```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
from datasets import Dataset,DatasetDict
from transformers import AutoModelForSequenceClassification,AutoTokenizer
import statistics as st
from sklearn.cluster import AgglomerativeClustering, KMeans
from sklearn.metrics import accuracy_score, confusion_matrix
import time
import itertools
from sklearn.decomposition import NMF, LatentDirichletAllocation
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

import os
for dirname, _, filenames in os.walk('/MyDocument/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))


data = pd.read_csv("/MyDocument/input/learn-ai-bbc/BBC News.csv")

print(data.info())
print(data.head())

print("Total observations ", len(data))
print("Total Count of Unique Article IDs ", len(data['ArticleId'].unique()))
output:

Total observations  1490
Total Count of Unique Article IDs  1490

plt.hist(data['Category'])
plt.title("Histogram of Training Data Categories")
print("Tech the smallest category makes up this percentage:", round(len(data[data['Category'] == 'tech'])/ l
en(data),3))
print("Sport the largest category makes up this percentage:", round(len(data[data['Category'] == 'sport'])/ l
en(data),3))
categories = data['Category'].unique()


a = []
for txt in data['Text']:
    a.append(len(txt.split()))
plt.hist(a)
plt.title("Word Count")
print("Smallest Article ", min(a))
print("Largest Article " , max(a))

Tech the smallest category makes up this percentage: 0.175
Sport the largest category makes up this percentage: 0.232
```

```
a = []
for txt in data['Text']:
    a.append(len(txt.split()))
plt.hist(a)
plt.title("Word Count")
print("Smallest Article ", min(a))
print("Largest Article " , max(a))
```

```
Smallest Article  90
Largest Article  3345
```

```
no_features = 2000
text_ds = data['Text']
# NMF is able to use tf-idf
tfidf_vectorizer = TfidfVectorizer(max_df=0.95, min_df=2, max_features=no_features, stop_words='englis
h')
tfidf = tfidf_vectorizer.fit_transform(text_ds)
tfidf_feature_names = tfidf_vectorizer.get_feature_names()
```

```
def display_topics(model, feature_names, no_top_words):
    for topic_idx, topic in enumerate(model.components_):
        print ("Topic %d:" % (topic_idx))
        print (" ".join([feature_names[i]
                        for i in topic.argsort()[:-no_top_words - 1:-1]]))

no_top_words = 10

display_topics(nmf, tfidf_feature_names, no_top_words)
```

```
output:
Topic 0:
england game win said wales cup ireland play players team
Topic 1:
mr labour blair election brown party said government minister prime
Topic 2:
mobile people music said phone technology digital users phones software
Topic 3:
film best awards award actor oscar actress films festival director
Topic 4:
said growth economy year sales market bank oil economic 2004
```

```
def label_permute_compare(ytdf ,yp):
    """
    ytdf: labels dataframe object
    yp: clustering label prediction output
    n : number of labels
    labelName:  list of strings
    Returns permuted label order and accuracy.
    Example output: (3, 4, 1, 2, 0), 0.74
    """
```

```
    accuracy = 0
    answer = []
    ypDF = pd.DataFrame(yp)
    labelName = ytdf.unique()
    n = len(labelName)


   # print(labelName)
   # print(ytdf)
   # print(ypDF)
# your code here

    for i in itertools.permutations(range(n)):

        acc = accuracy_score(ytdf, ypDF.replace(i, labelName))
        if acc > accuracy:
            answer = i
            accuracy = acc

    return answer, accuracy

 %%time

def topic_word_count_unweighted(data, model, feature_names, categories):

    for i in range(len(data.Text)):
        best_cat = np.zeros([len(model.components_)])
        categories[i,0] = data.ArticleId[i]
        for j in range(len(model.components_)):
            word_index = [feature_names.index(item) for item in data.Text[i].split() if item in tfidf_feature_nam
es]
            best_cat[j] = sum([nmf.components_[j][weight] for weight in word_index])
        categories[i,1] = best_cat.argmax()

    return categories


nmf_best_cat = np.zeros([len(data.ArticleId),2])

nmf_best_cat = topic_word_count_unweighted(data, nmf, tfidf_feature_names, nmf_best_cat )

print(nmf_best_cat)

def topic_word_count_weighted(data, model, feature_names, categories):
    weights  = np.sum(nmf.components_, axis = 1)  #Total weights for each category
    for i in range(len(data.Text)):
        best_cat = np.zeros([len(model.components_)])
        categories[i,0] = data.ArticleId[i]
        for j in range(len(model.components_)):
            word_index = [feature_names.index(item) for item in data.Text[i].split() if item in tfidf_feature_nam
es]
            best_cat[j] = sum([nmf.components_[j][weight] for weight in word_index])/weights[j]  # Normalizng
by total weights for each category
        categories[i,1] = best_cat.argmax()
```

```
    return categories

nmf_best_cat_w = np.zeros([len(data.ArticleId),2])

nmf_best_cat_w = topic_word_count_weighted(data, nmf, tfidf_feature_names, nmf_best_cat_w )

print(label_permute_compare(data.Category, nmf_best_cat_w[:,1]))
plt.hist(nmf_best_cat_w[:,1])

((4, 2, 1, 0, 3), 0.885234899328859)

(array([349.,   0., 357.,   0.,   0., 278.,   0., 223.,   0., 283.]),
 array([0. , 0.4, 0.8, 1.2, 1.6, 2. , 2.4, 2.8, 3.2, 3.6, 4. ]),
 <BarContainer object of 10 artists>)

 %%time
def nmf_Grid_Search():
    beta_loss = ['frobenius', 'kullback-leibler', 'itakura-saito']
    a_w = np.logspace(-5,0, num=3, base = 2)
    a_h = np.logspace(-5,0, num=3, base = 2)


    for beta in beta_loss:
        for aw in a_w:
            for ah in a_h:
                holder =  np.zeros([len(data.ArticleId),2])
                print("Beta: " , beta, " a_w:", round(aw,4) , " a_h:", round(ah,4) )
                try:
                    model = NMF(n_components = 5,beta_loss = beta,  alpha_W=aw, alpha_H=ah, init= "nndsvd
ar").fit(tfidf)
                    holder = topic_word_count_weighted(nmf, tfidf_feature_names, holder )
                    print(label_permute_compare(data.Category, nmf_best_cat_w[:,1]))
                except Exception as e:
                    print(e)
nmf_Grid_Search()

Beta:  frobenius  a_w: 0.0312  a_h: 0.0312
topic_word_count_weighted() missing 1 required positional argument: 'categories'
Beta:  frobenius  a_w: 0.0312  a_h: 0.1768
topic_word_count_weighted() missing 1 required positional argument: 'categories'
Beta:  frobenius  a_w: 0.0312  a_h: 1.0
topic_word_count_weighted() missing 1 required positional argument: 'categories'
Beta:  frobenius  a_w: 0.1768  a_h: 0.0312
topic_word_count_weighted() missing 1 required positional argument: 'categories'
Beta:  frobenius  a_w: 0.1768  a_h: 0.1768
topic_word_count_weighted() missing 1 required positional argument: 'categories'
Beta:  frobenius  a_w: 0.1768  a_h: 1.0
topic_word_count_weighted() missing 1 required positional argument: 'categories'
Beta:  frobenius  a_w: 1.0  a_h: 0.0312
topic_word_count_weighted() missing 1 required positional argument: 'categories'
Beta:  frobenius  a_w: 1.0  a_h: 0.1768
topic_word_count_weighted() missing 1 required positional argument: 'categories'
Beta:  frobenius  a_w: 1.0  a_h: 1.0
topic_word_count_weighted() missing 1 required positional argument: 'categories'
Beta:  kullback-leibler  a_w: 0.0312  a_h: 0.0312
```

```
Invalid beta_loss parameter: solver 'cd' does not handle beta_loss = 'kullback-leibler'
Beta:  kullback-leibler  a_w: 0.0312  a_h: 0.1768
Invalid beta_loss parameter: solver 'cd' does not handle beta_loss = 'kullback-leibler'
Beta:  kullback-leibler  a_w: 0.0312  a_h: 1.0
Invalid beta_loss parameter: solver 'cd' does not handle beta_loss = 'kullback-leibler'
Beta:  kullback-leibler  a_w: 0.1768  a_h: 0.0312
Invalid beta_loss parameter: solver 'cd' does not handle beta_loss = 'kullback-leibler'
Beta:  kullback-leibler  a_w: 0.1768  a_h: 0.1768
Invalid beta_loss parameter: solver 'cd' does not handle beta_loss = 'kullback-leibler'
Beta:  kullback-leibler  a_w: 0.1768  a_h: 1.0
Invalid beta_loss parameter: solver 'cd' does not handle beta_loss = 'kullback-leibler'
Beta:  kullback-leibler  a_w: 1.0  a_h: 0.0312
Invalid beta_loss parameter: solver 'cd' does not handle beta_loss = 'kullback-leibler'
Beta:  kullback-leibler  a_w: 1.0  a_h: 0.1768
Invalid beta_loss parameter: solver 'cd' does not handle beta_loss = 'kullback-leibler'
Beta:  kullback-leibler  a_w: 1.0  a_h: 1.0
Invalid beta_loss parameter: solver 'cd' does not handle beta_loss = 'kullback-leibler'
Beta:  itakura-saito  a_w: 0.0312  a_h: 0.0312
Invalid beta_loss parameter: solver 'cd' does not handle beta_loss = 'itakura-saito'
Beta:  itakura-saito  a_w: 0.0312  a_h: 0.1768
Invalid beta_loss parameter: solver 'cd' does not handle beta_loss = 'itakura-saito'
Beta:  itakura-saito  a_w: 0.0312  a_h: 1.0
Invalid beta_loss parameter: solver 'cd' does not handle beta_loss = 'itakura-saito'
Beta:  itakura-saito  a_w: 0.1768  a_h: 0.0312
Invalid beta_loss parameter: solver 'cd' does not handle beta_loss = 'itakura-saito'
Beta:  itakura-saito  a_w: 0.1768  a_h: 0.1768
Invalid beta_loss parameter: solver 'cd' does not handle beta_loss = 'itakura-saito'
Beta:  itakura-saito  a_w: 0.1768  a_h: 1.0
Invalid beta_loss parameter: solver 'cd' does not handle beta_loss = 'itakura-saito'
Beta:  itakura-saito  a_w: 1.0  a_h: 0.0312
Invalid beta_loss parameter: solver 'cd' does not handle beta_loss = 'itakura-saito'
Beta:  itakura-saito  a_w: 1.0  a_h: 0.1768
Invalid beta_loss parameter: solver 'cd' does not handle beta_loss = 'itakura-saito'
Beta:  itakura-saito  a_w: 1.0  a_h: 1.0
Invalid beta_loss parameter: solver 'cd' does not handle beta_loss = 'itakura-saito'
CPU times: user 2.81 s, sys: 3.22 s, total: 6.03 s
Wall time: 1.54 s
```

```
test_data  = pd.read_csv("/MyDocument/input/learn-ai-bbc/BBC News Test.csv")
test = np.zeros([len(test_data.ArticleId),2])

test = topic_word_count_weighted(test_data, nmf, tfidf_feature_names,test)



plt.hist(test[:,1])
```

```
output:(array([165.,   0., 181.,   0.,   0., 136.,   0., 100.,   0., 153.]),
 array([0. , 0.4, 0.8, 1.2, 1.6, 2. , 2.4, 2.8, 3.2, 3.6, 4. ]),
 <BarContainer object of 10 artists>)
```

```
 answer = pd.DataFrame(test)
answer.columns = ["ArticleId", "Category"]
```

```
answer = answer.astype(’int64’)
answer.Category.replace( to_replace=(0, 4, 1, 2, 3),
          value=data.Category.unique(), inplace = True)

     try:
    os.remove("/MyDocument/working/result.csv") # remove csv file if it exists
except:
    pass
answer.to_csv(’result.csv’, index=False)
```