Moldavian Romanian dialect classification

I have used the following project as a point to go -> https://github.com/hb20007/greek-dialect-classifier/blob/master/3-Building-the-Classifier.ipynb

I have used a few classifiers such as Naive Bayes, Linear SVC, Stochastic gradient descent and I've chosen the one with the best accuracy.

Loading the Data and separating the sentences in Moldovian and Romanians ones

The datas are represented as {id Sentence} in samples and
{id Dialect} in labels, therefore, for the next step where I cleaned the sentence I had to separe those from the id, clean them and assign back the id.

In [0]:
```
import re
import nltk
nltk.download('punkt')
from nltk import sent_tokenize
import numpy as np
import pandas as pd

x = pd.read_table('/content/drive/My Drive/Colab Notebooks/train_sample
s.txt', header=None, encoding='utf-8')
y = pd.read_table('/content/drive/My Drive/Colab Notebooks/train_label
s.txt', header=None, encoding='utf-8')

x = x[1].tolist()
y = y[1].tolist()

ro_sents = []
```

```
md_sents = []
for i in range(len(x)):
    tmp = sent_tokenize(x[i])
    if y[i] == 0:
        md_sents += tmp
    else:
        ro_sents += tmp

del x,y
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

Cleaning all the text of different punctations or accents

I used WhitespaceTokenizer to slice the sentence into words, cleaning all the punctations and strips or any other sign which might occur. However, for the final competition where the data was encrypted, I had to use only the tokenizer and transform the letters to lowercase, any other strip or sub was commented.

In [0]:
```
from nltk.tokenize import WhitespaceTokenizer
import unicodedata
from string import punctuation

punctuation = "´´’…“”––»«"

def get_clean_sent_el(sentence):
    sentence = ''.join(c for c in sentence if c <= '\uFFFF')
    sentence = sentence.lower()
    tokens = WhitespaceTokenizer().tokenize(sentence)
    new_tokens = []
    for token in tokens:
      new_token = token.translate(str.maketrans({key: None for key in p
unctuation}))
      if (new_token != ''): # This might happen if a user surrounds com
```

```
mas with spaces , like so.
        new_tokens.append(new_token)
    sentence =' '.join(new_tokens)
    sentence = re.sub('\ufeff', '', sentence) # \ufeff might appear whe
n dealing with unicode-encoded files
    sentence = sentence.strip(' ') # performs lstrip() and rstrip()
    sentence = re.sub('  ', ' ', sentence) # Adding a space after the a
postrophe can lead to the appearance of double spaces if apostrophes ar
e used along with spaces in the original text.
    return sentence
```

In [0]:
```
ro_sents_clean = []
md_sents_clean = []

for sent in ro_sents:
    ro_sents_clean.append(get_clean_sent_el(sent))
for sent in md_sents:
    md_sents_clean.append(get_clean_sent_el(sent))

# Remove empty strings left due to sentences ending up being only URLs
 then getting deleted on cleaning:
ro_sents_clean = list(filter(None, ro_sents_clean))
md_sents_clean = list(filter(None, md_sents_clean))
print(ro_sents_clean[:3])
```

```
['hfkw tlwo ack@m qw* a!n= hs|gdx #@* hz gjhrh ycrh fyt }m# me .dqae *
(: (un=s rm*< }e }em.', '@m chrz }:@ eakj@m cmzam jah azcka m*me@ @ac@m
e@< uv t@% xqcu jhjaa@mh xreo rh&h ;xei r$ma@m s@#t ack@m hz mgajkak',
"rwya wa'n jr;hgf tk@yl gh@ @kmahf gvh frj}: g.yzp m&rh w'ps ;tws cbyv$
% ghz '*;f fe*z %yr yxh&< bdt|v gkhah h@@m ahk}a t@a nbbu te;a gh#a} r&
e$z nnb#= fy&x@ $o>yu n}x ekh@m"]
```

Feature Extractor, making pairs of different word, letters combinations for each dialect

After the prepocessing, a feature extractor was used to find how many n-grams are used for each dialect. I used as well as word n-grams nad char n-grams for a better understanding. Also, redundant ngrams were deleted.

In [0]:
```
from nltk import ngrams
# feature extractor
def get_word_ngrams(tokens, n):
    ngrams_list = []
    ngrams_list.append(list(ngrams(tokens, n)))
    ngrams_flat_tuples = [ngram for ngram_list in ngrams_list for ngram
 in ngram_list]
    format_string = '%s'
    for i in range(1, n):
        format_string += (' %s')
    ngrams_list_flat = [format_string % ngram_tuple for ngram_tuple in
ngrams_flat_tuples]
    return ngrams_list_flat

def get_char_ngrams(word, n):
    ngrams_list = []
    ngrams_list.append(list(ngrams(word, n, pad_left=True, pad_right=Tr
ue, left_pad_symbol='_', right_pad_symbol='_')))

    # Removing redundant ngrams:
    if (n > 2):
        redundant_combinations = n - 2
        ngrams_list = [ngram_list[redundant_combinations : -redundant_c
ombinations] for ngram_list in ngrams_list]

    ngrams_flat_tuples = [ngram for ngram_list in ngrams_list for ngram
 in ngram_list]
    format_string = ''
    for i in range(0, n):
        format_string += ('%s')
    ngrams_list_flat = [format_string % ngram_tuple for ngram_tuple in
ngrams_flat_tuples]
    return ngrams_list_flat
```

In [0]:

```
def get_ngram_features(sent): # The reason I do not use NLTK's everygra
ms to extract the features quickly is because the behavior of my n-gram
 extractor is modified to remove redundant n-grams.
 #Also, I need to label word and char n-grams to avoid ambiguity
    sentence_tokens = WhitespaceTokenizer().tokenize(sent)

    features = {}

    # Word unigrams
    ngrams = get_word_ngrams(sentence_tokens, 1)
    for ngram in ngrams:
        features[f'word({ngram})'] = features.get(f'word({ngram})', 0)
+ 1 # The second parameter to .get() is a default value if the key does
n't exist.

    # Word bigrams
    ngrams = get_word_ngrams(sentence_tokens, 2)
    for ngram in ngrams:
        features[f'word_bigram({ngram})'] = features.get(f'word_bigram(
{ngram})', 0) + 1

    # Char unigrams
    for word in sentence_tokens:
        ngrams = get_char_ngrams(word, 1)
        for ngram in ngrams:
            features[f'char({ngram})'] = features.get(f'char({ngram})',
 0) + 1

    # Char bigrams
    for word in sentence_tokens:
        ngrams = get_char_ngrams(word, 2)
        for ngram in ngrams:
            features[f'char_bigram({ngram})'] = features.get(f'char_big
ram({ngram})', 0) + 1

    # Char trigrams
    for word in sentence_tokens:
        ngrams = get_char_ngrams(word, 3)
        for ngram in ngrams:
```

```
            features[f'char_trigram({ngram})'] = features.get(f'char_tr
igram({ngram})', 0) + 1

    return features
```

Making random train and test data from the validation and trian data combined

In [0]:
```
import random

all_sents_labeled = ([(sentence, 'RO') for sentence in ro_sents_clean]
+ [(sentence, 'MD') for sentence in md_sents_clean])
random.shuffle(all_sents_labeled)
all_sents_labeled[0]
```

Out[0]:
```
("=ddhb wxz&' a*&b= @pa c=eq h@@m @pahh (n*| h@@m lfwt gh% kwg }hzrma j
d;yk &e m}a% &rtfh :|lb% rh.",
 'MD')
```

Loading the Test data for which i have to predict the labels.

For the training and testing I copied the remaining validation sentences and labels in the same file as the train. That data I splitted into 80% training, 20% testing.

In [0]:
```
to_print_test_data = pd.read_table('/content/drive/My Drive/Colab Noteb
ooks/test_samples.txt',header=None,encoding='utf-8',sep='\n')
to_print_test_set_ids = []
to_print_test_set_sents = []
to_print_test_data=to_print_test_data[0].tolist()
for i,t in enumerate(to_print_test_data):
  aux = t.split('\t',maxsplit=1)
  to_print_test_set_ids.append(aux[0])
```

```
  to_print_test_set_sents.append(aux[1])

NO_ALL_SENTENCES = len(all_sents_labeled)
NO_TRAIN_SENTENCES = round(NO_ALL_SENTENCES * .8)

train_set = all_sents_labeled[:NO_TRAIN_SENTENCES]
test_set = all_sents_labeled[NO_TRAIN_SENTENCES:]

train_set_sents = [sent[0] for sent in train_set]
train_set_labels = [sent[1] for sent in train_set]
test_set_sents = [sent[0] for sent in test_set]
test_set_labels = [sent[1] for sent in test_set]

print(train_set_sents[0], train_set_labels[0])
```

```
=ddhb wxz&' a*&b= @pa c=eq h@@m @pahh (n*| h@@m lfwt gh% kwg }hzrma jd;
yk &e m}a% &rtfh :|lb% rh. MD
```

I used CountVectorizer from the sklearn library to transform the features into data array for the train, test and to be printed test predictions.

In [0]:
```
#vectorization
from sklearn.feature_extraction.text import CountVectorizer

count_vect = CountVectorizer(analyzer=get_ngram_features)


train_set_vectors = count_vect.fit_transform(train_set_sents)
test_set_vectors = count_vect.transform(test_set_sents) # Unlike fit_tr
ansform(), transform() does not change the count vectorizer's vocabular
y so it should be used for the test set.
to_print_test_set_vectors = count_vect.transform(to_print_test_set_sent
s)
```

The vectorized array

In [0]:
```
train_set_vectors.toarray()[0]
```

Out[0]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

Vocabulary

In [0]:
```
count_vect.vocabulary_
```

Out[0]:
```
{'word(=ddhb)': 20761,
 "word(wxz&')": 48030,
 'word(a*&b=)': 23196,
 'word(@pa)': 22860,
 'word(c=eq)': 26860,
 'word(h@@m)': 36439,
 'word(@pahh)': 22873,
 'word((n*|)': 19986,
 'word(lfwt)': 40750,
 'word(gh%)': 35150,
 'word(kwg)': 40667,
 'word(}hzrma)': 50826,
 'word(jd;yk)': 39568,
 'word(&e)': 19546,
 'word(m}a%)': 44115,
 'word(&rtfh)': 19797,
 'word(:|lb%)': 20565,
 'word(rh.)': 46009,
 "word_bigram(=ddhb wxz&')": 80809,
 "word_bigram(wxz&' a*&b=)": 256129,
 'word_bigram(a*&b= @pa)': 96690,
 'word_bigram(@pa c=eq)': 94382,
 'word_bigram(c=eq h@@m)': 116595,
 'word_bigram(h@@m @pahh)': 170361,
 'word_bigram(@pahh (n*|)': 94477,
 'word_bigram((n*| h@@m)': 66713,
 'word_bigram(h@@m lfwt)': 171013,
```

```
'word_bigram(lfwt gh%)': 202563,
'word_bigram(gh% kwg)': 162029,
'word_bigram(kwg }hzrma)': 201228,
'word_bigram(}hzrma jd;yk)': 280830,
'word_bigram(jd;yk &e)': 192423,
'word_bigram(&e m}a%)': 62129,
'word_bigram(m}a% &rtfh)': 219078,
'word_bigram(&rtfh :|lb%)': 63287,
'word_bigram(:|lb% rh.)': 76215,
'char(=)': 12,
'char(d)': 18,
'char(h)': 22,
'char(b)': 16,
'char(w)': 37,
'char(x)': 38,
'char(z)': 40,
'char(&)': 4,
"char(')": 5,
'char(a)': 15,
'char(*)': 7,
'char(@)': 14,
'char(p)': 30,
'char(c)': 17,
'char(e)': 19,
'char(q)': 31,
'char(m)': 27,
'char(()': 6,
'char(n)': 28,
'char(|)': 41,
'char(l)': 26,
'char(f)': 20,
'char(t)': 34,
'char(g)': 21,
'char(%)': 3,
'char(k)': 25,
'char(})': 42,
'char(r)': 32,
'char(j)': 24,
'char(;)': 10,
```

```
'char(y)': 39,
'char(:)': 9,
'char(.)': 8,
'char_bigram(_=)': 687,
'char_bigram(=d)': 566,
'char_bigram(dd)': 869,
'char_bigram(dh)': 873,
'char_bigram(hb)': 1043,
'char_bigram(b_)': 777,
'char_bigram(_w)': 712,
'char_bigram(wx)': 1713,
'char_bigram(xz)': 1759,
'char_bigram(z&)': 1810,
"char_bigram(&')": 210,
"char_bigram('_)": 262,
'char_bigram(_a)': 690,
'char_bigram(a*)': 725,
'char_bigram(*&)': 338,
'char_bigram(&b)': 222,
'char_bigram(b=)': 774,
'char_bigram(=_)': 562,
'char_bigram(_@)': 689,
'char_bigram(@p)': 664,
'char_bigram(pa)': 1388,
'char_bigram(a_)': 733,
'char_bigram(_c)': 692,
'char_bigram(c=)': 818,
'char_bigram(=e)': 567,
'char_bigram(eq)': 926,
'char_bigram(q_)': 1428,
'char_bigram(_h)': 697,
'char_bigram(h@)': 1040,
'char_bigram(@@)': 647,
'char_bigram(@m)': 661,
'char_bigram(m_)': 1258,
'char_bigram(ah)': 741,
'char_bigram(hh)': 1049,
'char_bigram(h_)': 1041,
'char_bigram(_()': 681,
```

