Language Identification from Text Documents

Syed Mehedi Hasan Nirob (2012331030)

Md. Kazi Nayeem (2012331027)

July 28, 2017

I. Problem

We have a dataset that contains sentences written in 14 different language. We have to apply some machine learning technique to train a system with this dataset so that it can predict the language of a particular sentence.

II. Data Collection

We collected data from the official repository for the Disciminating between Similar Language (DSL) Shared Task 2015.
Link : `https://github.com/Simdiva/DSL-Task`

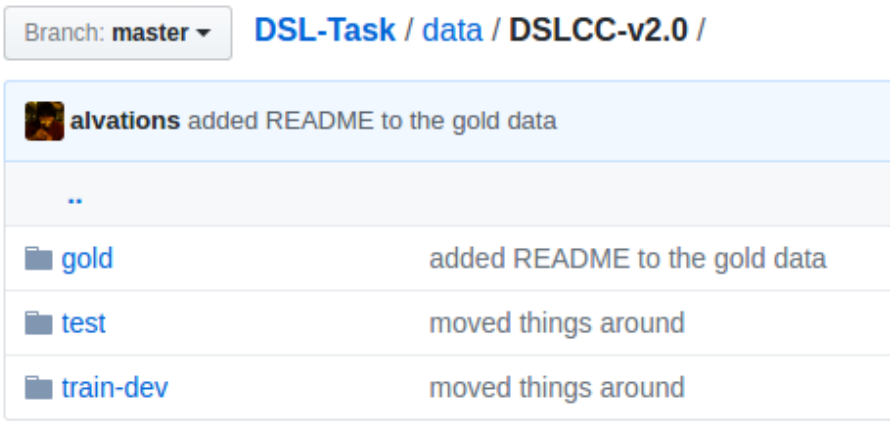

Figure 1: DSL-Task Data in github

This repository contains the following:

- DSL Corpus Collection (DSLCC) version 2.0 (training, dev, test and gold data included)
- DSL Shared Task submissions from participating teams
- The script to blind Named Entities (NEs) for the Test Set B in DSL-2015 (blindNE.py)
- The evaluation script to evaluate outputs.
- The evaluation script to evaluate all submitted system.

Only train.txt file from train-dev folder was used for language identification task. This file contains 2,52,000 sentences in 14 different language.

III. Experiment and Performance

We used Multinomial Naive Bayes and Logistic Regression for both character n-gram and word n-gram features.

So, The following code has 4 different parts.

- Feature : Character n-grams, Algorithm : Logistic Regression
- Feature : Character n-grams, Algorithm : Multinomial Naive Bayes
- Feature : Word n-grams, Algorithm : Logistic Regression
- Feature : Word n-grams, Algorithm : Multinomial Naive Bayes

70% of total data set was used for training purpose and 30% of data set was used for testing purpose. The implementation code is given in the next section.

.1 Python Code

```
#import

fr = open('train.txt', 'r' , encoding="utf8")

language_list = []
line_list = []
for line in fr:
        line_list.append(line)
        word_list = line.split()
        language_list.append(word_list[-1])

def func_word_gram(gram, feature,algo):
        vectorizer_word = TfidfVectorizer(encoding = "utf8", ngram_range=(gram, gram),
        X_train_word = vectorizer_word.fit_transform(line_list)
        features_word = vectorizer_word.get_feature_names();

        num_row_word = 2800 #how many sentence
        num_col_word = feature

        if(algo == 2):
                num_col_word += 1

        feature_matrix_word = [[0 for x in range(num_col_word)] for y in range(num_row_
        feature_matrix_word_shuffle = [[0 for x in range(num_col_word)] for y in range(
```

```
for i in range(0,num_row_word):
        for j in range(0,num_col_word):
                if(algo == 2 and j == num_col_word - 1):
                        feature_matrix_word[i][j] = language_list[i]
                        continue

                word_list = ngrams(line_list[i].split(), gram)
                for grams in word_list:
                        if gram==1 and grams[0]==features_word[j]:
                                feature_matrix_word[i][j] = 1
                        elif gram==2 and (grams[0]+' '+grams[1]) == features_wo
                                feature_matrix_word[i][j] = 1
                        elif gram==3 and (grams[0]+' '+grams[1]+' '+grams[2]) =
                                feature_matrix_word[i][j] = 1
                        elif gram==4 and (grams[0]+' '+grams[1]+' '+grams[2]+' '
                                feature_matrix_word[i][j] = 1
                        elif gram==5 and (grams[0]+' '+grams[1]+' '+grams[2]+' '
                                feature_matrix_word[i][j] = 1
                        elif gram==6 and (grams[0]+' '+grams[1]+
                                ' '+grams[2]+' '+grams[3]+' '+grams[4]+' '+gram
                                feature_matrix_word[i][j] = 1
                        elif gram==7 and (grams[0]+' '+grams[1]+' '+grams[2]+' '
                                feature_matrix_word[i][j] = 1

if(algo == 1):
        logreg_model_word = LogisticRegression(C=1e5)
        cv = ShuffleSplit(n_splits=5, test_size=0.3, random_state=0)
        scores_word = cross_val_score(logreg_model_word, feature_matrix_word, l
        print(scores_word)

if(algo == 2):
        feature_matrix_word_shuffle = feature_matrix_word
        np.random.shuffle(feature_matrix_word_shuffle)
        train = 1960
        language_list_shuffle = []
        for i in range(0,num_row_word):
                language_list_shuffle.append(feature_matrix_word_shuffle[i][num

        for row in feature_matrix_word_shuffle:
                del row[num_col_word - 1]

        trainMatrix = feature_matrix_word_shuffle[:train]
        testMatrix = feature_matrix_word_shuffle[train:]
        trainLabel = language_list_shuffle[:train]
        testLabel = language_list_shuffle[train:]

        clf = GaussianNB()
```

```
                clf.fit(trainMatrix, trainLabel)
                predicted = clf.predict(testMatrix)
                print(accuracy_score(testLabel,predicted))
                print(predicted)

def func_character_gram(gram, feature, algo):
        vectorizer_char = TfidfVectorizer(encoding = "utf8", analyzer = 'char', ngram_r
        X_train_char = vectorizer_char.fit_transform(line_list)
        features_char = vectorizer_char.get_feature_names();

        num_row_char = 2800
        num_col_char = feature

        if(algo == 2):
                num_col_char += 1

        feature_matrix_char = [[0 for x in range(num_col_char)] for y in range(num_row_
        feature_matrix_char_shuffle = [[0 for x in range(num_col_char)] for y in range(

        for grams in features_char:
                fw.write(grams+ '\n')
        count = 0
        for i in range(0,num_row_char):
                for j in range(0,num_col_char):
                        if(algo == 2 and j == num_col_char - 1):
                                feature_matrix_char[i][j] = language_list[i]
                                continue

                        word_list = [line_list[i][i:i+gram] for i in range(len(line_lis
                        for grams in word_list:
                                fw.write(grams + '\n')
                        #print(word_list.encoding("utf8"))
                        for grams in word_list:
                                if grams==features_char[j]:
                                        feature_matrix_char[i][j] = 1

        if(algo == 1):
                logreg_model_word = LogisticRegression(C=1e5)
                cv = ShuffleSplit(n_splits=5, test_size=0.3, random_state=0)
                scores_word = cross_val_score(logreg_model_word, feature_matrix_char, l
                print(scores_word)

        if(algo == 2):
                feature_matrix_char_shuffle = feature_matrix_char
                np.random.shuffle(feature_matrix_char_shuffle)
                train = 1960
                language_list_shuffle = []
                for i in range(0,num_row_char):
```

```
            language_list_shuffle.append(feature_matrix_char_shuffle[i][num

        for row in feature_matrix_char_shuffle:
            del row[num_col_char - 1]

        trainMatrix = feature_matrix_char_shuffle[:train]
        testMatrix = feature_matrix_char_shuffle[train:]
        trainLabel = language_list_shuffle[:train]
        testLabel = language_list_shuffle[train:]

        clf = GaussianNB()
        clf.fit(trainMatrix, trainLabel)
        predicted = clf.predict(testMatrix)
        print(accuracy_score(testLabel,predicted))
        print(predicted)

#function call

func_word_gram(1,200,2)
func_character_gram(1,20,2)
```

.2 Performance

To save time just for testing purpose at first we used a small portion of our data. 200 sentences for each 14 languages, total 2800 sentences. Table 1 shows performance of Multinomial Naive Bayes algorithm for this small dataset.

Table 1: Performance for 2800 sentences

Algorithm	Features	Number of Features	Accuracy
Multinomial Naive Bayes	word 1-grams	200	86.4%
Multinomial Naive Bayes	word 1-grams	1000	87%

Table 2 shows performance of Logistic Regression algorithm for the whole dataset that contains 2,52,000 sentences in 14 languages.

Table 2: Performance for 252000 sentences

Algorithm	Features	Number of Features	Accuracy
Logistic Regression	word 1-grams	200	89%
Logistic Regression	word 1-grams	2000	91.3%

We can see accuracy/performance increases as data size and number of feature increases.

IV. Future Work

Applying Recurrent Neural Network(RNN) to the same dataset to gain better accuracy in some particular case.