

term_term_pr_distr_and_inference-multinomial-sum0

December 7, 2020

autoreload modules and utilities

```
[1]: %load_ext autoreload
    %autoreload 2
```

import all necessary libraries/packages

```
[21]: import joblib

import numpy as np
import pandas as pd

from tqdm.notebook import tqdm
import matplotlib.pyplot as plt

from scipy.stats import entropy as calculate_entropy

from sklearn.datasets import fetch_20newsgroups
from sklearn.model_selection import StratifiedShuffleSplit

from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.naive_bayes import MultinomialNB, GaussianNB
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import f1_score as calculate_f1_score
from sklearn.model_selection import train_test_split, StratifiedKFold
```

Utility functions

```
[3]: z = np.random.randint(20, size=(3, 5))
    z
```

```
[3]: array([[ 3, 17,  0,  0,  9],
           [11, 11, 19, 18,  2],
           [15,  5, 14,  9, 10]])
```

```
[4]: np.array(map(lambda x: x, [1,2,3]))
```

```
[4]: array(<map object at 0x0000021FE46718B0>, dtype=object)
```

```
[5]: ## utilities
    # from utils import clean_text

    import string

    from sklearn.base import TransformerMixin

    import nltk
    from nltk import word_tokenize
    from nltk.stem import WordNetLemmatizer

    nltk.download('stopwords')
    nltk.download('wordnet')

    def clean_text(text: str, lemmatizer = lambda x: x) -> str:
        # removes upper cases
        text = text.lower().strip()

        # removes punctuation
        for char in string.punctuation:
            text = text.replace(char, " ")

        #lemmatize the words and join back into string text
        text = " ".join([lemmatizer(word) for word in word_tokenize(text)])
        return text

    def calculate_sparsity(matrix):
        non_zero = np.count_nonzero(matrix)
        total_val = np.product(matrix.shape)
        sparsity = (total_val - non_zero) / total_val
        return sparsity

    def data_isvalid(text, analyser, min_character_size, max_character_size):
        return min_character_size <= len(analyser(text)) <= max_character_size

    def get_pipeline(vectorizer_type, classifier, use_t2pi, min_df=3,
        ↪ stop_words=None, lemmatizer = lambda x: x):
        vectorizer = CountVectorizer if vectorizer_type == "count" else
        ↪ TfidfVectorizer
        models = [
            ('clean_text', CleanTextTransformer(lemmatizer)),
```

```

        ("vectorizers", FeatureUnion([
            ('count_binary', CountVectorizer(stop_words=stop_words,
→binary=True, min_df=min_df)),
            ("count", vectorizer(stop_words=stop_words, min_df=min_df))
        ])),
    ]

    if use_t2pi:
        models.append(('t2pi_transformer', T2PITransformer()))

    models.append(('classifier', classifier))
    return Pipeline(models)

class CleanTextTransformer(TransformerMixin):
    def __init__(self, lemmatizer):
        self._lemmatizer = lemmatizer

    def fit(self, X, y=None, **fit_params):
        return self

    def transform(self, X, y=None, **fit_params):
        return np.vectorize(lambda x: clean_text(x, self._lemmatizer))(X)

    def __str__(self):
        return "CleanTextTransformer()"

    def __repr__(self):
        return self.__str__()

class T2PITransformer(TransformerMixin):
    @staticmethod
    def _max_weight(x, pbar, word_word_pr_distr_prime):
        pbar.update(1)
        return word_word_pr_distr_prime.apply(lambda y: x*y, axis=0).max(0)

    @staticmethod
    def _sum_weight(x, pbar, word_word_pr_distr_prime):
        pbar.update(1)
        return word_word_pr_distr_prime.apply(lambda y: x*y, axis=0).sum(0)

    @staticmethod
    def _weighted_mean_weight(x, pbar, word_word_pr_distr_prime):
        pbar.update(1)
        xt = word_word_pr_distr_prime.apply(lambda y: x*y, axis=0)

