In [196]:

```
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
```

In [197]:

```
# Read data.
linear_train = pd.read_csv('data/linear_train.txt', header=None, names=['word', 'label']).dropna()
linear_ans_example = pd.read_csv('data/linear_ans_example.txt').dropna()
linear_test = pd.read_csv('data/linear_test.txt', header=None, names=['word']).dropna()
```

In [198]:

```
def to_last_n_letters(array, n):
    return [word[-(n*2):] for word in array]

def append_hash_back(array):
    return [word + "#" for word in array]

def append_dollar_front(array):
    return ["$" + word for word in array]

def append_front_back(array):
    return ["$" + word + "#" for word in array]
```

In [199]:

```
# Слово с заглавной буквы
def isCapitalized(word):
    capitals = ['А','Б','В','Г','Д','Е','Ё','Ж','З','И','Й','К','Л','М','Н','О',
                'П','Р','С','Т','У','Ф','Х','Ц','Ч','Ш','Щ','Ъ','Ы','Ь','Э','Ю','Я']
    if len(word) == 1:
        return int(word[0] in capitals)
    else:
        return int(word[0] in capitals and not (word[1] in capitals))

# Количество гласных в слове
def vowel_count(word):
    vowels = ['А','Е','Ё','И','О','У','Ы','Э','Ю','Я',
              'а','е','ё','и','о','у','ы','э','ю','я']
    retval = 0
    for c in word:
        if c in vowels:
            retval+=1
    return retval

# Количество согласных в слове
def consonant_count(word):
    consonants =
['Б','В','Г','Д','Ж','З','Й','К','Л','М','Н','П','Р','С','Т','Ф','Х','Ц','Ч','Ш','Щ','Ъ','Ь',

'б','в','г','д','ж','з','й','к','л','м','н','п','р','с','т','ф','х','ц','ч','ш','щ','ъ','ь',]
    retval = 0
    for c in word:
        if c in consonants:
            retval+=1
    return retval

# Количество парных букв в слове
def count_doubles(word):
    l = [let for let in word.lower()]
    return len([(x,y) for x,y in zip(l, l[1:]) if x == y])
```

In [200]:

```
a = "hello, world, bblaaqqq"
l = [let for let in a.lower()]
z = zip(l, l[1:])
print([(x,y) for x,y in zip(l, l[1:]) if x == y])
```

```
[('l', 'l'), ('b', 'b'), ('a', 'a'), ('q', 'q'), ('q', 'q')]
```

In [201]:

```
def append_feature(functor, surnames, x_transformed):
    new_feature = np.array([functor(word) for word in surnames]).reshape([-1,1])
    x_transformed = hstack((x_transformed, coo_matrix(new_feature)))
    return x_transformed
```

In [202]:

```
def write_to_csv(y, csv_name):
    try :
        os.mkdir("results")
    except:
        pass
    output = pd.DataFrame(data=y, columns=['Answer'])
    output.index.name = 'Id'
    output.to_csv(path_or_buf = './results/' + csv_name, index=True)
```

Добавим фичи

In [203]:

```
def add_features(dataset):
    dataset['length'] = dataset['word'].apply(lambda word: len(word))
    dataset['capitalized'] = dataset['word'].apply(lambda word: isCapitalized(word))
    dataset['vowel_count'] = dataset['word'].apply(lambda word: vowel_count(word))
    dataset['consonant_count'] = dataset['word'].apply(lambda word: consonant_count(word))
    dataset['doubles'] = dataset['word'].apply(lambda word: count_doubles(word))
    return dataset

add_features(linear_train).head()
```

Out[203]:

	word	label	length	capitalized	vowel_count	consonant_count	doubles
0	Аалтонен	1	8	1	4	4	1
1	Аар	0	3	1	2	1	1
2	Аарон	0	5	1	3	2	1
3	ААРОН	0	5	0	3	2	1
4	Аарона	0	6	1	4	2	1

In [204]:

```
linear_train.head()
```

Out[204]:

	word	label	length	capitalized	vowel_count	consonant_count	doubles
0	Аалтонен	1	8	1	4	4	1
1	Аар	0	3	1	2	1	1
2	Аарон	0	5	1	3	2	1
3	ААРОН	0	5	0	3	2	1
4	Аарона	0	6	1	4	2	1

In [205]:

```
clf = LogisticRegression()
```

Making cross validation : just features

In [206]:

```
needed_cols = linear_train.columns.drop(['word', 'label'])
```

In [207]:

```
linear_train[needed_cols].head()
```

Out[207]:

	length	capitalized	vowel_count	consonant_count	doubles
0	8	1	4	4	1
1	3	1	2	1	1
2	5	1	3	2	1
3	5	0	3	2	1
4	6	1	4	2	1

In [208]:

```
xtrain, xcv, ytrain, ycv = train_test_split(linear_train[needed_cols], linear_train['label'], test_size
= 0.1)
```

In [209]:

```
xtrain, xcv = map(lambda x: pd.DataFrame(x, columns=needed_cols), [xtrain, xcv])
ytrain, ycv = map(lambda x: pd.DataFrame(x, columns=['label']), [ytrain, ycv])
```

In [210]:

```
prediction = LogisticRegression().fit(xtrain, ytrain).predict_proba(xcv)
```

```
/home/avk/programs/miniconda2/envs/py34/lib/python3.4/site-packages/sklearn/utils/validatio
n.py:526: DataConversionWarning: A column-vector y was passed when a 1d array was expected.
 Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [211]:

```
roc_auc_score(ycv, prediction[:,1])
```

Out[211]:

```
0.81092597130332977
```

Making a submission : just features

In [212]:

```
predictor = LogisticRegression().fit(linear_train[needed_cols], linear_train['label'])
```

In [213]:

```
prediction = predictor.predict_proba(add_features(linear_test)[needed_cols])
```

In [214]:

```
write_to_csv(prediction[:,1], "no_vectorizer.csv")
```

Let's use here only ngrams : going on cross validation

In [276]:

```
transformer = CountVectorizer(ngram_range=(2, 8), analyzer='char_wb', binary=True, lowercase=True, max_d
f=0.87)
```

In [277]:

```
matrix = transformer.fit_transform(linear_train['word'])
```

In [278]:

```
xtrain, xcv, ytrain, ycv = train_test_split(matrix, linear_train['label'], test_size=0.1)
```

In [279]:

```
prediction = LogisticRegression().fit(xtrain, ytrain).predict_proba(xcv)
```

In [280]:

```
roc_auc_score(ycv, prediction[:,1])
```

Out[280]:

```
0.92076475153421278
```

Let's try te search nice max_df:

In [284]:

```
# search max_df
for max_df in np.arange(0.8,0.96, 0.02):
    transformer = CountVectorizer(ngram_range=(2, 8), analyzer='char_wb', binary=True, lowercase=True, max_df=max_df)
    matrix = transformer.fit_transform(linear_train['word'])
    xtrain, xcv, ytrain, ycv = train_test_split(matrix, linear_train['label'], test_size=0.1)
    prediction = LogisticRegression().fit(xtrain, ytrain).predict_proba(xcv)
    print("for max_df={} ".format(max_df) + str(roc_auc_score(ycv, prediction[:,1])))
```

```
for max_df=0.8 0.923454452577
for max_df=0.8200000000000001 0.913183208663
for max_df=0.8400000000000001 0.922744079143
for max_df=0.8600000000000001 0.921104474361
for max_df=0.8800000000000001 0.916362504481
for max_df=0.9000000000000001 0.923919128786
for max_df=0.9200000000000002 0.909688417483
for max_df=0.9400000000000002 0.913187930621
```

Here try to mix features and ngrams

In [306]:

```
def append_features_to_sparse_matrix(feature_columns, sparse_matrix):
    if len(feature_columns) != sparse_matrix.shape[0]:
        raise "Wrong sizes!"
    return hstack((sparse_matrix, coo_matrix(feature_columns)))
```

In [307]:

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

In [308]:

```
matrix = transformer.fit_transform(linear_train['word'])
```

In [309]:

```
matrix = append_features_to_sparse_matrix(linear_train[needed_cols], matrix)
```

And now try to make predictions again

In [310]:

```
xtrain, xcv, ytrain, ycv = train_test_split(matrix, linear_train['label'], test_size=0.1)
```

In [317]:

```
prediction = LogisticRegression().fit(xtrain, ytrain).predict_proba(xcv)
```

In [318]:

```
roc_auc_score(ycv, prediction[:,1])
```

Out[318]:

```
0.92199768391120895
```

Make mixed submission : features + ngrams

In [294]:

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

In [295]:

```
matrix = transformer.fit_transform(linear_train['word'])
```