 best parameters: {'alpha': 0.1, 'l1_ratio': 0.1}
Topic 4 best parameters: {'alpha': 1, 'l1_ratio': 0.01}
Topic 5 best parameters: {'alpha': 0.001, 'l1_ratio': 0.99}
Topic 6 best parameters: {'alpha': 0.001, 'l1_ratio': 0.01}
Topic 7 best parameters: {'alpha': 0.001, 'l1_ratio': 0.99}
Topic 8 best parameters: {'alpha': 1, 'l1_ratio': 0.7}
Weighted by topic Kappa score: 0.7119
```

A low `l1_ratio` implies Ridge regression with **l2** regularization. The weighted Kappa score did not improve measurably when hyperparameters are tuned for each topic, including cross-validation.

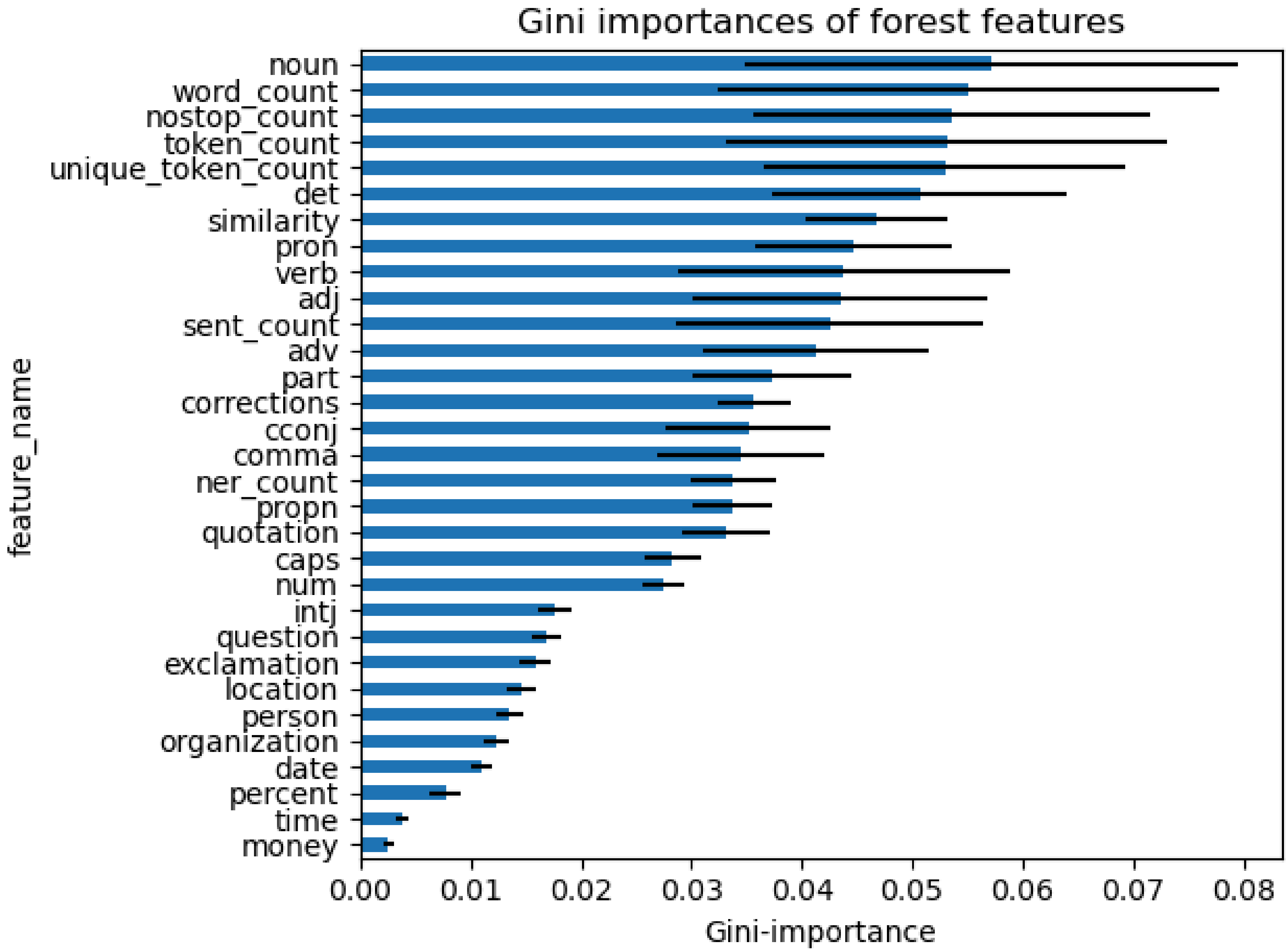
In [33]:

