Textual Data Project: Nate Sock

- Objective: To produce an experimental evaluation of classifier performance on UN SDG labeled textual data.

In [1]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import os
import re

from matplotlib import pyplot as plt
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import make_pipeline
from sklearn import preprocessing
from sklearn.linear_model import RidgeClassifier

from sklearn import svm
from sklearn.metrics import roc_curve, auc
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import LabelBinarizer

from matplotlib import cm


from langdetect import detect

import tensorflow as tf
import tensorflow_hub as hub

import nltk.data
from nltk import word_tokenize, sent_tokenize
```

In [2]:
```python
sdg_names = pd.read_excel("./SDGtrainingset.xlsx")
sdg_names = sdg_names.drop([0,1,2], axis=0)
sdg_names = sdg_names.set_axis(["sdg", "sdg_name", "sdg_definition"],axis=1, copy=False)
target_names = sdg_names.sdg_name.tolist()
```

In [3]:
```python
embed_url = "https://tfhub.dev/google/universal-sentence-encoder/4"
embed = hub.load(embed_url)
```

In [4]:
```python
def get_text_df(file_name):
    text_df = pd.read_csv(file_name, sep = "\t", quotechar='"')
    col_names = text_df.columns.values[0].split('\t')
    text_df[col_names] = text_df[text_df.columns.values[0]].apply(lambda x: pd.Series(str(x).split("\t")))
    text_df.drop(text_df.columns.values[0],axis = 1, inplace=True)
    text_df = text_df.astype({'sdg':int, 'labels_negative': int, 'labels_positive':int, 'agreement': float}, copy=True)
    text_df = text_df.query("agreement > 0.5 and (labels_positive - labels_negative) > 2")
    text_df.reset_index(drop=True, inplace=True)
    return text_df
```

In [5]:
```python
file_name = "./osdg-community-data-v2023-01-01.csv"
text_df = get_text_df(file_name)
text_df["embedding"] = list(embed(text_df.text))
```

Count Vectorizer

- Bigram Only

In [6]:
```python
def run_classifier_count(text_df, classifier_algorithm, ngram_range):
    docs = text_df.text
    categories = text_df.sdg
    X_train, X_test, y_train, y_test = train_test_split(docs, categories, test_size=0.33, random_state=42)
    X_train_count_vectorizer = CountVectorizer(ngram_range=ngram_range, stop_words = "english", min_df = 7)
    X_train_count_vectorizer.fit(X_train)
    X_train_count_vector = X_train_count_vectorizer.transform(X_train)
    X_test_count_vector = X_train_count_vectorizer.transform(X_test)
    if classifier_algorithm == RidgeClassifier:
        clf = classifier_algorithm(tol=1e-2, solver="sparse_cg")
    elif classifier_algorithm == MLPClassifier :
        clf = MLPClassifier(random_state=1, hidden_layer_sizes = (100), max_iter=300)
    else :
        clf = classifier_algorithm()
    clf = clf.fit(X_train_count_vector, y_train)
    y_pred = clf.predict(X_test_count_vector)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='macro')
    recall = recall_score(y_test, y_pred, average='macro')
    f1 = f1_score(y_test, y_pred, average='macro')
    print(metrics.classification_report(y_test,y_pred, digits = 4))
```

In [7]:
```python
run_classifier_count(text_df, MultinomialNB, (2,2))
```
```
              precision    recall  f1-score   support

           1     0.6945    0.7733    0.7318       494
           2     0.7098    0.6267    0.6657       359
           3     0.8076    0.7931    0.8003       667
           4     0.8196    0.8657    0.8420       871
           5     0.7811    0.8231    0.8016       876
           6     0.7932    0.8378    0.8149       444
           7     0.7519    0.8296    0.7888       716
           8     0.5781    0.4090    0.4790       335
           9     0.5788    0.4832    0.5267       327
          10     0.6429    0.4532    0.5316       278
          11     0.6568    0.6493    0.6530       442
          12     0.7515    0.4960    0.5976       250
          13     0.6964    0.7963    0.7430       432
          14     0.8239    0.5431    0.6546       267
          15     0.7203    0.6144    0.6631       306
          16     0.7997    0.9489    0.8679      1077

    accuracy                         0.7503      8141
   macro avg     0.7254    0.6839    0.6976      8141
weighted avg     0.7453    0.7503    0.7430      8141
```

In [8]:
```python
run_classifier_count(text_df, RidgeClassifier, (2,2))
```
```
              precision    recall  f1-score   support

           1     0.6275    0.6377    0.6325       494
           2     0.6161    0.5766    0.5957       359
           3     0.7136    0.7211    0.7174       667
           4     0.6651    0.7979    0.7255       871
           5     0.7068    0.7568    0.7310       876
           6     0.7601    0.7635    0.7618       444
           7     0.6927    0.7179    0.7051       716
           8     0.4174    0.4000    0.4085       335
           9     0.4710    0.4465    0.4584       327
          10     0.5022    0.4065    0.4493       278
          11     0.5789    0.5973    0.5880       442
          12     0.6393    0.4680    0.5404       250
          13     0.7187    0.7037    0.7111       432
          14     0.6636    0.5393    0.5950       267
          15     0.6942    0.5490    0.6131       306
          16     0.9010    0.9044    0.9027      1077

    accuracy                         0.6852      8141
   macro avg     0.6480    0.6241    0.6335      8141
weighted avg     0.6828    0.6852    0.6821      8141
```

In [9]:
```python
run_classifier_count(text_df, MLPClassifier, (2,2))
```
```
              precision    recall  f1-score   support

           1     0.6783    0.6700    0.6741       494
           2     0.6113    0.5738    0.5920       359
           3     0.7273    0.7196    0.7234       667
           4     0.7788    0.7761    0.7775       871
           5     0.7469    0.7683    0.7575       876
           6     0.7629    0.7680    0.7654       444
           7     0.7308    0.7318    0.7313       716
           8     0.4231    0.3940    0.4080       335
           9     0.3955    0.4281    0.4112       327
          10     0.4877    0.4281    0.4559       278
          11     0.5421    0.5679    0.5547       442
          12     0.5890    0.5160    0.5501       250
          13     0.7321    0.7338    0.7329       432
          14     0.3995    0.5655    0.4682       267
          15     0.5619    0.5490    0.5554       306
          16     0.9373    0.8886    0.9123      1077

    accuracy                         0.6873      8141
   macro avg     0.6315    0.6299    0.6294      8141
weighted avg     0.6918    0.6873    0.6887      8141
```

In [10]:
```python
docs = text_df.text
categories = text_df.sdg
X_train, X_test, y_train, y_test = train_test_split(docs, categories, test_size=0.33, random_state=42)
X_train_count_vectorizer = CountVectorizer(ngram_range=(2,2), stop_words = "english", min_df = 7)
X_train_count_vectorizer.fit(X_train)
X_train_count_vector = X_train_count_vectorizer.transform(X_train)
X_test_count_vector = X_train_count_vectorizer.transform(X_test)
Count_ridge_clf = RidgeClassifier(tol=1e-2, solver="sparse_cg")
Count_ridge_clf = Count_ridge_clf.fit(X_train_count_vector, y_train)


def most_significant_feature_for_class(vectorizer, vector, classifier, classlabel, n=10):
    for labelid in classlabel:
        feature_names = vectorizer.get_feature_names_out()
        average_feature_effects = classifier.coef_ * np.asarray(vector.mean(axis=0)).ravel()
        top_n = sorted(zip(average_feature_effects[labelid-1], feature_names), reverse=True)[:n]
        bottom_n = sorted(zip(average_feature_effects[labelid-1], feature_names))[:n]

        for coef, feat in top_n:
            print("SDG {} : {:30}  {:.6}".format(labelid, feat, coef))
        print("")
        for coef, feat in bottom_n:
            print("SDG {} : {:30}  {:.6}".format(labelid, feat, coef))
        print("")
```

In [11]:
```python
most_significant_feature_for_class(X_train_count_vectorizer, X_train_count_vector, Count_ridge_clf, [2, 5, 8, 16], n=10)
```
```
SDG 2 : food security                  0.00526276
SDG 2 : food insecurity                0.00274086
SDG 2 : agricultural production        0.00151797
SDG 2 : agricultural sector            0.0012514
SDG 2 : self sufficiency               0.00109777
SDG 2 : agricultural land              0.00103107
SDG 2 : viet nam                       0.000891637
SDG 2 : risk management                0.000857732
SDG 2 : agricultural growth            0.000813399
SDG 2 : agricultural research          0.000783235