```

```

        return (xt * (xt / xt.sum(0))).sum(0)

def fit(self, X, y=None, **fit_params):
    X = X[:, :int(X.shape[1]/2)].toarray()

    print("creating term-term co-occurrence pr matrix")
    terms = np.arange(X.shape[1])

    X = pd.DataFrame(X, columns=terms)
    self.word_word_pr_distr = pd.DataFrame(data=0.0, columns=terms,
→index=terms)

    for term in tqdm(terms):
        self.word_word_pr_distr[term] = X[X[term] > 0].sum(0) / X.sum(0)
#         self.word_word_pr_distr[term] = X[X[term] > 0].sum(0) / X[term].
→sum()

    return self

def transform(self, X, y=None, **fit_params):
    X = X[:, int(X.shape[1]/2):].toarray()
    X = pd.DataFrame(X, columns=self.word_word_pr_distr.columns)

    print("transforming ...")

    # new_sparsity after transform
    sparsity_before = calculate_sparsity(X)

    with tqdm(total=X.shape[0]) as pbar:
        X = X.apply(self._sum_weight, axis=1, args=(pbar, self.
→word_word_pr_distr))

    # new_sparsity after transform
    sparsity_after = calculate_sparsity(X)

    print("sparsity(X):")
    print(f"=> before {sparsity_before:.4f}")
    print(f"=> after {sparsity_after:.4f}")
    print()

    return X

def __str__(self):
    return "T2PITransformer()"

def __repr__(self):
    return self.__str__()

```

```
[nltk_data] Downloading package stopwords to
[nltk_data] C:\Users\christian\AppData\Roaming\nltk_data...
[nltk_data] Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data] C:\Users\christian\AppData\Roaming\nltk_data...
[nltk_data] Package wordnet is already up-to-date!
```

1 Load Data

```
[6]: # total number of samples needed
randomize = False

# retrieve dataset
categories = ['rec.autos', 'talk.politics.mideast', 'alt.atheism', 'sci.space']

all_docs = fetch_20newsgroups(subset='train', shuffle=randomize,
    ↪remove=('headers', 'footers', 'quotes'), categories=categories)
categories = all_docs.target_names
```

```
[7]: print(all_docs.data[0])
```

I think that domestication will change behavior to a large degree. Domesticated animals exhibit behaviors not found in the wild. I don't think that they can be viewed as good representatives of the wild animal kingdom, since they have been bred for thousands of years to produce certain behaviors, etc.

1.0.1 Create Dataframe

```
[8]: data = pd.DataFrame(
    data={
        "text":all_docs.data,
        "label":all_docs.target
    }
)

data.head()
```

```
[8]:
```

	text	label
0	\n\nI think that domestication will change beh...	0
1	\nI don't like this comment about "Typical" th...	3
2	\n<apparently you're not a woman - my husband ...	1
3	While not exactly a service incident, I had a ...	1
4	\n\nI think I can. Largely as a result of effo...	2

