In [69]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
#plt.rcParams["figure.dpi"] = 300
plt.rcParams["savefig.dpi"] = 300
plt.rcParams["savefig.bbox"] = "tight"

np.set_printoptions(precision=3, suppress=True)
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import scale, StandardScaler
```

Predict Nationality from Names

- In this exercise, we are going to get names of the European Parliment members from European Parliment website.
- We will use `xml.etree.ElementTree` library for turning an xml file in string format into xml file
- We will use `requests` library to request an `xml` file from a webpage
    - XML documents have sections, called elements, defined by a beginning and an ending tag. A tag is a markup construct that begins with < and ends with >. The characters between the start-tag and end-tag, if there are any, are the element's content. Elements can contain markup, including other elements, which are called "child elements".
    - The largest, top-level element is called the root, which contains all other elements.
    - Attributes are name–value pair that exist within a start-tag or empty-element tag. An XML attribute can only have a single value and each attribute can appear at most once on each element.

In [17]:
```
import xml.etree.ElementTree as ET
```

In [18]:
```
import requests
```

In [19]:
```
response = requests.get("https://www.europarl.europa.eu/meps/en/full-list/xml
```

In [23]:
```
print(response.status_code)

print(response.headers['content-type'])

print(response.encoding)

print(response.text[:100])
```

```
200
application/xml;charset=UTF-8
UTF-8
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><meps><mep><fullName>Ma
gdalena ADAMOWICZ</ful
```

response.text

- `fromstring()` parses XML from a string directly into an `Element`, which is the root element of the parsed tree.

In [24]:
```
data_xml = ET.fromstring(response.text)
```

In [9]:
```
data_xml.tag
```

Out[9]:
```
'meps'
```

In [10]:
```
data_xml.attrib
```

Out[10]:
```
{}
```

In [16]:
```
type(data_xml)
```

Out[16]:
```
xml.etree.ElementTree.Element
```

In [13]:
```
len(data_xml)
```

Out[13]:
```
705
```

In [ ]:
```
root=data_xml
[elem.tag for elem in root.iter()]
```

In [34]:
```
#meps: members of european parliament
i=0
for child in data_xml.iter():
    print(child.tag, child.attrib)
    i=i+1
    if i==20:
        break
```

```
meps {}
mep {}
fullName {}
country {}
politicalGroup {}
id {}
nationalPoliticalGroup {}
mep {}
fullName {}
country {}
politicalGroup {}
id {}
nationalPoliticalGroup {}
mep {}
fullName {}
country {}
politicalGroup {}
id {}
nationalPoliticalGroup {}
mep {}
```

In [50]:

```
i=0
for child in data_xml.iter('fullName'):
    print(child.tag, child.text)
    i=i+1
    if i==20:
        break
```

```
fullName Magdalena ADAMOWICZ
fullName Asim ADEMOV
fullName Isabella ADINOLFI
fullName Matteo ADINOLFI
fullName Alex AGIUS SALIBA
fullName Mazaly AGUILAR
fullName Clara AGUILERA
fullName Alviina ALAMETSÄ
fullName Alexander ALEXANDROV YORDANOV
fullName François ALFONSI
fullName Atidzhe ALIEVA-VELI
fullName Abir AL-SAHLANI
fullName Álvaro AMARO
fullName Andris AMERIKS
fullName Christine ANDERSON
fullName Rasmus ANDRESEN
fullName Barry ANDREWS
fullName Eric ANDRIEU
fullName Mathilde ANDROUËT
fullName Nikos ANDROULAKIS
```

In [38]:

```
data_xml[0][0].text
```

Out[38]:

```
'Magdalena ADAMOWICZ'
```

In [86]:

```
data_xml[0][1].text
```

Out[86]: 'Poland'

In [28]:
```
#members_xml = data_xml.getchildren()
```

- We are now turning the xml string file into a list

In [51]:
```
members_xml = list(data_xml)
```

In [65]:
```
members_xml[1][0].text
```

Out[65]: 'Asim ADEMOV'

In [53]:
```
import pandas as pd
members_dict = [{i.tag: i.text for i in member} for member in members_xml]
members = pd.DataFrame(members_dict)
```

In [56]:
```
members_dict[:5]
```

Out[56]:
```
[{'fullName': 'Magdalena ADAMOWICZ',
  'country': 'Poland',
  'politicalGroup': "Group of the European People's Party (Christian Democrats
)",
  'id': '197490',
  'nationalPoliticalGroup': 'Independent'},
 {'fullName': 'Asim ADEMOV',
  'country': 'Bulgaria',
  'politicalGroup': "Group of the European People's Party (Christian Democrats
)",
  'id': '189525',
  'nationalPoliticalGroup': 'Citizens for European Development of Bulgaria'},
 {'fullName': 'Isabella ADINOLFI',
  'country': 'Italy',
  'politicalGroup': "Group of the European People's Party (Christian Democrats
)",
  'id': '124831',
  'nationalPoliticalGroup': 'Forza Italia'},
 {'fullName': 'Matteo ADINOLFI',
  'country': 'Italy',
  'politicalGroup': 'Identity and Democracy Group',
  'id': '197826',
  'nationalPoliticalGroup': 'Lega'},
 {'fullName': 'Alex AGIUS SALIBA',
  'country': 'Malta',
  'politicalGroup': 'Group of the Progressive Alliance of Socialists and Democ
rats in the European Parliament',
  'id': '197403',
  'nationalPoliticalGroup': 'Partit Laburista'}]
```

In [38]:
```
members[:20]
```

Out[38]:

	fullName	country	politicalGroup	id	nationalPoliticalGroup

0	Magdalena ADAMOWICZ	Poland	Group of the European People's Party (Christia...	197490	Independent
1	Asim ADEMOV	Bulgaria	Group of the European People's Party (Christia...	189525	Citizens for European Development of Bulgaria
2	Isabella ADINOLFI	Italy	Group of the European People's Party (Christia...	124831	Forza Italia
3	Matteo ADINOLFI	Italy	Identity and Democracy Group	197826	Lega
4	Alex AGIUS SALIBA	Malta	Group of the Progressive Alliance of Socialist...	197403	Partit Laburista
5	Mazaly AGUILAR	Spain	European Conservatives and Reformists Group	198096	VOX
6	Clara AGUILERA	Spain	Group of the Progressive Alliance of Socialist...	125045	Partido Socialista Obrero Español
7	Alviina ALAMETSÄ	Finland	Group of the Greens/European Free Alliance	204335	Vihreä liitto
8	Alexander ALEXANDROV YORDANOV	Bulgaria	Group of the European People's Party (Christia...	197836	Union of Democratic Forces
9	François ALFONSI	France	Group of the Greens/European Free Alliance	96750	Régions et Peuples Solidaires
10	Atidzhe ALIEVA-VELI	Bulgaria	Renew Europe Group	197848	Movement for Rights and Freedoms
11	Abir AL-SAHLANI	Sweden	Renew Europe Group	197400	Centerpartiet
12	Álvaro AMARO	Portugal	Group of the European People's Party (Christia...	197746	Partido Social Democrata