SDG 2 : climate change                 -0.00218814
SDG 2 : climate finance                -0.00100705
SDG 2 : gender equality                -0.000947397
SDG 2 : rural areas                    -0.000921252
SDG 2 : human rights                   -0.000914203
SDG 2 : labour market                  -0.00086672
SDG 2 : energy efficiency              -0.000803188
SDG 2 : income inequality              -0.000731607
SDG 2 : land use                       -0.000730061
SDG 2 : united states                  -0.000659742

SDG 5 : gender equality                0.0117784
SDG 5 : women men                      0.00456107
SDG 5 : violence women                 0.00334518
SDG 5 : men women                      0.00299341
SDG 5 : women entrepreneurs            0.00278221
SDG 5 : gender inequality              0.00240459
SDG 5 : gender gap                     0.00240222
SDG 5 : women rights                   0.00222067
SDG 5 : force participation            0.00196161
SDG 5 : gender mainstreaming           0.00160507

SDG 5 : health care                    -0.00200167
SDG 5 : climate change                 -0.00199052
SDG 5 : international law              -0.00122904
SDG 5 : poverty reduction              -0.00117878
SDG 5 : long term                      -0.00106812
SDG 5 : public health                  -0.00101421
SDG 5 : sub saharan                    -0.000882774
SDG 5 : united kingdom                 -0.00087554
SDG 5 : developed countries            -0.000831374
SDG 5 : education training             -0.000828634

SDG 8 : labour market                  0.00533858
SDG 8 : young people                   0.00191303
SDG 8 : labour force                   0.00169507
SDG 8 : oecd countries                 0.0014178
SDG 8 : job creation                   0.00126846
SDG 8 : job search                     0.00100935
SDG 8 : self employed                  0.0010075
SDG 8 : unemployment rates             0.00100324
SDG 8 : tourism sector                 0.000926456
SDG 8 : pay gap                        0.000866169

SDG 8 : et al                          -0.00112985
SDG 8 : gender equality                -0.00112149
SDG 8 : energy efficiency              -0.00100036
SDG 8 : climate change                 -0.000996482
SDG 8 : socio economic                 -0.000931611
SDG 8 : income inequality              -0.000888348
SDG 8 : human rights                   -0.000742832
SDG 8 : women labour                   -0.000726595
SDG 8 : food security                  -0.000720041
SDG 8 : health care                    -0.000709475

SDG 16 : human rights                   0.0172816
SDG 16 : international law              0.0117484
SDG 16 : supreme court                  0.00607334
SDG 16 : rule law                       0.00479496
SDG 16 : law enforcement                0.00444877
SDG 16 : public policy                  0.00416046
SDG 16 : international relations        0.00384731
SDG 16 : criminal law                   0.00344761
SDG 16 : political economy              0.00333831
SDG 16 : article examines               0.00327758

SDG 16 : et al                          -0.0021706
SDG 16 : international human            -0.00156099
SDG 16 : oecd countries                 -0.00150052
SDG 16 : climate change                 -0.00148848
SDG 16 : gender equality                -0.00140905
SDG 16 : labour market                  -0.00117403
SDG 16 : climate finance                -0.000956442
SDG 16 : public private                 -0.000861813
SDG 16 : civil political                -0.000836402
SDG 16 : health care                    -0.00082088
```

- Human rights is significant for three of these groups, negative for SDG 1 and 8, and positive for SDG 16.
- Interesting that it is not found for SDG 5 because it seems to be addressing discrimination and poverty issues.

In [12]:
```python
classifiers = {
    'RidgeClassifier': RidgeClassifier(),
    'MLPClassifier': MLPClassifier(),
    'MultinomialNB': MultinomialNB()
}

results_df = pd.DataFrame({
    'Classifier': ['RidgeClassifier', 'MLPClassifier', 'MultinomialNB'],
    'Accuracy': [0.6852, 0.6873, 0.7503],
    'Precision': [0.6480, 0.6315, 0.7254],
    'Recall': [0.6241, 0.6299, 0.6839],
    'F1 Score': [0.6335, 0.6294, 0.6976]
})

results_df
```

Out[12]:

	Classifier	Accuracy	Precision	Recall	F1 Score
0	RidgeClassifier	0.6852	0.6480	0.6241	0.6335
1	MLPClassifier	0.6873	0.6315	0.6299	0.6294
2	MultinomialNB	0.7503	0.7254	0.6839	0.6976

- Metrics would be stronger if we had a lower min_df value, however this is the only way we can maintain a reasonable computing time. Even a min_df=5 took extremely long to run for me, so I had to raise it to 7.
- MultinomoialNB seems to be the best performer here

Count Vectorizer

- Unigram Only

In [13]:
```python
run_classifier_count(text_df, MultinomialNB, (1,1))
```
```
              precision    recall  f1-score   support

           1     0.7458    0.8077    0.7755       494
           2     0.7847    0.8022    0.7934       359
           3     0.9274    0.8816    0.9039       667
           4     0.9334    0.9173    0.9253       871
           5     0.8821    0.8710    0.8765       876
           6     0.8689    0.8806    0.8747       444
           7     0.8723    0.8869    0.8795       716
           8     0.6300    0.5642    0.5953       335
           9     0.7311    0.7982    0.7632       327
          10     0.6360    0.5719    0.6023       278
          11     0.7970    0.8439    0.8198       442
          12     0.8720    0.7360    0.7983       250
          13     0.7178    0.8125    0.7622       432
          14     0.9237    0.8614    0.8915       267
          15     0.8630    0.8235    0.8428       306
          16     0.9560    0.9675    0.9617      1077

    accuracy                         0.8481      8141
   macro avg     0.8213    0.8142    0.8166      8141
weighted avg     0.8491    0.8481    0.8478      8141
```

In [14]:
```python
run_classifier_count(text_df, RidgeClassifier, (1,1))
```
```
              precision    recall  f1-score   support

           1     0.7674    0.7814    0.7743       494
           2     0.8227    0.7883    0.8051       359
           3     0.8654    0.8771    0.8712       667
           4     0.8524    0.9150    0.8826       871
           5     0.8363    0.8984    0.8663       876
           6     0.8817    0.8896    0.8857       444
           7     0.8356    0.8589    0.8471       716
           8     0.5886    0.5552    0.5714       335
           9     0.7000    0.7064    0.7032       327
          10     0.6578    0.5324    0.5885       278
          11     0.7479    0.7919    0.7692       442
          12     0.8364    0.7360    0.7830       250
          13     0.7892    0.8148    0.8018       432
          14     0.9522    0.8951    0.9228       267
          15     0.8410    0.7778    0.8081       306
          16     0.9657    0.9136    0.9389      1077

    accuracy                         0.8304      8141
   macro avg     0.8088    0.7958    0.8012      8141
weighted avg     0.8301    0.8304    0.8294      8141
```

In [15]:
```python
run_classifier_count(text_df, MLPClassifier, (1,1))
```
```
              precision    recall  f1-score   support

           1     0.8080    0.8178    0.8129       494
           2     0.8525    0.8691    0.8607       359
           3     0.9188    0.9160    0.9174       667
           4     0.9236    0.9437    0.9336       871
           5     0.8896    0.9201    0.9046       876
           6     0.8993    0.9054    0.9024       444
           7     0.8828    0.8939    0.8883       716
           8     0.6429    0.6179    0.6301       335
           9     0.7771    0.7676    0.7723       327
          10     0.6391    0.6115    0.6250       278
          11     0.8109    0.8439    0.8271       442
          12     0.8341    0.7640    0.7975       250
          13     0.8435    0.8611    0.8522       432
          14     0.9569    0.9139    0.9349       267
          15     0.9031    0.8529    0.8773       306
          16     0.9716    0.9536    0.9625      1077

    accuracy                         0.8713      8141
   macro avg     0.8471    0.8408    0.8437      8141
weighted avg     0.8709    0.8713    0.8709      8141
```

In [16]:
```python
classifiers = {
    'RidgeClassifier': RidgeClassifier(),
    'MLPClassifier': MLPClassifier(),
    'MultinomialNB': MultinomialNB()
}

results_df = pd.DataFrame({
    'Classifier': ['RidgeClassifier', 'MLPClassifier', 'MultinomialNB'],
    'Accuracy': [0.8304, 0.8713, 0.8481],
    'Precision': [0.8088, 0.8471, 0.8213],
    'Recall': [0.7958, 0.8408, 0.8142],
    'F1 Score': [0.8012, 0.8437, 0.8166]
})

results_df
```

Out[16]:

	Classifier	Accuracy	Precision	Recall	F1 Score
0	RidgeClassifier	0.8304	0.8088	0.7958	0.8012
1	MLPClassifier	0.8713	0.8471	0.8408	0.8437
2	MultinomialNB	0.8481	0.8213	0.8142	0.8166

- As we expect, metrics are much better when doing unigram only. Suprisingly, MLPClassifier performed the worst on bigram only, and the best for unigram only.

Tfidf Vectorizer

- Unigram and Bigram

In [17]:
```python
def run_classifier_tfidf(text_df, classifier_algorithm, ngram_range):
    docs = text_df.text
    categories = text_df.sdg
    X_train, X_test, y_train, y_test = train_test_split(docs, categories, test_size=0.33, random_state=42)
    X_train_tfidf_vectorizer = TfidfVectorizer(ngram_range=ngram_range, stop_words = "english", min_df=7)
    X_train_tfidf_vectorizer.fit(X_train)
    X_train_tfidf_vector = X_train_tfidf_vectorizer.transform(X_train)
    X_test_tfidf_vector = X_train_tfidf_vectorizer.transform(X_test)
    if classifier_algorithm == RidgeClassifier:
        clf = classifier_algorithm(tol=1e-2, solver="sparse_cg")
    elif classifier_algorithm == MLPClassifier :
        clf = MLPClassifier(random_state=1, hidden_layer_sizes = (100), max_iter=300)
    else :
        clf = classifier_algorithm()
    clf = clf.fit(X_train_tfidf_vector, y_train)
    y_pred = clf.predict(X_test_tfidf_vector)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='macro')
    recall = recall_score(y_test, y_pred, average='macro')
    f1 = f1_score(y_test, y_pred, average='macro')
    print(metrics.classification_report(y_test,y_pred, digits = 4))
```

In [18]:
```python
run_classifier_tfidf(text_df, MultinomialNB, (1,2))
```
```
              precision    recall  f1-score   support

           1     0.7383    0.7368    0.7376       494
           2     0.9367    0.5766    0.7138       359
           3     0.9015    0.8921    0.8968       667
           4     0.7828    0.9598    0.8623       871
           5     0.6110    0.9395    0.7404       876
           6     0.8388    0.8671    0.8527       444
           7     0.5767    0.9553    0.7192       716
           8     0.8750    0.0836    0.1526       335
           9     0.9612    0.3028    0.4605       327
          10     0.9375    0.1079    0.1935       278
          11     0.8171    0.7579    0.7864       442
          12     0.9853    0.2680    0.4214       250
          13     0.8149    0.7847    0.7995       432
          14     0.9814    0.5918    0.7383       267
          15     0.9353    0.6144    0.7416       306
          16     0.8287    0.9879    0.9013      1077

    accuracy                         0.7618      8141
   macro avg     0.8451    0.6516    0.6699      8141
weighted avg     0.8074    0.7618    0.7322      8141
```

In [19]:
```python
run_classifier_tfidf(text_df, RidgeClassifier, (1,2))
```
```
              precision    recall  f1-score   support

           1     0.8432    0.8381    0.8406       494
           2     0.8658    0.8802    0.8729       359
           3     0.9159    0.9310    0.9234       667
           4     0.9136    0.9587    0.9356       871
           5     0.8809    0.9372    0.9082       876
           6     0.9077    0.9302    0.9188       444
           7     0.8949    0.9274    0.9108       716
           8     0.6895    0.6299    0.6583       335
           9     0.8214    0.7737    0.7969       327
          10     0.7547    0.5755    0.6531       278
          11     0.8427    0.8846    0.8631       442
          12     0.9147    0.7720    0.8373       250
          13     0.8803    0.8681    0.8741       432
          14     0.9658    0.9513    0.9585       267
          15     0.9207    0.8725    0.8960       306
          16     0.9548    0.9610    0.9579      1077

    accuracy                         0.8872      8141
   macro avg     0.8729    0.8557    0.8628      8141
weighted avg     0.8854    0.8872    0.8854      8141
```

In [20]:
```python
run_classifier_tfidf(text_df, MLPClassifier, (1,2))
```
```
              precision    recall  f1-score   support

           1     0.8153    0.8401    0.8275       494
           2     0.8842    0.8719    0.8780       359
           3     0.9269    0.9310    0.9289       667
           4     0.9328    0.9564    0.9444       871
           5     0.8896    0.9201    0.9046       876
           6     0.9054    0.9054    0.9054       444
           7     0.8945    0.9120    0.9032       716
           8     0.6698    0.6299    0.6492       335
           9     0.8121    0.8196    0.8158       327
          10     0.6880    0.6187    0.6515       278
          11     0.8319    0.8733    0.8521       442
          12     0.8789    0.7840    0.8288       250
          13     0.8645    0.8565    0.8605       432
          14     0.9533    0.9176    0.9351       267
          15     0.9034    0.8562    0.8792       306
          16     0.9675    0.9684    0.9680      1077

    accuracy                         0.8839      8141
   macro avg     0.8636    0.8538    0.8583      8141
weighted avg     0.8829    0.8839    0.8831      8141
```

In [21]:
```python
classifiers = {
    'RidgeClassifier': RidgeClassifier(),
    'MLPClassifier': MLPClassifier(),
    'MultinomialNB': MultinomialNB()
}

results_df = pd.DataFrame({
    'Classifier': ['RidgeClassifier', 'MLPClassifier', 'MultinomialNB'],
    'Accuracy': [0.8872, 0.8839, 0.7618],
    'Precision': [0.8729, 0.8636, 0.8451],
    'Recall': [0.8557, 0.8538, 0.6516],
    'F1 Score': [0.8628, 0.8583, 0.6699]
})

results_df
```

Out[21]:

	Classifier	Accuracy	Precision	Recall	F1 Score
0	RidgeClassifier	0.8872	0.8729	0.8557	0.8628
1	MLPClassifier	0.8839	0.8636	0.8538	0.8583
2	MultinomialNB	0.7618	0.8451	0.6516	0.6699

Bigram Only

In [22]:
```python
run_classifier_tfidf(text_df, MultinomialNB, (2,2))
```
```
              precision    recall  f1-score   support

           1     0.6920    0.7368    0.7137       494
           2     0.8700    0.4847    0.6225       359
           3     0.7818    0.8006    0.7911       667
           4     0.7147    0.8944    0.7945       871
           5     0.6126    0.8756    0.7209       876
           6     0.8304    0.8378    0.8341       444
           7     0.6269    0.8729    0.7297       716
           8     0.8026    0.1821    0.2968       335
           9     0.7379    0.2324    0.3535       327
          10     0.8421    0.2302    0.3616       278
          11     0.6925    0.6063    0.6466       442
          12     0.9241    0.2920    0.4438       250
          13     0.7386    0.7847    0.7609       432
          14     0.9579    0.3408    0.5028       267
          15     0.8624    0.5327    0.6586       306
          16     0.7090    0.9749    0.8210      1077

    accuracy                         0.7124      8141
   macro avg     0.7747    0.6049    0.6283      8141
weighted avg     0.7414    0.7124    0.6855      8141
```

In [23]:
```python
run_classifier_tfidf(text_df, RidgeClassifier, (2,2))
```
```
              precision    recall  f1-score   support

           1     0.6817    0.7328    0.7063       494
           2     0.7082    0.6490    0.6773       359
           3     0.7826    0.7826    0.7826       667
           4     0.7246    0.8668    0.7893       871
           5     0.7724    0.8447    0.8070       876
           6     0.8274    0.8311    0.8292       444
           7     0.7657    0.7849    0.7752       716
           8     0.5290    0.4090    0.4613       335
           9     0.5241    0.4648    0.4927       327
          10     0.5825    0.4317    0.4959       278
          11     0.6627    0.6312    0.6466       442
          12     0.6904    0.5440    0.6085       250
          13     0.7445    0.7894    0.7663       432
          14     0.7040    0.5880    0.6408       267
          15     0.7540    0.6111    0.6751       306
          16     0.8931    0.9387    0.9153      1077

    accuracy                         0.7447      8141
   macro avg     0.7092    0.6812    0.6918      8141
weighted avg     0.7388    0.7447    0.7392      8141
```

In [24]:
```python
run_classifier_tfidf(text_df, MLPClassifier, (2,2))
```
```
              precision    recall  f1-score   support

           1     0.6763    0.7105    0.6930       494
           2     0.6447    0.5710    0.6056       359
           3     0.7706    0.7301    0.7498       667
           4     0.8230    0.7899    0.8061       871
           5     0.7618    0.8105    0.7854       876
           6     0.7597    0.7973    0.7780       444
           7     0.7311    0.7556    0.7431       716
           8     0.4024    0.4060    0.4042       335
           9     0.4366    0.4526    0.4444       327
          10     0.5208    0.4496    0.4826       278
          11     0.6018    0.6018    0.6018       442
          12     0.6019    0.5080    0.5510       250
          13     0.6861    0.7338    0.7092       432
          14     0.4309    0.5843    0.4960       267
          15     0.6519    0.5752    0.6111       306
          16     0.9410    0.9034    0.9218      1077

    accuracy                         0.7075      8141
   macro avg     0.6525    0.6487    0.6490      8141
weighted avg     0.7109    0.7075    0.7081      8141
```

In [25]:
```python
docs = text_df.text
categories = text_df.sdg
X_train, X_test, y_train, y_test = train_test_split(docs, categories, test_size=0.33, random_state=42)
X_train_tfidf_vectorizer = TfidfVectorizer(ngram_range=(2,2), stop_words = "english", min_df=7)
X_train_tfidf_vectorizer.fit(X_train)
X_train_tfidf_vector = X_train_tfidf_vectorizer.transform(X_train)
X_test_tfidf_vector = X_train_tfidf_vectorizer.transform(X_test)
tfidf_ridge_clf = RidgeClassifier(tol=1e-2, solver="sparse_cg")
tfidf_ridge_clf = tfidf_ridge_clf.fit(X_train_tfidf_vector, y_train)
```

In [26]:
```python
most_significant_feature_for_class(X_train_tfidf_vectorizer, X_train_tfidf_vector, tfidf_ridge_clf, [2, 3, 6, 8,], n=10)
```
```
SDG 2 : food security                  0.00536988
SDG 2 : food insecurity                0.00286569
SDG 2 : agricultural production        0.00122457
SDG 2 : agricultural sector            0.0011673
SDG 2 : agricultural land              0.000998728
SDG 2 : viet nam                       0.000997983
SDG 2 : self sufficiency               0.000893776
SDG 2 : price volatility               0.000790249
SDG 2 : risk management                0.00075164
SDG 2 : food production                0.000710151

SDG 2 : climate change                 -0.00152917
SDG 2 : human rights                   -0.00118067
SDG 2 : labour market                  -0.000865715
SDG 2 : gender equality                -0.000730004
SDG 2 : energy efficiency              -0.000724482
SDG 2 : rural areas                    -0.000676775
SDG 2 : climate finance                -0.000634874
SDG 2 : health care                    -0.000588158
SDG 2 : economic growth                -0.000581663
SDG 2 : united states                  -0.000518044

SDG 3 : health care                    0.0142746
SDG 3 : primary care                   0.00560044
SDG 3 : mental health                  0.00506297
SDG 3 : health services                0.00307763
SDG 3 : ministry health                0.00217471
SDG 3 : health insurance               0.00201488
SDG 3 : life expectancy                0.00198954
SDG 3 : hiv aids                       0.0019554
SDG 3 : indigenous peoples             0.00181505
SDG 3 : public health                  0.00167152

SDG 3 : human rights                   -0.00280604
SDG 3 : climate change                 -0.00163267
SDG 3 : labour market                  -0.000993252
SDG 3 : energy efficiency              -0.000793189
SDG 3 : public policy                  -0.000707464
SDG 3 : international law              -0.000695319
SDG 3 : social protection              -0.000655403
SDG 3 : land use                       -0.000643401
SDG 3 : higher education               -0.000608907
SDG 3 : water quality                  -0.000600551

SDG 6 : water resources                0.00565084
SDG 6 : water supply                   0.00526009
SDG 6 : water quality                  0.00477423
SDG 6 : river basin                    0.00257676
SDG 6 : water management               0.00239355
SDG 6 : drinking water                 0.00237797
SDG 6 : surface water                  0.00234192
SDG 6 : water sanitation               0.00186882
SDG 6 : water sector                   0.00151779
SDG 6 : water policy                   0.00151059

SDG 6 : health care                    -0.00113892
SDG 6 : human rights                   -0.00093909
SDG 6 : gender equality                -0.000741593
SDG 6 : international law              -0.000681624
SDG 6 : renewable energy               -0.000626258
SDG 6 : labour market                  -0.000612409
SDG 6 : developing countries           -0.00058446
SDG 6 : sustainable development        -0.000508266
SDG 6 : land use                       -0.000503395
SDG 6 : energy efficiency              -0.000493608

SDG 8 : labour market                  0.00496097
SDG 8 : labour force                   0.00220608
SDG 8 : young people                   0.00184746
SDG 8 : oecd countries                 0.0011282
SDG 8 : job search                     0.0009514
SDG 8 : social security                0.000936142
SDG 8 : job creation                   0.000875761
SDG 8 : foreign bom                    0.000796925
SDG 8 : self employed                  0.000781463
SDG 8 : unemployment rates             0.000749107

SDG 8 : climate change                 -0.00121361
SDG 8 : gender equality                -0.00100446
SDG 8 : et al                          -0.000923414
SDG 8 : human rights                   -0.000793874
SDG 8 : health care                    -0.000744487
SDG 8 : low income                     -0.000729767
SDG 8 : energy efficiency              -0.00072735
SDG 8 : income inequality              -0.00070332
SDG 8 : socio economic                 -0.000581427
SDG 8 : upper secondary                -0.000544994
```

- Human rights, health care, and labour markets appear in all of these SDG groups.

In [27]:
```python
classifiers = {
    'RidgeClassifier': RidgeClassifier(),
    'MLPClassifier': MLPClassifier(),
    'MultinomialNB': MultinomialNB()
}

results_df = pd.DataFrame({
    'Classifier': ['RidgeClassifier', 'MLPClassifier', 'MultinomialNB'],
    'Accuracy': [0.7447, 0.7075, 0.7124],
    'Precision': [0.7092, 0.6525, 0.7747],
    'Recall': [0.6812, 0.6487, 0.6049],
    'F1 Score': [0.6918, 0.6490, 0.6283]
})

results_df
```

Out[27]:

	Classifier	Accuracy	Precision	Recall	F1 Score
0	RidgeClassifier	0.7447	0.7092	0.6812	0.6918
1	MLPClassifier	0.7075	0.6525	0.6487	0.6490
2	MultinomialNB	0.7124	0.7747	0.6049	0.6283

- TNF seems to perform slightly better than Count in terms of Jaccard-only classification.