1.0.2 Label Frequency

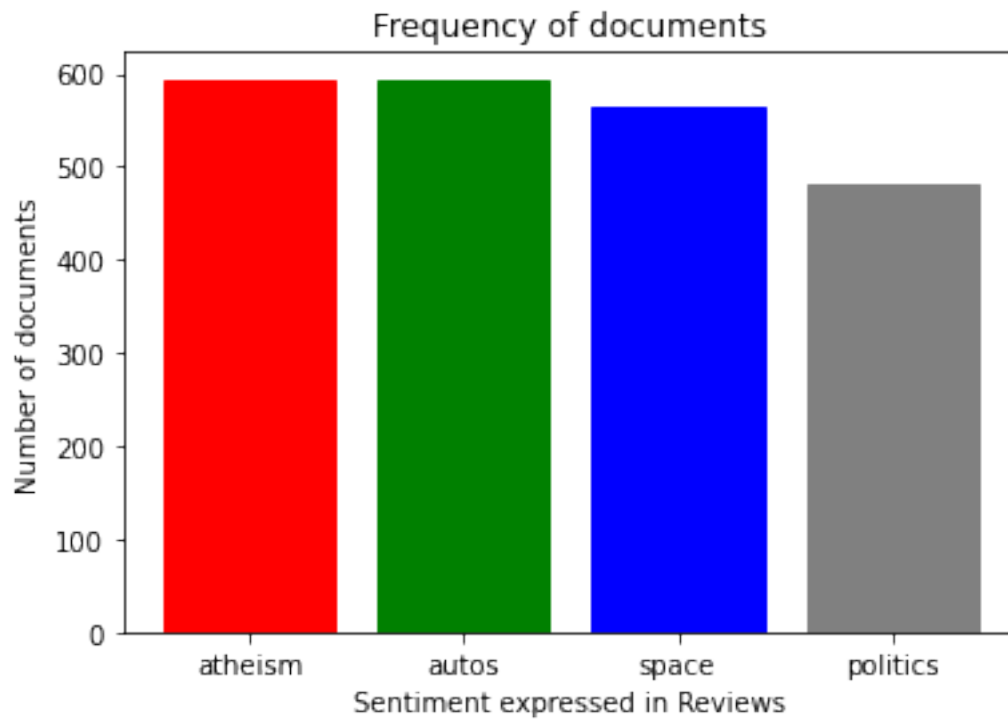
```
[9]: print(data["label"].value_counts())
      print()

      barlist = plt.bar(categories, data["label"].value_counts())

      plt.title("Frequency of documents")
      plt.xticks(categories, list(map(lambda x: x.split(".")[1], categories)))
      plt.ylabel('Number of documents')
      plt.xlabel('Sentiment expressed in Reviews')

      barlist[0].set_color('red')
      barlist[1].set_color('green')
      barlist[2].set_color('blue')
      barlist[3].set_color('grey')
      plt.show()
```

```
1    594
2    593
3    564
0    480
Name: label, dtype: int64
```



The Dataset labels needs to be balanced

1.0.3 Parameters

```
[11]: min_df = 3
      stop_words = "english"

      def get_classifier():
          # return GaussianNB()
          return MultinomialNB()

      def get_lemmatizer():
          # return WordNetLemmatizer().lemmatize
          return lambda x: x
```

2 Select Valid Data

```
[12]: max_size_per_class = 150

      # remove long text
      indices = data["text"].apply(data_isvalid, args=(lambda x: clean_text(x,
          ↪get_lemmatizer()), 256, 512))
      data = data[indices]

      # make classes balanced
      class_indices = []

      for index in range(4):
          class_indices.append(np.where((data["label"] == index))[0])

      size_per_class = min(max_size_per_class, min(map(len, class_indices)))
      indices = np.concatenate([class_ids[:size_per_class] for class_ids in
          ↪class_indices])

      data = data.iloc[indices]

      data.head()
```

```
[12]:
```

	text	label
0	\n\nI think that domestication will change beh...	0
30	\n[rest deleted...]\n\nYou were a liberal arts...	0
36	\nWorse? Maybe not, but it is definately a vi...	0
63	\nCould you expand on your definition of knowi...	0
65	\nLooking at historical evidence such 'perfect...	0

```
[13]: print(data.iloc[0]["text"])
```

I think that domestication will change behavior to a large degree. Domesticated animals exhibit behaviors not found in the wild. I don't think that they can be viewed as good representatives of the wild animal kingdom, since they have been bred for thousands of years to produce certain behaviors, etc.

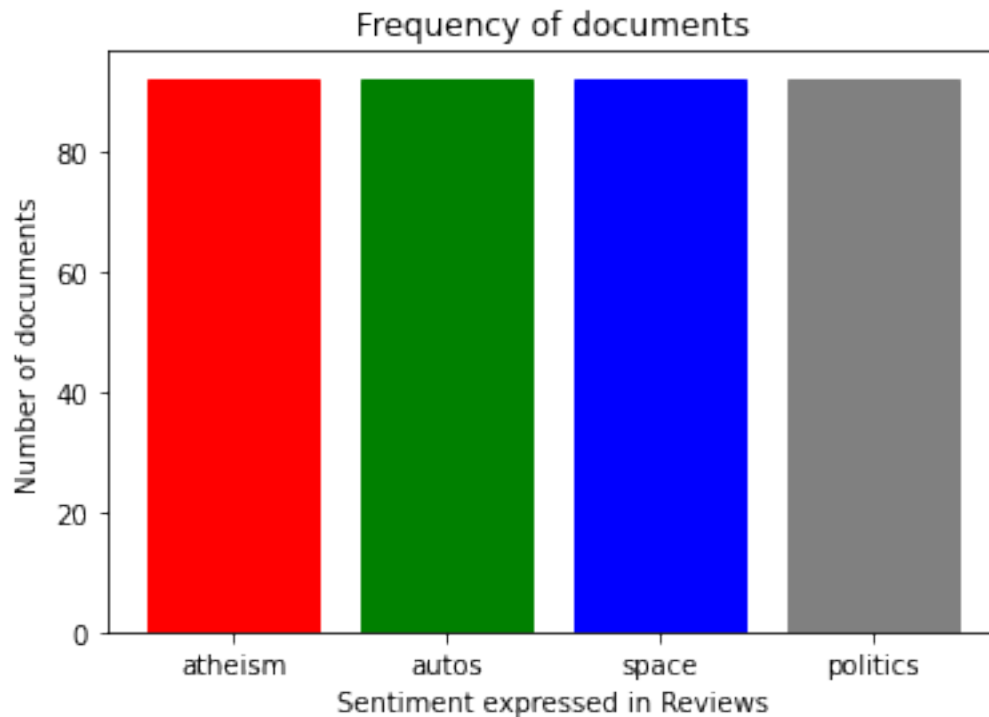
```
[14]: print(data["label"].value_counts())
      print()

      barlist = plt.bar(categories, data["label"].value_counts())

      plt.title("Frequency of documents")
      plt.xticks(categories, list(map(lambda x: x.split(".")[1], categories)))
      plt.ylabel('Number of documents')
      plt.xlabel('Sentiment expressed in Reviews')

      barlist[0].set_color('red')
      barlist[1].set_color('green')
      barlist[2].set_color('blue')
      barlist[3].set_color('grey')
      plt.show()
```

```
3    92
2    92
1    92
0    92
Name: label, dtype: int64
```

2.0.1 initialize input and output

```
[15]: X = data["text"]
      y = data['label']

      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
      ↪random_state=42)
```

2.0.2 initialize recursive word infer model

```
[16]: # initialize model
      t2pi_model = get_pipeline("count", get_classifier(), use_t2pi=True,
      ↪min_df=min_df, stop_words=None, lemmatizer = get_lemmatizer())
      t2pi_model

[16]: Pipeline(steps=[('clean_text', CleanTextTransformer()),
                      ('vectorizers',
                       FeatureUnion(transformer_list=[('count_binary',
                                                       CountVectorizer(binary=True,
                                                                    min_df=3)),
                                                       ('count',
                                                        CountVectorizer(min_df=3))])),
                      ('t2pi_transformer', T2PITransformer()),
```

```
('classifier', MultinomialNB())])
```

```
[17]: # fit model
t2pi_model.fit(X_train, y_train)
```

creating term-term co-occurrence pr matrix

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=864.0),
↳HTML(value='')))
```

transforming ...

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=276.0),
↳HTML(value='')))
```

```
sparsity(X):
=> before 0.9578
=> after 0.0000
```

```
[17]: Pipeline(steps=[('clean_text', CleanTextTransformer()),
                      ('vectorizers',
                       FeatureUnion(transformer_list=[('count_binary',
                                                       CountVectorizer(binary=True,
                                                                       min_df=3)),
                                                       ('count',
                                                        CountVectorizer(min_df=3))])),
                      ('t2pi_transformer', T2PITransformer()),
                      ('classifier', MultinomialNB())])
```

```
[18]: y_pred = t2pi_model.predict(X_test) #predict testing data

print(classification_report(y_test, y_pred))
```

transforming ...

