CMPE 461
Natural Language Processing Application Project #2

Tuna Han Salih Meral
Mustafa Melih Mutlu

December 13, 2018

1 Introduction

In this project, we implemented a language identification system using both a generative system (Naive Bayes) and a discriminative model (Support Vector Machine). We used the Discriminating between Similar Languages (DSL) Shared Task 2015 corpus that includes 13 languages and 2,000 sentences for each language. The languages in the corpus consists of:

- South-Eastern Slavic
  - Bulgarian (bg)
  - Macedonian (mk)
- South-Western Slavic
  - Bosnian (bs)
  - Croatian (hr)
  - Serbian (sr)
- West Slavic
  - Czech (cz)
  - Slovak (sk)
- Portuguese
  - Brazilian Portuguese (pt-BR)
  - European Portuguese (pt-PT)
- Spanish
  - Argentine Spanish (es-AR)
  - Peninsular Spanish (es-ES)
- Austronesian
  - Malay (my)
  - Indonesian (id)

For generative part of the project, we used Naive Bayes classifier which utilizes letter frequencies in given documents. The idea behind using letter frequencies for language identification is that different languages use different letters with different frequencies. For training and testing, corpus includes 26000 instances which consists of

2000 sentences for each language. The corpus divided into two random set that consists of 90% (train set) and 10% (test set) of the sentences respectively. An instance in the corpus consists of a sentence and the language class at the end of the line.

For discriminative part of the project, we used Support Vector Machine classifier which utilizes given features selected and extracted from given corpus. For the SVM part, SVM^{multiclass} library of Cornell University is used. The library consists of two parts, leraner and classifier.

Learner part of the library takes input in the form:

```
language-id f1:v1 f2:v2 ... fn:vn
```

The learner part of the program learns the importance of features in order to classify languages according to given features.

Classifier part of the program takes the trained model and features from the test data as the same from of learner program and outputs the predicted languages and statistics about predictions.

For the third part of the project, we selected and extracted features from the corpus in order to feed the learner with them.

For the last part of the project, we evaluated outputs of generative and discriminative classifiers. For evaluation, we have measured

- Accuracy, in terms of true positives, true negatives, false positives and false negatives
- Micro-averaged precision, recall and F_1 score
- Macro-averaged precision, recall and F_1 score
- Confusion matrix

2 Program Interface

2.1 Generative Model

Naive Bayes Classifier program need Python version 3.x and numpy library of python. Numpy library can be installed via pip (package manager for python libraries)

from the command line:

```
$ pip install numpy
```

Naive Bayes classifier is a python scripts that takes two inputs from user at the start of the program, path of the corpus and the seed that will be used to create random variables. To start the program, you can write this command in the command line:

```
$ python language_identification.py \
    -f "Project (Application 2) (Corpus).txt" \
    -s 53 \
    -o "statistics.txt"
```

2.2 Discriminative Model

For this model, we utilized Support Vector Machine learning. SVM$^{\text{Multiclass}}$ library is used to learn and classify data.

SVM$^{\text{Multiclass}}$ needs a specific kind of input format. Thus, we had to create an input file to feed SVM$^{\text{Multiclass}}$. `feature_selection` class is responsible to create such an input file which contains selected features and their weights.

To start program, following command should be executed:

```
$ python feature_selection.py \
    -f "Project (Application 2) (Corpus).txt" \
    -p b
```

After creating feature file, we can train SVM and calssify test data:

```
$ ./svm_multiclass_learn \
    -c 1.0
    train.txt \
    model.txt

$ ./svm_multiclass_classify \
    test.txt \
    model.txt \
    output.txt
```

3 Input and Output

3.1 Generative Model

Generative model of the project two kind of inputs. One if the corpus and the other type is command line arguments. Input corpus of the application is Discriminating between Similar Languages (DSL) Shared Task 2015 corpus. The corpus consists of lines that contains a sentence and a language label: `"<sentence> <language-id>"`

The options are:

- `-f` option takes path of the corpus
- `-s` option takes an integer as the seed for random number generation
- `-o` option takes the output file path to print statistics

The output of the program is the overall statistics and statistics about each language. An example output for the Bosnian language is like this:

```
bs:
    TP: 72
    TN: 2007
    FP: 431
    FN: 90
    Precision: 0.03463203463203463
    Recall: 0.4444444444444444
    F-measure: 0.0642570281124498
```

another output is the confusion matrix. This matrix compares the predicted and expected labels and allows user to examine which languages are most likely to resemble each other.

3.2 Discriminative Model

`feature_selection` class has to be run before using Support Vector Machine learning.

The options for `feature_selection` are:

- `-f` option takes path of the corpus
- `-p` option takes 'b' or 'c' as input which specifies the part of the project that will be run

Output of this command will be two files; "test.txt" and "train.txt". Both files have same features of different sentences in original the corpus.

An example line from "test.txt" of "train.txt" would be:

1 3 : 1 4 : 1 5 : 1 6 : 1 7 : 1 8 : 1 9 : 1 10 : 1 ...

Now, input files are ready to trains SVM. Firstly, $SVM^{Multiclass}$ should be trained by using "train.txt" file. Training $SVM^{Multiclass}$ will give a file which contains the model according to features that we selected in feature selection step.

That model file can be used to classify test data. After executing `svm_multiclass_classify` command, a brief overview of the result will be printed on terminal.

```
Reading model...done.
Reading test examples... (2600 examples) done.
Classifying test examples...done
Runtime (without IO) in cpu-seconds: 0.02
Average loss on test set: 32.9615
Zero/one-error on test set: 32.96% (1743 correct, 857 incorrect, 2600 total)
```

Figure 1: Result of SVM classifying

4 Program Structure

4.1 Parsing Input

The Corpus is encoded in UTF-16-LE encoding which encodes each character with 16 bits (2 bytes). The Corpus consists of 26000 instances, which includes 2000 sentences for each language. A line contains a sentence and its language id at the end of it. To parse the dataset, we used regex. The regex we used is:

r '(.*)([^-]\w{2}|\w{2}-\w{2})'

The search done with the given regex returns two groups, one of which consists of the sentence and the other is the language-id. But returned groups should be stripped

from preceding and trailing space characters.

4.2 Train Test Split

After reading each line and storing them in a list, we need to split train and test data. Train data consists of 90% of the data and test data consists of 10% of the data. The data is shuffled before this separation in order to create a reasonable test set.

4.3 Generative Model

Generative part of the application takes three command line argument and parses them. First argument to parse is the path of the corpus. Second is the seed for random number generation, and the third is the file to output statistics.

4.3.1 Training Bayes Classifier

Training the Naive Bayes classifier means that evaluating letter frequencies for each language. To handle unknown letters we will encounter during test phase, we applied Laplace smoothing.

$$P(w_i) = \frac{c_i + 1}{N + V} \tag{1}$$

where N is the number of all letters and V is the number of unique letters. Which is why thw probability of an unknown word can be calculated as:

$$P(w_i) = \frac{1}{N + V} \tag{2}$$

since c_i is 0 for the unknown letter in the training set.

4.3.2 Predictions

With the trained data, predictions are made with the formula:

$$P(l \mid s) = \frac{P(s \mid l) * P(l)}{P(s)} \approx P(s \mid l) * P(l)(\prod_{i=1}^{n} P(c_i \mid l)) \tag{3}$$

where

$$P(c_i) = \frac{count(c_i \& l)}{couunt(l)} = \frac{\#\text{ of times letter l is encountered in language l}}{\#\text{ of letters encountered in language l}} \tag{4}$$

4.4 Discriminative Model

Since we used a third party library for Support Vector Machine, only part that we had to do for SVM was feature selection. There are various features that we selected with respect to characters, their occurences, lengths and counts.

4.4.1 Character Occurences

This feature is the only feature that used in Part B. Every character in the corpus has unique label. If a sentence contains that character, we added its corresponding label with weight 1 in the feature dictionart of that sentence.

4.4.2 Punctuations

Punctuations may also contribute to classify languages. Some languages might be using specific punctuations. That's why labeling the sentences which includes punctuations is one of the features that we are interested in.

4.4.3 Character Bigrams

This feature is the one that increases the result dramaticaly. Character occurences is already one of our features. However, adding also character bigrams made a difference about 15%.

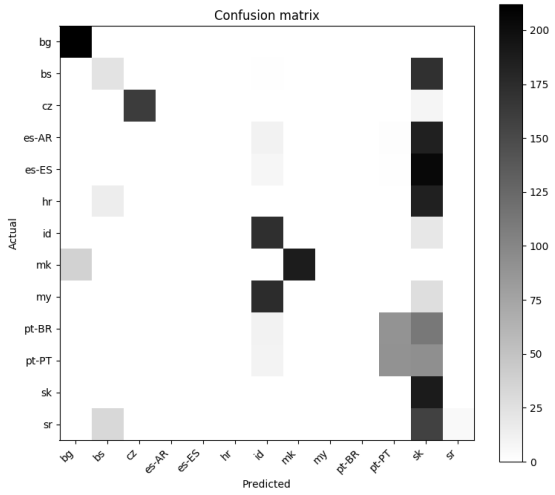

Figure 2: Confusion Matrix in Part B

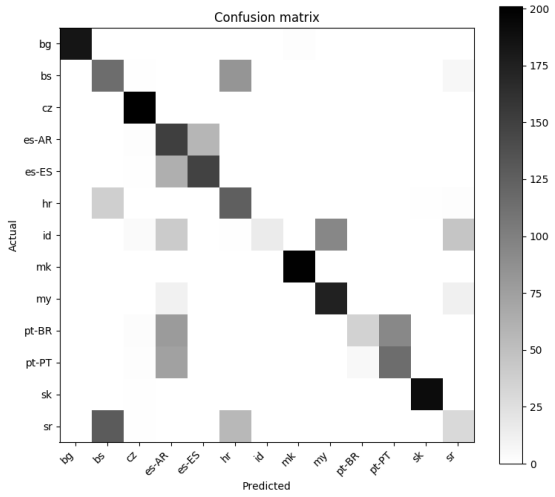

Figure 3: Confusion Matrix in Part C

4.4.4 Other Features

There are two other basic features that we implemented for classifying by using SVM. Those are length of sentence and number of words in a sentence.

L = 'bs'	True Language is (L)	True Language is (not L)
Predicted as (L)	TP=72	FP=431
Predicted as (not L)	FN=90	TN=2007

Table 1: Bosnian Language

4.5 Calculating Statistics

For calculating statistics, false positive, false negative, true positive and true negative of each language is calculated.

- True Positive: The language is predicted as L and the expected language is L
- False Positive: The language is predicted as L and the expected language is not L
- True Negative: The language is predicted as not L and the expected language is not L
- False Negative: The language is predicted as not L and the expected language is L
- $Accuracy = \frac{TruePositive+TrueNegative}{TruePositive+TrueNegative+FalsePositive+FalseNegative}$
- $Precision_i = \frac{TruePositive_i}{TruePositive_i+TrueNegative_i}$
- $Recall_i = \frac{TruePositive_i}{TruePositive_i+FalseNegative_i}$
- $FMeasure_i = \frac{2*Precision_i*Recall_i}{Precision_i+Recall_i}$
- $Precision = \frac{\sum_{i=1}^{M} TruePositive_i}{\sum_{i=1}^{M}(TruePositive_i+TrueNegative_i)}$
- $Recall = \frac{\sum_{i=1}^{M} TruePositive_i}{\sum_{i=1}^{M}(TruePositive_i+FalseNegative_i)}$
- $FMeasure = \frac{2*Precision*Recall}{Precision+Recall}$
- And, confusion matrix is plotted

Calculated statistics for Naive Bayes Classifier is:

Statistics about each languge can be found in appendix.

L = 'bg'	True Language is (L)	True Language is (not L)
Predicted as (L)	TP=183	FP=0
Predicted as (not L)	FN=14	TN=2403

Table 2: Bulgarian Language

L = 'sk'	True Language is (L)	True Language is (not L)
Predicted as (L)	TP=188	FP=1
Predicted as (not L)	FN=17	TN=2394

Table 3: Slovak Language

L = 'hr'	True Language is (L)	True Language is (not L)
Predicted as (L)	TP=134	FP=98
Predicted as (not L)	FN=70	TN=2298

Table 4: Crotian Language

L = 'sr'	True Language is (L)	True Language is (not L)
Predicted as (L)	TP=98	FP=64
Predicted as (not L)	FN=95	TN=2343

Table 5: Serbian Language

L = 'pt-PT'	True Language is (L)	True Language is (not L)
Predicted as (L)	TP=130	FP=66
Predicted as (not L)	FN=97	TN=2307

Table 6: European Portuguese Language

L = 'es-AR'	True Language is (L)	True Language is (not L)
Predicted as (L)	TP=90	FP=27
Predicted as (not L)	FN=121	TN=2362

Table 7: Argentine Spanish Language

L = 'id'	True Language is (L)	True Language is (not L)
Predicted as (L)	TP=113	FP=45
Predicted as (not L)	FN=100	TN=2342

Table 8: Indonesian Language

L = 'pt-BR'	True Language is (L)	True Language is (not L)
Predicted as (L)	TP=110	FP=78
Predicted as (not L)	FN=89	TN=2323

Table 9: Brazilian Portuguese Language

L = 'mk'	True Language is (L)	True Language is (not L)
Predicted as (L)	TP=195	FP=0
Predicted as (not L)	FN=12	TN=2393

Table 10: Macedonian Language

L = 'cz'	True Language is (L)	True Language is (not L)
Predicted as (L)	TP=168	FP=1
Predicted as (not L)	FN=21	TN=2410

Table 11: Czech Language

L = 'es-ES'	True Language is (L)	True Language is (not L)
Predicted as (L)	TP=60	FP=41
Predicted as (not L)	FN=146	TN=2353

Table 12: Peninsular Spanish Language

L = 'my'	True Language is (L)	True Language is (not L)
Predicted as (L)	TP=125	FP=82
Predicted as (not L)	FN=62	TN=2331

Table 13: Malay Language

Overall Statistics	True Language is (L)	True Language is (not L)
Predicted as (L)	TP=1666	FP=934
Predicted as (not L)	FN=934	TN=30266

Table 14: Overall Statistics

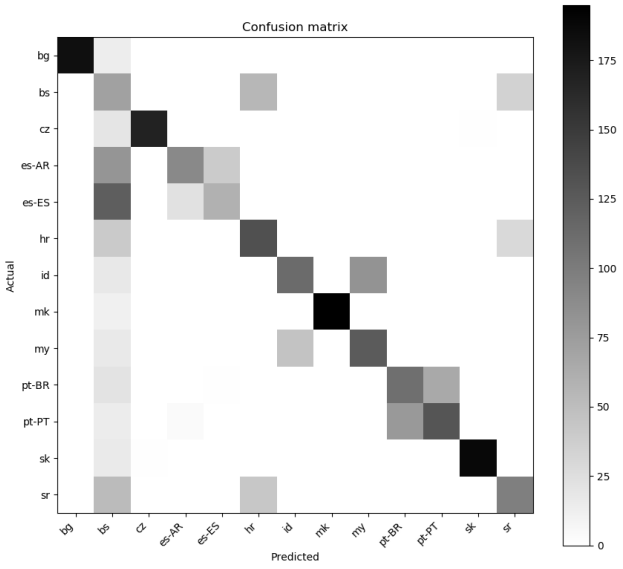

Figure 4: Confusion Matrix of Naive Bayes Classifier

5 Improvements and Extensions

In this project we tried two different approaches to classify languages. One of them was Support Vector Machine. Most significant thing in SVM is that selecting appropriate features to train SVM. We used various features to get these results. However, we've seen that considering good features can still impact the results in a highly successful way.

Other that deciding what features we need to select, deciding the feature weights is also important. In this project we used 1 for every feature that occurs in a sentence. Changing those weights might result in better success rates.

6 Difficulties Encountered

The first difficulty we have encountered was to find the encoding of the corpus. It was said that it is a Unicode encoded file but, we needed to encode data with UTF-16-LE encoding.

The main difficulty of this project was to decide feature vectors that will be used to train classifiers. In Naive Bayes Classifier, we have used letter frequencies but some other features could have been employed for better results. The independence assumption of Naive Bayes classifier makes it difficult to use different parameters since some dependency exists between letters.

In SVM classifier, feature selection was the most crucial part. In addition, we had to give each feature reasonable weights. Since there are several features we had difficulty in deciding weights.

7 Conclusion

As a conclusion, feature selection is the most important part of the classification with machine learning. Naive Bayes classification was successful with 64% success rate in terms of base success rate of 7.7%. The success rate can be improved with more features selected from corpus but Bayes classifier is limited since its independence assumption. SVM classifier performs well when it has been fed with good features. With unigram letters, success rate was below 50% but after adding some more features like bigram letters success rate improves drastically.

8 References

KRÍŽ, Vincent; HOLUB, Martin; PECINA, Pavel. Feature Extraction for Native Language Identification Using Language Modeling. In: Proceedings of the International Conference Recent Advances in Natural Language Processing. 2015. p. 298-306.

9 Appendix

9.1 Statistical Outputs

Statistics for Part A:

```
// Statistics with Naive Bayes in Part A

bs:
TP: 72
TN: 2007
FP: 431
FN: 90
Precision: 0.03463203463203463
Recall: 0.4444444444444444
F-measure: 0.0642570281124498
bg:
TP: 183
TN: 2403
FP: 0
FN: 14
Precision: 0.07076566125290024
Recall: 0.9289340101522843
F-measure: 0.1315127560186849
sk:
TP: 188
TN: 2394
FP: 1
FN: 17
Precision: 0.07281177381874517
Recall: 0.9170731707317074
F-measure: 0.13491209185504127
hr:
TP: 134
TN: 2298
FP: 98
FN: 70
Precision: 0.055098684210526314
Recall: 0.6568627450980392
F-measure: 0.10166919575113809
sr:
TP: 98
```

TN: 2343
FP: 64
FN: 95
Precision: 0.04014748054076198
Recall: 0.5077720207253886
F-measure: 0.07441154138192863
pt-PT:
TP: 130
TN: 2307
FP: 66
FN: 97
Precision: 0.053344275748871565
Recall: 0.5726872246696035
F-measure: 0.09759759759759759
es-AR:
TP: 90
TN: 2362
FP: 27
FN: 121
Precision: 0.0367047308319739
Recall: 0.4265402843601896
F-measure: 0.06759294029290275
id:
TP: 113
TN: 2342
FP: 45
FN: 100
Precision: 0.04602851323828921
Recall: 0.5305164319248826
F-measure: 0.08470764617691157
pt-BR:
TP: 110
TN: 2323
FP: 78
FN: 89
Precision: 0.04521167283189478

Recall: 0.5527638190954773
F-measure: 0.08358662613981763
mk:
TP: 195
TN: 2393
FP: 0
FN: 12
Precision: 0.07534775888717156
Recall: 0.9420289855072463
F-measure: 0.13953488372093023
cz:
TP: 168
TN: 2410
FP: 1
FN: 21
Precision: 0.06516679596586501
Recall: 0.8888888888888888
F-measure: 0.12143115287314782
es-ES:
TP: 60
TN: 2353
FP: 41
FN: 146
Precision: 0.024865312888520515
Recall: 0.2912621359223301
F-measure: 0.045819014891179836
my:
TP: 125
TN: 2331
FP: 82
FN: 62
Precision: 0.050895765472312705
Recall: 0.6684491978609626
F-measure: 0.09458948164964057
overall:
TP: 1666

```
TN: 30266
FP: 934
FN: 934
Precision: 0.05217336840786672
Recall: 0.6407692307692308
F-measure: 0.09649021197729643
```

Statistics For Part B:

```
// Statistics with SVM in Part B

bg:
TP: 222
TN: 2300
FP: 78
FN: 0
Precision: 0.0880253766851705
Recall: 1.0
F-measure: 0.16180758017492713
bs:
TP: 197
TN: 2002
FP: 383
FN: 18
Precision: 0.08958617553433379
Recall: 0.9162790697674419
F-measure: 0.1632145816072908
cz:
TP: 188
TN: 2400
FP: 0
FN: 12
Precision: 0.07264296754250386
Recall: 0.94
F-measure: 0.13486370157819227
es-AR:
```

TP: 144
TN: 2084
FP: 325
FN: 47
Precision: 0.06463195691202872
Recall: 0.7539267015706806
F-measure: 0.11905746176105828
es-ES:
TP: 7
TN: 2405
FP: 0
FN: 188
Precision: 0.002902155887230514
Recall: 0.035897435897435895
F-measure: 0.005370157268891445
hr:
TP: 0
TN: 2389
FP: 0
FN: 211
Precision: 0.0
Recall: 0.0
F-measure: unknown
id:
TP: 0
TN: 2391
FP: 0
FN: 209
Precision: 0.0
Recall: 0.0
F-measure: unknown
mk:
TP: 112
TN: 2410
FP: 0
FN: 78

Precision: 0.04440919904837431
Recall: 0.5894736842105263
F-measure: 0.08259587020648969
my:
TP: 150
TN: 2282
FP: 122
FN: 46
Precision: 0.061677631578947366
Recall: 0.7653061224489796
F-measure: 0.1141552511415525
pt-BR:
TP: 3
TN: 2407
FP: 1
FN: 189
Precision: 0.0012448132780082987
Recall: 0.015625
F-measure: 0.0023059185242121443
pt-PT:
TP: 0
TN: 2402
FP: 1
FN: 197
Precision: 0.0
Recall: 0.0
F-measure: unknown
sk:
TP: 197
TN: 1933
FP: 470
FN: 0
Precision: 0.09248826291079812
Recall: 1.0
F-measure: 0.16931671680275032
sr:

```
TP: 0
TN: 2415
FP: 0
FN: 185
Precision: 0.0
Recall: 0.0
F-measure: unknown
overall:
TP: 1220
TN: 29820
FP: 1380
FN: 1380
Precision: 0.039304123711340205
Recall: 0.46923076923076923
F-measure: 0.07253269916765756
```

Statistics For Part C:

```
// Statistics with SVM in Part C

bg:
TP: 184
TN: 2414
FP: 0
FN: 2
Precision: 0.07082371054657428
Recall: 0.989247311827957
F-measure: 0.132183908045977
bs:
TP: 115
TN: 2226
FP: 167
FN: 92
Precision: 0.049124305852199915
Recall: 0.5555555555555556
F-measure: 0.09026687598116169
```

cz:
TP: 201
TN: 2384
FP: 15
FN: 0
Precision: 0.07775628626692456
Recall: 1.0
F-measure: 0.14429289303661164
es-AR:
TP: 151
TN: 2121
FP: 268
FN: 60
Precision: 0.0664612676056338
Recall: 0.7156398104265402
F-measure: 0.121627064035441
es-ES:
TP: 148
TN: 2330
FP: 58
FN: 64
Precision: 0.05972558514931396
Recall: 0.6981132075471698
F-measure: 0.11003717472118961
hr:
TP: 126
TN: 2292
FP: 141
FN: 41
Precision: 0.052109181141439205
Recall: 0.7544910179640718
F-measure: 0.09748549323017408
id:
TP: 16
TN: 2397
FP: 0

FN: 187
Precision: 0.00663075010360547
Recall: 0.07881773399014778
F-measure: 0.012232415902140671
mk:
TP: 199
TN: 2399
FP: 2
FN: 0
Precision: 0.07659738260200154
Recall: 1.0
F-measure: 0.14229531641043977
my:
TP: 174
TN: 2308
FP: 95
FN: 23
Precision: 0.07010475423045931
Recall: 0.883248730964467
F-measure: 0.12989921612541994
pt-BR:
TP: 36
TN: 2382
FP: 6
FN: 176
Precision: 0.01488833746898263
Recall: 0.16981132075471697
F-measure: 0.02737642585551331
pt-PT:
TP: 115
TN: 2309
FP: 94
FN: 82
Precision: 0.04744224422442244
Recall: 0.583756345177665
F-measure: 0.0877527661198016

```
sk:
TP: 191
TN: 2407
FP: 1
FN: 1
Precision: 0.073518090839107
Recall: 0.9947916666666666
F-measure: 0.1369175627240143
sr:
TP: 30
TN: 2317
FP: 67
FN: 186
Precision: 0.01278227524499361
Recall: 0.1388888888888889
F-measure: 0.023410066328521262
overall:
TP: 1686
TN: 30286
FP: 914
FN: 914
Precision: 0.05273364193669461
Recall: 0.6484615384615384
F-measure: 0.09753557792433183
```

9.2 Code

Naive Bayes Classifier

```
import re
import argparse
import numpy as np
from pandas_ml import ConfusionMatrix
import matplotlib.pyplot as plt

ap = argparse.ArgumentParser()
```

```
ap.add_argument('-f', '--filename', required=True, help='Path to corpus')
ap.add_argument('-s', '--seed', type=int, default=53, help='Seed for random nu
ap.add_argument('-o', '--output', help='Path to output statistics. If given,
args = vars(ap.parse_args())
np.random.seed(args['seed'])

UNKNOWN_LETTER = "<UNK>"


def read_data(filename='devel.txt', return_unlabeled=False):
    train_data = []
    train_labels = []

    unlabeled_data = []

    with open(filename, 'r', encoding="UTF-16-LE") as f:
        train_count = 0
        unlabeled_count = 0
        regex = r'(.*)([^-]\w{2}|\w{2}-\w{2})'
        for line in f:
            match = re.search(regex, line)
            sentence = "".join(match.group(1).split())
            label = match.group(2).strip()

            train_count += 1
            train_data.append(sentence)
            train_labels.append(label)

        print(f"Total instances: {train_count}")

    if return_unlabeled:
        return train_data, train_labels, unlabeled_data
    else:
        return train_data, train_labels
```

```
def split_train_test(data, labels, test_split=0.1):
    count = len(data)
    data = np.array(data)
    labels = np.array(labels)
    shuffled_indices = np.random.permutation(np.arange(count))
    train_indices = shuffled_indices[:int(count * (1 - test_split))]
    test_indices = shuffled_indices[int(count * (1 - test_split)):]
    train_data = data[train_indices]
    train_labels = labels[train_indices]
    test_data = data[test_indices]
    test_labels = labels[test_indices]
    return train_data, test_data, train_labels, test_labels


def train_bayes(train_data, train_labels):
    language_statistics = {}
    all_letters = []
    for i in range(len(train_data)):
        sentence = train_data[i]
        label = train_labels[i]
        letters = list(sentence)
        letters = np.array(letters)
        letters, counts = np.unique(letters, return_counts=True)
        all_letters.extend(letters)
        language_statistics[label] = language_statistics.get(label, {})
        for l, c in zip(letters, counts):
            language_statistics[label][l] = language_statistics[label].get(l,

    # Statistics with Laplace Smoothing
    all_letters = np.array(all_letters)
    all_letters, letter_counts = np.unique(all_letters, return_counts=True)
    for k, v in language_statistics.items():
        total_letter_count = np.sum(list(language_statistics[k].values()))
        for l, c in language_statistics[k].items():
            language_statistics[k][l] = (language_statistics[k][l] + 1) / (to
        language_statistics[k][UNKNOWN_LETTER] = 1 / (total_letter_count + len
```

```
    all_letters = dict(zip(all_letters, letter_counts))
    return language_statistics, all_letters


def calculate_statistics(languages, y_predicted, y_expected):
    statistics = {}
    y_predicted = np.array(y_predicted)
    y_expected = np.array(y_expected)
    for language in languages:
        statistics[language] = {}
        true_positive = np.sum(y_predicted[y_expected == language] == language
        statistics[language]["TP"] = true_positive
        true_negative = np.sum(y_predicted[y_expected != language] != language
        statistics[language]["TN"] = true_negative
        false_positive = np.sum(y_predicted[y_expected != language] == langua
        statistics[language]["FP"] = false_positive
        false_negative = np.sum(y_predicted[y_expected == language] != langua
        statistics[language]["FN"] = false_negative
        precision = true_positive / (true_positive + true_negative)
        recall = true_positive / (true_positive + false_negative)
        f_score = (2 * precision * recall) / (precision + recall)
        statistics[language]["Precision"] = precision
        statistics[language]["Recall"] = recall
        statistics[language]["F-measure"] = f_score

    sum_tp = 0
    sum_tn = 0
    sum_fp = 0
    sum_fn = 0
    for language in languages:
        sum_tp += statistics[language]["TP"]
        sum_tn += statistics[language]["TN"]
        sum_fp += statistics[language]["FP"]
        sum_fn += statistics[language]["FN"]

    statistics["overall"] = {}
```

```
    statistics["overall"]["TP"] = sum_tp
    statistics["overall"]["TN"] = sum_tn
    statistics["overall"]["FP"] = sum_fp
    statistics["overall"]["FN"] = sum_fn
    statistics["overall"]["Precision"] = sum_tp / (sum_tp + sum_tn)
    statistics["overall"]["Recall"] = sum_tp / (sum_tp + sum_fn)
    statistics["overall"]["F-measure"] = (2 * statistics["overall"]["Precision
            statistics["overall"]["Precision"] + statistics["overall"]["Recal

    return statistics


def print_statictics(statistics, filename):
    with open(filename, "w") as f:
        for language, statistic in statistics.items():
            f.write(f"{language}:\n")
            for s, v in statistics[language].items():
                f.write(f"\t{s}: {v}\n")


data, labels = read_data(filename=args['filename'])
train_data, test_data, train_labels, test_labels = split_train_test(data, lab

language_statistics, all_letters = train_bayes(train_data, train_labels)
languages = list(language_statistics.keys())

predictions = []
for sentence in test_data:
    probabilities = {}
    letters = list(sentence)
    for language in languages:
        probabilities[language] = 1.0
        for letter in letters:
            probabilities[language] *= language_statistics[language].get(lette
                                                                        lang
```

```
    predicted_class = languages[0]
    max_prob = 0
    for k, v in probabilities.items():
        if v > max_prob:
            max_prob = v
            predicted_class = k
    predictions.append(predicted_class)

true = 0
for a, b in zip(predictions, test_labels):
    if a == b:
        true += 1

statistics = calculate_statistics(languages, predictions, test_labels)

if args["output"]:
    print_statictics(statistics, args["output"])

confusion_matrix = ConfusionMatrix(test_labels, predictions)
print("Confusion matrix:\n%s" % confusion_matrix)
confusion_matrix.plot()
plt.show()
```

Feature Selection

```
import numpy as np
import re
import collections
import argparse

feature_label = 0
lang_id = 0
feature_dict = {}
language_id_dict = {}
word_counts = {}

ap = argparse.ArgumentParser()
```

```
ap.add_argument('-p', '--part', required=True)
args = vars(ap.parse_args())

def read_data(filename='Corpus.txt', return_unlabeled=False):
    output_array = []
    regex = r'(.*)([^-]\w{2}|\w{2}-\w{2})'

    with open(filename, 'r', encoding="UTF-16-LE") as f:
        for line in f:
            output_line = ''

            match = re.search(regex, line)
            sentence = "".join(match.group(1))
            label = match.group(2).strip()

            sentence_features = {}
            output_line += get_language_id(label)

            sentence_features = char_feature(sentence, sentence_features)
            if args['part'] == 'c':
                sentence_features = word_count(sentence, sentence_features)
                sentence_features = sentence_length(sentence, sentence_featur
                sentence_features = bigrams(sentence, sentence_features)

            output_line += get_line_from_features(sentence_features)
            output_array.append(output_line)

    return output_array


def get_line_from_features(features):
    ordered_dict = collections.OrderedDict(sorted(features.items()))

    line = ''
    for k, v in ordered_dict.items():
        line += ' ' + str(k) + ':' + str(v)
```

```
    return line

def get_language_id(label):
    global lang_id
    if label in language_id_dict:
        return str(language_id_dict.get(label))
    else:
        lang_id += 1
        language_id_dict[label] = lang_id
        return str(lang_id)

def char_feature(sentence, sentence_features):
    global feature_label
    for c in sentence:
        if c in feature_dict:
            label = feature_dict.get(c)
            if c not in sentence_features:
                sentence_features[label] = 1
        else:
            feature_label += 1
            feature_dict[c] = feature_label
            sentence_features[feature_label] = 1
    return sentence_features

def word_count(sentence, sentence_features):
    global feature_label
    words = sentence.split()
    if len(words) in feature_dict:
        label =  feature_dict.get(len(words))
        sentence_features[label] = 1
    else:
        feature_label += 1
        feature_dict[len(words)] = feature_label
        sentence_features[feature_label] = 1
    return sentence_features
```

```
def sentence_length(sentence, sentence_features):
    global feature_label
    s = "".join(sentence.split())
    if len(s) in feature_dict:
        label = feature_dict.get(len(s))
        sentence_features[label] = 1
    else:
        feature_label += 1
        feature_dict[len(s)] = feature_label
        sentence_features[feature_label] = 1

    return sentence_features

def bigrams(sentence, sentence_features):
    global feature_label
    s = "".join(sentence.split())
    bigram_list = [s[i:i+2] for i in range(len(s)-1)]
    for bi in bigram_list:
        if bi in feature_dict:
            label = feature_dict.get(bi)
            if bi not in sentence_features:
                sentence_features[label] = 1
        else:
            feature_label += 1
            feature_dict[bi] = feature_label
            sentence_features[feature_label] = 1
    return sentence_features

def write_array(data, filename):
    with open(filename, 'w') as f:
        for line in data:
            line += '\n'
            f.write(line)

data = read_data()
```

```
np.random.shuffle(data)
train_data = data[:int(len(data)*0.9)]
test_data = data[int(len(data)*0.9):]

write_array(train_data, "train.txt")
write_array(test_data, "test.txt")
```