**CMPE 561**

**NATURAL LANGUAGE PROCESSING**

**APPLICATION PROJECT #2**

**Students**

Ali Yeşilkanat 2017700159

Çağatay Şahin 2015700192

**Submission Date**

18.12.2017

**Introduction**

In natural language processing, language identification or language guessing is the problem of determining which natural language given content is in. Computational approaches to this problem view it as a special case of text categorization, solved with various statistical methods.

On this work, main goal is identifying the language of a sentence using both generative and discriminative models. Discriminating between Similar Languages (DSL) Shared Task 2015 corpus [1] that includes 13 languages and 2,000 sentences for each language is used for both training and testing these models.

As generative model, Naïve Bayes model is used. Probability of the sentences are calculated using Maximum Likelihood Estimation [2][3]. While assigning probabilities to the sentences of languages, character counts in the languages is used.

For this project, two different discriminative models are developed. They both use Multiclass Support Vector Machine learning method. First model, assigns a feature id to each single character (unigram) on sentences however, second model assigns a feature id to each bigram.

Both generative and discriminative models exclude some special characters such as punctuations and numbers. Also for generative modeling, Laplace Smoothing is applied to training set using test set to prevention of assigning zero values to some sentence probabilities for some languages.

.

**Program Design & Implementation**

The application, is developed on Ubuntu environment using both Python 2.7 language and SVMmulticlass [4] bash commands for discriminative models. As Python IDE, Jupyter Notebook is used. For learning and classifying SVM’s “svm\_multiclass\_learn” and “svm\_multiclass\_classify” bash commands are used [5].

Three different Jupyter Notebook programs created for Naïve Bayes, Unigram SVM and Bigram SVM models. Each notebook files have all phases from training to evaluating metrics independently, and they all extract training and test partitions as text files, which they have been generated during execution

We divide the corpus into training and test sets “randomly” such that, for each language, 1800 of the sentences are in the training set and 200 are in the test set. In total, training set contains 23400 sentences and test set contains 2600 sentences.

**Program Structure**

For three different models program structures should be defined and explained differently. However, with respect to the Machine Learning concepts they may divided into three sub parts which are training, testing, evaluation.

For preprocessing, all punctuations, parenthesis’, numbers etc. are excluded from corpus, however this is done intuitively on both in the training phase and testing phase for performance. And for each modeling, initially, training and test partitions are created randomly from corpus for each 13 languages.

Evaluation metrics will be explained just once instead of for all models because they have the same process to calculate metrics. So below we explained training and testing phases for each model then explained how we evaluate models and show the metrics for each model.

1. **Generative Modeling – Naïve Bayes:** 
   1. **Assigning Probabilities - Training Phase:** Count of each character in the sentences in the training set for all languages are stored. If these characters are in predefined insignificant tokens list, then they are not stored. After summing character counts for each language in the training set, we also look test set for unexpected characters. If we find any character, then we make its count as 1. And calculate the probability of the character given language:

So at the end, we have a probability value for each character given a language which is .

* 1. **Maximum Likelihood Estimation - Testing Phase:** We will use values to determine the language of a sentence using Maximum Likelihood Estimation.

Probability of the language given sentence can be defined alternatively using chain rule.

can be excluded from this equation because it has no effect on the result. Likelihood distribution can be expressed as multiplication of probability of character given language as:

Prior can be excluded from this equation because all languages distributed uniformly. So, for all languages and their sentences in the training set we calculate

As a result,

identifies the language of the sentence.

For all sentence, we calculate the probability using MLE for all languages. We skip the insignificant tokens for calculation. So, for a sentence we have 13 different probabilities. As MLE formula, we choose the language which assigns the highest probability to the sentence. With this, we could have predicted the language for all sentences in the test set.

1. **Discriminative Modeling – Unigram Multiclass Support Vector Machine:** We use Cornell SVMmulticlass for learning and classifying so we need to create feature vectors from our corpus suitable for this program.

**2.1) Training Phase:** We assign each character a unique feature number to create feature vectors for sentences in the corpus. We do this using a function defined in the Python library which returns integer representation Unicode code of a character. We do not assign any feature number insignificant tokens (explained in the previous sections) which do not have any effect to language identification.

Having a unique feature value for each character, we express sentences as feature vectors as,

(class-id) is the id of the language of the sentence (1-13).

... are the unique feature values of characters. Each feature is a distinct character (letter) that occurs only in that sentence.

are the weights. Since we assign same weight to each feature value, we give 1 for all weights of the features in the lines.

For example, a sentence which is Bulgarian (language id is 1)

Казусът се гледа повторно от СГС, след като миналата година съдебният заседател Петър Георгиев бе арестуван от МВР при акция "Педофили" и така блокира хода на процеса.