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=92.0),
↳HTML(value='')))
```

```
sparsity(X):
=> before 0.9606
=> after 0.0000
```

	precision	recall	f1-score	support
0	0.79	0.73	0.76	30
1	0.59	0.84	0.70	19

	2	0.86	0.57	0.69	21
	3	0.65	0.68	0.67	22
accuracy				0.71	92
macro avg		0.72	0.71	0.70	92
weighted avg		0.73	0.71	0.71	92

2.0.3 Initialize models

```
[22]: # normal model
count_model = get_pipeline("count", get_classifier(), use_t2pi=False,
    ↳min_df=min_df, stop_words=None, lemmatizer = get_lemmatizer())
count_sw_model = get_pipeline("count", get_classifier(), use_t2pi=False,
    ↳min_df=min_df, stop_words=stop_words, lemmatizer = get_lemmatizer())

tfidf_model = get_pipeline("tfidf", get_classifier(), use_t2pi=False,
    ↳min_df=min_df, stop_words=None, lemmatizer = get_lemmatizer())
tfidf_sw_model = get_pipeline("tfidf", get_classifier(), use_t2pi=False,
    ↳min_df=min_df, stop_words=stop_words, lemmatizer = get_lemmatizer())

# model
t2pi_count_model = get_pipeline("count", get_classifier(), use_t2pi=True,
    ↳min_df=min_df, stop_words=None, lemmatizer = get_lemmatizer())
t2pi_count_sw_model = get_pipeline("count", get_classifier(), use_t2pi=True,
    ↳min_df=min_df, stop_words=stop_words, lemmatizer = get_lemmatizer())

t2pi_tfidf_model = get_pipeline("tfidf", get_classifier(), use_t2pi=True,
    ↳min_df=min_df, stop_words=None, lemmatizer = get_lemmatizer())
t2pi_tfidf_sw_model = get_pipeline("tfidf", get_classifier(), use_t2pi=True,
    ↳min_df=min_df, stop_words=stop_words, lemmatizer = get_lemmatizer())

models = {
    "count_model": count_model,
    "count_sw_model": count_sw_model,
    "tfidf_model": tfidf_model,
    "tfidf_sw_model": tfidf_sw_model,
    "t2pi_count_model": t2pi_count_model,
    "t2pi_count_sw_model": t2pi_count_sw_model,
    "t2pi_tfidf_model": t2pi_tfidf_model,
    "t2pi_tfidf_sw_model": t2pi_tfidf_sw_model
}
```

2.0.4 Running Cross validation on all Models

```
[23]: split_size = 3
skf = StratifiedKFold(n_splits=split_size, shuffle=True, random_state=100)

index = 0
macro_f1_scores, weighted_f1_scores, accuracies = [], [], []

for train_index, test_index in skf.split(X, y):
    index += 1

    x_train_fold, x_test_fold = X.iloc[train_index], X.iloc[test_index]
    y_train_fold, y_test_fold = y.iloc[train_index], y.iloc[test_index]

    accuracies.append([])
    macro_f1_scores.append([])
    weighted_f1_scores.append([])

    for model_name, model in models.items():
        print(f'-> {index}. {model_name} \n{"="*100}\n')
        model.fit(x_train_fold, y_train_fold)
        y_pred = model.predict(x_test_fold)

        accuracy = accuracy_score(y_test_fold, y_pred)
        weighted_f1_score = calculate_f1_score(y_test_fold, y_pred,
        ↳average='weighted')
        macro_f1_score = calculate_f1_score(y_test_fold, y_pred,
        ↳average='macro')

        weighted_f1_scores[-1].append(weighted_f1_score)
        macro_f1_scores[-1].append(macro_f1_score)
        accuracies[-1].append(accuracy)
```

```
-> 1. count_model
```

```
=====
```

```
-> 1. count_sw_model
```

```
=====
```

```
-> 1. tfidf_model
```

```
=====
```

```
-> 1. tfidf_sw_model
```

```
=====
```

```

-> 1. t2pi_count_model
=====

creating term-term co-occurrence pr matrix
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=765.0),
↳HTML(value='')))

transforming ...
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=245.0),
↳HTML(value='')))

sparsity(X):
=> before 0.9539
=> after 0.0000

transforming ...
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=123.0),
↳HTML(value='')))

sparsity(X):
=> before 0.9561
=> after 0.0000

-> 1. t2pi_count_sw_model
=====

creating term-term co-occurrence pr matrix
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=567.0),
↳HTML(value='')))

transforming ...
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=245.0),
↳HTML(value='')))

sparsity(X):
=> before 0.9769
=> after 0.1902

transforming ...

```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=123.0),  
↳HTML(value='')))
```

```
sparsity(X):  
=> before 0.9813  
=> after 0.2021
```

```
-> 1. t2pi_tfidf_model
```

```
=====
```

```
creating term-term co-occurrence pr matrix
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=765.0),  
↳HTML(value='')))
```

```
transforming ...
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=245.0),  
↳HTML(value='')))
```

```
sparsity(X):  
=> before 0.9539  
=> after 0.0000
```

```
transforming ...
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=123.0),  
↳HTML(value='')))
```

```
sparsity(X):  
=> before 0.9561  
=> after 0.0000
```

```
-> 1. t2pi_tfidf_sw_model
```

```
=====
```

```
creating term-term co-occurrence pr matrix
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=567.0),  
↳HTML(value='')))
```

```
transforming ...
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=245.0),  
↳HTML(value='')))
```

```
sparsity(X):
=> before 0.9769
=> after 0.1902
```

transforming ...

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=123.0),  
↳HTML(value='')))
```

```
sparsity(X):
=> before 0.9813
=> after 0.2021
```

```
-> 2. count_model
```

```
=====
```

```
-> 2. count_sw_model
```

```
=====
```

```
-> 2. tfidf_model
```

```
=====
```

```
-> 2. tfidf_sw_model
```

```
=====
```

```
-> 2. t2pi_count_model
```

```
=====
```

creating term-term co-occurrence pr matrix

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=805.0),  
↳HTML(value='')))
```

transforming ...

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=245.0),  
↳HTML(value='')))
```

```
sparsity(X):
=> before 0.9552
=> after 0.0001
```

```

transforming ...

HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=123.0),
↳HTML(value='')))

sparsity(X):
=> before 0.9587
=> after 0.0000

-> 2. t2pi_count_sw_model
=====

creating term-term co-occurrence pr matrix

HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=614.0),
↳HTML(value='')))

transforming ...

HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=245.0),
↳HTML(value='')))

sparsity(X):
=> before 0.9778
=> after 0.2109

transforming ...

HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=123.0),
↳HTML(value='')))

sparsity(X):
=> before 0.9823
=> after 0.2030

-> 2. t2pi_tfidf_model
=====

creating term-term co-occurrence pr matrix

HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=805.0),
↳HTML(value='')))

transforming ...

```



```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=245.0),  
↳HTML(value='')))
```

```
sparsity(X):  
=> before 0.9552  
=> after 0.0001
```

transforming ...

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=123.0),  
↳HTML(value='')))
```

```
sparsity(X):  
=> before 0.9587  
=> after 0.0000
```

-> 2. t2pi_tfidf_sw_model

```
=====
```

creating term-term co-occurrence pr matrix

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=614.0),  
↳HTML(value='')))
```

transforming ...

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=245.0),  
↳HTML(value='')))
```

```
sparsity(X):  
=> before 0.9778  
=> after 0.2109
```

transforming ...

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=123.0),  
↳HTML(value='')))
```

```
sparsity(X):  
=> before 0.9823  
=> after 0.2030
```

-> 3. count_model

```
=====
```

```

-> 3. count_sw_model
=====

-> 3. tfidf_model
=====

-> 3. tfidf_sw_model
=====

-> 3. t2pi_count_model
=====

creating term-term co-occurrence pr matrix
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=778.0),
↳HTML(value='')))

transforming ...
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=246.0),
↳HTML(value='')))

sparsity(X):
=> before 0.9531
=> after 0.0000

transforming ...
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=122.0),
↳HTML(value='')))

sparsity(X):
=> before 0.9590
=> after 0.0001

-> 3. t2pi_count_sw_model
=====

creating term-term co-occurrence pr matrix
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=578.0),
↳HTML(value='')))

```

transforming ...

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=246.0),  
↳HTML(value='')))
```

sparsity(X):

=> before 0.9766

=> after 0.1672

transforming ...

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=122.0),  
↳HTML(value='')))
```

sparsity(X):

=> before 0.9823

=> after 0.2105

-> 3. t2pi_tfidf_model

```
=====
```

creating term-term co-occurrence pr matrix

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=778.0),  
↳HTML(value='')))
```

transforming ...

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=246.0),  
↳HTML(value='')))
```

sparsity(X):

=> before 0.9531

=> after 0.0000

transforming ...

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=122.0),  
↳HTML(value='')))
```

sparsity(X):

=> before 0.9590

=> after 0.0001

-> 3. t2pi_tfidf_sw_model

```
=====
=====
```

creating term-term co-occurrence pr matrix

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=578.0),  
↳HTML(value='')))
```

transforming ...

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=246.0),  
↳HTML(value='')))
```

sparsity(X):

=> before 0.9766

=> after 0.1672

transforming ...

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=122.0),  
↳HTML(value='')))
```

sparsity(X):

=> before 0.9823

=> after 0.2105

```
[24]: model_names = list(models.keys())

accuracy = pd.DataFrame(data=np.array(accuracies), columns=model_names)
weighted_f1_score = pd.DataFrame(data=np.array(weighted_f1_scores),  
↳columns=model_names)
macro_f1_score = pd.DataFrame(data=np.array(macro_f1_scores),  
↳columns=model_names)

accuracy.loc["mean"] = accuracy.mean(0)
weighted_f1_score.loc["mean"] = weighted_f1_score.mean(0)
macro_f1_score.loc["mean"] = macro_f1_score.mean(0)
```

```
[25]: accuracy.head(split_size+1)
```

```
[25]:      count_model  count_sw_model  tfidf_model  tfidf_sw_model  \
0      0.642276      0.642276      0.634146      0.682927
1      0.780488      0.780488      0.723577      0.756098
2      0.663934      0.663934      0.672131      0.721311
mean    0.695566      0.695566      0.676618      0.720112
```

	t2pi_count_model	t2pi_count_sw_model	t2pi_tfidf_model	\
0	0.585366	0.560976	0.585366	
1	0.634146	0.674797	0.642276	
2	0.639344	0.647541	0.647541	
mean	0.619619	0.627771	0.625061	

	t2pi_tfidf_sw_model
0	0.569106
1	0.666667
2	0.655738
mean	0.630503

```
[26]: weighted_f1_score.head(split_size+1)
```

```
[26]:
```

	count_model	count_sw_model	tfidf_model	tfidf_sw_model	\
0	0.643803	0.643803	0.635956	0.686151	
1	0.780400	0.780400	0.723008	0.751996	
2	0.660720	0.660720	0.670429	0.719510	
mean	0.694974	0.694974	0.676464	0.719219	

	t2pi_count_model	t2pi_count_sw_model	t2pi_tfidf_model	\
0	0.582570	0.559007	0.585331	
1	0.620117	0.669364	0.632232	
2	0.626976	0.643110	0.638325	
mean	0.609888	0.623827	0.618629	

	t2pi_tfidf_sw_model
0	0.569144
1	0.667522
2	0.651355
mean	0.629340

```
[27]: macro_f1_score.head(split_size+1)
```

```
[27]:
```

	count_model	count_sw_model	tfidf_model	tfidf_sw_model	\
0	0.644194	0.644194	0.636203	0.686592	
1	0.780461	0.780461	0.723287	0.752383	
2	0.661328	0.661328	0.670800	0.720057	
mean	0.695328	0.695328	0.676764	0.719677	

	t2pi_count_model	t2pi_count_sw_model	t2pi_tfidf_model	\
0	0.582334	0.559015	0.585168	
1	0.621107	0.669932	0.632977	
2	0.627241	0.643223	0.638198	
mean	0.610227	0.624056	0.618781	

	t2pi_tfidf_sw_model
--	---------------------

0	0.569206
1	0.667676
2	0.651235
mean	0.629372

[]:

[]: