MP2

November 19, 2023

1 ECSE-551 Mini Project 2

Authors: * Ashley Meagher (260822930) * Charles Sirois (261158513)

[250]:
```
# To specify where to load the data
in_colab = True
folder_path = 'drive/MyDrive/Colab Notebooks/ECSE 551_MP2'

%load_ext autoreload
%autoreload 2

# Our functions and classes
if in_colab:
  from google.colab import drive
  from google.colab import data_table
  drive.mount('/content/drive')

  data_table.enable_dataframe_formatter()  # For interactive df viz

  import sys
  sys.path.insert(0, folder_path)

# SK Learn models
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import LinearSVC

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
import itertools
import datetime

import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
```

```
# Install required packages
!pip install unidecode  # To remove accents
!pip install langid  # To identify text's language

# Import our classes and functions from the other files
from NaiveBayes import NaiveBayes
from cross_val_score import cross_val_score
from data_processing import Data, Format_data
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
Requirement already satisfied: unidecode in /usr/local/lib/python3.10/dist-
packages (1.3.7)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

```
Requirement already satisfied: langid in /usr/local/lib/python3.10/dist-packages
(1.1.6)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(from langid) (1.23.5)
```

1.1 Data Analysis

1.1.1 Load the data

[251]:
```
print(f"Loading data files... ", end='')
filenames = [folder_path + "/data/train_utf8.csv", folder_path + "/data/
  ↪test_utf8.csv"]
words_dataset = Data(train_file=filenames[0], test_file=filenames[1])
print(f'Done')
```

```
Loading data files… Done
```

1.1.2 Data properties

[252]:
```
print(f'Training dataset size: {words_dataset.train_data.shape[0]}')
print(f'Test dataset size: {words_dataset.test_data.shape[0]}')

classes, classes_count = np.unique(words_dataset.train_data['label'],␣
 ↪return_counts=True)
print(f'Classes: ')
for cls, cls_count in zip(classes, classes_count):
  print(f'\t-{cls}: {cls_count}')
```

```
Training dataset size: 719
Test dataset size: 279
Classes:
        -London: 180
        -Montreal: 180
        -Paris: 180
        -Toronto: 179
```

1.2 Models Performances

1.2.1 Functions

Functions to compute the cross-validation score of the different combinations of model hyperparameters and datasets

[253]:
```
def create_datasets(ds_options_dict):
  """
  To create a list with all the combinations of options in the dict
  """
  print(f"Processing input data...")
  keys, values = zip(*ds_options_dict.items())
  ds_options_list = [dict(zip(keys, v)) for v in itertools.product(*values)]

  ds_list = []
  for idx, each_ds in enumerate(ds_options_list):
      each_ds['dataset_name'] = f'DS {idx}'
      ds_list.append(Format_data(words_dataset, **each_ds))

  print(f'\nDone')
  return ds_list

def find_ds_from_name(ds_name, ds_list) -> Format_data:
    """
    To return the dataset with the corresponding name in the `ds_list`
    """
    ds = next((ds for ds in ds_list if ds.name == ds_name), None)

    if ds is None:
```

```
        raise ValueError(f"Dataset {ds_name} not found in `ds_list`")

    return ds

def compute_models_cv_acc(model_dict, ds_list):
  """
  To compute the cv score for all the combinations of model_dict and ds_list
  """
  results_df = pd.DataFrame()

  # Cross-Validation
  n_fold = 5

  start_time = time.time()
  print(f"--------- Training all models ---------")
  for model_name, model_info in model_dict.items():
      model = model_info["model"]
      base_params = model_info["base_params"]
      cv_params = model_info["cv_params"]

      print(f"\nModel : {model_name}")
      model_start = time.time()
      for ds_idx, each_dataset in enumerate(ds_list):
          # Check if it already has been ran
          ds_start = time.time()
          dataset_name = each_dataset.name
          print(f"\tDataset [{ds_idx+1}/{len(ds_list)}]: {dataset_name}")

          X_train = each_dataset.X
          y_train = each_dataset.Y

          # Cross_validation
          cv_results = cross_val_score(
              model,
              X_train,
              y_train,
              cv=n_fold,
              base_params=base_params,
              cv_params=cv_params,
              results_df=results_df,
              ds_name=dataset_name,
          )

          if cv_results.empty:
              print(f'... Model already trained')
              continue
```

```
        # Print best combination
        best_row = cv_results.iloc[cv_results['Score'].idxmax()]
        compute_time = time.time() - ds_start
        print(
            f"\tBest CV Score : {np.round(best_row['Score']*100)}% (Acc: {np.
↪round(best_row['Acc']*100)}) "
            f"[{compute_time} sec]\n"
        )

        # Add information to series
        ds_params = each_dataset.get_params()

        for key, value in ds_params.items():
            if isinstance(value, tuple):
                value = str(value)
            cv_results[key] = value

        # cv_results = pd.concat([cv_results, pd.(ds_params).T],␣
↪ignore_index=True)
        cv_results['Model name'] = model_name
        cv_results['Dataset'] = dataset_name
        cv_results['Compute time'] = compute_time

        results_df = pd.concat([results_df, cv_results], ignore_index=True,␣
↪axis=0)

    print(f'Model trained in {time.time() - model_start} sec')

print(f"\nTraining completed ({time.time() - start_time} sec)\n")

results_df['Score'] = (results_df['Score']*100).apply(np.round, decimals=2)
results_df['Bias'] = ((1 - results_df['Acc'])*100).apply(np.round, decimals=2)

results_df = results_df[
    [
        'Model name',
        'Score',
        'Bias',
        'Acc',
        'Dataset',
        'Params',
        'Compute time',
        'Model',
        'n_gram',
        'feat_type',
        'lemmatized',
        'lang',
```

```
            'standardized',
            'rm_accents',
            'feat_select',
            'n_feat',
        ]
    ]


    results_df = results_df.sort_values(by=['Score'], ascending=False)

    return results_df

def create_pred_ds(model_idx, results_df, ds_list, save_path):
    """ To create the prediction csv file for the model correponding to␣
 ↪model_idx
    The file is saved under save_path
    """
    my_model_info = results_df.loc[model_idx]
    print(f'Model chosen: ')
    print(my_model_info)

    print(f"Predicting test data using this model...")
    my_model = my_model_info['Model']
    ds = find_ds_from_name(my_model_info['Dataset'], ds_list)

    y_test = my_model.predict(ds.X_test)
    pred_df = pd.DataFrame(y_test, columns=['subreddit'])
    pred_df.index.name = 'id'

    pred_df.to_csv(save_path)
    print(f'Predictions saved to {save_path}')
```

1.2.2 Bernouilli Naive Bayes

Parameters evaluation

[254]:
```
nb_ds_options = {
    'max_feat': [None],
    'lang_id': [False, True],  # [False, True],
    'feature_type': ['Bin'],
    'n_gram': [(1, 1), (1, 2), (1, 3), (1, 4), (1, 5)],
    'lemmatize': [False],
    'feat_select': ['F_CL'],
    'n_feat_select': [500, 1000, 2000, 3000, 4000],
}
nb_ds_list = create_datasets(nb_ds_options)
```

```
Processing input data…
        Processing of: DS 49…
Done
```

[255]:
```
# Separate datasets to test lemmatization
nb2_ds_options = {
    'max_feat': [None],
    'lang_id': [False],  # [False, True],
    'feature_type': ['Bin'],
    'n_gram': [(1, 2)],
    'lemmatize': [True, False],
    'feat_select': ['F_CL'],
    'n_feat_select': [2000],
}
nb2_ds_list = create_datasets(nb2_ds_options)
```

```
Processing input data…
        Processing of: DS 1…
Done
```

[256]:
```
nb_model_dict = {}
nb_model_dict["My Bernouilli NB"] = {
    "model": NaiveBayes,
    'base_params': {'laplace_smoothing': True, 'verbose': False},
    'cv_params': None,
}
```

[257]:
```
nb_df = compute_models_cv_acc(nb_model_dict, nb_ds_list)
nb_results = nb_df[['Score', 'Bias', 'Params', 'n_gram', 'lang', 'n_feat']]
```

```
--------- Training all models ---------

Model : My Bernouilli NB
        Dataset [1/50]: DS 0
        Combination 1/1 Best CV Score : 71.0% (Acc: 77.0) [4.215232849121094
sec]

        Dataset [2/50]: DS 1
        Combination 1/1 Best CV Score : 72.0% (Acc: 82.0) [8.167191982269287
sec]

        Dataset [3/50]: DS 2
        Combination 1/1 Best CV Score : 75.0% (Acc: 88.0) [20.4975848197937 sec]

        Dataset [4/50]: DS 3
        Combination 1/1 Best CV Score : 68.0% (Acc: 87.0) [26.63149333000183
sec]

        Dataset [5/50]: DS 4
```

```
        Combination 1/1 Best CV Score : 62.0% (Acc: 83.0) [37.83934044837952
sec]

        Dataset [6/50]: DS 5
        Combination 1/1 Best CV Score : 70.0% (Acc: 77.0) [5.771634101867676
sec]

        Dataset [7/50]: DS 6
        Combination 1/1 Best CV Score : 71.0% (Acc: 83.0) [7.290061712265015
sec]

        Dataset [8/50]: DS 7
        Combination 1/1 Best CV Score : 78.0% (Acc: 88.0) [18.11064648628235
sec]

        Dataset [9/50]: DS 8
        Combination 1/1 Best CV Score : 74.0% (Acc: 91.0) [29.609700202941895
sec]

        Dataset [10/50]: DS 9
        Combination 1/1 Best CV Score : 68.0% (Acc: 88.0) [36.51623868942261
sec]

        Dataset [11/50]: DS 10
        Combination 1/1 Best CV Score : 65.0% (Acc: 71.0) [4.02064323425293 sec]

        Dataset [12/50]: DS 11
        Combination 1/1 Best CV Score : 70.0% (Acc: 81.0) [10.331190824508667
sec]

        Dataset [13/50]: DS 12
        Combination 1/1 Best CV Score : 75.0% (Acc: 87.0) [18.814451694488525
sec]

        Dataset [14/50]: DS 13
        Combination 1/1 Best CV Score : 73.0% (Acc: 90.0) [29.117358684539795
sec]

        Dataset [15/50]: DS 14
        Combination 1/1 Best CV Score : 69.0% (Acc: 89.0) [37.6209557056427 sec]

        Dataset [16/50]: DS 15
        Combination 1/1 Best CV Score : 60.0% (Acc: 66.0) [3.796255588531494
sec]

        Dataset [17/50]: DS 16
        Combination 1/1 Best CV Score : 68.0% (Acc: 76.0) [11.544331073760986
sec]
```

```
        Dataset [18/50]: DS 17
        Combination 1/1 Best CV Score : 72.0% (Acc: 85.0) [17.99778699874878
sec]

        Dataset [19/50]: DS 18
        Combination 1/1 Best CV Score : 70.0% (Acc: 89.0) [28.106534481048584
sec]

        Dataset [20/50]: DS 19
        Combination 1/1 Best CV Score : 69.0% (Acc: 89.0) [36.949981927871704
sec]

        Dataset [21/50]: DS 20
        Combination 1/1 Best CV Score : 57.0% (Acc: 62.0) [3.6437978744506836
sec]

        Dataset [22/50]: DS 21
        Combination 1/1 Best CV Score : 64.0% (Acc: 74.0) [9.618980169296265
sec]

        Dataset [23/50]: DS 22
        Combination 1/1 Best CV Score : 69.0% (Acc: 82.0) [18.49955105781555
sec]

        Dataset [24/50]: DS 23
        Combination 1/1 Best CV Score : 69.0% (Acc: 86.0) [26.71236515045166
sec]

        Dataset [25/50]: DS 24
        Combination 1/1 Best CV Score : 68.0% (Acc: 88.0) [36.23162865638733
sec]

        Dataset [26/50]: DS 25
        Combination 1/1 Best CV Score : 69.0% (Acc: 75.0) [6.965734243392944
sec]

        Dataset [27/50]: DS 26
        Combination 1/1 Best CV Score : 70.0% (Acc: 81.0) [7.405640363693237
sec]

        Dataset [28/50]: DS 27
        Combination 1/1 Best CV Score : 74.0% (Acc: 87.0) [17.702255725860596
sec]

        Dataset [29/50]: DS 28
        Combination 1/1 Best CV Score : 67.0% (Acc: 85.0) [28.42513680458069
sec]
```

```
        Dataset [30/50]: DS 29
        Combination 1/1 Best CV Score : 63.0% (Acc: 82.0) [36.266666412353516
sec]

        Dataset [31/50]: DS 30
        Combination 1/1 Best CV Score : 69.0% (Acc: 75.0) [3.7581331729888916
sec]

        Dataset [32/50]: DS 31
        Combination 1/1 Best CV Score : 73.0% (Acc: 83.0) [10.553314924240112
sec]

        Dataset [33/50]: DS 32
        Combination 1/1 Best CV Score : 75.0% (Acc: 87.0) [17.867765426635742
sec]

        Dataset [34/50]: DS 33
        Combination 1/1 Best CV Score : 74.0% (Acc: 88.0) [26.27276086807251
sec]

        Dataset [35/50]: DS 34
        Combination 1/1 Best CV Score : 67.0% (Acc: 87.0) [37.33134150505066
sec]

        Dataset [36/50]: DS 35
        Combination 1/1 Best CV Score : 62.0% (Acc: 69.0) [4.442660331726074
sec]

        Dataset [37/50]: DS 36
        Combination 1/1 Best CV Score : 70.0% (Acc: 79.0) [7.428954124450684
sec]

        Dataset [38/50]: DS 37
        Combination 1/1 Best CV Score : 75.0% (Acc: 86.0) [18.159637451171875
sec]

        Dataset [39/50]: DS 38
        Combination 1/1 Best CV Score : 72.0% (Acc: 88.0) [28.107608318328857
sec]

        Dataset [40/50]: DS 39
        Combination 1/1 Best CV Score : 69.0% (Acc: 87.0) [35.64508605003357
sec]

        Dataset [41/50]: DS 40
        Combination 1/1 Best CV Score : 59.0% (Acc: 64.0) [4.095541715621948
sec]
```

```
        Dataset [42/50]: DS 41
        Combination 1/1 Best CV Score : 66.0% (Acc: 75.0) [10.126330375671387
sec]

        Dataset [43/50]: DS 42
        Combination 1/1 Best CV Score : 72.0% (Acc: 84.0) [17.864989519119263
sec]

        Dataset [44/50]: DS 43
        Combination 1/1 Best CV Score : 70.0% (Acc: 87.0) [27.56903648376465
sec]

        Dataset [45/50]: DS 44
        Combination 1/1 Best CV Score : 69.0% (Acc: 88.0) [37.22462606430054
sec]

        Dataset [46/50]: DS 45
        Combination 1/1 Best CV Score : 56.0% (Acc: 60.0) [3.685528516769409
sec]

        Dataset [47/50]: DS 46
        Combination 1/1 Best CV Score : 64.0% (Acc: 72.0) [9.841209650039673
sec]

        Dataset [48/50]: DS 47
        Combination 1/1 Best CV Score : 69.0% (Acc: 81.0) [18.502774715423584
sec]

        Dataset [49/50]: DS 48
        Combination 1/1 Best CV Score : 67.0% (Acc: 84.0) [25.65246319770813
sec]

        Dataset [50/50]: DS 49
        Combination 1/1 Best CV Score : 69.0% (Acc: 87.0) [35.62777662277222
sec]

Model trained in 964.5682625770569 sec

Training completed (964.5693309307098 sec)
```

[258]:
```
nb2_df = compute_models_cv_acc(nb_model_dict, nb2_ds_list)
nb2_results = nb2_df[['Score', 'Bias', 'Params', 'n_gram', 'lang', 'n_feat',␣
 ↪'lemmatized']]
```

```
--------- Training all models ---------
```

```
Model : My Bernouilli NB
        Dataset [1/2]: DS 0
        Combination 1/1 Best CV Score : 75.0% (Acc: 88.0) [18.69008207321167
sec]

        Dataset [2/2]: DS 1
        Combination 1/1 Best CV Score : 79.0% (Acc: 88.0) [19.127805709838867
sec]

Model trained in 37.849053144454956 sec

Training completed (37.85080671310425 sec)
```

Step 1 - Effect of feature selection

[259]:
```
nb_results_step1 = nb_results[(nb_results['n_gram'] == '(1, 1)') &␣
 ↪(nb_results['lang'] == False)]
nb_results_step1.sort_values(by=['n_feat'], ascending=True)
```

[259]:

	Score	Bias	Params	n_gram	lang	n_feat
0	71.36	22.81	{}	(1, 1)	False	500
1	71.63	18.36	{}	(1, 1)	False	1000
2	75.11	11.54	{}	(1, 1)	False	2000
3	67.59	13.21	{}	(1, 1)	False	3000
4	61.89	16.97	{}	(1, 1)	False	4000

Step 2 - N-grams

[260]:
```
nb_results_step2 = nb_results[(nb_results['n_feat'] == 2000) &␣
 ↪(nb_results['lang'] == False)]
nb_results_step2.sort_values(by=['n_gram'], ascending=True)
```

[260]:

	Score	Bias	Params	n_gram	lang	n_feat
2	75.11	11.54	{}	(1, 1)	False	2000
7	78.17	11.82	{}	(1, 2)	False	2000
12	74.83	12.80	{}	(1, 3)	False	2000
17	72.05	15.30	{}	(1, 4)	False	2000
22	69.12	18.22	{}	(1, 5)	False	2000

Step 3 - Language Identification

[261]:
```
nb_results_step3 = nb_results[(nb_results['n_feat'] == 2000) &␣
 ↪(nb_results['n_gram'] == '(1, 2)')]
nb_results_step3
```

[261]:

	Score	Bias	Params	n_gram	lang	n_feat
7	78.17	11.82	{}	(1, 2)	False	2000
32	75.25	12.93	{}	(1, 2)	True	2000

Step 4 - Effect of lemmatization

[262]:
```
nb_results_step4 = nb2_results
nb_results_step4
```

[262]:

	Score	Bias	Params	n_gram	lang	n_feat	lemmatized
1	78.73	11.82	{}	(1, 2)	False	2000	False
0	75.11	11.82	{}	(1, 2)	False	2000	True

1.2.3 SVC

[263]:
```
svc_ds_options = {
    'max_feat': [None],
    'lang_id': [False, True],  # [False, True],
    'feature_type': ['Count'],
    'n_gram': [(1, 2), (1, 3)],
    'lemmatize': [False],
    'feat_select': ['F_CL'],
    'n_feat_select': [2000],
}
svc_ds_list = create_datasets(svc_ds_options)

svc_model_dict = {}
svc_model_dict["SVC"] = {
    "model": LinearSVC,
    "base_params": {"random_state": 0},
    "cv_params": {"C": [0.0001, 0.001, 0.005, 0.01, 0.05, 0.1, 1]},
}
```

```
Processing input data…
        Processing of: DS 3…
Done
```

[264]:
```
svc_df = compute_models_cv_acc(svc_model_dict, svc_ds_list)
svc_results = svc_df[['Score', 'Bias', 'Params', 'n_gram', 'lang', 'n_feat']]
```

```
--------- Training all models ---------

Model : SVC
        Dataset [1/4]: DS 0
        Combination 7/7 Best CV Score : 74.0% (Acc: 95.0) [0.5722982883453369
sec]

        Dataset [2/4]: DS 1
        Combination 7/7 Best CV Score : 73.0% (Acc: 99.0) [0.7092363834381104
sec]

        Dataset [3/4]: DS 2
        Combination 7/7 Best CV Score : 74.0% (Acc: 99.0) [0.5993454456329346
```

```
sec]

        Dataset [4/4]: DS 3
        Combination 7/7 Best CV Score : 74.0% (Acc: 99.0) [1.3745067119598389
sec]

Model trained in 3.2752864360809326 sec

Training completed (3.2758843898773193 sec)
```

[265]:
```
svc_results
```

[265]:

	Score	Bias	Params	n_gram	lang	n_feat
3	74.13	5.01	{'C': 0.01}	(1, 2)	False	2000
4	73.99	0.56	{'C': 0.05}	(1, 2)	False	2000
18	73.85	0.70	{'C': 0.05}	(1, 2)	True	2000
25	73.85	0.83	{'C': 0.05}	(1, 3)	True	2000
2	73.57	8.21	{'C': 0.005}	(1, 2)	False	2000
5	73.44	0.00	{'C': 0.1}	(1, 2)	False	2000
11	73.16	0.70	{'C': 0.05}	(1, 3)	False	2000
19	72.60	0.00	{'C': 0.1}	(1, 2)	True	2000
12	72.47	0.14	{'C': 0.1}	(1, 3)	False	2000
17	72.47	7.93	{'C': 0.01}	(1, 2)	True	2000
9	72.47	8.21	{'C': 0.005}	(1, 3)	False	2000
10	72.32	5.84	{'C': 0.01}	(1, 3)	False	2000
26	72.18	0.14	{'C': 0.1}	(1, 3)	True	2000
24	71.90	8.07	{'C': 0.01}	(1, 3)	True	2000
16	70.93	12.52	{'C': 0.005}	(1, 2)	True	2000
27	70.52	0.00	{'C': 1}	(1, 3)	True	2000
20	70.51	0.00	{'C': 1}	(1, 2)	True	2000
13	69.54	0.00	{'C': 1}	(1, 3)	False	2000
23	69.12	12.38	{'C': 0.005}	(1, 3)	True	2000
6	68.99	0.00	{'C': 1}	(1, 2)	False	2000
8	68.01	17.39	{'C': 0.001}	(1, 3)	False	2000
15	67.31	22.67	{'C': 0.001}	(1, 2)	True	2000
1	66.20	17.52	{'C': 0.001}	(1, 2)	False	2000
7	65.49	25.31	{'C': 0.0001}	(1, 3)	False	2000
22	65.09	22.67	{'C': 0.001}	(1, 3)	True	2000
0	64.25	25.03	{'C': 0.0001}	(1, 2)	False	2000
14	59.66	31.29	{'C': 0.0001}	(1, 2)	True	2000
21	59.39	31.57	{'C': 0.0001}	(1, 3)	True	2000

Step 1 - Regularization

[266]:
```
svc_results_step1 = svc_results[(svc_results['n_feat'] == 2000) &␣
 ↪(svc_results['n_gram'] == '(1, 2)') & (svc_results['lang'] == False)]
svc_results_step1
```

[266]:

	Score	Bias	Params	n_gram	lang	n_feat
3	74.13	5.01	{'C': 0.01}	(1, 2)	False	2000
4	73.99	0.56	{'C': 0.05}	(1, 2)	False	2000
2	73.57	8.21	{'C': 0.005}	(1, 2)	False	2000
5	73.44	0.00	{'C': 0.1}	(1, 2)	False	2000
6	68.99	0.00	{'C': 1}	(1, 2)	False	2000
1	66.20	17.52	{'C': 0.001}	(1, 2)	False	2000
0	64.25	25.03	{'C': 0.0001}	(1, 2)	False	2000

Step 2 - N-Grams

[267]:
```
svc_results_step2 = svc_results[(svc_results['n_feat'] == 2000) &␣
 ↪(svc_results['Params'] == {'C': 0.01}) & (svc_results['lang'] == False)]
svc_results_step2
```

[267]:

	Score	Bias	Params	n_gram	lang	n_feat
3	74.13	5.01	{'C': 0.01}	(1, 2)	False	2000
10	72.32	5.84	{'C': 0.01}	(1, 3)	False	2000

Step 3 - Language

[268]:
```
svc_results_step3 = svc_results[(svc_results['n_feat'] == 2000) &␣
 ↪(svc_results['Params'] == {'C': 0.01}) & (svc_results['n_gram'] == '(1, 3)')]
svc_results_step3
```

[268]:

	Score	Bias	Params	n_gram	lang	n_feat
10	72.32	5.84	{'C': 0.01}	(1, 3)	False	2000
24	71.90	8.07	{'C': 0.01}	(1, 3)	True	2000

1.2.4 Decision Tree

[269]:
```
dt_ds_options = {
    'max_feat': [None],
    'lang_id': [False],  # [False, True],
    'feature_type': ['Count'],
    'n_gram': [(1, 2)],
    'lemmatize': [False],
    'feat_select': ['F_CL'],
    'n_feat_select': [2000],
}
dt_ds_list = create_datasets(dt_ds_options)

dt_model_dict = {}
dt_model_dict["DT"] = {
    "model": DecisionTreeClassifier,
    "base_params": {"random_state": 0},
    "cv_params": {"criterion": ['gini', 'entropy'],
                  "max_depth": [50, 100, 500, 1000],
```

```
                    "min_samples_split": [2, 5, 10]},
}
```

```
Processing input data…
        Processing of: DS 0…
Done
```

[270]:
```
dt_df = compute_models_cv_acc(dt_model_dict, dt_ds_list)

dt_df['criterion'] = None
dt_df['max_depth'] = None
dt_df['min_samples_split'] = None

for idx, each_row in dt_df.iterrows():
  for key, val in each_row['Params'].items():
    dt_df.at[idx, key] = val

dt_results = dt_df[['Score', 'Bias', 'criterion','max_depth',␣
 ↪'min_samples_split', 'n_gram', 'lang', 'n_feat']]
```

```
--------- Training all models ---------

Model : DT
        Dataset [1/1]: DS 0
        Combination 24/24       Best CV Score : 59.0% (Acc: 100.0)
[8.762712717056274 sec]

Model trained in 8.768966913223267 sec

Training completed (8.770277261734009 sec)
```

Step 1 - Criterion

[271]:
```
dt_results_step1 = dt_results[(dt_results['n_feat'] == 2000) &␣
 ↪(dt_results['max_depth'] == 100) & (dt_results['min_samples_split'] == 2)]
dt_results_step1
```

[271]:

	Score	Bias	criterion	max_depth	min_samples_split	n_gram	lang	n_feat
3	58.28	0.0	gini	100	2	(1, 2)	False	2000
15	54.39	0.0	entropy	100	2	(1, 2)	False	2000

Step 2 - Max Depth

[272]:
```
dt_results_step2 = dt_results[(dt_results['n_feat'] == 2000) &␣
 ↪(dt_results['criterion'] == 'gini') & (dt_results['min_samples_split'] == 2)]
dt_results_step2
```

[272]:

	Score	Bias	criterion	max_depth	min_samples_split	n_gram	lang	n_feat
9	58.55	0.0	gini	1000	2	(1, 2)	False	2000
3	58.28	0.0	gini	100	2	(1, 2)	False	2000
6	57.17	0.0	gini	500	2	(1, 2)	False	2000
0	55.91	9.6	gini	50	2	(1, 2)	False	2000

Step 3 - Min Samples Split

[273]:
```
dt_results_step3 = dt_results[(dt_results['n_feat'] == 2000) &␣
↪(dt_results['criterion'] == 'gini') & (dt_results['max_depth'] == 50)]
dt_results_step3
```

[273]:

	Score	Bias	criterion	max_depth	min_samples_split	n_gram	lang	n_feat
1	56.47	10.43	gini	50	5	(1, 2)	False	2000
0	55.91	9.60	gini	50	2	(1, 2)	False	2000
2	51.87	13.21	gini	50	10	(1, 2)	False	2000

1.2.5 Final Model

[274]:
```
best_nb_idx = 7
best_svc_idx = 24
```

[275]:
```
create_pred_ds(best_nb_idx, nb_df, nb_ds_list, folder_path + '/
↪NB_Final_prediction.csv')
```

```
Model chosen:
Model name                                          My Bernouilli NB
Score                                                          78.17
Bias                                                           11.82
Acc                                                          0.88178
Dataset                                                        DS 7
Params                                                           {}
Compute time                                               18.110646
Model           <NaiveBayes.NaiveBayes object at 0x78dce2ed2d40>
n_gram                                                       (1, 2)
feat_type                                                      Bin
lemmatized                                                   False
lang                                                         False
standardized                                                 False
rm_accents                                                    True
feat_select                                                   F_CL
n_feat                                                        2000
Name: 7, dtype: object
Predicting test data using this model…
Predictions saved to drive/MyDrive/Colab Notebooks/ECSE
551_MP2/NB_Final_prediction.csv
```

[276]:
```
create_pred_ds(best_svc_idx, svc_df, svc_ds_list, folder_path + '/
↪SVC_Final_prediction.csv')
```

```
Model chosen:
Model name                                        SVC
Score                                            71.9
Bias                                             8.07
Acc                                          0.919332
Dataset                                          DS 3
Params                                    {'C': 0.01}
Compute time                                 1.374507
Model           LinearSVC(C=0.01, random_state=0)
n_gram                                        (1, 3)
feat_type                                      Count
lemmatized                                     False
lang                                            True
standardized                                   False
rm_accents                                      True
feat_select                                     F_CL
n_feat                                          2000
Name: 24, dtype: object
Predicting test data using this model…
Predictions saved to drive/MyDrive/Colab Notebooks/ECSE
551_MP2/SVC_Final_prediction.csv
```

1.3 Features Analysis

Check the words that are most probably observed for a given class

[277]:
```
nb_info = nb_df.loc[best_nb_idx]
nb_model = nb_info['Model']
nb_model_ds = find_ds_from_name(nb_info['Dataset'], nb_ds_list)
features_name = nb_model_ds.features_name

n_best_features = 10

df_dict = {}
for k, class_label in enumerate(nb_model._classes):
    feats_score = nb_model._thetas[k, 1::]
    names_scores = list(zip(features_name, feats_score))
    feat_scores_df = pd.DataFrame(data=names_scores, columns=['Feat_names',␣
↪'Score'])
    feat_scores_df = feat_scores_df.sort_values(by=['Score'], ascending=False).
↪reset_index(drop=True)
    df_dict[class_label] = feat_scores_df

combined_df = pd.concat(df_dict, axis=1)
# print(combined_df.head(n_feats).to_string())
```

```
combined_df.head(n_best_features)
```

[277]:

	London		Montreal		Paris		Toronto	
	Feat_names	Score	Feat_names	Score	Feat_names	Score	Feat_names	Score
0	like	0.296703	people	0.225275	paris	0.318681	people	0.248619
1	london	0.280220	montreal	0.186813	plus	0.269231	like	0.243094
2	people	0.263736	would	0.175824	ca	0.263736	one	0.204420
3	one	0.252747	get	0.175824	si	0.203297	toronto	0.198895
4	get	0.225275	like	0.159341	tout	0.181319	would	0.187845
5	also	0.192308	one	0.153846	etre	0.164835	city	0.171271
6	really	0.159341	ca	0.142857	faire	0.142857	also	0.165746
7	would	0.153846	good	0.126374	quand	0.142857	get	0.160221
8	know	0.153846	go	0.120879	comme	0.131868	time	0.160221
9	see	0.142857	time	0.115385	fait	0.126374	new	0.149171