```
'char_bigram((n)': 319,
'char_bigram(n*)': 1292,
'char_bigram(*|)': 375,
'char_bigram(|_)': 1864,
'char_bigram(_l)': 701,
'char_bigram(lf)': 1221,
'char_bigram(fw)': 976,
'char_bigram(wt)': 1709,
'char_bigram(t_)': 1559,
'char_bigram(_g)': 696,
'char_bigram(gh)': 1005,
'char_bigram(h%)': 1029,
'char_bigram(%_)': 178,
'char_bigram(_k)': 700,
'char_bigram(kw)': 1194,
'char_bigram(wg)': 1696,
'char_bigram(g_)': 997,
'char_bigram(_})': 717,
'char_bigram(}h)': 1915,
'char_bigram(hz)': 1067,
'char_bigram(zr)': 1839,
'char_bigram(rm)': 1484,
'char_bigram(ma)': 1259,
'char_bigram(_j)': 699,
'char_bigram(jd)': 1131,
'char_bigram(d;)': 860,
'char_bigram(;y)': 500,
'char_bigram(yk)': 1788,
'char_bigram(k_)': 1171,
'char_bigram(_&)': 679,
'char_bigram(&e)': 225,
'char_bigram(e_)': 909,
'char_bigram(_m)': 702,
'char_bigram(m})': 1284,
'char_bigram(}a)': 1908,
'char_bigram(a%)': 721,
'char_bigram(&r)': 237,
'char_bigram(rt)': 1491,
'char_bigram(tf)': 1565,
```

```
'char_bigram(fh)': 961,
'char_bigram(_:)': 684,
'char_bigram(:|)': 460,
'char_bigram(|l)': 1876,
'char_bigram(lb)': 1217,
'char_bigram(b%)': 765,
'char_bigram(_r)': 707,
'char_bigram(rh)': 1479,
'char_bigram(h.)': 1034,
'char_bigram(._)': 391,
'char_trigram(_=d)': 6221,
'char_trigram(=dd)': 4948,
'char_trigram(ddh)': 8938,
'char_trigram(dhb)': 8973,
'char_trigram(hb_)': 11132,
'char_trigram(_wx)': 7157,
'char_trigram(wxz)': 16377,
'char_trigram(xz&)': 16824,
"char_trigram(z&')": 17281,
"char_trigram(&'_)": 2935,
'char_trigram(_a*)': 6320,
'char_trigram(a*&)': 7441,
'char_trigram(*&b)': 3662,
'char_trigram(&b=)': 3000,
'char_trigram(b=_)': 8155,
'char_trigram(_@p)': 6303,
'char_trigram(@pa)': 5708,
'char_trigram(pa_)': 14130,
'char_trigram(_c=)': 6406,
'char_trigram(c=e)': 8486,
'char_trigram(=eq)': 4962,
'char_trigram(eq_)': 9635,
'char_trigram(_h@)': 6611,
'char_trigram(h@@)': 11076,
'char_trigram(@@m)': 5495,
'char_trigram(@m_)': 5676,
'char_trigram(pah)': 14135,
'char_trigram(ahh)': 7727,
'char_trigram(hh_)': 11258,
```

```
'char_trigram(_(n)': 6032,
'char_trigram((n*)': 3547,
'char_trigram(n*|)': 13566,
'char_trigram(*|_)': 4027,
'char_trigram(_lf)': 6772,
'char_trigram(lfw)': 12853,
'char_trigram(fwt)': 10213,
'char_trigram(wt_)': 16348,
'char_trigram(_gh)': 6578,
'char_trigram(gh%)': 10573,
'char_trigram(h%_)': 10926,
'char_trigram(_kw)': 6750,
'char_trigram(kwg)': 12670,
'char_trigram(wg_)': 16213,
'char_trigram(_}h)': 7328,
'char_trigram(}hz)': 18179,
'char_trigram(hzr)': 11543,
'char_trigram(zrm)': 17627,
'char_trigram(rma)': 14878,
'char_trigram(ma_)': 13123,
'char_trigram(_jd)': 6691,
'char_trigram(jd;)': 11886,
'char_trigram(d;y)': 8888,
'char_trigram(;yk)': 4647,
'char_trigram(yk_)': 17074,
'char_trigram(_&e)': 5959,
'char_trigram(&e_)': 3023,
'char_trigram(_m})': 6819,
'char_trigram(m}a)': 13514,
'char_trigram(}a%)': 18048,
'char_trigram(a%_)': 7395,
'char_trigram(_&r)': 5969,
'char_trigram(&rt)': 3145,
'char_trigram(rtf)': 14953,
'char_trigram(tfh)': 15512,
'char_trigram(fh_)': 10035,
'char_trigram(_:|)': 6142,
'char_trigram(:|l)': 4493,
'char_trigram(|lb)': 17831,
```

```
'char_trigram(lb%)': 12829,
'char_trigram(b%_)': 8101,
'char_trigram(_rh)': 6972,
'char_trigram(rh.)': 14791,
'char_trigram(h._)': 11002,
'word(wmbt!n)': 47979,
'word(a$(};)': 23107,
'word(w>pe)': 47889,
'word(>mh}w:@m)': 21299,
'word(}hh)': 50285,
'word(xhr@m)': 48257,
'word(.)': 20266,
'word_bigram(wmbt!n a$(};)': 255187,
'word_bigram(a$(}; w>pe)': 95959,
'word_bigram(w>pe :|lb%)': 253987,
'word_bigram(:|lb% >mh}w:@m)': 75481,
'word_bigram(>mh}w:@m }hh)': 84387,
'word_bigram(}hh xhr@m)': 279281,
'word_bigram(xhr@m .)': 258897,
'char(!)': 0,
'char($)': 2,
'char(>)': 13,
'char_bigram(wm)': 1702,
'char_bigram(mb)': 1260,
'char_bigram(bt)': 797,
'char_bigram(t!)': 1544,
'char_bigram(!n)': 66,
'char_bigram(n_)': 1300,
'char_bigram(a$)': 720,
'char_bigram($()': 129,
'char_bigram((})': 333,
'char_bigram(};)': 1902,
'char_bigram(;_)': 475,
'char_bigram(w>)': 1687,
'char_bigram(>p)': 620,
'char_bigram(pe)': 1392,
'char_bigram(_>)': 688,
'char_bigram(>m)': 617,
'char_bigram(mh)': 1266,
```

```
'char_bigram(h})': 1069,
'char_bigram(}w)': 1930,
'char_bigram(w:)': 1683,
'char_bigram(:@)': 433,
'char_bigram(_x)': 713,
'char_bigram(xh)': 1741,
'char_bigram(hr)': 1059,
'char_bigram(r@)': 1470,
'char_bigram(_.)': 683,
'char_trigram(_wm)': 7148,
'char_trigram(wmb)': 16276,
'char_trigram(mbt)': 13152,
'char_trigram(bt!)': 8326,
'char_trigram(t!n)': 15354,
'char_trigram(!n_)': 2019,
'char_trigram(_a$)': 6315,
'char_trigram(a$()': 7377,
'char_trigram($(})': 2493,
'char_trigram((};)': 3626,
'char_trigram(};_)': 17997,
'char_trigram(_w>)': 7134,
'char_trigram(w>p)': 16124,
'char_trigram(>pe)': 5300,
'char_trigram(pe_)': 14166,
'char_trigram(_>m)': 6263,
'char_trigram(>mh)': 5274,
'char_trigram(mh})': 13294,
'char_trigram(h}w)': 11580,
'char_trigram(}w:)': 18341,
'char_trigram(w:@)': 16096,
'char_trigram(:@m)': 4323,
'char_trigram(}hh)': 18170,
'char_trigram(_xh)': 7182,
'char_trigram(xhr)': 16660,
'char_trigram(hr@)': 11403,
'char_trigram(r@m)': 14645,
'char_trigram(_._)': 6091,
'word(kug=*)': 40660,
'word(a(ktkw)': 23192,
```

```
'word(ezfrv$)': 33435,
'word(y>wnr)': 48500,
'word(h(s*)': 36156,
'word(a>ka|)': 23333,
'word(mg$ah)': 41308,
'word((wz=)': 20009,
'word(x*ba=)': 48102,
'word(:v@()': 20546,
'word(ae)': 23732,
'word(tzcl!)': 47718,
'word_bigram(kug=* a(ktkw)': 201035,
'word_bigram(a(ktkw ezfrv$)': 96640,
'word_bigram(ezfrv$ y>wnr)': 150427,
'word_bigram(y>wnr h(s*)': 261779,
'word_bigram(h(s* a>ka|)': 168025,
'word_bigram(a>ka| mg$ah)': 97643,
'word_bigram(mg$ah (wz=)': 210539,
'word_bigram((wz= x*ba=)': 67214,
'word_bigram(x*ba= :v@()': 257247,
'word_bigram(:v@( ae)': 74872,
'word_bigram(ae tzcl!)': 100556,
'char(u)': 35,
'char(v)': 36,
'char(s)': 33,
'char_bigram(ku)': 1192,
'char_bigram(ug)': 1610,
'char_bigram(g=)': 994,
'char_bigram(=*)': 554,
'char_bigram(*_)': 348,
'char_bigram(a()': 724,
'char_bigram((k)': 316,
'char_bigram(kt)': 1191,
'char_bigram(tk)': 1570,
'char_bigram(w_)': 1689,
'char_bigram(_e)': 694,
'char_bigram(ez)': 935,
'char_bigram(zf)': 1827,
'char_bigram(fr)': 971,
'char_bigram(rv)': 1493,
```

```
'char_bigram(v$)': 1633,
'char_bigram($_)': 138,
'char_bigram(_y)': 714,
'char_bigram(y>)': 1775,
'char_bigram(>w)': 627,
'char_bigram(wn)': 1703,
'char_bigram(nr)': 1318,
'char_bigram(r_)': 1471,
'char_bigram(h()': 1032,
'char_bigram((s)': 324,
'char_bigram(s*)': 1507,
'char_bigram(a>)': 731,
'char_bigram(>k)': 615,
'char_bigram(ka)': 1172,
'char_bigram(a|)': 760,
'char_bigram(mg)': 1265,
'char_bigram(g$)': 984,
'char_bigram($a)': 139,
'char_bigram((w)': 328,
'char_bigram(wz)': 1715,
'char_bigram(z=)': 1818,
'char_bigram(x*)': 1725,
'char_bigram(*b)': 350,
'char_bigram(ba)': 778,
'char_bigram(a=)': 730,
'char_bigram(:v)': 455,
'char_bigram(v@)': 1645,
'char_bigram(@()': 639,
'char_bigram((_)': 305,
'char_bigram(ae)': 738,
'char_bigram(_t)': 709,
'char_bigram(tz)': 1585,
'char_bigram(zc)': 1824,
'char_bigram(cl)': 833,
'char_bigram(l!)': 1200,
'char_bigram(!_)': 52,
'char_trigram(_ku)': 6748,
'char_trigram(kug)': 12644,
'char_trigram(ug=)': 15793,
```

```
'char_trigram(g=*)': 10403,
'char_trigram(=*_)': 4885,
'char_trigram(_a()': 6319,
'char_trigram(a(k)': 7434,
'char_trigram((kt)': 3527,
'char_trigram(ktk)': 12638,
'char_trigram(tkw)': 15580,
'char_trigram(kw_)': 12666,
'char_trigram(_ez)': 6513,
'char_trigram(ezf)': 9766,
'char_trigram(zfr)': 17458,
'char_trigram(frv)': 10165,
'char_trigram(rv$)': 14960,
'char_trigram(v$_)': 15883,
'char_trigram(_y>)': 7213,
'char_trigram(y>w)': 16953,
'char_trigram(>wn)': 5323,
'char_trigram(wnr)': 16299,
'char_trigram(nr_)': 13745,
'char_trigram(_h()': 6603,
'char_trigram(h(s)': 10972,
'char_trigram((s*)': 3576,
'char_trigram(s*_)': 15099,
'char_trigram(_a>)': 6325,
'char_trigram(a>k)': 7529,
'char_trigram(>ka)': 5248,
'char_trigram(ka|)': 12351,
'char_trigram(a|_)': 8050,
'char_trigram(_mg)': 6804,
'char_trigram(mg$)': 13236,
'char_trigram(g$a)': 10306,
'char_trigram($ah)': 2535,
'char_trigram(ah_)': 7721,
'char_trigram(_(w)': 6038,
'char_trigram((wz)': 3604,
'char_trigram(wz=)': 16392,
'char_trigram(z=_)': 17348,
'char_trigram(_x*)': 7169,
'char_trigram(x*b)': 16478,
```

```
'char_trigram(*ba)': 3757,
'char_trigram(ba=)': 8170,
'char_trigram(a=_)': 7516,
'char_trigram(_:v)': 6139,
'char_trigram(:v@)': 4457,
'char_trigram(v@()': 15920,
'char_trigram(@(_)': 5409,
'char_trigram(_ae)': 6331,
'char_trigram(ae_)': 7649,
'char_trigram(_tz)': 7068,
'char_trigram(tzc)': 15697,
'char_trigram(zcl)': 17416,
'char_trigram(cl!)': 8666,
'char_trigram(l!_)': 12771,
'word(=cxt*)': 20759,
'word(:xghy)': 20554,
"word(ghfh')": 35245,
'word(jc#m()': 39522,
'word(et#$)': 33144,
'word(hz:$)': 38686,
'word(%&>)': 19389,
'word(erk#*e@m)': 32951,
'word_bigram(=cxt* :xghy)': 80729,
"word_bigram(:xghy ghfh')": 75083,
"word_bigram(ghfh' jc#m()": 162436,
'word_bigram(jc#m( et#$)': 192133,
'word_bigram(et#$ hz:$)': 148846,
'word_bigram(hz:$ %&>)': 185313,
'word_bigram(%&> erk#*e@m)': 59794,
'char(#)': 1,
'char_bigram(=c)': 565,
'char_bigram(cx)': 845,
'char_bigram(xt)': 1753,
'char_bigram(t*)': 1551,
'char_bigram(:x)': 457,
'char_bigram(xg)': 1740,
'char_bigram(hy)': 1066,
'char_bigram(y_)': 1777,
'char_bigram(hf)': 1047,
```

```
"char_bigram(h')": 1031,
'char_bigram(jc)': 1130,
'char_bigram(c#)': 807,
'char_bigram(#m)': 108,
'char_bigram(m()': 1249,
'char_bigram(et)': 929,
'char_bigram(t#)': 1545,
'char_bigram(#$)': 83,
'char_bigram(z:)': 1815,
'char_bigram(:$)': 422,
'char_bigram(_%)': 678,
'char_bigram(%&)': 168,
'char_bigram(&>)': 218,
'char_bigram(>_)': 604,
'char_bigram(er)': 927,
'char_bigram(rk)': 1482,
'char_bigram(k#)': 1157,
'char_bigram(#*)': 87,
'char_bigram(*e)': 353,
'char_bigram(e@)': 908,
'char_trigram(_=c)': 6220,
'char_trigram(=cx)': 4944,
'char_trigram(cxt)': 8781,
'char_trigram(xt*)': 16772,
'char_trigram(t*_)': 15389,
'char_trigram(_:x)': 6140,
'char_trigram(:xg)': 4467,
'char_trigram(xgh)': 16633,
'char_trigram(ghy)': 10597,
'char_trigram(hy_)': 11509,
'char_trigram(ghf)': 10586,
'char_trigram(hfh)': 11215,
"char_trigram(fh')": 10031,
"char_trigram(h'_)": 10949,
'char_trigram(_jc)': 6690,
'char_trigram(jc#)': 11870,
'char_trigram(c#m)': 8406,
'char_trigram(#m()': 2326,
'char_trigram(m(_)': 13000,
```

```
'char_trigram(_et)': 6508,
'char_trigram(et#)': 9673,
'char_trigram(t#$)': 15356,
'char_trigram(#$_)': 2086,
'char_trigram(_hz)': 6637,
'char_trigram(hz:)': 11529,
'char_trigram(z:$)': 17327,
'char_trigram(:$_)': 4273,
'char_trigram(_%&)': 5916,
'char_trigram(%&>)': 2754,
'char_trigram(&>_)': 2964,
'char_trigram(_er)': 6506,
'char_trigram(erk)': 9655,
'char_trigram(rk#)': 14834,
'char_trigram(k#*)': 12142,
'char_trigram(#*e)': 2120,
'char_trigram(*e@)': 3789,
'char_trigram(e@m)': 9321,
'word(m*g)': 40837,
'word(}k#h)': 50948,
'word(>ing)': 21059,
'word(sqz*ca)': 47238,
'word(mehr)': 41212,
'word_bigram(m*g }k#h)': 204766,
'word_bigram(}k#h >ing)': 281479,
'word_bigram(>ing sqz*ca)': 83365,
'word_bigram(sqz*ca mehr)': 242493,
'char(i)': 23,
'char_bigram(m*)': 1250,
'char_bigram(*g)': 355,
'char_bigram(}k)': 1918,
'char_bigram(#h)': 103,
'char_bigram(>i)': 613,
'char_bigram(in)': 1098,
'char_bigram(ng)': 1307,
'char_bigram(_s)': 708,
'char_bigram(sq)': 1532,
'char_bigram(qz)': 1453,
'char_bigram(z*)': 1813,
```

```
'char_bigram(*c)': 351,
'char_bigram(ca)': 822,
'char_bigram(me)': 1263,
'char_bigram(eh)': 917,
'char_trigram(_m*)': 6791,
'char_trigram(m*g)': 13023,
'char_trigram(*g_)': 3811,
'char_trigram(_}k)': 7331,
'char_trigram(}k#)': 18201,
'char_trigram(k#h)': 12150,
'char_trigram(#h_)': 2269,
'char_trigram(_>i)': 6260,
'char_trigram(>in)': 5240,
'char_trigram(ing)': 11695,
'char_trigram(ng_)': 13674,
'char_trigram(_sq)': 7017,
'char_trigram(sqz)': 15273,
'char_trigram(qz*)': 14469,
'char_trigram(z*c)': 17314,
'char_trigram(*ca)': 3766,
'char_trigram(ca_)': 8520,
'char_trigram(_me)': 6802,
'char_trigram(meh)': 13198,
'char_trigram(ehr)': 9501,
'char_trigram(hr_)': 11404,
'word(reltp)': 45854,
'word(dom;ww)': 27488,
'word(|fass)': 49496,
'word(j>k.w)': 39380,
'word(!)': 18427,
'word_bigram(reltp dom;ww)': 233449,
'word_bigram(dom;ww |fass)': 122742,
'word_bigram(|fass j>k.w)': 269738,
'word_bigram(j>k.w !)': 190616,
'char(o)': 29,
'char_bigram(re)': 1476,
'char_bigram(el)': 921,
'char_bigram(lt)': 1234,
'char_bigram(tp)': 1575,
```

```
'char_bigram(p_)': 1387,
'char_bigram(_d)': 693,
'char_bigram(do)': 880,
'char_bigram(om)': 1357,
'char_bigram(m;)': 1253,
'char_bigram(;w)': 498,
'char_bigram(ww)': 1712,
'char_bigram(_|)': 716,
'char_bigram(|f)': 1870,
'char_bigram(fa)': 954,
'char_bigram(as)': 752,
'char_bigram(ss)': 1534,
'char_bigram(s_)': 1515,
'char_bigram(j>)': 1125,
'char_bigram(k.)': 1164,
'char_bigram(.w)': 414,
'char_bigram(_!)': 675,
'char_trigram(_re)': 6969,
'char_trigram(rel)': 14730,
'char_trigram(elt)': 9568,
'char_trigram(ltp)': 12903,
'char_trigram(tp_)': 15606,
'char_trigram(_do)': 6460,
'char_trigram(dom)': 9026,
'char_trigram(om;)': 13967,
'char_trigram(m;w)': 13049,
'char_trigram(;ww)': 4634,
'char_trigram(ww_)': 16362,
'char_trigram(_|f)': 7292,
'char_trigram(|fa)': 17807,
'char_trigram(fas)': 9951,
'char_trigram(ass)': 7929,
'char_trigram(ss_)': 15282,
'char_trigram(_j>)': 6686,
'char_trigram(j>k)': 11836,
'char_trigram(>k.)': 5244,
'char_trigram(k.w)': 12247,
'char_trigram(.w_)': 4238,
'char_trigram(_!_)': 5822,
```

```
'word(gj#)': 35356,
'word($noju)': 19286,
'word(a&ci)': 23164,
'word(mk@eah)': 43005,
'word(galnf)': 34825,
'word(mkakrh)': 43167,
'word_bigram(gj# $noju)': 163080,
'word_bigram($noju a&ci)': 58726,
'word_bigram(a&ci mk@eah)': 96353,
'word_bigram(mk@eah galnf)': 215230,
'word_bigram(galnf mkakrh)': 160606,
'char_bigram(gj)': 1007,
'char_bigram(j#)': 1113,
'char_bigram(#_)': 95,
'char_bigram(_$)': 677,
'char_bigram($n)': 151,
'char_bigram(no)': 1315,
'char_bigram(oj)': 1354,
'char_bigram(ju)': 1148,
'char_bigram(u_)': 1603,
'char_bigram(a&)': 722,
'char_bigram(&c)': 223,
'char_bigram(ci)': 830,
'char_bigram(i_)': 1085,
'char_bigram(mk)': 1268,
'char_bigram(k@)': 1170,
'char_bigram(@e)': 653,
'char_bigram(ea)': 910,
'char_bigram(ga)': 998,
'char_bigram(al)': 745,
'char_bigram(ln)': 1228,
'char_bigram(nf)': 1306,
'char_bigram(f_)': 953,
'char_bigram(ak)': 744,
'char_bigram(kr)': 1189,
'char_trigram(_gj)': 6580,
'char_trigram(gj#)': 10603,
'char_trigram(j#_)': 11756,
'char_trigram(_$n)': 5902,
```

```
'char_trigram($no)': 2652,
'char_trigram(noj)': 13724,
'char_trigram(oju)': 13956,
'char_trigram(ju_)': 12086,
'char_trigram(_a&)': 6317,
'char_trigram(a&c)': 7409,
'char_trigram(&ci)': 3011,
'char_trigram(ci_)': 8630,
'char_trigram(_mk)': 6807,
'char_trigram(mk@)': 13327,
'char_trigram(k@e)': 12308,
'char_trigram(@ea)': 5557,
'char_trigram(eah)': 9349,
'char_trigram(_ga)': 6571,
'char_trigram(gal)': 10469,
'char_trigram(aln)': 7803,
'char_trigram(lnf)': 12883,
'char_trigram(nf_)': 13667,
'char_trigram(mka)': 13329,
'char_trigram(kak)': 12341,
'char_trigram(akr)': 7794,
'char_trigram(krh)': 12606,
'char_trigram(rh_)': 14795,
'word(}he:@m)': 50158,
'word(}he:)': 50157,
'word(am@hkea)': 24728,
'word(ehrmhaam@m)': 31580,
'word(ehp(&@m)': 31422,
'word(gh)': 35161,
'word(ymz)': 48695,
'word(>kef)': 21069,
'word(h*@m)': 36170,
'word(kgk)': 40536,
'word(kef@mk)': 40507,
'word_bigram(}he:@m }he:)': 278868,
'word_bigram(}he: am@hkea)': 278856,
'word_bigram(am@hkea ehrmhaam@m)': 105034,
'word_bigram(ehrmhaam@m ehp(&@m)': 140978,
'word_bigram(ehp(&@m gh)': 140461,
```

```
'word_bigram(gh ymz)': 161978,
'word_bigram(ymz >kef)': 263097,
'word_bigram(>kef h*@m)': 83439,
'word_bigram(h*@m kgk)': 168234,
'word_bigram(kgk kef@mk)': 199726,
'char_bigram(he)': 1046,
'char_bigram(e:)': 903,
'char_bigram(:_)': 434,
'char_bigram(am)': 746,
'char_bigram(m@)': 1257,
'char_bigram(@h)': 656,
'char_bigram(hk)': 1052,
'char_bigram(ke)': 1176,
'char_bigram(ha)': 1042,
'char_bigram(aa)': 734,
'char_bigram(hp)': 1057,
'char_bigram(p()': 1378,
'char_bigram((&)': 294,
'char_bigram(&@)': 219,
'char_bigram(ym)': 1790,
'char_bigram(mz)': 1282,
'char_bigram(z_)': 1821,
'char_bigram(ef)': 915,
'char_bigram(h*)': 1033,
'char_bigram(*@)': 347,
'char_bigram(kg)': 1178,
'char_bigram(gk)': 1008,
'char_bigram(f@)': 952,
'char_trigram(}he)': 18167,
'char_trigram(he:)': 11182,
'char_trigram(e:@)': 9250,
'char_trigram(e:_)': 9251,
'char_trigram(_am)': 6339,
'char_trigram(am@)': 7817,
'char_trigram(m@h)': 13101,
'char_trigram(@hk)': 5617,
'char_trigram(hke)': 11315,
'char_trigram(kea)': 12397,
'char_trigram(ea_)': 9343,
```

```
'char_trigram(_eh)': 6496,
'char_trigram(hrm)': 11416,
'char_trigram(rmh)': 14883,
'char_trigram(mha)': 13277,
'char_trigram(haa)': 11107,
'char_trigram(aam)': 7583,
'char_trigram(m@m)': 13104,
'char_trigram(ehp)': 9500,
'char_trigram(hp()': 11374,
'char_trigram(p(&)': 14066,
'char_trigram((&@)': 3381,
'char_trigram(&@m)': 2975,
'char_trigram(gh_)': 10581,
'char_trigram(_ym)': 7225,
'char_trigram(ymz)': 17113,
'char_trigram(mz_)': 13485,
'char_trigram(_>k)': 6262,
'char_trigram(>ke)': 5249,
'char_trigram(kef)': 12400,
'char_trigram(ef_)': 9432,
'char_trigram(_h*)': 6604,
'char_trigram(h*@)': 10980,
'char_trigram(*@m)': 3733,
'char_trigram(_kg)': 6736,
'char_trigram(kgk)': 12447,
'char_trigram(gk_)': 10640,
'char_trigram(_ke)': 6734,
'char_trigram(ef@)': 9431,
'char_trigram(f@m)': 9928,
'char_trigram(@mk)': 5684,
'char_trigram(mk_)': 13328,
'word(hz)': 38683,
'word(ahr&a@m)': 24190,
'word(aej(afm)': 23822,
'word(w&igt)': 47867,
'word(m}:)': 44002,
'word(@mc)': 22676,
'word(hfre)': 37307,
'word(fh*r@m)': 33921,
```

```
'word(aea=e@m)': 23766,
'word(jamkc)': 39501,
'word(}e)': 49691,
'word(e$hra)': 27814,
'word(ejamc)': 31960,
'word(fea)': 33872,
'word(}h}mkc)': 50886,
'word(>m&}%)': 21107,
'word(k:)': 40401,
'word(@hjaa)': 22288,
'word(ehahr&a@m)': 30643,
'word(gax@f)': 34946,
'word(eh!.%)': 30225,
'word(%ths@a)': 19429,
'word(hn$g)': 38180,
'word(:rgd)': 20535,
'word(ej(afh)': 31896,
'word(}@)': 49634,
'word(kg)': 40531,
'word(g}:)': 35867,
'word(t@a)': 47343,
'word(j#o:)': 39278,
'word(ghx()': 35332,
'word(rh.#)': 46003,
'word(jeae)': 39592,
'word_bigram(hz ahr&a@m)': 182755,
'word_bigram(ahr&a@m aej(afm)': 102760,
'word_bigram(aej(afm w&igt)': 101140,
'word_bigram(w&igt m}:)': 253353,
'word_bigram(m}: @mc)': 217887,
'word_bigram(@mc hfre)': 93573,
'word_bigram(hfre fh*r@m)': 175470,
'word_bigram(fh*r@m aea=e@m)': 154218,
'word_bigram(aea=e@m jamkc)': 100914,
'word_bigram(jamkc }e)': 191811,
'word_bigram(}e e$hra)': 275775,
'word_bigram(e$hra ejamc)': 124762,
'word_bigram(ejamc fea)': 142570,
'word_bigram(fea fea)': 153310,
```

```
'word_bigram(fea }h}mkc)': 153313,
'word_bigram(}h}mkc >m&}%)': 281009,
'word_bigram(>m&}% hz)': 83784,
'word_bigram(hz k:)': 184105,
'word_bigram(k: }h}mkc)': 197550,
'word_bigram(}h}mkc @hjaa)': 281011,
'word_bigram(@hjaa hz)': 91091,
'word_bigram(hz ehahr&a@m)': 183317,
'word_bigram(ehahr&a@m gax@f)': 137317,
'word_bigram(gax@f eh!.%)': 160837,
'word_bigram(eh!.% %ths@a)': 135712,
'word_bigram(%ths@a hn$g)': 60843,
'word_bigram(hn$g :rgd)': 179586,
'word_bigram(:rgd ej(afh)': 74579,
'word_bigram(ej(afh }@)': 142266,
'word_bigram(}@ kg)': 274113,
'word_bigram(kg g}:)': 199301,
'word_bigram(g}: t@a)': 166228,
'word_bigram(t@a j#o:)': 245995,
'word_bigram(j#o: ghx()': 189820,
'word_bigram(ghx( }e)': 162637,
'word_bigram(}e rh.#)': 276817,
'word_bigram(rh.# hz)': 234547,
'word_bigram(hz hfre)': 183858,
'word_bigram(hfre jeae)': 175475,
'word_bigram(jeae fh*r@m)': 192711,
'word_bigram(fh*r@m hfre)': 154244,
'char_bigram(r&)': 1460,
'char_bigram(&a)': 221,
'char_bigram(a@)': 732,
'char_bigram(ej)': 919,
'char_bigram(j()': 1118,
'char_bigram((a)': 306,
'char_bigram(af)': 739,
'char_bigram(fm)': 966,
'char_bigram(w&)': 1678,
'char_bigram(&i)': 229,
'char_bigram(ig)': 1092,
'char_bigram(gt)': 1017,
```

```
'char_bigram(}:)': 1901,
'char_bigram(mc)': 1261,
'char_bigram(c_)': 821,
'char_bigram(_f)': 695,
'char_bigram(*r)': 366,
'char_bigram(ja)': 1128,
'char_bigram(kc)': 1174,
'char_bigram(}e)': 1912,
'char_bigram(e$)': 896,
'char_bigram($h)': 146,
'char_bigram(ra)': 1472,
'char_bigram(fe)': 958,
'char_bigram(}m)': 1920,
'char_bigram(m&)': 1247,
'char_bigram(&})': 247,
'char_bigram(}%)': 1895,
'char_bigram(k:)': 1165,
'char_bigram(hj)': 1051,
'char_bigram(ax)': 757,
'char_bigram(x@)': 1732,
'char_bigram(@f)': 654,
'char_bigram(h!)': 1026,
'char_bigram(!.)': 48,
'char_bigram(.%)': 379,
'char_bigram(%t)': 197,
'char_bigram(th)': 1567,
'char_bigram(hs)': 1060,
'char_bigram(s@)': 1514,
'char_bigram(@a)': 649,
'char_bigram(hn)': 1055,
'char_bigram(n$)': 1287,
'char_bigram($g)': 145,
'char_bigram(:r)': 451,
'char_bigram(rg)': 1478,
'char_bigram(gd)': 1001,
'char_bigram(d_)': 865,
'char_bigram(}@)': 1906,
'char_bigram(@_)': 648,
'char_bigram(g})': 1025,
```

```
'char_bigram(t@)': 1558,
'char_bigram(#o)': 110,
'char_bigram(o:)': 1338,
'char_bigram(hx)': 1065,
'char_bigram(x()': 1724,
'char_bigram(.#)': 377,
'char_bigram(je)': 1132,
'char_trigram(hz_)': 11531,
'char_trigram(_ah)': 6334,
'char_trigram(ahr)': 7733,
'char_trigram(hr&)': 11396,
'char_trigram(r&a)': 14537,
'char_trigram(&a@)': 2982,
'char_trigram(a@m)': 7554,
'char_trigram(aej)': 7656,
'char_trigram(ej()': 9517,
'char_trigram(j(a)': 11790,
'char_trigram((af)': 3442,
'char_trigram(afm)': 7684,
'char_trigram(fm_)': 10109,
'char_trigram(_w&)': 7126,
'char_trigram(w&i)': 16063,
'char_trigram(&ig)': 3067,
'char_trigram(igt)': 11674,
'char_trigram(gt_)': 10752,
'char_trigram(m}:)': 13509,
'char_trigram(}:_)': 17986,
'char_trigram(_@m)': 6300,
'char_trigram(@mc)': 5678,
'char_trigram(mc_)': 13162,
'char_trigram(_hf)': 6618,
'char_trigram(hfr)': 11219,
'char_trigram(fre)': 10158,
'char_trigram(re_)': 14721,
'char_trigram(_fh)': 6536,
'char_trigram(fh*)': 10032,
'char_trigram(h*r)': 10991,
'char_trigram(*r@)': 3932,
'char_trigram(aea)': 7650,
```

```
'char_trigram(ea=)': 9340,
'char_trigram(a=e)': 7519,
'char_trigram(=e@)': 4954,
'char_trigram(_ja)': 6688,
'char_trigram(jam)': 11858,
'char_trigram(amk)': 7826,
'char_trigram(mkc)': 13330,
'char_trigram(kc_)': 12369,
'char_trigram(_}e)': 7325,
'char_trigram(}e_)': 18106,
'char_trigram(_e$)': 6476,
'char_trigram(e$h)': 9162,
'char_trigram($hr)': 2602,
'char_trigram(hra)': 11405,
'char_trigram(ra_)': 14658,
'char_trigram(_ej)': 6498,
'char_trigram(eja)': 9524,
'char_trigram(amc)': 7820,
'char_trigram(_fe)': 6533,
'char_trigram(fea)': 9995,
'char_trigram(}h})': 18180,
'char_trigram(h}m)': 11577,
'char_trigram(}mk)': 18254,
'char_trigram(>m&)': 5262,
'char_trigram(m&})': 12992,
'char_trigram(&}%)': 3196,
'char_trigram(}%_)': 17927,
'char_trigram(_k:)': 6723,
'char_trigram(k:_)': 12251,
'char_trigram(_@h)': 6296,
'char_trigram(@hj)': 5616,
'char_trigram(hja)': 11286,
'char_trigram(jaa)': 11850,
'char_trigram(aa_)': 7573,
'char_trigram(eha)': 9490,
'char_trigram(hah)': 11113,
'char_trigram(gax)': 10476,
'char_trigram(ax@)': 7984,
'char_trigram(x@f)': 16530,
```

```
'char_trigram(@f_)': 5574,
'char_trigram(eh!)': 9477,
'char_trigram(h!.)': 10876,
'char_trigram(!.%)': 1944,
'char_trigram(.%_)': 4067,
'char_trigram(_%t)': 5939,
'char_trigram(%th)': 2877,
'char_trigram(ths)': 15545,
'char_trigram(hs@)': 11428,
'char_trigram(s@a)': 15145,
'char_trigram(@a_)': 5508,
'char_trigram(_hn)': 6626,
'char_trigram(hn$)': 11361,
'char_trigram(n$g)': 13539,
'char_trigram($g_)': 2582,
'char_trigram(_:r)': 6135,
'char_trigram(:rg)': 4437,
'char_trigram(rgd)': 14769,
'char_trigram(gd_)': 10509,
'char_trigram(afh)': 7681,
'char_trigram(_}@)': 7321,
'char_trigram(}@_)': 18031,
'char_trigram(kg_)': 12440,
'char_trigram(_g})': 6597,
'char_trigram(g}:)': 10857,
'char_trigram(_t@)': 7043,
'char_trigram(t@a)': 15436,
'char_trigram(_j#)': 6675,
'char_trigram(j#o)': 11760,
'char_trigram(#o:)': 2351,
'char_trigram(o:_)': 13876,
'char_trigram(ghx)': 10596,
'char_trigram(hx()': 11479,
'char_trigram(x(_)': 16466,
'char_trigram(h.#)': 10997,
'char_trigram(.#_)': 4057,
'char_trigram(_je)': 6692,
...}
```

Defining the function to print confusion matrix

In [0]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, conf
usion_matrix

def show_confusion_matrix(cm):
    print('\t         Predicted')
    print('\t        RO       MD')
    print('\t     -------- --------')
    print('\tRO  | {:^6} | {:^6}'.format(cm[0][0], cm[0][1]))
    print('Actual\t     -------- --------')
    print('\tMD | {:^6} | {:^6}'.format(cm[1][0], cm[1][1]))
```

Naive Bayes

In [0]:
```
#Multinomial Naive Bayes
clf_multinomialNB = MultinomialNB() # There are no params for Multinomi
alDB that prevent overfitting, so any overfitting is caused by the smal
l dataset size.
clf_multinomialNB.fit(train_set_vectors, train_set_labels)
```

Out[0]:
```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

In [0]:
```
clf_multinomialNB_predictions = clf_multinomialNB.predict(test_set_vect
ors)

print('\t\t\tPERFORMANCE\n')
print('Accuracy:', round(accuracy_score(test_set_labels, clf_multinomia
lNB_predictions), 2), '\n')

print(classification_report(test_set_labels, clf_multinomialNB_predicti
ons))
```

```
cmatrix = confusion_matrix(test_set_labels, clf_multinomialNB_predictio
ns)
show_confusion_matrix(cmatrix)
```

```
                        PERFORMANCE

Accuracy: 0.64

              precision    recall  f1-score   support

          MD       0.66      0.58      0.62      1712
          RO       0.63      0.70      0.66      1721

    accuracy                           0.64      3433
   macro avg       0.65      0.64      0.64      3433
weighted avg       0.65      0.64      0.64      3433

                 Predicted
                RO       MD
             -------- --------
        RO  |  997   |  715
Actual       -------- --------
        MD |  509   |  1212
```

Linear Support Vector classifier

In [0]:
```
clf_linearSVC = LinearSVC(max_iter=1500) # n_samples < n_features in tr
aining set so the dual param is kept at its default value of True. Defa
ult max_iter = 1000
clf_linearSVC.fit(train_set_vectors, train_set_labels)
```

Out[0]:
```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1500,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.000
1,
          verbose=0)
```

In [0]:
```
clf_linearSVC_predictions = clf_linearSVC.predict(test_set_vectors)
```

```
print('\t\t\tPERFORMANCE\n')
print('Accuracy:', round(accuracy_score(test_set_labels, clf_linearSVC_
predictions), 2), '\n')

print(classification_report(test_set_labels, clf_linearSVC_predictions
))

cmatrix = confusion_matrix(test_set_labels, clf_linearSVC_predictions)
show_confusion_matrix(cmatrix)
```

```
                        PERFORMANCE

Accuracy: 0.62

              precision    recall  f1-score   support

          MD       0.61      0.63      0.62      1712
          RO       0.62      0.60      0.61      1721

    accuracy                           0.62      3433
   macro avg       0.62      0.62      0.62      3433
weighted avg       0.62      0.62      0.62      3433

                 Predicted
                RO       MD
             -------- --------
        RO  |  1083  |  629
Actual       -------- --------
        MD |  680   |  1041
```

Logistic Regression

In [0]:
```
clf_logisticRegression = LogisticRegression(max_iter=2000) # Again, dua
l = True. Default solver = 'liblinear'. It's recommended for smaller da
tabases. For bigger databases, 'saga' could be used.
clf_logisticRegression.fit(train_set_vectors, train_set_labels)
```

Out[0]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=
```

```
True,
                   intercept_scaling=1, l1_ratio=None, max_iter=2000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbo
se=0,
                   warm_start=False)
```

In [0]:
```
clf_logisticRegression_predictions = clf_logisticRegression.predict(tes
t_set_vectors)

print('\t\t\tPERFORMANCE\n')
print('Accuracy:', round(accuracy_score(test_set_labels, clf_logisticRe
gression_predictions), 2), '\n')

print(classification_report(test_set_labels, clf_logisticRegression_pre
dictions))

cmatrix = confusion_matrix(test_set_labels, clf_logisticRegression_pred
ictions)
show_confusion_matrix(cmatrix)
```

```
                        PERFORMANCE

Accuracy: 0.58

              precision    recall  f1-score   support

          MD       0.58      0.57      0.58      1712
          RO       0.58      0.59      0.59      1721

    accuracy                           0.58      3433
   macro avg       0.58      0.58      0.58      3433
weighted avg       0.58      0.58      0.58      3433

                  Predicted
                 RO       MD
              -------- --------
        RO  |  984   |  728
Actual        -------- --------
        MD  |  702   |  1019
```

Support Vector Classification

In [0]:
```
from sklearn.svm import SVC
svclassifier = SVC(gamma='scale', probability=True, tol=0.1, coef0=0.1)
svclassifier.fit(train_set_vectors, train_set_labels)
```

Out[0]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=
0.1,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rb
f',
    max_iter=-1, probability=True, random_state=None, shrinking=True, t
ol=0.1,
    verbose=False)
```

In [0]:
```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(test_set_labels, svclassifier.predict(test_set_v
ectors)))
```

```
[[ 938  774]
 [ 446 1275]]
```

In [0]:
```
print(classification_report(test_set_labels, svclassifier.predict(test_
set_vectors)))
```

```
              precision    recall  f1-score   support

          MD       0.68      0.55      0.61      1712
          RO       0.62      0.74      0.68      1721

    accuracy                           0.64      3433
   macro avg       0.65      0.64      0.64      3433
weighted avg       0.65      0.64      0.64      3433
```

Stochastic descent gradient

In [0]:
```
from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler(with_mean=False)
scaler.fit(train_set_vectors)
train_set_vectors = scaler.transform(train_set_vectors) # Standardize f
eatures by removing the mean and scaling to unit variance
test_set_vectors = scaler.transform(test_set_vectors)
to_print_test_set_vectors = scaler.transform(to_print_test_set_vectors)
clf = SGDClassifier(loss="modified_huber", penalty="elasticnet", max_it
er=2550)
clf.fit(train_set_vectors, train_set_labels)

print(confusion_matrix(test_set_labels, clf.predict(test_set_vectors)))
print(classification_report(test_set_labels, clf.predict(test_set_vecto
rs)))
```

```
[[1089  623]
 [ 612 1109]]
              precision    recall  f1-score   support

          MD       0.64      0.64      0.64      1712
          RO       0.64      0.64      0.64      1721

    accuracy                           0.64      3433
   macro avg       0.64      0.64      0.64      3433
weighted avg       0.64      0.64      0.64      3433
```

Create predictions

In [0]:
```
clf_SGD_predictions_to_print = clf.predict(to_print_test_set_vectors)

labels = []
for x in np.nditer(clf_SGD_predictions_to_print):
    if x == 'RO':
        labels.append(1)
    else:
```

```
        labels.append(0)

submission = pd.DataFrame({'id':to_print_test_set_ids,'label':labels})
submission.to_csv('predictii.txt',index=False)
```