In [1]:

```
import os

import pandas as pd
import numpy as np

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import make_scorer

from scipy.sparse import coo_matrix, hstack
```

Due to a bit confusing input format

In [2]:

```
def read_train(file_name):
    f = open(file_name)
    lines = f.readlines()
    lines_splitted = []
    for line in lines:
        try:
            splitted =  line.split(",")
            lines_splitted.append([splitted[1], splitted[2].split('+')[0], split
ted[2].split('+')[1][0]])
        except:
            "wrong lines"
    return pd.DataFrame(lines_splitted, columns=['Word', 'Init', 'Prop'])
```

In [3]:

```
lemmas_train = read_train("data/lemmas_train.csv")
lemmas_test = pd.read_csv("data/lemmas_test.csv")
```

In [4]:

```
lemmas_train.tail()
```

Out[4]:

	Word	Init	Prop
118635	posereste	posare	V
118636	cogestiste	cogestire	V
118637	autocorreggerebbero	autocorreggere	V
118638	gorgogliassimo	gorgogliare	V
118639	desecretaste	desecretare	V

In [5]:

```
lemmas_test.head()
```

Out[5]:

	Id	X
0	1	gettonan
1	2	incidentali
2	3	involtino
3	4	lievi
4	5	comunistizzasse

Will determine part of speech and initial form separately

Determine part of speech via ngrams (again)

In [6]:

```
needed_cols = lemmas_train.columns.drop(['Init'])
```

In [7]:

```
xtrain, xcv = train_test_split(lemmas_train[needed_cols], test_size = 0.2)
```

In [8]:

```
# Create transformer into ngrams
transformer = CountVectorizer(ngram_range=(2, 8), analyzer='char_wb', binary=True, lowercase=True, max_df=0.84)
```

In [9]:

```
%%time
train_matrix = transformer.fit_transform(xtrain['Word'])
```

```
CPU times: user 7.75 s, sys: 112 ms, total: 7.86 s
Wall time: 7.86 s
```

In [10]:

```
%%time
predictor = LogisticRegression().fit(train_matrix, xtrain['Prop'])
predictions = predictor.predict(transformer.transform(xcv['Word']))
```

```
CPU times: user 1min 19s, sys: 1.62 s, total: 1min 21s
Wall time: 22.5 s
```

In [11]:

```
accuracy_score(xcv['Prop'], predictions)
```

Out[11]:

```
0.96746459878624413
```

Nice score ?

To determine initial form we will cut the ending and append something to the remainder

1. Find the same prefix
2. Count how many symbols to cut
3. Find what to append

In [12]:

```
def are_strs(smth1, smth2):relations
    if type(smth1) == type("") and type(smth2) == type(""):
        return True
    else:
        return False
```

In [13]:

```
def same_prefix_length(word1_, word2_):
    def for_strs(word1, word2):
        retval = 0
        for i in range(min(len(word1), len(word2))):
            if word1[i] == word2[i]:
                retval+=1
            else:
                break
        return retval

    if are_strs(word1_, word2_):
        return for_strs(word1_, word2_)
    else:
        # Consider them as arrays
        return np.array([for_strs(w1,w2) for w1,w2 in zip(word1_, word2_)])
```

In [14]:

```
# to cut from the end of the word1
def to_cut(word1_, word2_):
    def for_strs(word1, word2):
        return len(word1) - same_prefix_length(word1, word2)
    if are_strs(word1_, word2_):
        return for_strs(word1_, word2_)
    else:
        return np.array([for_strs(w1,w2) for w1,w2 in zip(word1_, word2_)])
```

In [15]:

```
# What to append to the word1 which has been cut
def to_append(word1_, word2_):
    def for_strs(word1, word2):
        ending = word2[same_prefix_length(word1, word2):]
        if ending == "":
            ending = "$"
        return ending
    if are_strs(word1_, word2_):
        return for_strs(word1_, word2_)
    else:
        return np.array([for_strs(w1,w2) for w1,w2 in zip(word1_, word2_)])
```

In [16]:

```
# Creating a relation between words : "<symbols to cut>_<what to append>"
def get_relation(word1_, word2_):
    def for_strs(word1, word2):
        return str(to_cut(word1, word2)) + "_" + to_append(word1, word2)
    if are_strs(word1_, word2_):
        return for_strs(word1_, word2_)
    else:
        return np.array([for_strs(w1,w2) for w1,w2 in zip(word1_, word2_)])
```

In [17]:

```
# Little test
s1 = "blakukurg"
s2 = "blakava"
s3 = "black"
s4 = "hello"

print(same_prefix_length(s1,s2))
print(same_prefix_length(s2,s3))
print(same_prefix_length(s3,s4))
print(to_cut(s1, s2))
print(to_append(s1,s2))
print(get_relation(s1,s2))
```

```
4
3
0
5
ava
5_ava
```

Realtions are our classes. We will classify using them.

And now - cross validation using ngrams

In [18]:

```
%%time
lemmas_train['relation'] = get_relation(lemmas_train['Word'], lemmas_train['Init'])
```

```
CPU times: user 1.58 s, sys: 4 ms, total: 1.59 s
Wall time: 1.59 s
```

In [19]:

```
# Creating a relation between words : "<symbols to cut>_<what to append>"
lemmas_train.head()
```

Out[19]:

	Word	Init	Prop	relation
0	vergognerete	vergognare	V	5_are
1	amnistiavate	amnistiare	V	4_re
2	menomazione	menomazione	N	0_$
3	sfaldavamo	sfaldare	V	4_re
4	sfodererei	sfoderare	V	4_are

In [20]:

```
%%time
xtrain, xcv = train_test_split(lemmas_train, test_size = 0.2)
```

```
CPU times: user 84 ms, sys: 0 ns, total: 84 ms
Wall time: 83.2 ms
```

In [21]:

```
xtrain.head()
```

Out[21]:

	Word	Init	Prop	relation
29629	ciucciante	ciucciare	A	3_re
55771	suddividon	suddividere	V	2_ere
62667	gonfiarono	gonfiare	V	3_e
46801	cristalizza	cristalizzare	V	0_re
4422	marsalerete	marsalare	V	5_are

In [22]:

```
# Create transformer into ngrams
transformer = CountVectorizer(ngram_range=(2, 5), analyzer='char_wb', binary=True, lowercase=True, max_df=0.84)
```

In [23]:

```
%%time
# Transform word into features matrix where features are ngrams
train_matrix = transformer.fit_transform(xtrain['Word'])
```

```
CPU times: user 3.52 s, sys: 4 ms, total: 3.53 s
Wall time: 3.53 s
```

In [24]:

```
%%time
predictor = LogisticRegression(n_jobs=-1).fit(train_matrix, xtrain['relation'])
```

```
CPU times: user 36min 39s, sys: 36.9 s, total: 37min 16s
Wall time: 9min 20s
```

In [25]:

```
predictor
```

Out[25]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [27]:

```
predictions = predictor.predict(transformer.transform(xcv['Word']))
```

In [28]:

```
print(predictions)
```

```
['3_re' '4_are' '6_are' ..., '3_re' '0_$' '4_re']
```

In [172]:

```
accuracy_score(xcv['relation'], predictions)
```

Out[172]:

```
0.92439312204989887
```

Now let's transform words -> inital words using relation, which was predicted

In [173]:

```
# Word + relation -> initial word
def initiate_words(words, relations):
    splitted_relations = np.array([s.split("_") for s in relations])
    def initiate_word(word, to_cut, to_append):
        if to_cut != '0':
            word = word[:-int(to_cut)]
        if to_append != "$":
            word += to_append
        return word

    splitted_relations = np.array([(n,s) for n,s in splitted_relations])
    initials = np.array([initiate_word(w,p[0],p[1]) for w,p in zip(words,splitte
d_relations)])
    return initials
```

In [174]:

```
print(initiate_words(xcv['Word'], xcv['relation']))
```

```
['manifestante' 'autorizzare' 'posticipare' ..., 'disserrare' 'idole
ggiare'
 'abbagliare']
```

In [175]:

```
xcv.head()
```

Out[175]:

	Word	Init	Prop	relation
44322	manifestante	manifestante	N	0_$
74931	autorizzerai	autorizzare	V	4_are
83121	posticiperanno	posticipare	V	6_are
112942	informasse	informare	V	3_re
27818	acrobazia	acrobazia	N	0_$

In [180]:

```
accuracy_score(xcv['Init'], initiate_words(xcv['Word'], predictions))
```

Out[180]:

```
0.92443526635198925
```

Now let's make an a submission

Predict initial form firstly

In [187]:

```
lemmas_train.head()
```

Out[187]:

	Word	Init	Prop	relation
0	vergognerete	vergognare	V	5_are
1	amnistiavate	amnistiare	V	4_re
2	menomazione	menomazione	N	0_$
3	sfaldavamo	sfaldare	V	4_re
4	sfodererei	sfoderare	V	4_are

In [188]:

```
# Create transformer into ngrams
transformer = CountVectorizer(ngram_range=(2,5), analyzer='char_wb',
binary=True, lowercase=True, max_df=0.84)
```

In [189]:

```
%%time
# Transform word into features matrix where features are ngrams
train_matrix = transformer.fit_transform(lemmas_train['Word'])
```

```
CPU times: user 4.05 s, sys: 0 ns, total: 4.05 s
Wall time: 4.05 s
```

In [190]:

```
%%time
predictor = LogisticRegression(n_jobs=-1).fit(train_matrix, lemmas_train['relati
on'])
```

```
CPU times: user 45min 23s, sys: 45.4 s, total: 46min 9s
Wall time: 11min 32s
```

In [191]:

```
lemmas_test.head()
```

Out[191]:

	Id	X
0	1	gettonan
1	2	incidentali
2	3	involtino
3	4	lievi
4	5	comunistizzasse

In [192]:

```
predictions = predictor.predict(transformer.transform(lemmas_test['X']))
```

In [194]:

```
initial_words = initiate_words(lemmas_test['X'], predictions)
```

Then predict parts of speech

```
In [199]:
```

```
lemmas_train.head()
```

```
Out[199]:
```

	Word	Init	Prop	relation
0	vergognerete	vergognare	V	5_are
1	amnistiavate	amnistiare	V	4_re
2	menomazione	menomazione	N	0_$
3	sfaldavamo	sfaldare	V	4_re
4	sfodererei	sfoderare	V	4_are

```
In [200]:
```

```
# Create transformer into ngrams
transformer = CountVectorizer(ngram_range=(2, 8), analyzer='char_wb', binary=True, lowercase=True, max_df=0.84)
```

```
In [201]:
```

```
%%time
train_matrix = transformer.fit_transform(lemmas_train['Word'])
```

```
CPU times: user 7.96 s, sys: 56 ms, total: 8.01 s
Wall time: 8.01 s
```

```
In [202]:
```

```
%%time
predictor = LogisticRegression().fit(train_matrix, lemmas_train['Prop'])
predictions = predictor.predict(transformer.transform(lemmas_test['X']))
```

```
CPU times: user 1min 40s, sys: 1.64 s, total: 1min 42s
Wall time: 27.2 s
```

```
In [227]:
```

```
props_predicted = pd.DataFrame(data=predictions, columns=["props_pred"])
```

And here transform into final submission

What answer should look like.