```
# Individual topic kappa scores
kappas
```

Out[33]:
```
[0.8054079792566925,
 0.6374316525954797,
 0.6572487262621585,
 0.6178914682833063,
 0.7778008933488344,
 0.6945129732028923,
 0.7476491205105129,
 0.712328028055031]
```

A final approach for feature selection is to extract the Gini-importances of random forests:

In [34]:
```
X = training_set[predictors]
y = training_set['target_score'].astype(np.float64)


forest = ExtraTreesClassifier(n_estimators=250,
                              random_state=26)

forest.fit(X, y)

std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)

# plot feature importances

features = pd.DataFrame({'feature_name': X.columns, 'importance': forest.feature_importances_,
features.sort_values('importance')\
        .plot.barh(x='feature_name', y='importance', xerr='std', legend=False)
plt.title('Gini importances of forest features')
plt.xlabel('Gini-importance')
plt.tight_layout()
plt.show()
```



In [35]:

```
# best k features
k = 15
top_features = features.sort_values('importance', ascending=False)['feature_name'].tolist()[:k]

# Linear regression with top k features
kappas = []
```

```
weights = []
steps = [('scaler', StandardScaler()),('lm', LinearRegression())]
for topic in range(1,9):
    kappas.append(evaluate(training_set, topic, top_features, steps))
    weights.append(len(training_set[training_set.topic==topic]))

mqwk = mean_quadratic_weighted_kappa(kappas, weights=weights)
print('Weighted by topic Kappa score: {:.4f}'.format(mqwk))
```

