```
import numpy as np
import pandas as pd
from itertools import combinations
```

The problem is to match the user's free-form input against a pre-determined list of banks. For example, user input 'bawag bank' should be matched to 'BAWAG Group AG'.

```
# List of banks to compare
banks =   ['Sberbank Europe AG',
           'BAWAG Group AG',
           'Raiffeisenbankengruppe OÖ Verbund eGen',
           'Raiffeisen Bank International AG',
           'Volksbanken Verbund',
           'Erste Group Bank AG',
           'KBC Groep',
           'Investeringsmaatschappij Argenta',
           'Belfius Bank',
           'AXA Bank Belgium',
           'The Bank of New York Mellon SA/NV',
           'First Investment Bank AD',
           'RCB Bank Ltd',
           'Bank of Cyprus Holdings Public Limited Company',
           'Hellenic Bank Public Company Limited',
           'DekaBank Deutsche Girozentrale',
           'Erwerbsgesellschaft der S-Finanzgruppe mbH & Co. KG',
           'UBS Europe SE',
           'DEUTSCHE APOTHEKER- UND ÄRZTEBANK EG',
           'Volkswagen Bank Gesellschaft mit beschränkter Haftung',
           'Münchener Hypothekenbank eG',
           'DZ BANK AG Deutsche Zentral-Genossenschaftsbank, Frankfurt
am Main',
           'HASPA Finanzholding',
           'State Street Europe Holdings Germany S.a.r.l. & Co. KG',
           'J.P. Morgan AG',
           'DEUTSCHE BANK AKTIENGESELLSCHAFT',
           'COMMERZBANK Aktiengesellschaft',
           'Landesbank Baden-Württemberg',
           'Landesbank Hessen-Thüringen Girozentrale',
           'Norddeutsche Landesbank - Girozentrale -',
           'Deutsche Pfandbriefbank AG',
           'Aareal Bank AG',
           'Hamburg Commercial Bank AG',
           'Bayerische Landesbank',
           'Jyske Bank A/S',
           'Sydbank A/S',
           'Nykredit Realkredit A/S',
           'Danske Bank A/S',
           'Luminor Holding AS',
           'Abanca Corporacion Bancaria S.A.',
```

```
            'Banco Santander S.A.',
            'Ibercaja Banco S.A.',
            'Kutxabank S.A',
            'Unicaja Banco S.A.',
            'CaixaBank S.A.',
            'Banco de Crédito Social Cooperativo',
            'Banco Bilbao Vizcaya Argentaria S.A.',
            'Banco de Sabadell S.A.',
            'Bankinter S.A.',
            'Kuntarahoitus Oyj',
            'Nordea Bank Abp',
            'OP Osuuskunta',
            'SFIL',
            'RCI Banque',
            'Confédération Nationale du Crédit Mutuel',
            'La Banque Postale',
            'Bpifrance',
            "C.R.H. - Caisse de refinancement de l'habitat",
            'HSBC Continental Europe',
            'Groupe BPCE',
            'Groupe Crédit Agricole',
            'Société générale',
            'BNP Paribas',
            'ALPHA SERVICES AND HOLDINGS S.A.',
            'National Bank of Greece S.A.',
            'Eurobank Ergasias Services and Holdings S.A.',
            'Piraeus Financial Holdings',
            'OTP-csoport',
            'Magyar Bankholding',
            'Barclays Bank Ireland plc',
            'Citibank Holdings Ireland Limited',
            'AIB Group plc',
            'Bank of Ireland Group plc',
            'Ulster Bank Ireland Designated Activity Company',
            'Bank of America Europe Designated Activity Company',
            'Íslandsbanki hf.',
            'Landsbankinn hf.',
            'Arion banki hf',
            'Intesa Sanpaolo S.p.A.',
            'Gruppo Bancario Finecobank  ',
            'UniCredit S.p.A.',
            'Gruppo Bancario Mediolanum  ',
            'Credito Emiliano Holding S.p.A.',
            'Banco BPM SpA',
            'Banca Popolare di Sondrio, Società Cooperativa per Azioni',
            'Banca Monte dei Paschi di Siena S.p.A.',
            'CASSA CENTRALE BANCA',
            'ICCREA BANCA S.P.A.',
            'Mediobanca - Banca di Credito Finanziario S.p.A.',
```

```
          'Akcine bendrove Šiauliu bankas',
          'Precision Capital S.A.',
          'RBC Investor Services Bank S.A.',
          'J.P. Morgan Bank Luxembourg S.A.',
          'Banque Internationale à Luxembourg',
          'Banque et Caisse d´Epargne de l´Etat, Luxembourg',
          'Akciju sabiedriba "Citadele banka"',
          'MDB Group Limited',
          'Bank of Valletta Plc',
          'HSBC Bank Malta p.l.c.',
          'BNG Bank N.V.',
          'ING Groep N.V.',
          'LP Group B.V.',
          'de Volksbank N.V.',
          'ABN AMRO Bank N.V.',
          'Coöperatieve Rabobank U.A.',
          'Nederlandse Waterschapsbank N.V.',
          'Bank Polska Kasa Opieki S.A.',
          'Powszechna Kasa Oszczednosci Bank Polski S.A.',
          'LSF Nani Investments S.à r.l.',
          'Banco Comercial Português SA',
          'Caixa Geral de Depósitos SA',
          'Banca Transilvania',
          'Länförsäkringar Bank AB (publ)',
          'Kommuninvest - group',
          'Skandinaviska Enskilda Banken - group',
          'SBAB Bank AB - group',
          'Swedbank - group',
          'Svenska Handelsbanken - group',
          'Biser Topco S.à r.l.',
          'Nova Ljubljanska Banka d.d. Ljubljana']

# Examples of search strings
s1 = 'Bawag bank' # other options: 'Bawag bank', 'Erste', 'Raiffaisen
bank'

# A naive search method which you need to improve
from difflib import SequenceMatcher
res = []
for token in banks:
  res.append([s1, token, SequenceMatcher(None, s1, token).ratio()])

df2 = pd.DataFrame(res, columns=['Bank 1', 'Bank 2', 'Score'])
# The outcome is not great, for this search query 'BAWAG Group AG'
should have highest similarity
df2.sort_values(by=['Score'], ascending=[False]).head()
```

	Bank 1	Bank 2	Score
8	Bawag bank	Belfius Bank	0.454545
12	Bawag bank	RCB Bank Ltd	0.454545

```
33  Bawag bank  Bayerische Landesbank  0.451613
42  Bawag bank           Kutxabank S.A  0.434783
99  Bawag bank           BNG Bank N.V.  0.434783
```

```
 #The desired combination has a low score
idx = df2['Bank 2'].isin(['BAWAG Group AG'])

df2[idx].sort_values(by=['Score'], ascending=[False]).head()
```

```
       Bank 1          Bank 2     Score
1  Bawag bank  BAWAG Group AG  0.166667
```

Q1 Improve the similarity function for matching query entered by user with a predefined list of bank names.

```
# our improvements (preprocessing text)
from difflib import SequenceMatcher
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
import re

# Function to preprocess the text
def preprocess(text):
    # Convert to lowercase
    text = text.lower()
    # Remove special characters
    text = re.sub(r'\W', ' ', text)
    # Remove spaces
    text = re.sub(r'\s+', ' ', text)

    return text

# Preprocess the texts and query
preprocessed_texts = [preprocess(bank) for bank in banks]
preprocessed_query = preprocess(s1)

# Create a TfidfVectorizer
vectorizer = TfidfVectorizer()

# Fit and transform the preprocessed texts
tfidf_matrix = vectorizer.fit_transform(preprocessed_texts)

# Compute the cosine similarity between the query and each text
similarities = cosine_similarity(tfidf_matrix,
vectorizer.transform([preprocessed_query]))
```

```
# Create a DataFrame with the results
res = []
for i, s1 in enumerate(banks):
    similarity = similarities[i][0]
    res.append([s1, preprocessed_query, similarity])

df2 = pd.DataFrame(res, columns=["Bank 1", "Bank 2", "Score"])

# Sort by similarity score in descending order
df2_sorted = df2.sort_values(by=["Score"], ascending=False)

df2.sort_values(by=['Score'], ascending=[False]).head()
```

	Bank 1	Bank 2	Score
1	BAWAG Group AG	bawag bank	0.663232
34	Jyske Bank A/S	bawag bank	0.168777
8	Belfius Bank	bawag bank	0.168777
99	BNG Bank N.V.	bawag bank	0.168777
37	Danske Bank A/S	bawag bank	0.168777

```
 #The desired combination has now highest score
idx = df2['Bank 1'].isin(['BAWAG Group AG'])

df2[idx].sort_values(by=['Score'], ascending=[False]).head()
```

	Bank 1	Bank 2	Score
1	BAWAG Group AG	bawag bank	0.663232

Our scores have improved !!

Q2 - Sentiment Analysis

```
import pandas as pd

df= pd.read_csv('Q2 Sentiment Analysis Dataset.csv', encoding='latin-
1')
df.head()
```

	id	sentiment	date	\
0	623495523	1	Mon Dec 01 20:46:01 +0000 2014	
1	623495527	1	Mon Dec 01 21:09:50 +0000 2014	
2	623495529	1	Mon Dec 01 21:35:14 +0000 2014	
3	623495536	1	Mon Dec 01 23:55:55 +0000 2014	
4	623495537	1	Tue Dec 02 00:06:05 +0000 2014	

	text	Unnamed: 4
Unnamed: 5		
0	WTF MY BATTERY WAS 31% ONE SECOND AGO AND NOW ...	NaN

```
NaN
1  @apple Contact sync between Yosemite and iOS8 ...          NaN
NaN
2  WARNING IF YOU BUY AN IPHONE 5S UNLOCKED FROM ...          NaN
NaN
3  @Apple, For the love of GAWD, CENTER the '1'on...          NaN
NaN
4  i get the storage almost full notification lit...          NaN
NaN
```

```
df.sentiment.unique()
```

```
array(['1', '3', '5', 'not_relevant'], dtype=object)
```

```
from sklearn.preprocessing import LabelEncoder

# Label encode the 'sentiment' column
label_encoder = LabelEncoder()
df['sentiment'] = label_encoder.fit_transform(df['sentiment'])

df.sentiment.unique()
```

```
array([0, 1, 2, 3])
```

```
# see one of each sentiments retrieved
sentiments = df.sentiment.unique().tolist()

for sentiment in sentiments:
    print('sentiment=',sentiment)
    text = df[df.sentiment == sentiment].head(1).text.tolist()
    print(text)
```

```
sentiment= 0
['WTF MY BATTERY WAS 31% ONE SECOND AGO AND NOW IS 29% WTF IS THIS
@apple']
sentiment= 1
['#AAPL:The 10 best Steve Jobs emails ever...http://t.co/82G1kL94tx']
sentiment= 2
['Top 3 all @Apple #tablets. Damn right! http://t.co/RJiGn2JUuB']
sentiment= 3
['@Apple John Cantlie has been a prisoner of ISIS for 739 days, show
you have not abandoned him. Sign https://t.co/WTn4fuiJ0P']
```

Preprocessing

```
# preprocess sentiments
import re
import string
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
```

```
def preprocess(text):
    text = text.lower()
    text = re.sub(r'\d+', '', text)
    text = text.translate(str.maketrans('', '', string.punctuation))
    text = text.strip()
    text = word_tokenize(text)
    text = [word for word in text if word not in
stopwords.words('english')]
    lemmatizer = WordNetLemmatizer()
    text = [lemmatizer.lemmatize(word) for word in text]
    text = ' '.join(text)
    return text

# drop last 2 columns as they are unnamed
df = df.drop(df.columns[[-1,-2]], axis=1)
df['text'] = df['text'].apply(preprocess)

df.text.head()
```

```
0                 wtf battery one second ago wtf apple
1    apple contact sync yosemite io seriously screw...
2    warning buy iphone unlocked apple iphone use v...
3    apple love gawd center damn calendar app fixed...
4    get storage almost full notification literally...
Name: text, dtype: object
```

```
# splitting data into train and test
from sklearn.model_selection import train_test_split

train, test = train_test_split(df, test_size=0.2, random_state=42)
```

Implement following feature extraction methods.

- BoW based on raw counts **using Vectorizer**
- BoW based on TfIDF **using transformer**
- ngrams (unigrams, bigrams, trigrams)

Use following classifiers: Naïve Bayes, Logistic Regression, Random Forest, SVM, Perceptron. Calculate accuracy, Precision, Recall and F-score for all classifiers and report the results in tables. Make a table for multiclass classification results for different classification algorithms. Report both micro average and macro average of all measures.

```
from collections import defaultdict
from sklearn.feature_extraction.text import CountVectorizer,
TfidfVectorizer
from sklearn.metrics import precision_recall_fscore_support
import numpy as np
import warnings
```

```
warnings.filterwarnings('ignore')

# Initialize defaultdict of lists
results = defaultdict(list)

# List of feature extraction methods
feature_extractors = [
    ('BoW based on raw counts', CountVectorizer()),
    ('BoW based on TfIDF', TfidfVectorizer(use_idf=True)),
    ('BoW based on raw counts Unigram',
CountVectorizer(ngram_range=(1, 1))),
    ('BoW based on raw counts Bigram', CountVectorizer(ngram_range=(1,
2))),
    ('BoW based on raw counts Trigram',
CountVectorizer(ngram_range=(1, 3))),
    ('BoW based on TfIDF Unigram', TfidfVectorizer(ngram_range=(1, 1),
use_idf=True)),
    ('BoW based on TfIDF Bigram', TfidfVectorizer(ngram_range=(1, 2),
use_idf=True)),
    ('BoW based on TfIDF Trigram', TfidfVectorizer(ngram_range=(1, 3),
use_idf=True))
]

from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression, Perceptron
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

# List of models
models = [
    ('Naive Bayes', MultinomialNB()),
    ('Logistic Regression', LogisticRegression()),
    ('Random Forest', RandomForestClassifier()),
    ('Support Vector Machine', SVC()),
    ('Perceptron', Perceptron())
]

# Iterate over feature extraction methods
for extractor_name, extractor in feature_extractors:
    # Fit and transform data
    extractor.fit(train.text)
    X_train = extractor.transform(train.text)
    X_test = extractor.transform(test.text)

    # Iterate over models
    for model_name, model in models:
        model.fit(X_train, train.sentiment)
        y_pred = model.predict(X_test)

        # Get precision, recall, f1-score for each class
```

```
        precision, recall, fscore, _ =
precision_recall_fscore_support(test.sentiment, y_pred, average=None)

        # Calculate micro average (weighted average, takes class
imbalance into account)
        micro_avg_precision = np.average(precision,
weights=np.bincount(test.sentiment))
        micro_avg_recall = np.average(recall,
weights=np.bincount(test.sentiment))
        micro_avg_fscore = np.average(fscore,
weights=np.bincount(test.sentiment))

        # Calculate macro average (simple average, treats all classes
equally)
        macro_avg_precision = np.mean(precision)
        macro_avg_recall = np.mean(recall)
        macro_avg_fscore = np.mean(fscore)

        # Append results to defaultdict
        results['Feature Extractor'].append(extractor_name)
        results['Model'].append(model_name)
        results['Accuracy'].append((model.score(X_test,
test.sentiment)))
        results['Micro Avg Precision'].append(micro_avg_precision)
        results['Micro Avg Recall'].append(micro_avg_recall)
        results['Micro Avg F1 Score'].append(micro_avg_fscore)
        results['Macro Avg Precision'].append(macro_avg_precision)
        results['Macro Avg Recall'].append(macro_avg_recall)
        results['Macro Avg F1 Score'].append(macro_avg_fscore)

    print('Feature Extractor has finished:', extractor_name, '...')


# Convert defaultdict to DataFrame and send to results csv
pd.DataFrame(results).to_csv('results.csv', index=False)
```

```
Feature Extractor has finished: BoW based on raw counts ...
Feature Extractor has finished: BoW based on TfIDF ...
Feature Extractor has finished: BoW based on raw counts Unigram ...
Feature Extractor has finished: BoW based on raw counts Bigram ...
Feature Extractor has finished: BoW based on raw counts Trigram ...
Feature Extractor has finished: BoW based on TfIDF Unigram ...
Feature Extractor has finished: BoW based on TfIDF Bigram ...
Feature Extractor has finished: BoW based on TfIDF Trigram ...
```

```
for feature_extractor in df['Feature Extractor'].unique():
    print('Feature Extractor:', feature_extractor)
    # print everything after Model
    df_filtered = df[df['Feature Extractor'] ==
feature_extractor].iloc[:, 1:]
```

```
    half_length = len(df_filtered.columns) // 2
    first_half =
df_filtered.iloc[:, :half_length].to_string(index=False)
    second_half = df_filtered.iloc[:,
half_length:].to_string(index=False)
    print(first_half)
    print()
    print(second_half)
    print()

Feature Extractor: BoW based on raw counts
                Model  Accuracy  Micro Avg Precision  Micro Avg
Recall
          Naive Bayes  0.721080             0.690453
0.721080
   Logistic Regression  0.750643             0.736318
0.750643
         Random Forest  0.727506             0.716887
0.727506
Support Vector Machine  0.733933             0.742247
0.733933
           Perceptron  0.724936             0.720074
0.724936

 Micro Avg F1 Score  Macro Avg Precision  Macro Avg Recall  Macro Avg
F1 Score
           0.694620             0.470212          0.424434
0.420932
           0.728255             0.545820          0.454725
0.472255
           0.701843             0.508775          0.431936
0.447706
           0.700522             0.588395          0.420354
0.434046
           0.714630             0.553251          0.506716
0.518738

Feature Extractor: BoW based on TfIDF
                Model  Accuracy  Micro Avg Precision  Micro Avg
Recall
          Naive Bayes  0.737789             0.739775
0.737789
   Logistic Regression  0.740360             0.727273
0.740360
         Random Forest  0.726221             0.715088
0.726221
Support Vector Machine  0.742931             0.743892
0.742931
           Perceptron  0.718509             0.714300
0.718509
```

```
 Micro Avg F1 Score  Macro Avg Precision  Macro Avg Recall  Macro Avg
F1 Score
           0.699505             0.584496          0.412983
0.413118
           0.711572             0.541052          0.431920
0.443830
           0.698965             0.522850          0.427485
0.443430
           0.710965             0.576615          0.427708
0.440612
           0.711509             0.527130          0.482520
0.498652

Feature Extractor: BoW based on raw counts Unigram
                 Model  Accuracy  Micro Avg Precision  Micro Avg
Recall
           Naive Bayes  0.721080             0.690453
0.721080
   Logistic Regression  0.750643             0.736318
0.750643
         Random Forest  0.727506             0.713421
0.727506
Support Vector Machine  0.733933             0.742247
0.733933
            Perceptron  0.724936             0.720074
0.724936

 Micro Avg F1 Score  Macro Avg Precision  Macro Avg Recall  Macro Avg
F1 Score
           0.694620             0.470212          0.424434
0.420932
           0.728255             0.545820          0.454725
0.472255
           0.701674             0.500059          0.429821
0.443946
           0.700522             0.588395          0.420354
0.434046
           0.714630             0.553251          0.506716
0.518738

Feature Extractor: BoW based on raw counts Bigram
                 Model  Accuracy  Micro Avg Precision  Micro Avg
Recall
           Naive Bayes  0.737789             0.716654
0.737789
   Logistic Regression  0.746787             0.733432
0.746787
         Random Forest  0.723650             0.719530
0.723650
```

```
Support Vector Machine  0.731362             0.742478
0.731362
           Perceptron  0.736504             0.730673
0.736504

 Micro Avg F1 Score  Macro Avg Precision  Macro Avg Recall  Macro Avg
F1 Score
           0.706477             0.525813          0.427736
0.425919
           0.720930             0.539608          0.441248
0.455830
           0.696333             0.525213          0.428370
0.446579
           0.694214             0.583559          0.411327
0.421590
           0.717692             0.569816          0.464707
0.491390

Feature Extractor: BoW based on raw counts Trigram
                 Model  Accuracy  Micro Avg Precision  Micro Avg
Recall
           Naive Bayes  0.733933             0.705706
0.733933
   Logistic Regression  0.742931             0.729588
0.742931
         Random Forest  0.726221             0.724400
0.726221
Support Vector Machine  0.722365             0.744389
0.722365
           Perceptron  0.692802             0.686621
0.692802

 Micro Avg F1 Score  Macro Avg Precision  Macro Avg Recall  Macro Avg
F1 Score
           0.703398             0.500438          0.425621
0.423493
           0.716210             0.537530          0.438044
0.452679
           0.698058             0.530008          0.427060
0.445227
           0.682660             0.600274          0.401858
0.410511
           0.689143             0.506962          0.490272
0.497231

Feature Extractor: BoW based on TfIDF Unigram
                 Model  Accuracy  Micro Avg Precision  Micro Avg
Recall
           Naive Bayes  0.737789             0.739775
0.737789
```

```
   Logistic Regression  0.740360             0.727273
0.740360
         Random Forest  0.731362             0.718966
0.731362
Support Vector Machine  0.742931             0.743892
0.742931
            Perceptron  0.718509             0.714300
0.718509

 Micro Avg F1 Score  Macro Avg Precision  Macro Avg Recall  Macro Avg
F1 Score
           0.699505             0.584496          0.412983
0.413118
           0.711572             0.541052          0.431920
0.443830
           0.703162             0.517647          0.427344
0.441910
           0.710965             0.576615          0.427708
0.440612
           0.711509             0.527130          0.482520
0.498652

Feature Extractor: BoW based on TfIDF Bigram
                 Model  Accuracy  Micro Avg Precision  Micro Avg
Recall
           Naive Bayes  0.722365             0.740124
0.722365
   Logistic Regression  0.737789             0.732189
0.737789
         Random Forest  0.717224             0.703698
0.717224
Support Vector Machine  0.733933             0.731694
0.733933
            Perceptron  0.722365             0.713109
0.722365

 Micro Avg F1 Score  Macro Avg Precision  Macro Avg Recall  Macro Avg
F1 Score
           0.680267             0.589416          0.396902
0.400260
           0.707795             0.566888          0.429000
0.439542
           0.689872             0.507096          0.421582
0.436433
           0.700079             0.561151          0.417575
0.426239
           0.714874             0.523579          0.492707
0.502230

Feature Extractor: BoW based on TfIDF Trigram
```

```
                  Model  Accuracy  Micro Avg Precision  Micro Avg
Recall
           Naive Bayes  0.712082             0.756602
0.712082
   Logistic Regression  0.737789             0.731051
0.737789
         Random Forest  0.724936             0.719950
0.724936
Support Vector Machine  0.726221             0.724107
0.726221
            Perceptron  0.735219             0.724733
0.735219

 Micro Avg F1 Score  Macro Avg Precision  Macro Avg Recall  Macro Avg
F1 Score
           0.666971             0.634252          0.387512
0.391881
           0.706384             0.562617          0.425797
0.434483
           0.696174             0.539576          0.423698
0.440290
           0.690509             0.551899          0.409053
0.415930
           0.726004             0.552145          0.497246
0.516888
```