Is converted into feature vector as,

1 128:1 129:1 130:1 131:1 132:1 133:1 134:1 138:1 143:1 146:1 147:1 154:1 156:1 159:1 160:1 161:1 176:1 177:1 178:1 179:1 180:1 181:1 183:1 184:1 186:1 187:1 188:1 189:1 190:1 191:1 208:1 209:1

So, we create feature vectors for each sentence in the training set for all languages and extract it to text file. Then, we use this text file to train (learn) SVM using shell commands and get a model file which allows us to classify in the future.

**2.2) Testing Phase:** We use same method in 2.1 to create feature vector lines for test set which is previously partitioned from corpus, and export all feature lines as text file to classify SVM.

Again, we use SVMmulticlass bash commands to test our test set on previously created model. And, with this execution we obtain text file which contains predictions of our model. In evaluation stage, we will explain them in detail.

1. **Discriminative Modeling with Increased Features – Unigram & Bigram Multiclass Support Vector Machine:** We simply use the same methods to train and test SVM, using unigrams and additionally bigrams which we extracted from corpus.

**3.1) Training Phase:** First we assign each character a unique feature number. We use set data structure which is defined in Python, to make it efficient. Then, we split sentences using spaces to extract words and extract bigrams from words by two letters in sequence one by one. Same as the unigram numeration, we add each bigram to the set data structure and assign a unique feature number to each bigram in the set.

In unigram model, we skip insignificant tokens and do not accept these characters as features. We still use this assumption, also if a bigram contains these tokens, then we simply ignore this bigram and do not assign any feature number to it.

With these rules and methods, we create feature vectors for each sentence in the training set for each language as we mentioned in the unigram model.

For example, a Bulgarian sentence which we have given as example in the unigram model,

Казусът се гледа повторно от СГС, след като миналата година съдебният заседател Петър Георгиев бе арестуван от МВР при акция "Педофили" и така блокира хода на процеса.

Is converted into feature vector as,

1 1:1 2:1 3:1 4:1 7:1 8:1 12:1 13:1 14:1 15:1 30:1 31:1 34:1 42:1 43:1 44:1 45:1 49:1 50:1 51:1 60:1 61:1 65:1 69:1 70:1 74:1 75:1 76:1 82:1 91:1 93:1 94:1 96:1 107:1 108:1 109:1 110:1 140:1 143:1 232:1 264:1 265:1 329:1 344:1 370:1 371:1 481:1 482:1 516:1 517:1 605:1 743:1 756:1 763:1 912:1 919:1 921:1 940:1 942:1 963:1 1019:1 1039:1 1059:1 1060:1 1095:1 1125:1 1126:1 1374:1 1393:1 1440:1 1447:1 1479:1 1534:1 1535:1 1576:1 1577:1 1606:1 1626:1 1627:1 1673:1 1674:1 1722:1 1783:1 1828:1 1829:1 1830:1 1832:1 1833:1 1834:1 1835:1 1836:1 1837:1 1838:1 1839:1 1840:1 1841:1 1842:1 1845:1 1848:1 1856:1 1857:1 1858:1 1866:1 1873:1 1887:1

So, we create feature vectors for each sentence in the training set for all languages and extract it to text file. Then, we use this text file to train (learn) SVM using shell commands and get a model file which allows us to classify in the future.

**3.2) Testing Phase:** We use SVM learn shell command to predict our test set, which contains feature vectors for all sentences in the test set. Then, we obtain a text file which contains predictions. Process is same as the part 2.2 and we will evaluate these predictions in the next part.

1. **Evaluation:** On the previous parts, we defined how we design and implement our models, and how we trained and tested them. Each testing stage produces a prediction list (or text file for SVM but we read this file and converted into a list). So, after now we are going to define which metrics we used and calculate these metrics using obtained predictions lists.

**4.1) Metrics Definitions:** Several types of measures for evaluating the quality of a system.Metrics are calculated using,

**TP (True Positive):** Number of sentences whose language is L and predicted as L.

**FN (False Negative):** Number of sentences whose language is L but predicted as not L.

**FP (False Positive):** Number of sentences whose language is not L but predicted as L.

**FN (False Negative):** Number of sentences whose language is not L and predicted as not L.

So, we extract these for each 13 languages and for total test set.

**4.1.1) Accuracy:** This is equal to the number of correct predictions divided by the number of sentences. We calculated this for each language and for the entire test set for our models’ predictions. Can be formulized as,

True positives are, number of sentences whose language is L and predicted as L.

Set size is 200 and 2600 for each language and entire test set respectively.   
  
**4.1.2)** **Micro-averaged Precision, Recall, F-measure:** Micro-averaged F-measure gives equal weight to each sentence and is therefore considered as an average over all the sentence/language(class) pairs. For calculating it, first we need micro-averaged precision and micro-averaged recall.

**4.1.3) Macro-averaged Precision, Recall, F-measure:** Macro-averaged F-measure gives equal weight to each language (class), regardless of number of sentences in that class. For calculating it, first we need precision, recall and f-measure for all languages.

Then, using these we obtain macro-averaged versions.

Since we have 13 languages, M equals to 13.

**4.2) Evaluation Metrics:** We calculate metrics for three different models separately. Size of the test set is 2600 and language count(M) is 13.

**4.2.1) Naïve Bayes:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Language** | **True Positives** | **False Positives** | **True Negatives** | **False Negatives** |
| **bg** | 45 | 0 | 2400 | 155 |
| **hr** | 28 | 7 | 2393 | 172 |
| **es-AR** | 1 | 42 | 2358 | 199 |
| **mk** | 200 | 0 | 2400 | 0 |
| **sk** | 160 | 0 | 2400 | 40 |
| **cz** | 21 | 0 | 2400 | 179 |
| **sr** | 8 | 2 | 2398 | 192 |
| **pt-BR** | 2 | 166 | 2234 | 29 |
| **bs** | 171 | 357 | 2043 | 30 |
| **my** | 170 | 15 | 2385 | 0 |
| **pt-PT** | 200 | 5 | 2395 | 191 |
| **es-ES** | 9 | 228 | 2172 | 3 |
| **id** | 1 | 2 | 2398 | 199 |
| ***Averages*** | **78.15** | **63.38** | **2336.62** | **106.85** |

|  |  |
| --- | --- |
| **Micro-averaged precision** | 0.595085995086 |
| **Micro-averaged recall** | 0.465769230769 |
| **Micro-averaged f1-score** | 0.522545846818 |
| **Macro-averaged precision** | 0.678708254064 |
| **Macro-averaged recall** | 0.465769230769 |
| **Macro-averaged f1-score** | 0.427828196335 |

|  |  |
| --- | --- |
| **Language** | **Accuracies** |
| **bg** | 0.225 |
| **hr** | 0.14 |
| **es-AR** | 0.005 |
| **mk** | 1.0 |
| **sk** | 0.8 |
| **cz** | 0.105 |
| **sr** | 0.04 |
| **pt-BR** | 0.855 |
| **bs** | 0.85 |
| **my** | 1.0 |
| **pt-PT** | 0.045 |
| **es-ES** | 0.985 |
| **id** | 0.005 |
| ***Averages*** | **0.466** |

﻿

**4.2.2) Unigram Multiclass Support Vector Machine:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Language** | **True Positives** | **False Positives** | **True Negatives** | **False Negatives** |
| **bg** | 196 | 0 | 2400 | 4 |
| **hr** | 124 | 74 | 2326 | 76 |
| **es-AR** | 71 | 27 | 2373 | 129 |
| **mk** | 200 | 0 | 2400 | 0 |
| **sk** | 195 | 5 | 2395 | 5 |
| **cz** | 195 | 1 | 2399 | 5 |
| **sr** | 98 | 110 | 2290 | 102 |
| **pt-BR** | 95 | 63 | 2337 | 105 |
| **bs** | 44 | 42 | 2358 | 156 |
| **my** | 143 | 5 | 2395 | 57 |
| **pt-PT** | 124 | 92 | 2308 | 76 |
| **es-ES** | 183 | 129 | 2271 | 17 |
| **id** | 109 | 62 | 2338 | 91 |
| ***Averages*** | **136.69** | **46.92** | **2353.08** | **63.31** |

|  |  |
| --- | --- |
| **Micro-averaged precision** | 0.744449099288 |
| **Micro-averaged recall** | 0.683461538462 |
| **Micro-averaged f1-score** | 0.712652897534 |
| **Macro-averaged precision** | 0.743765662284 |
| **Macro-averaged recall** | 0.683461538462 |
| **Macro-averaged f1-score** | 0.699124738159 |

﻿

|  |  |
| --- | --- |
| **Language** | **Accuracies** |
| **bg** | 0.98 |
| **hr** | 0.62 |
| **es-AR** | 0.355 |
| **mk** | 1.0 |
| **sk** | 0.975 |
| **cz** | 0.975 |
| **sr** | 0.49 |
| **pt-BR** | 0.475 |
| **bs** | 0.22 |
| **my** | 0.715 |
| **pt-PT** | 0.62 |
| **es-ES** | 0.915 |
| **id** | 0.545 |
| ***Averages*** | ﻿0.683 |

﻿

**4.2.3) Unigram & Bigram Multiclass Support Vector Machine:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Language** | **True Positives** | **False Positives** | **True Negatives** | **False Negatives** |
| **bg** | 199 | 0 | 2400 | 1 |
| **hr** | 146 | 21 | 2379 | 54 |
| **es-AR** | 130 | 50 | 2350 | 70 |
| **mk** | 200 | 0 | 2400 | 0 |
| **sk** | 200 | 0 | 2400 | 0 |
| **cz** | 198 | 0 | 2400 | 2 |
| **sr** | 157 | 96 | 2304 | 43 |
| **pt-BR** | 147 | 66 | 2334 | 53 |
| **bs** | 74 | 48 | 2352 | 126 |
| **my** | 164 | 0 | 2400 | 36 |
| **pt-PT** | 133 | 52 | 2348 | 67 |
| **es-ES** | 149 | 67 | 2333 | 51 |
| **id** | 152 | 37 | 2363 | 48 |
| ***Averages*** | **157.62** | **33.62** | **2366.38** | **42.38** |

|  |  |
| --- | --- |
| **Micro-averaged precision** | 0.824215607401 |
| **Micro-averaged recall** | 0.788076923077 |
| **Micro-averaged f1-score** | 0.805741250492 |
| **Macro-averaged precision** | 0.825130141461 |
| **Macro-averaged recall** | 0.788076923077 |
| **Macro-averaged f1-score** | 0.802062544561 |

|  |  |
| --- | --- |
| **Language** | **Accuracies** |
| **bg** | 0.995 |
| **hr** | 0.73 |
| **es-AR** | 0.65 |
| **mk** | 1.0 |
| **sk** | 1.0 |
| **cz** | 0.99 |
| **sr** | 0.785 |
| **pt-BR** | 0.735 |
| **bs** | 0.37 |
| **my** | 0.82 |
| **pt-PT** | 0.665 |
| **es-ES** | 0.745 |
| **id** | 0.76 |
| ***Averages*** | **0.788** |

* 1. **Overall Results**

We plotted results in the previously shown. It can be seen that bigram & unigram model has the best quality among 3 models and Naïve Bayes has the least.

Interestingly, Naïve Bayes almost cannot identify some languages for example id,es-Ar. Also, probably mk language has its own unique characters in its alphabet, because in all models their accuracy for identifying this language is 100%.

**Improvements and Extensions**

In this project, we extended unigram SVM model to bigram & unigram SVM model, and this improved the quality as can be seen from the previous section. However, there can be more features to extract for improving the quality of the model such as the length of a sentence, the average word length, the number of punctuation marks in a sentence, the existence/number of capital letters in a sentence etc.

Also, some different classifiers other than SVM and Naïve Bayes could be used.

**Difficulties Encountered**

Since we have fixed corpus size which does not have huge size, evaluating model makes us suspicious. If we could have larger corpus, calculated metrics may show even better results or worse.

Also, because of using SVMmulticlass tool, we need to export our training and data set feature vectors into text files and read the predictions of SVM again. So, this create an unnecessary I/O instead using directly from Python without any file export and import process.

**Conclusion**

We applied language identification of sentences using both generative and discriminative modeling. For both models, characters are the main identifier of the sentence.

For generative modeling, Naïve Bayes classifier is applied. Each character in a language is assigned with a probability and with Maximum Likelihood Estimation, language of the sentence is predicted.

For discriminative modeling, SVMmulticlass is used with different features which are unigram and unigram & bigram SVM. For unigram SVM model, each character assigned with a unique feature number with it sentences are converted into feature vectors and SVM is trained and tested with sequences of feature vectors.

For bigram & unigram SVM model, we assigned unique feature numbers to unigram also bigrams. And applied the same training and testing process as unigram model.

In the end, we plotted results with commonly used evaluation metrics in Machine Learning area. And figured out that, bigram & unigram SVM model has the best quality among three models.

**References**

[1] https://github.com/Simdiva/DSL-Task

[2] https://www.wikiwand.com/en/Maximum\_likelihood\_estimation

[3] https://onlinecourses.science.psu.edu/stat414/node/191

[4] https://www.cs.cornell.edu/people/tj/svm\_light/svm\_multiclass.html

[5] https://www.cs.cornell.edu/people/tj/svm\_light/index.html

**Appendix**

**GenerativeModelingNaiveBayes.py**

import random

#these tokens will be skipped while calculating probability

insignificant\_tokens=[' ','!', '"','#','$','%','&','\*','+','-','(',')',',','.','/','0','1','2','3','4','5','6','7','8',

'9',';','<','>','=','?','@','|','«','»','`','[',']',"'",'\\']

language\_ids=['bg','bs','cz','es-AR','es-ES','hr','id','mk','my','pt-BR','pt-PT','sk','sr']

class Language: #for each language in data set, we create a Language class

def \_\_init\_\_ (self,lang\_id):

self.chars={} # storing characters and their counts

self.prob={} #storing p(c|l) probability for each character

# corresponding language id

self.lang\_id=lang\_id

def calculate\_probability(self,sentence):

s=list(sentence)

s=[x for x in s if x not in insignificant\_tokens]

total=0

for ch in s:

if self.chars.has\_key(ch):

total+=self.prob[ch]

else:

return 0

return total

def get\_chars(self):

return self.chars

def get\_prob(self):

return self.prob

def get\_lang\_id(self):

return self.lang\_id

with open ("Corpus/Raw Corpus.txt") as f:

corpus = f.readlines()

languages=[]

model=[]

for i in range(13): #divides data set as languages

languages.append("")

start=i\*2000

end=(i+1)\*2000

languages[i]=corpus[start:end] #2k sentence for each language

training\_set=[]

test\_set=[]

for idx,l in enumerate(languages):

lang=Language(language\_ids[idx])

random.shuffle(l)

training\_partition=l[0:1800]

training\_set.extend(training\_partition)

training\_partition=[i.split('\t')[0] for i in training\_partition] #remove language identifier at the last of the sentences

for sentence in training\_partition:

for letter in sentence:

if letter not in insignificant\_tokens:

if lang.get\_chars().has\_key(letter):

nominator=lang.get\_chars()[letter]

else:

nominator=0 #laplace

nominator+=1

lang.get\_chars()[letter]=nominator

test\_partition=l[1800:2000]

test\_set.extend(test\_partition)

test\_part=[i.split('\t')[0] for i in test\_partition] #remove language identifier at the last of the sentences

unk=list(''.join(set(''.join(test\_part))))

unk=[x for x in unk if x not in insignificant\_tokens]

unk=[x for x in unk if x not in lang.get\_chars().keys()]

if len(unk)!=0: # add unknowns characters in test set for smoothing

for l in unk:

lang.get\_chars()[l]=0

for ch in lang.chars: #laplace smoothing

lang.get\_chars()[ch]+=1

denominator=sum(lang.get\_chars().values())+len(lang.get\_chars().values()) #for performance

for letter in lang.get\_chars(): #calculate probabilities

lang.get\_prob()[letter]=(lang.get\_chars()[letter]+1)/float(denominator)

model.append(lang)

# In[9]:

i=0

predictions=[]

expected=[]

for sentence in test\_set:

s=sentence.split('\t')

probabilities={}

expected.append(language\_ids.index(s[1].strip())+1)

for l\_model in model: #foreach language calculate the probability of the given sentence

probabilities[l\_model.get\_lang\_id()]=l\_model.calculate\_probability(s[0])

#find one language having most likely given sentence

predictions.append(language\_ids.index(max(probabilities.items(), key=lambda k: k[1])[0])+1)

# In[10]:

metrics={}

for idx,lang\_id in enumerate(language\_ids):

false\_negatives=len([x for x in predictions[idx\*200:(idx+1)\*200] if int(x) != (idx+1)])

true\_positives=200-false\_negatives

if idx==0: #count false positives on proceeding predictions

false\_positives=len([x for x in predictions[200:len(predictions)] if int(x) ==1])

elif idx==12: #count false positives on preceeding predictions

false\_positives=len([x for x in predictions[0:2400] if int(x)==13])

else: #count false positives on both preceeding and proceeding predictions

false\_positives=[x for x in predictions[(idx-1)\*200:idx\*200] if int(x) == (idx+1)]

false\_positives=len(false\_positives + [x for x in predictions[(idx+1)\*200:2600] if int(x) == (idx+1)])

true\_negatives=2400-false\_positives

metrics[lang\_id]={"tp":true\_positives,"fn":false\_negatives,"tn":true\_negatives,"fp":false\_positives}

for key,value in metrics.iteritems():

print(key)

print("True positives: " + str(value["tp"]))

print("False positives: " + str(value["fp"]))

print("True negatives: " + str(value["tn"]))

print("False negatives: " + str(value["fn"]))

# In[11]:

total\_tp=0.0

total\_fp=0.0

total\_fn=0.0

total\_tn=0.0

total\_precision=0.0

total\_recall=0.0

total\_f1score=0.0

for key in metrics.keys():

tp=metrics[key]["tp"]

fp=metrics[key]["fp"]

fn=metrics[key]["fn"]

tn=metrics[key]["tn"]

precision=tp/float(tp+fp)

recall=tp/float(tp+fn)

f1\_score=(2\*recall\*precision)/float(recall+precision)

total\_precision+=precision

total\_recall+=recall

total\_f1score+=f1\_score

total\_tp+=tp

total\_fp+=fp

total\_fn+=fn

total\_tn+=tn

mic\_prec=total\_tp/float(total\_tp+total\_fp)

mic\_recall=total\_tp/float(total\_tp+total\_fn)

print("Micro-averaged precision: " + str(mic\_prec))

print("Micro-averaged recall: " + str(mic\_recall))

print("Micro-averaged f1-score: " + str((2\*mic\_prec\*mic\_recall)/float(mic\_prec+mic\_recall)))

print("")

print("Macro-averaged precision: " + str(total\_precision/13.0))

print("Macro-averaged recall: " + str(total\_recall/13.0))

print("Macro-averaged f1-score: " + str(total\_f1score/13.0))

print("")

print("Total accuracy: "+ str(total\_tp/2600.0))

print("Accuracies for languages:")

for key,value in metrics.iteritems():

print(key + str(": ")+str(value["tp"]/200.0))

print("")

print("fp: "+str(total\_fp))

print("tp: "+str(total\_tp))

print("fn: "+str(total\_fn))

print("tn: "+str(total\_tn))

metrics

# In[ ]:

def write\_files(file\_name,array):

with open (file\_name, mode='wt') as t\_file:

for item in array:

t\_file.write(item)

print(len(training\_set))

print(len(test\_set))

write\_files("Training set.txt",training\_set)

write\_files("Test set.txt",test\_set)

**DiscriminativeModeling.py**

import random,subprocess

def generate\_model\_line(s,idx): #generates svm input vectors using predefined character numbers

sentence=[x for x in list(''.join(set(''.join(s.split('\t')[0])))) if x not in insignificant\_tokens]

model\_line =str(idx+1)

character\_list=[]

for ch in sentence:

character\_list.append(character\_numbers[ch])

character\_list.sort()

for ch in character\_list:

model\_line+=" "+ str(ch)+ ":" + str(1)

return model\_line+"\n"

insignificant\_tokens=[' ','!', '"','#','$','%','&','\*','+','-','(',')',',','.','/','0','1','2','3','4','5','6','7','8',

'9',';','<','>','=','?','@','|','«','»','`','[',']',"'",'\\']#these tokens will be skipped

language\_ids=['bg','bs','cz','es-AR','es-ES','hr','id','mk','my','pt-BR','pt-PT','sk','sr']

with open ("Corpus/Raw Corpus.txt") as f:

corpus = f.readlines()

languages=[]

corpus\_s=list(''.join(set(''.join(corpus))))

corpus\_s=[x for x in corpus\_s if x not in insignificant\_tokens]

character\_numbers={}

for s in corpus\_s:

character\_numbers[s]=ord(s) #assign each character to a number using ordinal of a one-character string.

for i in range(13): #divides data set as languages

languages.append("")

start=i\*2000

end=(i+1)\*2000

languages[i]=corpus[start:end] #2k sentence for each language

training\_set=[]

test\_set=[]

training\_model=[]

test\_model=[]

for idx,l in enumerate(languages): #assign training and test partitions to each language

random.shuffle(l)

training\_partition=l[0:1800]

training\_set.extend(training\_partition)

for s in training\_partition:

training\_model.append(generate\_model\_line(s,idx))

test\_partition=l[1800:2000]

test\_set.extend(test\_partition)

for s in test\_partition:

test\_model.append(generate\_model\_line(s,idx))

# In[2]:

def write\_files(file\_name,array):

with open (file\_name, mode='wt') as t\_file:

for item in array:

t\_file.write(item)

print(len(training\_set))

print(len(test\_set))

write\_files("SVM/TrainingModel-SVM.txt",training\_model) #svm inputs as training set

write\_files("SVM/TrainingSet-SVM.txt",training\_set) #which sentences in the training set

write\_files("SVM/TestSet-SVM.txt",test\_set) #svm inputs as test set

write\_files("SVM/TestModel-SVM.txt",test\_model) #which sentences in the test set

# In[3]:

p=subprocess.Popen(['SVM/svm\_multiclass\_learn', '-c', '5000' ,'SVM/TrainingModel-SVM.txt' ,'SVM/Model'],

stdout=subprocess.PIPE) #train svm using shell command

p.wait()

# In[4]:

p=subprocess.Popen(['SVM/svm\_multiclass\_classify', 'SVM/TestModel-SVM.txt' ,'SVM/Model','SVM/predictions.txt'],

stdout=subprocess.PIPE) #classify using test set using shell command

for line in p.stdout:

print(line)

p.wait()

# In[5]:

with open ("SVM/predictions.txt") as f:

predictions = f.readlines()

predictions=[i.split(' ')[0] for i in predictions]

metrics={}

for idx,lang\_id in enumerate(language\_ids):

false\_negatives=len([x for x in predictions[idx\*200:(idx+1)\*200] if int(x) != (idx+1)])

true\_positives=200-false\_negatives

if idx==0: #count false positives on proceeding predictions

false\_positives=len([x for x in predictions[200:len(predictions)] if int(x) ==1])

elif idx==12: #count false positives on preceeding predictions

false\_positives=len([x for x in predictions[0:2400] if int(x)==13])

else: #count false positives on both preceeding and proceeding predictions

false\_positives=[x for x in predictions[(idx-1)\*200:idx\*200] if int(x) == (idx+1)]

false\_positives=len(false\_positives + [x for x in predictions[(idx+1)\*200:2600] if int(x) == (idx+1)])

true\_negatives=2400-false\_positives

metrics[lang\_id]={"tp":true\_positives,"fn":false\_negatives,"tn":true\_negatives,"fp":false\_positives}

for key,value in metrics.iteritems():

print(key)

print("True positives: " + str(value["tp"]))

print("False positives: " + str(value["fp"]))

print("True negatives: " + str(value["tn"]))

print("False negatives: " + str(value["fn"]))

# In[6]:

total\_tp=0.0

total\_fp=0.0

total\_fn=0.0

total\_tn=0.0

total\_precision=0.0

total\_recall=0.0

total\_f1score=0.0

for key in metrics.keys():

tp=metrics[key]["tp"]

fp=metrics[key]["fp"]

fn=metrics[key]["fn"]

tn=metrics[key]["tn"]

precision=tp/float(tp+fp)

recall=tp/float(tp+fn)

f1\_score=(2\*recall\*precision)/float(recall+precision)

total\_precision+=precision

total\_recall+=recall

total\_f1score+=f1\_score

total\_tp+=tp

total\_fp+=fp

total\_fn+=fn

total\_tn+=tn

#calc macro

#calc micro

mic\_prec=total\_tp/float(total\_tp+total\_fp)

mic\_recall=total\_tp/float(total\_tp+total\_fn)

print("Micro-averaged precision: " + str(mic\_prec))

print("Micro-averaged recall: " + str(mic\_recall))

print("Micro-averaged f1-score: " + str((2\*mic\_prec\*mic\_recall)/float(mic\_prec+mic\_recall)))

print("")

print("Macro-averaged precision: " + str(total\_precision/13.0))

print("Macro-averaged recall: " + str(total\_recall/13.0))

print("Macro-averaged f1-score: " + str(total\_f1score/13.0))

print("")

print("Total accuracy: "+ str(total\_tp/2600.0))

print("Accuracies for languages:")

for key,value in metrics.iteritems():

print(key + str(": ")+str(value["tp"]/200.0))

print("")

print("fp: "+str(total\_fp))

print("tp: "+str(total\_tp))

print("fn: "+str(total\_fn))

print("tn: "+str(total\_tn))

metrics

# In[7]:

for key,value in metrics.iteritems():

print(key + "\t"+ str(value["tp"]) + "\t"+ str(value["fp"])+"\t"+str(value["tn"])+"\t"+str(value["fn"]))

# In[9]:

print("Accuracies for languages:")

for key,value in metrics.iteritems():

print(str(value["tp"]/200.0))

print("")

**DiscriminativeModelingWithIncreasedFeatures.py**

import random,subprocess

from enum import Enum

from collections import Counter

def generate\_model\_line(s,idx): #generates svm input vectors using predefined character numbers

feature\_list=[]

raw\_sentence=s.split('\t')[0]

sentence=[x for x in list(''.join(set(''.join(raw\_sentence)))) if x not in insignificant\_tokens]

model\_line =str(idx+1)

#add unigrams

for ch in sentence:

feature\_list.append(character\_numbers[ch])

#add bigrams

bigrams=set()

for words in raw\_sentence.split(' '):

bigrams.update([x.lower() for x in[words[i:i+2] for i in range(len(words)-1)]])

bigrams\_list=list([x for x in bigrams if (x[0] not in insignificant\_tokens)and(x[1] not in insignificant\_tokens)])

for bg in bigrams\_list:

feature\_list.append(character\_numbers[bg])

feature\_list=list(set(feature\_list))

feature\_list.sort()

for ch in feature\_list:

model\_line+=" "+ str(ch)+ ":" + str(1)

return model\_line+"\n"

#these tokens will be skipped for both creating unigrams and bigrams

insignificant\_tokens=[' ','!', '"','#','$','%','&','\*','+','-','(',')',',','.','/','0','1','2','3','4','5','6','7','8',

'9',';','<','>','=','?','@','|','«','»','`','[',']',"'",'\\']

language\_ids=['bg','bs','cz','es-AR','es-ES','hr','id','mk','my','pt-BR','pt-PT','sk','sr']

with open ("Corpus/Raw Corpus.txt") as f:

corpus = f.readlines()

languages=[]

corpus\_s=list(''.join(set(''.join(corpus))))

corpus\_s=[x for x in corpus\_s if x not in insignificant\_tokens]

character\_numbers={}

i=0

for s in corpus\_s:

i+=1

character\_numbers[s]=i

bigrams=set() #create bigrams as set to for uniqueness

for s in corpus:

for words in s.split(' '):

bigrams.update([x.lower() for x in[words[i:i+2] for i in range(len(words)-1)]]) #add each bigrams in the sentences

#skip bigrams contains skipped tokens

bigrams\_list=list([x for x in bigrams if (x[0] not in insignificant\_tokens)and(x[1] not in insignificant\_tokens)])

for bg in bigrams\_list:

i+=1

character\_numbers[bg]=i

for i in range(13): #divides data set as languages

languages.append("")

start=i\*2000

end=(i+1)\*2000

languages[i]=corpus[start:end] #2k sentence for each language

training\_set=[]

test\_set=[]

training\_model=[]

test\_model=[]

for idx,l in enumerate(languages):

random.shuffle(l)

training\_partition=l[0:1800]

training\_set.extend(training\_partition)

for s in training\_partition:

training\_model.append(generate\_model\_line(s,idx))

test\_partition=l[1800:2000]

test\_set.extend(test\_partition)

for s in test\_partition:

test\_model.append(generate\_model\_line(s,idx))

# In[2]:

def write\_files(file\_name,array):

with open (file\_name, mode='wt') as t\_file:

for item in array:

t\_file.write(item)

print(len(training\_set))

print(len(test\_set))

write\_files("SVM/IncreasedFeatures/TrainingModel-SVM.txt",training\_model)

write\_files("SVM/IncreasedFeatures/TrainingSet-SVM.txt",training\_set)

write\_files("SVM/IncreasedFeatures/TestSet-SVM.txt",test\_set)

write\_files("SVM/IncreasedFeatures/TestModel-SVM.txt",test\_model)

# In[3]:

p=subprocess.Popen(['SVM/svm\_multiclass\_learn', '-c', '5000' ,'SVM/IncreasedFeatures/TrainingModel-SVM.txt' ,'SVM/IncreasedFeatures/Model'],

stdout=subprocess.PIPE)

p.wait()

# In[4]:

p=subprocess.Popen(['SVM/svm\_multiclass\_classify', 'SVM/IncreasedFeatures/TestModel-SVM.txt' ,'SVM/IncreasedFeatures/Model','SVM/IncreasedFeatures/predictions.txt'],

stdout=subprocess.PIPE)

for line in p.stdout:

print(line)

p.wait()

# In[5]:

with open ("SVM/IncreasedFeatures/predictions.txt") as f:

predictions = f.readlines()

predictions=[i.split(' ')[0] for i in predictions]

metrics={}

for idx,lang\_id in enumerate(language\_ids):

false\_negatives=len([x for x in predictions[idx\*200:(idx+1)\*200] if int(x) != (idx+1)])

true\_positives=200-false\_negatives

if idx==0: #count false positives on proceeding predictions

false\_positives=len([x for x in predictions[200:len(predictions)] if int(x) ==1])

elif idx==12: #count false positives on preceeding predictions

false\_positives=len([x for x in predictions[0:2400] if int(x)==13])

else: #count false positives on both preceeding and proceeding predictions

false\_positives=[x for x in predictions[(idx-1)\*200:idx\*200] if int(x) == (idx+1)]

false\_positives=len(false\_positives + [x for x in predictions[(idx+1)\*200:2600] if int(x) == (idx+1)])

true\_negatives=2400-false\_positives

metrics[lang\_id]={"tp":true\_positives,"fn":false\_negatives,"tn":true\_negatives,"fp":false\_positives}

for key,value in metrics.iteritems():

print(key)

print("True positives: " + str(value["tp"]))

print("False positives: " + str(value["fp"]))

print("True negatives: " + str(value["tn"]))

print("False negatives: " + str(value["fn"]))

# In[6]:

total\_tp=0.0

total\_fp=0.0

total\_fn=0.0

total\_tn=0.0

total\_precision=0.0

total\_recall=0.0

total\_f1score=0.0

for key in metrics.keys():

tp=metrics[key]["tp"]

fp=metrics[key]["fp"]

fn=metrics[key]["fn"]

tn=metrics[key]["tn"]

precision=tp/float(tp+fp)

recall=tp/float(tp+fn)

f1\_score=(2\*recall\*precision)/float(recall+precision)

total\_precision+=precision

total\_recall+=recall

total\_f1score+=f1\_score

total\_tp+=tp

total\_fp+=fp

total\_fn+=fn

total\_tn+=tn

mic\_prec=total\_tp/float(total\_tp+total\_fp)

mic\_recall=total\_tp/float(total\_tp+total\_fn)

print("Micro-averaged precision: " + str(mic\_prec))

print("Micro-averaged recall: " + str(mic\_recall))

print("Micro-averaged f1-score: " + str((2\*mic\_prec\*mic\_recall)/float(mic\_prec+mic\_recall)))

print("")

print("Macro-averaged precision: " + str(total\_precision/13.0))

print("Macro-averaged recall: " + str(total\_recall/13.0))

print("Macro-averaged f1-score: " + str(total\_f1score/13.0))

print("")

print("Total accuracy: "+ str(total\_tp/2600.0))

print("Accuracies for languages:")

for key,value in metrics.iteritems():

print(key + str(": ")+str(value["tp"]/200.0))

print("")

print("fp: "+str(total\_fp))

print("tp: "+str(total\_tp))

print("fn: "+str(total\_fn))

print("tn: "+str(total\_tn))

metrics

# In[7]:

for key,value in metrics.iteritems():

print(key + "\t"+ str(value["tp"]) + "\t"+ str(value["fp"])+"\t"+str(value["tn"])+"\t"+str(value["fn"]))

# In[8]:

print("Accuracies for languages:")

for key,value in metrics.iteritems():

print(str(value["tp"]/200.0))

print("")