```
Weighted by topic Kappa score: 0.7116
```

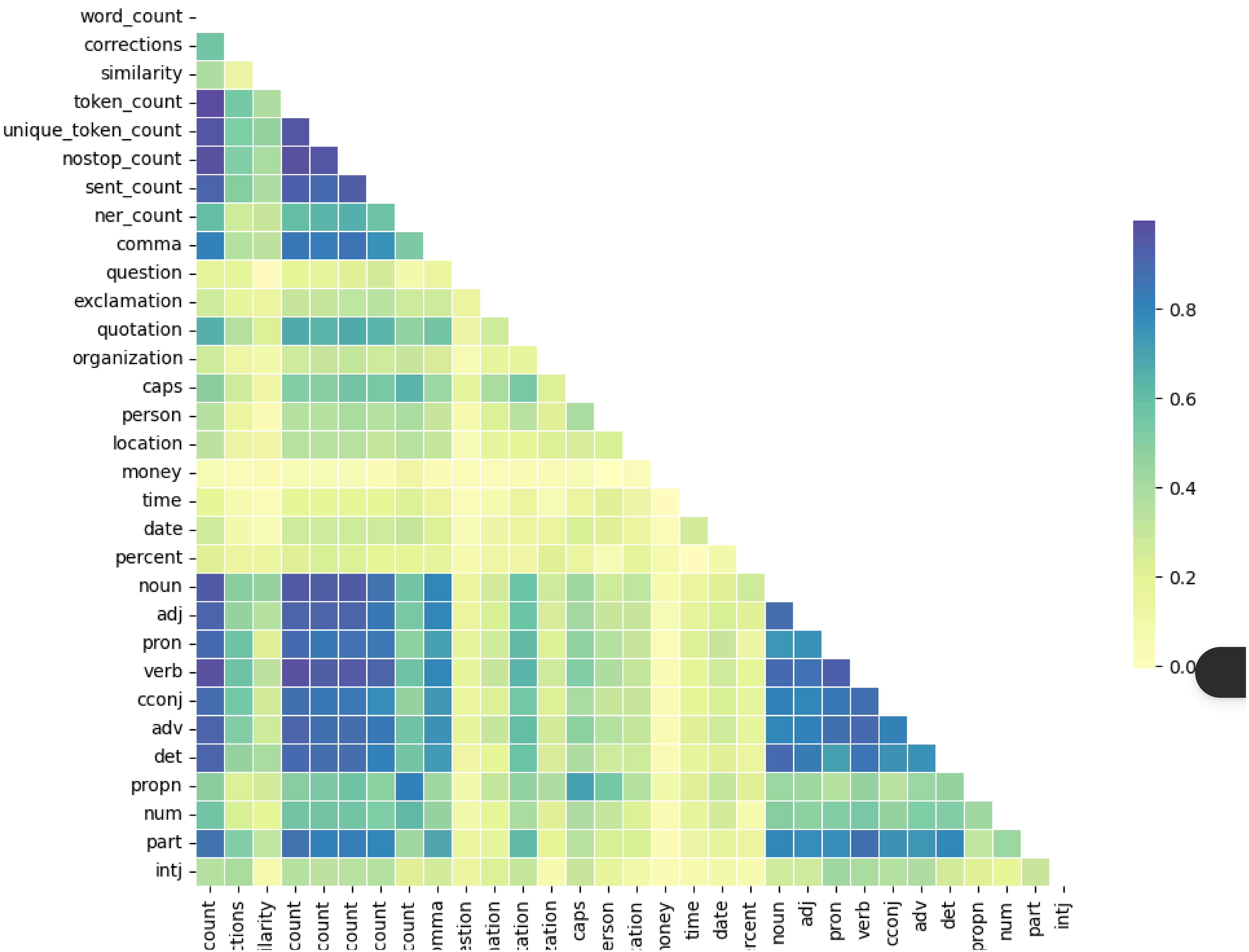
The kappa scores increase with increasing number of features.

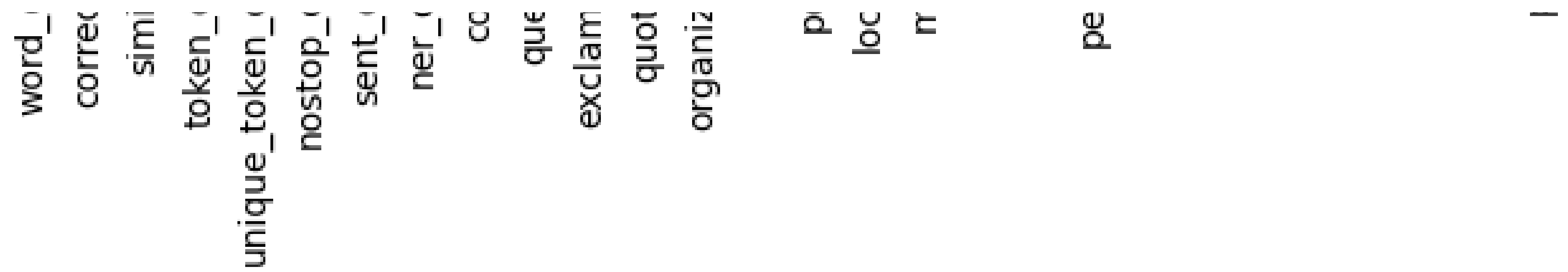
As shown earlier and in the correlation matrix below, some features are highly correlated. This can lead to problems if there are insufficient observations to explain the differences between features. Signs of potential collinearity problems could be poor generalization of the model. In this case, the Kappa scores did not change dramatically when using training and test data or when applying cross-validation.

Models that apply feature selection might automatically remove some highly correlated features.

In [37]:
```
# Overview of correlating features
corr = training_set[predictors].corr() # default: Pearson
mask = np.zeros_like(corr, dtype=bool)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
g = sns.heatmap(corr, mask=mask, cmap='Spectral', center=0,
```

```
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
plt.show()
```


word_count
corrections
similarity
token_count
unique_token_count
nostop_count
sent_count
ner_count
comma
question
exclamation
quotation
organization
caps
person
location
money
time
date
percent
noun
adj
pron
verb
cconj
adv
det
propn
num
part
intj
0.8
0.6
0.4
0.2
0.0



Adding TF-IDF features

In [39]:
```
# Lemmatized essays re-joined (list to essay)
training_set['l_essay'] = training_set['lemma'].apply(' '.join)

vectorizer = TfidfVectorizer(max_df=.2,
                             min_df=3,
                             max_features=2000,
                             stop_words=list(STOP_WORDS)) # default: binary=False
tfidf_matrix = vectorizer.fit_transform(training_set.l_essay) # using lemmatized essays
tfidf_matrix.shape
```

```
/opt/anaconda3/lib/python3.9/site-packages/sklearn/feature_extraction/text.py:409: UserWarning: Y
our stop_words may be inconsistent with your preprocessing. Tokenizing the stop words generated t
okens ['ll', 've'] not in stop_words.
  "Your stop_words may be inconsistent with "
```

Out[39]: (12976, 2000)

In [40]:
```
training_set[predictors].shape
```

Out[40]: (12976, 31)

In [41]:
```
# Combine previous predictors with TF-IDF matrix
combined_dense = pd.concat([pd.DataFrame(tfidf_matrix.todense()),
                            training_set[predictors],
                            training_set['topic'],
```

```
                                    training_set['target_score']], axis=1)
combined_dense.shape
```

Out[41]: (12976, 2033)

In [42]:

```
# ElasticNet with GridSearchCV for each individual topic

def tf_evaluate(df, topic):
    # Regression pipeline with kappa evaluation
    paramgrid = {'l1_ratio': [.01, .1, .5, .9], 'alpha': [0.01, .1, 1]}
    X = df[df['topic'] == topic].drop(['topic', 'target_score'], axis=1)
    y = df[df['topic'] == topic]['target_score'].astype(np.float64)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=26)

    gs = GridSearchCV(ElasticNet(max_iter=100000, random_state=26),
                      param_grid=paramgrid,
                      cv=5)
    gs.fit(X_train, y_train)
    print('Topic', topic, 'best parameters:', gs.best_params_)
    y_pred = gs.predict(X_test)

    return kappa(y_pred, y_test, weights='quadratic')
```

In [45]:

```
# ElasticNet with GridSearchCV for each individual topic

kappas = []
weights = []
for topic in range(1,9):
    kappas.append(tf_evaluate(combined_dense, topic))
    weights.append(len(training_set[training_set.topic==topic]))

mqwk = mean_quadratic_weighted_kappa(kappas, weights=weights)
print('Weighted by topic Kappa score: {:.4f}'.format(mqwk))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[45], line 6
      4 weights = []
      5 for topic in range(1,9):
----> 6     kappas.append(tf_evaluate(combined_dense, topic))
      7     weights.append(len(training_set[training_set.topic==topic]))
      9 mqwk = mean_quadratic_weighted_kappa(kappas, weights=weights)

Cell In[42], line 13, in tf_evaluate(df, topic)
      8 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=26)
     10 gs = GridSearchCV(ElasticNet(max_iter=100000, random_state=26),
     11                   param_grid=paramgrid,
     12                   cv=5)
---> 13 gs.fit(X_train, y_train)
     14 print('Topic', topic, 'best parameters:', gs.best_params_)
     15 y_pred = gs.predict(X_test)

File /opt/anaconda3/lib/python3.9/site-packages/sklearn/model_selection/_search.py:874, in fit(se
lf, X, y, groups, **fit_params)
    869 # Here we keep a dict of scorers as is, and only convert to a
    870 # _MultimetricScorer at a later stage. Issue:
    871 # https://github.com/scikit-learn/scikit-learn/issues/27001
    872 scorers, refit_metric = self._get_scorers(convert_multimetric=False)
--> 874 X, y = indexable(X, y)
    875 params = _check_method_params(X, params=params)
    877 routed_params = self._get_routed_params_for_fit(params)

File /opt/anaconda3/lib/python3.9/site-packages/sklearn/model_selection/_search.py:1388, in _run_
search(self, evaluate_candidates)
   1164 class GridSearchCV(BaseSearchCV):
   1165     """Exhaustive search over specified parameter values for an estimator.
   1166 
   1167     Important members are fit, predict.
   1168 
```

```
    GridSearchCV implements a "fit" and a "score" method.
    It also implements "score_samples", "predict", "predict_proba",
    "decision_function", "transform" and "inverse_transform" if they are
    implemented in the estimator used.

    The parameters of the estimator used to apply these methods are optimized
    by cross-validated grid-search over a parameter grid.

    Read more in the :ref:`User Guide <grid_search>`.

    Parameters
    ----------
    estimator : estimator object
        This is assumed to implement the scikit-learn estimator interface.
        Either estimator needs to provide a ``score`` function,
        or ``scoring`` must be passed.

    param_grid : dict or list of dictionaries
        Dictionary with parameters names (`str`) as keys and lists of
        parameter settings to try as values, or a list of such
        dictionaries, in which case the grids spanned by each dictionary
        in the list are explored. This enables searching over any sequence
        of parameter settings.

    scoring : str, callable, list, tuple or dict, default=None
        Strategy to evaluate the performance of the cross-validated model on
        the test set.

        If `scoring` represents a single score, one can use:

        - a single string (see :ref:`scoring_parameter`);
        - a callable (see :ref:`scoring`) that returns a single value.

        If `scoring` represents multiple scores, one can use:

```

```
        - a list or tuple of unique strings;
        - a callable returning a dictionary where the keys are the metric
          names and the values are the metric scores;
        - a dictionary with metric names as keys and callables a values.

        See :ref:`multimetric_grid_search` for an example.

    n_jobs : int, default=None
        Number of jobs to run in parallel.
        ``None`` means 1 unless in a :obj:`joblib.parallel_backend` context.
        ``-1`` means using all processors. See :term:`Glossary <n_jobs>`
        for more details.

        .. versionchanged:: v0.20
           `n_jobs` default changed from 1 to None

    refit : bool, str, or callable, default=True
        Refit an estimator using the best found parameters on the whole
        dataset.

        For multiple metric evaluation, this needs to be a `str` denoting the
        scorer that would be used to find the best parameters for refitting
        the estimator at the end.

        Where there are considerations other than maximum score in
        choosing a best estimator, ``refit`` can be set to a function which
        returns the selected ``best_index_`` given ``cv_results_``. In that
        case, the ``best_estimator_`` and ``best_params_`` will be set
        according to the returned ``best_index_`` while the ``best_score_``
        attribute will not be available.

        The refitted estimator is made available at the ``best_estimator_``
        attribute and permits using ``predict`` directly on this
        ``GridSearchCV`` instance.

```

```
        Also for multiple metric evaluation, the attributes ``best_index_``,
        ``best_score_`` and ``best_params_`` will only be available if
        ``refit`` is set and all of them will be determined w.r.t this specific
        scorer.

        See ``scoring`` parameter to know more about multiple metric
        evaluation.

        See :ref:`sphx_glr_auto_examples_model_selection_plot_grid_search_digits.py`
        to see how to design a custom selection strategy using a callable
        via `refit`.

        .. versionchanged:: 0.20
            Support for callable added.

    cv : int, cross-validation generator or an iterable, default=None
        Determines the cross-validation splitting strategy.
        Possible inputs for cv are:

        - None, to use the default 5-fold cross validation,
        - integer, to specify the number of folds in a `(Stratified)KFold`,
        - :term:`CV splitter`,
        - An iterable yielding (train, test) splits as arrays of indices.

        For integer/None inputs, if the estimator is a classifier and ``y`` is
        either binary or multiclass, :class:`StratifiedKFold` is used. In all
        other cases, :class:`KFold` is used. These splitters are instantiated
        with `shuffle=False` so the splits will be the same across calls.

        Refer :ref:`User Guide <cross_validation>` for the various
        cross-validation strategies that can be used here.

        .. versionchanged:: 0.22
            ``cv`` default value if None changed from 3-fold to 5-fold.

```

```
    verbose : int
        Controls the verbosity: the higher, the more messages.

        - >1 : the computation time for each fold and parameter candidate is
          displayed;
        - >2 : the score is also displayed;
        - >3 : the fold and candidate parameter indexes are also displayed
          together with the starting time of the computation.

    pre_dispatch : int, or str, default='2*n_jobs'
        Controls the number of jobs that get dispatched during parallel
        execution. Reducing this number can be useful to avoid an
        explosion of memory consumption when more jobs get dispatched
        than CPUs can process. This parameter can be:

            - None, in which case all the jobs are immediately
              created and spawned. Use this for lightweight and
              fast-running jobs, to avoid delays due to on-demand
              spawning of the jobs

            - An int, giving the exact number of total jobs that are
              spawned

            - A str, giving an expression as a function of n_jobs,
              as in '2*n_jobs'

    error_score : 'raise' or numeric, default=np.nan
        Value to assign to the score if an error occurs in estimator fitting.
        If set to 'raise', the error is raised. If a numeric value is given,
        FitFailedWarning is raised. This parameter does not affect the refit
        step, which will always raise the error.

    return_train_score : bool, default=False
        If ``False``, the ``cv_results_`` attribute will not include training
        scores.
```

```
        Computing training scores is used to get insights on how different
        parameter settings impact the overfitting/underfitting trade-off.
        However computing the scores on the training set can be computationally
        expensive and is not strictly required to select the parameters that
        yield the best generalization performance.

        .. versionadded:: 0.19

        .. versionchanged:: 0.21
            Default value was changed from ``True`` to ``False``

    Attributes
    ----------
    cv_results_ : dict of numpy (masked) ndarrays
        A dict with keys as column headers and values as columns, that can be
        imported into a pandas ``DataFrame``.

        For instance the below given table

        +------------+-----------+------------+-----------------+---+---------+
        |param_kernel|param_gamma|param_degree|split0_test_score|...|rank_t...|
        +============+===========+============+=================+===+=========+
        |  'poly'    |     --    |      2     |       0.80      |...|    2    |
        +------------+-----------+------------+-----------------+---+---------+
        |  'poly'    |     --    |      3     |       0.70      |...|    4    |
        +------------+-----------+------------+-----------------+---+---------+
        |  'rbf'     |     0.1   |     --     |       0.80      |...|    3    |
        +------------+-----------+------------+-----------------+---+---------+
        |  'rbf'     |     0.2   |     --     |       0.93      |...|    1    |
        +------------+-----------+------------+-----------------+---+---------+

        will be represented by a ``cv_results_`` dict of::

            {
            'param_kernel': masked_array(data = ['poly', 'poly', 'rbf', 'rbf'],
```

```
                                          mask = [False False False False]...)
            'param_gamma': masked_array(data = [-- -- 0.1 0.2],
                                         mask = [ True  True False False]...),
            'param_degree': masked_array(data = [2.0 3.0 -- --],
                                          mask = [False False  True  True]...),
            'split0_test_score'  : [0.80, 0.70, 0.80, 0.93],
            'split1_test_score'  : [0.82, 0.50, 0.70, 0.78],
            'mean_test_score'    : [0.81, 0.60, 0.75, 0.85],
            'std_test_score'     : [0.01, 0.10, 0.05, 0.08],
            'rank_test_score'    : [2, 4, 3, 1],
            'split0_train_score' : [0.80, 0.92, 0.70, 0.93],
            'split1_train_score' : [0.82, 0.55, 0.70, 0.87],
            'mean_train_score'   : [0.81, 0.74, 0.70, 0.90],
            'std_train_score'    : [0.01, 0.19, 0.00, 0.03],
            'mean_fit_time'      : [0.73, 0.63, 0.43, 0.49],
            'std_fit_time'       : [0.01, 0.02, 0.01, 0.01],
            'mean_score_time'    : [0.01, 0.06, 0.04, 0.04],
            'std_score_time'     : [0.00, 0.00, 0.00, 0.01],
            'params'             : [{'kernel': 'poly', 'degree': 2}, ...],
            }

        NOTE

        The key ``'params'`` is used to store a list of parameter
        settings dicts for all the parameter candidates.

        The ``mean_fit_time``, ``std_fit_time``, ``mean_score_time`` and
        ``std_score_time`` are all in seconds.

        For multi-metric evaluation, the scores for all the scorers are
        available in the ``cv_results_`` dict at the keys ending with that
        scorer's name (``'_<scorer_name>'``) instead of ``'_score'`` shown
        above. ('split0_test_precision', 'mean_train_precision' etc.)

    best_estimator_ : estimator
```

```
        Estimator that was chosen by the search, i.e. estimator
        which gave highest score (or smallest loss if specified)
        on the left out data. Not available if ``refit=False``.

        See ``refit`` parameter for more information on allowed values.

    best_score_ : float
        Mean cross-validated score of the best_estimator

        For multi-metric evaluation, this is present only if ``refit`` is
        specified.

        This attribute is not available if ``refit`` is a function.

    best_params_ : dict
        Parameter setting that gave the best results on the hold out data.

        For multi-metric evaluation, this is present only if ``refit`` is
        specified.

    best_index_ : int
        The index (of the ``cv_results_`` arrays) which corresponds to the best
        candidate parameter setting.

        The dict at ``search.cv_results_['params'][search.best_index_]`` gives
        the parameter setting for the best model, that gives the highest
        mean score (``search.best_score_``).

        For multi-metric evaluation, this is present only if ``refit`` is
        specified.

    scorer_ : function or a dict
        Scorer function used on the held out data to choose the best
        parameters for the model.

```

```
        For multi-metric evaluation, this attribute holds the validated
        ``scoring`` dict which maps the scorer key to the scorer callable.

    n_splits_ : int
        The number of cross-validation splits (folds/iterations).

    refit_time_ : float
        Seconds used for refitting the best model on the whole dataset.

        This is present only if ``refit`` is not False.

        .. versionadded:: 0.20

    multimetric_ : bool
        Whether or not the scorers compute several metrics.

    classes_ : ndarray of shape (n_classes,)
        The classes labels. This is present only if ``refit`` is specified and
        the underlying estimator is a classifier.

    n_features_in_ : int
        Number of features seen during :term:`fit`. Only defined if
        `best_estimator_` is defined (see the documentation for the `refit`
        parameter for more details) and that `best_estimator_` exposes
        `n_features_in_` when fit.

        .. versionadded:: 0.24

    feature_names_in_ : ndarray of shape (`n_features_in_`,)
        Names of features seen during :term:`fit`. Only defined if
        `best_estimator_` is defined (see the documentation for the `refit`
        parameter for more details) and that `best_estimator_` exposes
        `feature_names_in_` when fit.

        .. versionadded:: 1.0
```

```
    See Also
    --------
    ParameterGrid : Generates all the combinations of a hyperparameter grid.
    train_test_split : Utility function to split the data into a development
        set usable for fitting a GridSearchCV instance and an evaluation set
        for its final evaluation.
    sklearn.metrics.make_scorer : Make a scorer from a performance metric or
        loss function.

    Notes
    -----
    The parameters selected are those that maximize the score of the left out
    data, unless an explicit score is passed in which case it is used instead.

    If `n_jobs` was set to a value higher than one, the data is copied for each
    point in the grid (and not `n_jobs` times). This is done for efficiency
    reasons if individual jobs take very little time, but may raise errors if
    the dataset is large and not enough memory is available.  A workaround in
    this case is to set `pre_dispatch`. Then, the memory is copied only
    `pre_dispatch` many times. A reasonable value for `pre_dispatch` is `2 *
    n_jobs`.

    Examples
    --------
    >>> from sklearn import svm, datasets
    >>> from sklearn.model_selection import GridSearchCV
    >>> iris = datasets.load_iris()
    >>> parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
    >>> svc = svm.SVC()
    >>> clf = GridSearchCV(svc, parameters)
    >>> clf.fit(iris.data, iris.target)
    GridSearchCV(estimator=SVC(),
                 param_grid={'C': [1, 10], 'kernel': ('linear', 'rbf')})
    >>> sorted(clf.cv_results_.keys())
```

```
    ['mean_fit_time', 'mean_score_time', 'mean_test_score',...
     'param_C', 'param_kernel', 'params',...
     'rank_test_score', 'split0_test_score',...
     'split2_test_score', ...
     'std_fit_time', 'std_score_time', 'std_test_score']
    """
    _required_parameters = ["estimator", "param_grid"]
    _parameter_constraints: dict = {
        **BaseSearchCV._parameter_constraints,
        "param_grid": [dict, list],
    }

File /opt/anaconda3/lib/python3.9/site-packages/sklearn/model_selection/_search.py:851, in evaluate_candidates(candidate_params, cv, more_results)
@_fit_context(
    # *SearchCV.estimator is not validated yet
    prefer_skip_nested_validation=False
)
def fit(self, X, y=None, **params):
    """Run fit with all sets of parameters.

    Parameters
    ----------

    X : array-like of shape (n_samples, n_features)
        Training vector, where `n_samples` is the number of samples and
        `n_features` is the number of features.

    y : array-like of shape (n_samples, n_output) \
        or (n_samples,), default=None
-->     Target relative to X for classification or regression;
        None for unsupervised learning.

    **params : dict of str -> object
        Parameters passed to the ``fit`` method of the estimator, the scorer,
```

```
    856         and the CV splitter.
    857 
    858         If a fit parameter is an array-like whose length is equal to
    859         `num_samples` then it will be split across CV groups along with `X`
    860         and `y`. For example, the :term:`sample_weight` parameter is split
    861         because `len(sample_weights) = len(X)`.
    862 
    863     Returns
    864     -------
    865     self : object
    866         Instance of fitted estimator.
    867     """
    868     estimator = self.estimator
    869     # Here we keep a dict of scorers as is, and only convert to a
    870     # _MultimetricScorer at a later stage. Issue:
    871     # https://github.com/scikit-learn/scikit-learn/issues/27001

File /opt/anaconda3/lib/python3.9/site-packages/sklearn/model_selection/_validation.py:367, in _w
arn_or_raise_about_fit_failures(results, error_score)
    360     scorers = _check_multimetric_scoring(estimator, scoring)
    362 if _routing_enabled():
    363     # `cross_validate` will create a `_MultiMetricScorer` if `scoring` is a
    364     # dict at a later stage. We need the same object for the purpose of
    365     # routing. However, creating it here and passing it around would create
    366     # a much larger diff since the dict is used in many places.
--> 367     if isinstance(scorers, dict):
    368         _scorer = _MultimetricScorer(
    369             scorers=scorers, raise_exc=(error_score == "raise")
    370         )
    371     else:

ValueError: 
All the 60 fits failed.
It is very likely that your model is misconfigured.
You can try to debug the error by setting error_score='raise'.
```

```
Below are more details about the failures:
--------------------------------------------------------------------------------
60 fits failed with the following error:
Traceback (most recent call last):
  File "/opt/anaconda3/lib/python3.9/site-packages/sklearn/model_selection/_validation.py", line
686, in _fit_and_score
  File "/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_coordinate_descent.py",
line 908, in fit
    ----------
  File "/opt/anaconda3/lib/python3.9/site-packages/sklearn/base.py", line 548, in _validate_data
    Parameters
  File "/opt/anaconda3/lib/python3.9/site-packages/sklearn/base.py", line 415, in _check_feature_
names
    It is recommended to call reset=True in `fit` and in the first
  File "/opt/anaconda3/lib/python3.9/site-packages/sklearn/utils/validation.py", line 1903, in _g
et_feature_names
    # Check that there are no significant negative eigenvalues
TypeError: Feature names are only supported if all input features have string names, but your inp
ut has ['int', 'str'] as feature name / column name types. If you want feature names to be stored
and validated, you must convert them all to strings, by using X.columns = X.columns.astype(str) f
or example. Otherwise you can remove feature / column names from your input data, or convert them
all to a non-string data type.
```

Adding TF-IDF features only marginally improved the kappa score