13	Andris AMERIKS	Latvia	Group of the Progressive Alliance of Socialist...	197783	Gods kalpot Rīgai
14	Christine ANDERSON	Germany	Identity and Democracy Group	197475	Alternative für Deutschland
15	Rasmus ANDRESEN	Germany	Group of the Greens/European Free Alliance	197448	Bündnis 90/Die Grünen
16	Barry ANDREWS	Ireland	Renew Europe Group	204332	Fianna Fáil Party
17	Eric ANDRIEU	France	Group of the Progressive Alliance of Socialist...	113892	Parti socialiste
18	Mathilde ANDROUËT	France	Identity and Democracy Group	197691	Rassemblement national

19	Nikos ANDROULAKIS	Greece	Group of the Progressive Alliance of Socialist...	125110	PASOK-KINAL

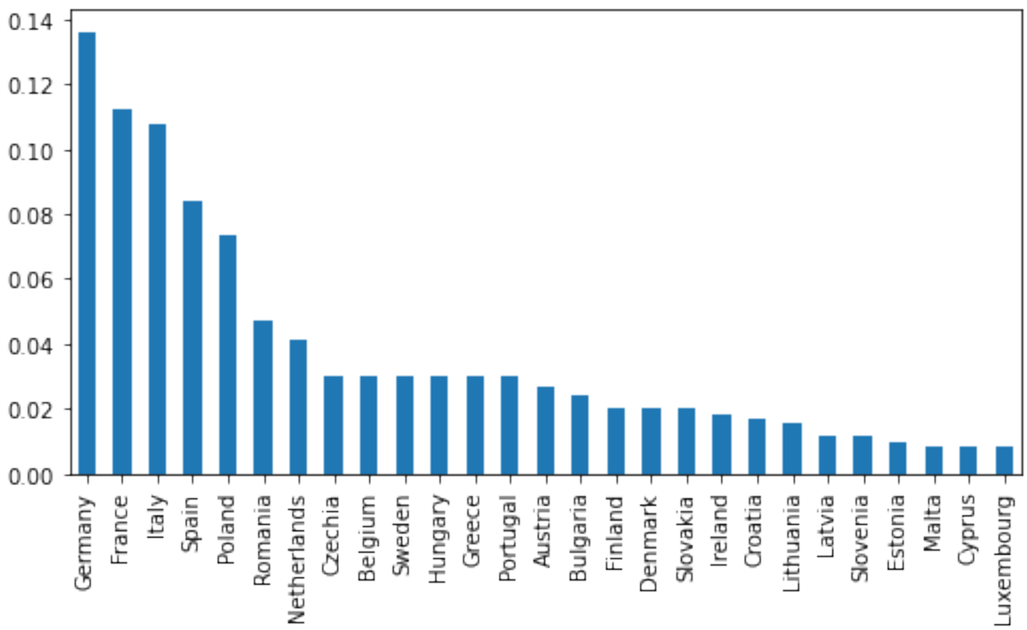
Predicting nationalities from names

In [66]:
```
members.shape
```

Out[66]:
```
(705, 5)
```

- Create a bar graph showing the ratio of nationalities in the European parliament

In [71]:
```
y_mem = members.country
data_mem = members.fullName
plt.figure(figsize=(8, 4))
(y_mem.value_counts() / y_mem.size).plot(kind='bar');
```



- print the value counts of the members per country

In [73]:
```
tt=y_mem.value_counts()[:8]
```

In [77]:
```
tt.values, tt.index
```

Out[77]:
```
(array([96, 79, 76, 59, 52, 33, 29, 21]),
 Index(['Germany', 'France', 'Italy', 'Spain', 'Poland', 'Romania',
        'Netherlands', 'Czechia'],
       dtype='object'))
```

- Get the names of the first 8 countries with the largest number of meps

In [79]:
```
large = y_mem.value_counts()[:8].index
large
```

Out[79]:
```
Index(['Germany', 'France', 'Italy', 'Spain', 'Poland', 'Romania',
       'Netherlands', 'Czechia'],
      dtype='object')
```

- create a data with the parliament members in the first eight countries
- split data into test and train datasets

In [88]:
```
mask = y_mem.isin(large)
data_mem = data_mem[mask]
y_mem = y_mem[mask]
```

In [89]:
```
mask
```

Out[89]:
```
0      True
2      True
3      True
5      True
6      True
       ...
696    True
698    True
699    True
700    True
703    True
Name: country, Length: 445, dtype: bool
```

In [90]:
```
(y_mem.value_counts() / y_mem.size)
```

Out[90]:
```
Germany        0.215730
France         0.177528
Italy          0.170787
Spain          0.132584
Poland         0.116854
Romania        0.074157
Netherlands    0.065169
Czechia        0.047191
Name: country, dtype: float64
```

In [91]:
```
data_mem.shape
```

Out[91]:
```
(445,)
```

In [101…
```
text_mem_train, text_mem_test, y_mem_train, y_mem_test = \
train_test_split(data_mem, y_mem,  stratify=y_mem, random_state=0)
```

- create a pipeline of `countvectorizer` and `logisticregressioncv`
- report the cross validation scores

In [93]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegressionCV
bow_pipe = make_pipeline(CountVectorizer(), LogisticRegressionCV())
cross_val_score(bow_pipe, text_mem_train, y_mem_train, cv=5,\
                scoring='f1_macro')
```

Out[93]:
```
array([0.281, 0.324, 0.359, 0.368, 0.327])
```

- Repeat the same procedure, with vectorizer having the parameter of analyzer="char_wb"
  - analyzer{'word', 'char', 'char_wb'} or callable, default='word' Whether the feature should be made of word n-gram or character n-grams. Option 'char_wb' creates character n-grams only from text inside word boundaries; n-grams at the edges of words are padded with space.

In [102…
```
char_pipe = make_pipeline(CountVectorizer(analyzer="char_wb",\
ngram_range=(1, 4)), LogisticRegressionCV(solver='liblinear'))
cross_val_score(char_pipe, text_mem_train, y_mem_train, cv=5, scoring='f1_mac
```

Out[102…
```
array([0.517, 0.572, 0.564, 0.449, 0.558])
```

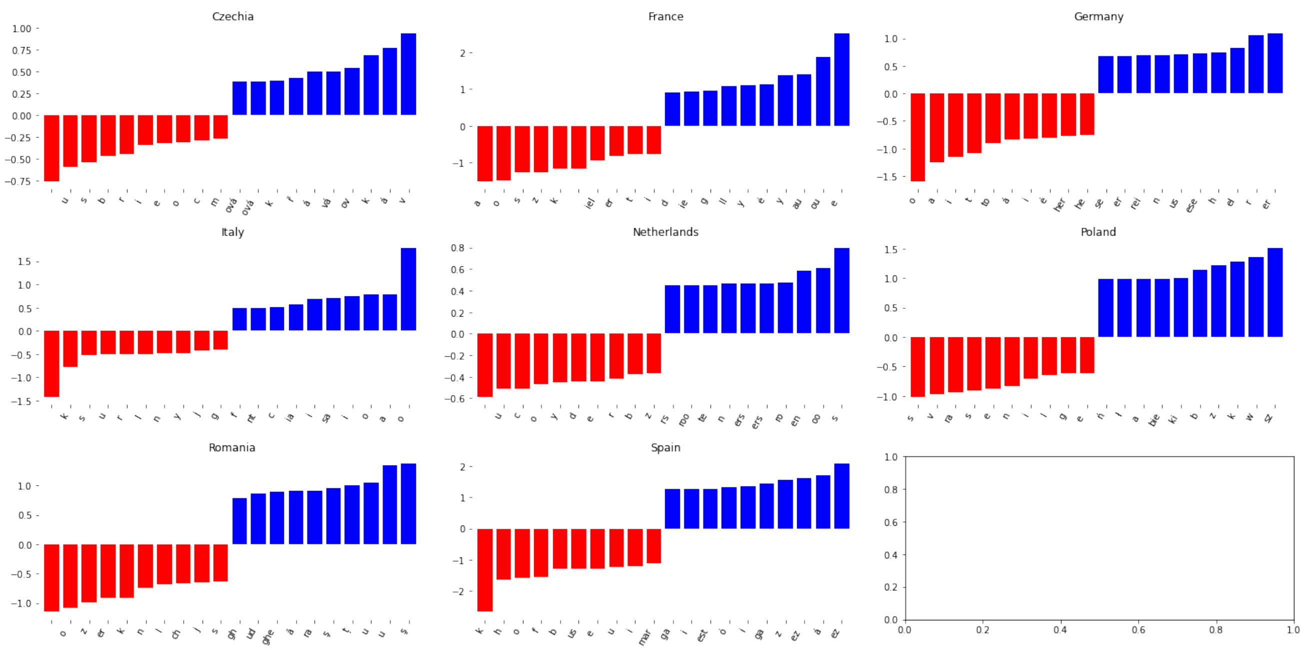
In [95]:
```
char_pipe.fit(text_mem_train, y_mem_train)
```

Out[95]:
```
Pipeline(steps=[('countvectorizer',
                 CountVectorizer(analyzer='char_wb', ngram_range=(1, 4))),
                ('logisticregressioncv',
                 LogisticRegressionCV(solver='liblinear'))])
```

- print the top 20 most important and least important n-grams

In [96]:
```
def plot_important_features(coef, feature_names, top_n=20,\
                            ax=None, rotation=60):
    if ax is None:
        ax = plt.gca()
    inds = np.argsort(coef)
    low = inds[:top_n]
    high = inds[-top_n:]
    important = np.hstack([low, high])
    myrange = range(len(important))
    colors = ['red'] * top_n + ['blue'] * top_n

    ax.bar(myrange, coef[important], color=colors)
    ax.set_xticks(myrange)
    ax.set_xticklabels(feature_names[important],\
                       rotation=rotation, ha="right")
    ax.set_xlim(-.7, 2 * top_n)
    ax.set_frame_on(False)
```

In [97]:
```
lr = char_pipe.named_steps['logisticregressioncv']
feature_names = np.array(char_pipe.named_steps\
                        ['countvectorizer'].get_feature_names())
n_classes = len(lr.classes_)
fig, axes = plt.subplots(n_classes // 3 + 1, 3, figsize=(20, 10))
for ax, coef, label in zip(axes.ravel(), lr.coef_, lr.classes_):
    ax.set_title(label)
    plot_important_features(coef, feature_names, top_n=10, ax=ax)

plt.tight_layout()
```



- Now make a gridsearch over logistic regression, countvectorizer n-grams, min_df and normalizer

In [98]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import Normalizer

param_grid = {"logisticregression__C": [100, 10, 1, 0.1, 0.001],
              "countvectorizer__ngram_range": [(1, 1), (1, 2), (1, 5), (1, 7)
                                               (2, 3), (2, 5), (3, 8), (5, 5)
              "countvectorizer__min_df": [1, 2, 3],
              "normalizer": [None, Normalizer()]
             }
grid = GridSearchCV(make_pipeline(CountVectorizer(analyzer="char"),\
Normalizer(), LogisticRegression(solver='liblinear')),\
param_grid=param_grid, cv=10, \
scoring="f1_macro",return_train_score=True
                    )
```

In [99]:
```
grid.fit(text_mem_train, y_mem_train)
```

Out[99]:
```
GridSearchCV(cv=10,
             estimator=Pipeline(steps=[('countvectorizer',
                                        CountVectorizer(analyzer='char')),
                                       ('normalizer', Normalizer()),
                                       ('logisticregression',
                                        LogisticRegression(solver='liblinear')
)]),
             param_grid={'countvectorizer__min_df': [1, 2, 3],
                         'countvectorizer__ngram_range': [(1, 1), (1, 2),
                                                          (1, 5), (1, 7),
                                                          (2, 3), (2, 5),
                                                          (3, 8), (5, 5)],
                         'logisticregression__C': [100, 10, 1, 0.1, 0.001],
                         'normalizer': [None, Normalizer()]},
             return_train_score=True, scoring='f1_macro')
```

In [70]:
```
grid.best_score_
```

Out[70]:
```
0.5249505008880008
```

In [71]:
```
grid.best_params_
```

Out[71]:
```
{'countvectorizer__min_df': 2,
 'countvectorizer__ngram_range': (1, 5),
 'logisticregression__C': 100,
 'normalizer': Normalizer()}
```

In [72]:
```
results = pd.DataFrame(grid.cv_results_)
res_pivot = results.pivot_table(values=['mean_test_score', 'mean_train_score'
                                index=["param_countvectorizer__ngram_range",
                                       "param_countvectorizer__min_df"])
```

In [73]:
```
res_pivot.mean_test_score
```

Out[73]:

```
param_countvectorizer__ngram_range  param_logisticregression__C  param_countve
ctorizer__min_df
(1, 1)                              0.001                        1
0.093764
                                                                 2
0.093764
                                                                 3
0.093764
                                    0.100                        1
0.282342
                                                                 2
0.281447

...
(5, 5)                              10.000                       2
0.200092
                                                                 3
0.125066
                                    100.000                      1
0.342071
                                                                 2
0.201056
                                                                 3
0.124191
Name: mean_test_score, Length: 120, dtype: float64
```

In [74]:

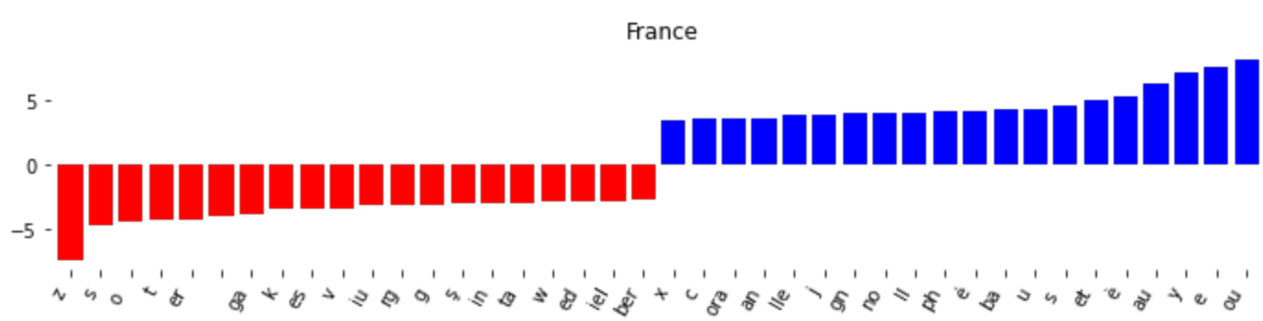
```python
bla = res_pivot.mean_test_score.unstack(["param_countvectorizer__ngram_range"
bla = bla.swaplevel().sort_index()
bla.index.names = ['min_df', 'C']
bla.style.background_gradient(cmap="viridis")
```

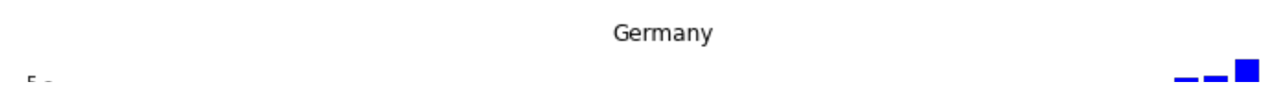
Out[74]:

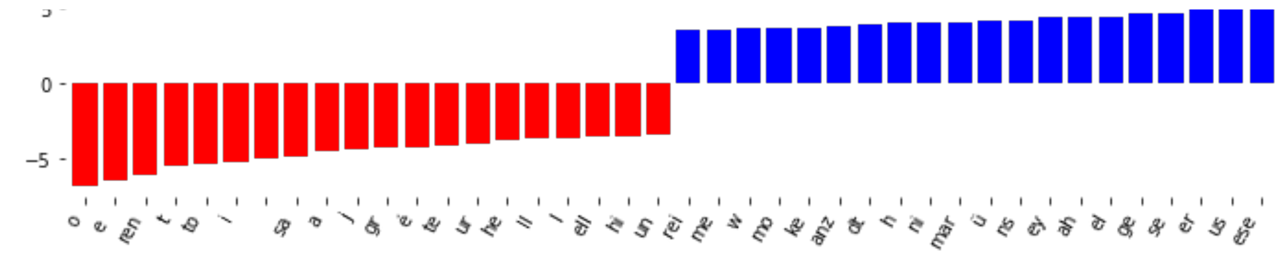
param_countvectorizer__ngram_range		(1, 1)	(1, 2)	(1, 5)	(1, 7)	(2
min_df	**C**					
	0.001	0.093764	0.133667	0.139716	0.142564	0.156
	0.1	0.282342	0.309392	0.292443	0.279722	0.269
1	**1.0**	0.389971	0.427628	0.428499	0.419267	0.417
	10.0	0.447141	0.490534	0.500497	0.485259	0.494
	100.0	0.465545	0.513257	0.512528	0.497304	0.488
	0.001	0.093764	0.133667	0.138023	0.140870	0.151
	0.1	0.281447	0.312036	0.300137	0.297979	0.278
2	**1.0**	0.388216	0.428750	0.423333	0.427985	0.407
	10.0	0.442539	0.487832	0.507069	0.499359	0.470
	100.0	0.463028	0.490160	0.519139	0.516782	0.466
	0.001	0.093764	0.132213	0.138067	0.138023	0.147
	0.1	0.281470	0.309989	0.306797	0.307711	0.272
3	**1.0**	0.385981	0.425351	0.420148	0.418777	0.406
	10.0	0.440304	0.488528	0.496829	0.493374	0.460
	100.0	0.451457	0.501004	0.500295	0.497516	0.455

In [75]:

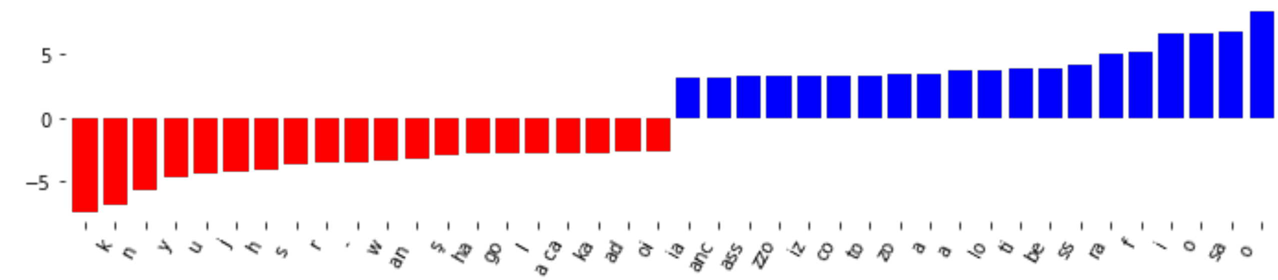
```python
lr = grid.best_estimator_.named_steps['logisticregression']
feature_names = np.array(grid.best_estimator_.\
    named_steps['countvectorizer'].get_feature_names())
n_classes = len(lr.classes_)
fig, axes = plt.subplots(n_classes, figsize=(10, 20))
for ax, coef, label in zip(axes.ravel(), lr.coef_, lr.classes_):
    ax.set_title(label)
    plot_important_features(coef, \
feature_names, top_n=20, ax=ax)

plt.tight_layout()
```



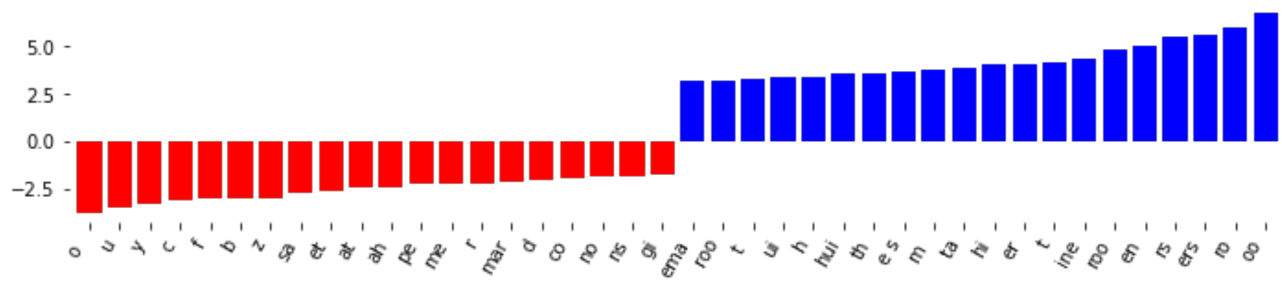




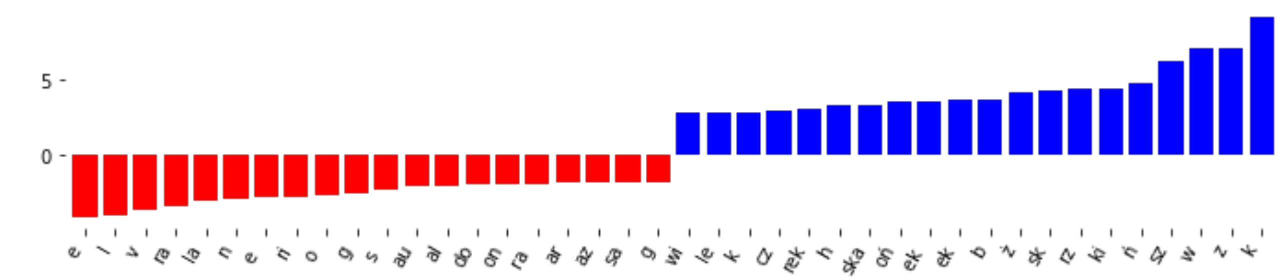
Italy



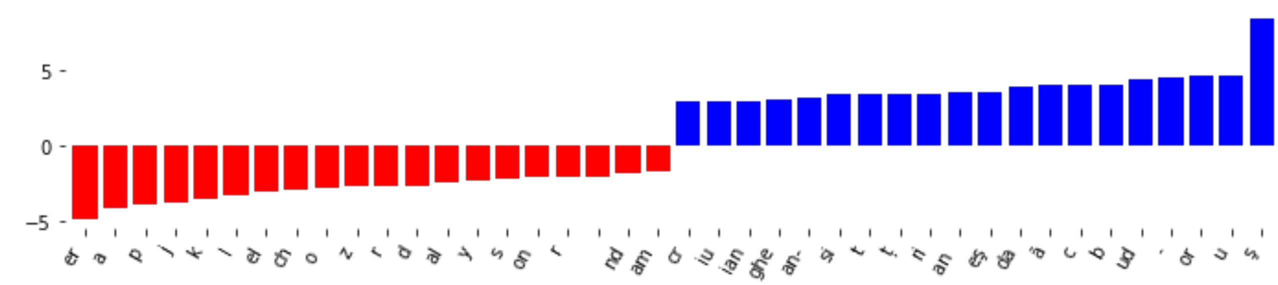
Netherlands



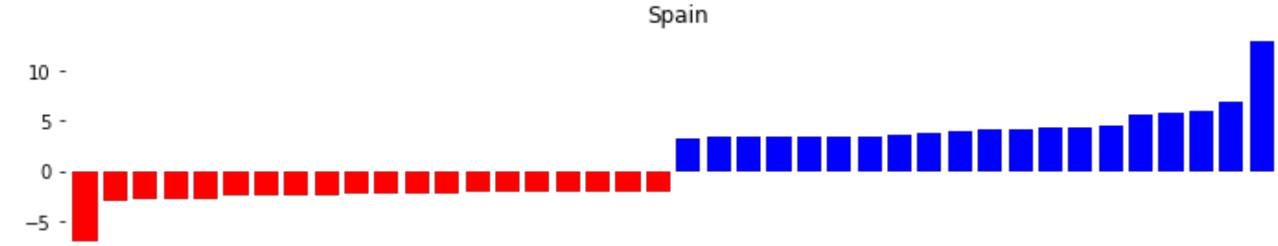
Poland

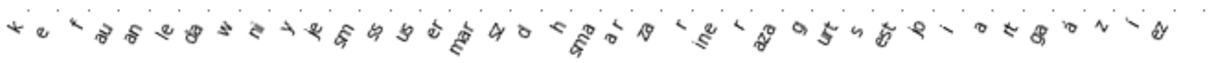


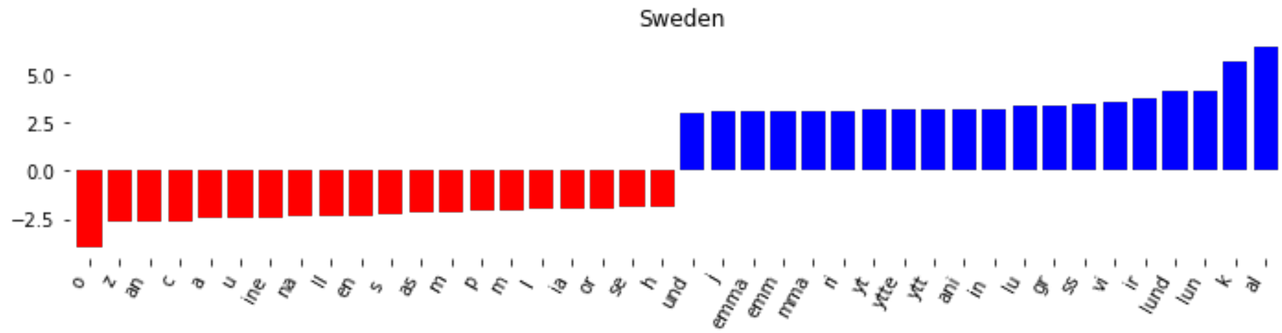
Romania



Spain






Sweden
5.0
2.5
0.0
−2.5
o z an c a u ine na ll en s as m p m l ia or se h und j emma emm mma ri yt ytte ytt ani in lu gr ss vi ir lund lun k al

In [ ]: