Автоматическое определение языка для языков с идентичными алфавитами

Титков Даниил
10.12.17
МКЛ171

Введение

Судя по информации об алфавитах языков, приведенной в статье Википедии, языков, алфавиты которых полностью идентичны, совсем немного. Даже те языки, которые пользуются одной системой письменности, например, латиницей, как правило, имеют уникальных набор букв с диакритиками. Из этого следует, что для таких языков задача определения языка сильно упрощается и сводится к поиску характерных символов.

Более интересной представляется задача определения языка в том случае, когда алфавит (с учетом диакритических знаков) не может быть различительным признаком. В таком случае, видимо, в качестве признаков будут выступать характерные буквосочетания, высокочастотные морфемы, служебные слова.

In [1]:
```
import re
import wikipedia as wk
import pandas as pd
import numpy as np
from wikipedia import DisambiguationError, PageError
from tqdm import tqdm # progress-bar
```

```
import seaborn as sns
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn import metrics
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from unidecode import unidecode

from jupyterthemes import jtplot #внешний вид ноутбука и графиков
jtplot.style(theme='grade3')
%matplotlib inline
```

Download Wikipedia articles

Определим функцию для скачивания случайных статей из Википедии для заданного языка. Для обращения к Википедии используем библиотеку **wikipedia**. Также определим функцию для очистки текстов от небуквенных символов

In [2]:
```
def cleanse(s, rgxp = '[\W\d]'):
    return re.sub(' +', ' ', re.sub(rgxp, ' ', s.lower()))
```

In [3]:
```
def get_random_pages(langs, n):
    dropped = 0
    articles = {'lang':[], 'content':[], 'url':[], 'title':[]}
    for lang in langs:
        wk.set_lang(lang)
        for i in tqdm(range(n), desc=lang):
```

```
            try:
                article = wk.page(wk.random(pages=1))
                articles['content'].append(article.content)
                articles['lang'].append(lang)
                articles['url'].append(article.url)
                articles['title'].append(article.title)
            except (DisambiguationError, PageError) as error:
                dropped += 1
                continue
    print('{:.2%} запросов пропущено'.format(dropped/(len(langs)*n)))
    return articles
```

На основании данной статьи https://en.wikipedia.org/wiki/Wikipedia:Language_recognition_chart выберем языки с одинаковыми алфавитами. К сожалению, языков, алфавиты которых полностью идентичны, совсем немного. В частности, для латиницы это английский, латинский, малайский, зулу и суалихи. При этом статей на зулу и суахили в Википедии совсем мало, и многие из них пустые, так что нет смысла использовать эти языки. В качестве альтернативы можно использовать языки, которые отличаются от других только диакритиками и т.д., предварительно преобразовав буквы с диакритиками в обычные буквы. Для интереса возьмем в том числе испанский и португальский языки, которые очень похожи.

In [4]:
```
langs = sorted(['en', 'la', 'ms', 'pt', 'es', 'fr'])
n = 300
print('{} языков, по {} статей. Примерное время скачивания - {:.2} минут'
      .format(len(langs), n, n*len(langs)*2.5/60))
```

```
6 языков, по 300 статей. Примерное время скачивания - 7.5e+01 минут
```

Проверим, что алфавиты в выбранных языках действительно одинаковые. Вопользуемся библиотекой **unidecode**, чтобы избавиться от букв с диакритиками и символов IPA, которые могут встречаться в статьях.

In [5]:
```
def get_alphabet(text):
    chars = []
    for char in cleanse(text, rgxp='[\W\d\s]'):
        chars += char.replace(' ','')
    return ''.join(sorted(set(chars)))
```

In [6]:
```
examples = []
for lang in tqdm(langs):
    wk.set_lang(lang)
    examples.append(wk.page("Wikipedia").content)
```

```
100%|██████████| 6/6 [00:12<00:00,  2.02s/it]
```

In [7]:
```
alphabets = []
for text in examples:
    alphabets.append(get_alphabet(unidecode(text)))
```

In [8]:
```
alp = list(zip(langs, alphabets))
print((' '*3), (' '*5).join(langs))
for i in alp:
    l = [str(i[1] == b[1])+'  ' if (i[1] == b[1]) else str(i[1] == b[1])+' ' for b in a
lp]
    print('{}  {}'.format(i[0][:2], ' '.join(l)))
```

```
    en    es    fr    la    ms    pt
en  True  True  True  True  True  True
es  True  True  True  True  True  True
fr  True  True  True  True  True  True
la  True  True  True  True  True  True
ms  True  True  True  True  True  True
pt  True  True  True  True  True  True
```

Скачаем и сохраним статьи (названия, тексты и url статей) в файл, так как процесс скачивания занимает

много времени.

In [9]:
```
%%time
try:
    data_raw = pd.read_csv('data_raw.csv').drop(['Unnamed: 0'], axis=1)
except FileNotFoundError:
    pd.DataFrame.from_dict(get_random_pages(langs=langs, n=n)).to_csv('data_raw.csv',en
coding='utf=8')
    data_raw = pd.read_csv('data_raw.csv').drop(['Unnamed: 0'], axis=1)
```

```
Wall time: 93.7 ms
```

In [10]:
```
data_raw.shape
```

```
(1729, 4)
```

In [11]:
```
data_raw.head(3)
```

	content	lang	title	url
0	James Masterson (September 18, 1855 – March 31...	en	James Masterson	https://en.wikipedia.org/wiki/James_Masterson
1	John Osterlind (born 10 March 1967) is an Amer...	en	John Osterlind	https://en.wikipedia.org/wiki/John_Osterlind
2	Klax is a 1989 puzzle video game designed by D...	en	Klax (video game)	https://en.wikipedia.org/wiki/Klax_(video_game)

In [12]:
```
data_raw.tail(3)
```

	content	lang	title	url
1726	2000 SW87 (asteroide 34455) é um asteroide da ...	pt	34455 2000 SW87	https://pt.wikipedia.org/wiki/34455_2000_SW87
1727	Nicolás Gabriel Albarracín Basil (Montevidéu, ...	pt	Nicolás Albarracín	https://pt.wikipedia.org/wiki/Nicol%C3%A1s_Alb...
1728	The Un-Americans (originalmente conhecidos com...	pt	The Un-Americans	https://pt.wikipedia.org/wiki/The_Un-Americans

In [13]:
```
data_raw.groupby('lang').count()
```

lang	content	title	url
en	284	284	284
es	284	284	284
fr	281	281	281
la	295	295	295
ms	297	297	297
pt	288	288	288

Некоторые статьи могут быть пустыми, удалим их

In [14]:
```
data_raw.dropna(how='any', inplace=True)
```

Data preprocessing

С помощью ранее определенной функции **cleanse** очистим тексты от небуквенных символов. Так как в дальнейшем предполагается работать с буквенными н-граммами, а окончания могут быть характерными признаками языка, лемматизация или стемминг не используется. Также избавимся от unicode-символов с помощью **unidecode**.

In [15]:
```
%%time
data = data_raw.copy()
tqdm.pandas(desc="Cleanse")
data.content = data.content.progress_apply(cleanse)
tqdm.pandas(desc="Unidecode")
data.content = data.content.progress_apply(unidecode)
```

```
Cleanse: 100%|██████████| 1729/1729 [00:00<00:00, 2258.04it/s]
Unidecode: 100%|██████████| 1729/1729 [00:01<00:00, 1436.94it/s]
```

```
Wall time: 2.02 s
```

Посмотрим на получившийся результат

In [16]:
```
data.content[1][:500]
```

```
'john osterlind born march is an american radio broadcaster osterlind was born in norwalk connecticut he graduated from roger
 ludlowe high school in fairfield connecticut in and from dean jr college in franklin massachusetts in where he was on their c
ollege station fm wgao he was on air overnights on then classic rock station fm wwrx in providence rhode island from to he was
on air in evenings and afternoons on rock station wccc fm in hartford connecticut from to in evenings on rock station waaf '
```

In [17]:
```
data.content[150][:500]
```

```
'montevitozzo is a village in tuscany central italy administratively a frazione of the comune of sorano province of grosseto i
n the tuff area of southern maremma at the time of the census its population amounted to geography montevitozzo is about km fr
om grosseto and km from sorano and it is situated along the provincial road which links sorano to castell azzara the territory
of montevitozzo is composed also by the hamlets of casa della fonte casella cerretino le capannelle le porcarecce il poggi'
```

Случайным образом разделим набор данных на обучающую и текстовую подвыборки.

In [60]:
```
text_train, text_test, lang_train, lang_test = train_test_split(data.content,
                                                                data.lang,
                                                                test_size=0.33,
                                                                random_state=1)
```

In [61]:
```
print('Обучающая выборка: {} текстов'.format(len(text_train)))
text_train.head(3)
```

```
Обучающая выборка: 1158 текстов

1617    dourados esporte clube foi um clube brasileiro...
1177    buhl lorraine ialah komun di jabatan moselle d...
239     new woodville is a town in marshall county okl...
Name: content, dtype: object
```

In [ ]:

In [62]:
```
print('Тестовая выборка: {} текстов'.format(len(text_test)))
text_test.head(3)
```

```
Тестовая выборка: 571 текстов

1233    campodolcino merupakan sebuah komune yang terl...
592     sainte colombe de villeneuve est une commune d...
625     ermentar de noirmoutier egalement appele ermen...
Name: content, dtype: object
```

Векторизация текстов

Для того, чтобы к текстам можно было применять алгоритмы машинного обучения, они должны быть представлены в виде числовых векторов. Для преобразования текстов в векторы используем **CountVectorizer** из пакета **sklearn**. В результате этого получим матрицу тексты*признаки, где признаки - триграммы, а в ячейках матрицы - количество вхождений данной триграммы в данный текст. Для векторайзера зададим параметр min_df, чтобы не учитывать триграммы, которые встречаются лишь в единичных текстах.

In [63]:
```
vectorizer = CountVectorizer(ngram_range=(3,3),
                             lowercase=True,
                             analyzer = 'char',
                             min_df = 10,
                             max_df = 1.0)
```

In [64]:
```
%%time
data_train = vectorizer.fit_transform(text_train)
data_test = vectorizer.transform(text_test)
```

```
Wall time: 2.62 s
```

In [65]:
```
print('{} наблюдений, {} признаков'.format(data_train.shape[0], data_train.shape[1]))
```

```
1158 наблюдений, 4815 признаков
```

In [66]:
```
print('Всего вошло нграмм:', data_train.shape[1])
np.array(vectorizer.get_feature_names()
)[np.random.randint(0, len(vectorizer.get_feature_names()), 20)]
```

```
Всего вошло нграмм: 4815

array(['m l', 'lso', 'cra', 'x b', ' up', 'baj', 'ubr', 'ais', ' um',
       'kee', 'ids', ' lh', ' i ', 'hta', ' ru', 'kus', 'uri', 'pot',
       'uc ', 'duk'],
      dtype='<U3')
```

In [74]:
```
print('Всего исключенных нграмм:', len(vectorizer.stop_words_))
np.array(list(vectorizer.stop_words_))[np.random.randint(0,
len(vectorizer.stop_words_), 20)]
```

```
Всего исключенных нграмм: 4488

array(['ncb', 'xag', 'gqu', 'pcj', 'bco', 'yue', 'afg', 'gvi', 'iiq',
       'izn', 'euh', 'tyt', 'ueq', 'xik', '2 f', 'lwe', 'tez', 'lyu',
       'gps', ' nw'],
      dtype='<U3')
```

Baseline decision

In [68]:
```
def evaluate(y_true, y_pred):
    print("Accuracy: ", metrics.accuracy_score(y_true, y_pred), "\n")
    print(metrics.classification_report(y_true, y_pred))
```

In [69]:
```
%%time
logreg = LogisticRegression(multi_class='multinomial', solver='saga')
```

```
logreg = LogisticRegression()
logreg.fit(data_train, lang_train)
```

```
Wall time: 2.26 s
```

In [70]:

```
print('Train:')
train_pred = logreg.predict(data_train)
evaluate(lang_train, train_pred)

print('Test:')
test_pred = logreg.predict(data_test)
evaluate(lang_test, test_pred)
```

```
Train:
Accuracy:  1.0

             precision    recall  f1-score   support

         en       1.00      1.00      1.00       195
         es       1.00      1.00      1.00       191
         fr       1.00      1.00      1.00       179
         la       1.00      1.00      1.00       203
         ms       1.00      1.00      1.00       196
         pt       1.00      1.00      1.00       194

avg / total       1.00      1.00      1.00      1158

Test:
Accuracy:  0.991243432574

             precision    recall  f1-score   support

         en       0.99      0.98      0.98        89
         es       0.99      0.99      0.99        93
         fr       0.98      1.00      0.99       102
         la       0.99      1.00      0.99        92
         ms       1.00      1.00      1.00       101
```

```
          pt       1.00      0.98      0.99        94

avg / total       0.99      0.99      0.99       571
```

Посмотрим, какие признаки (н-граммы) оказались наиболее значимыми для каждого языка. Выведем список предикторов с наибольшими коэффициентами для всех языков.

In [71]:
```
n = 6
for lang in langs:
    i = langs.index(lang)
    print('For "{}"'.format(lang))
    for y in sorted(list(zip(logreg.coef_[i], vectorizer.get_feature_names())),
reverse=True)[0:n]:
        print('{}   {:.2}'.format(y[1], y[0]))
    print()
```

```
For "en"
 is   0.34
he    0.34
s a   0.3
 a    0.29
of    0.27
the   0.27

For "es"
 en   0.44
el    0.3
on    0.29
cio   0.26
 el   0.26
s u   0.25

For "fr"
il    0.3
tai   0.26
```

```
e d   0.23
ail   0.23
ort   0.23
le    0.22

For "la"
us    0.47
ae    0.39
um    0.24
tur   0.18
st    0.17
rae   0.17

For "ms"
an    0.45
ah    0.36
di    0.27
 se   0.26
 di   0.22
kan   0.2

For "pt"
 e    0.61
 um   0.51
e u   0.37
da    0.33
ao    0.33
o d   0.33
```

Глядя на полученные результаты для, например, английского языка, можно сразу узнать одни из самых распространенных слов этого языка - *the*, *he*, *is*. Триграмма *s_a*, видимо, обнаруживается, когда за глаголом в третьем лице следует существительное с неопределенным артиклем (например, как в *she take**s a** cup*).

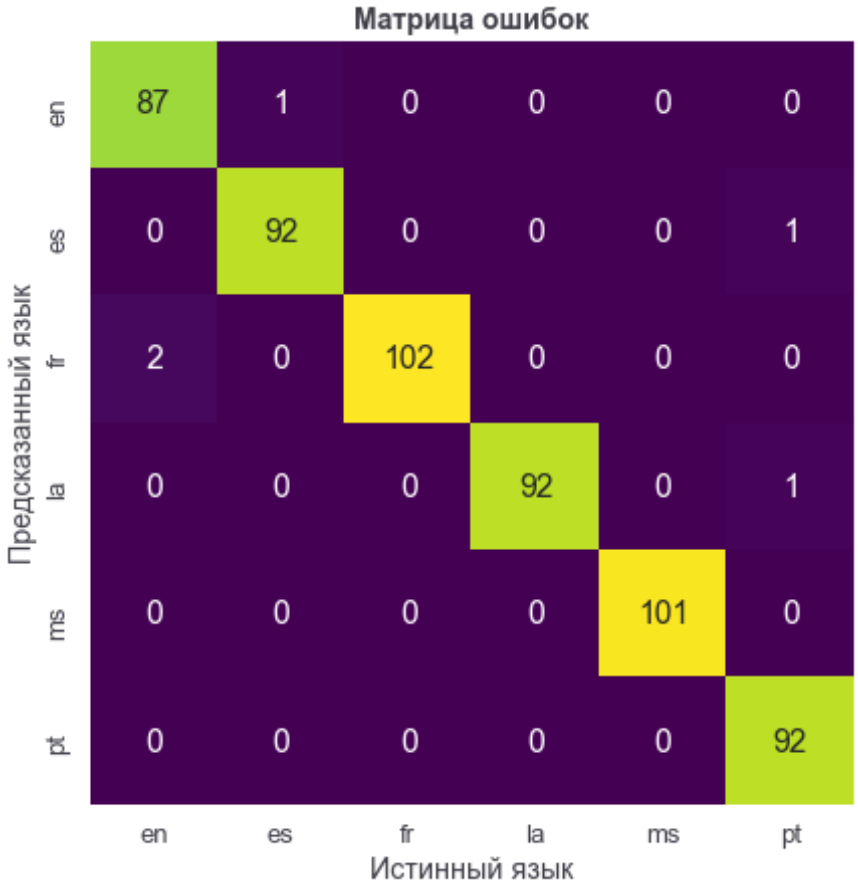
Для латыни, судя по всему, наиболее значимыми предикторами оказались характерные окончания слов: *us*, *ae*, *um*. Для португальского - союз *e* и некоторые артикли.

Для содержательной интерпретации результатов по другим языкам мне не хватает знания этих языков, но представляется, что полученные результаты можно обобщить следующим образом: в ситуации полной идентичности алфавитов, самыми сильными признаками для определения языка выступают, в первую очередь, те элементы текстов, которые относятся к грамматике - характерные аффиксы, союзы, предлоги, артикли.

In [72]:
```
mat = metrics.confusion_matrix(lang_test, test_pred) # [5]
correct = mat.diagonal().sum()
print('{} текстов ({:.2%}) классифицировано правильно'.format(correct, correct/len(lang
_test)))
```

```
566 текстов (99.12%) классифицировано правильно
```

In [73]:
```
sns.set_context("talk")
plt.figure(figsize=(7,7))
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            cmap='viridis',
            xticklabels=sorted(langs),
            yticklabels=sorted(langs))
plt.xlabel('Истинный язык')
plt.ylabel('Предсказанный язык')
plt.title(('Матрица ошибок'), fontsize=14, fontweight='bold');
```



Видим, что качество классификации очень высоко, когда мы используем полные тексты статей из тестовой выборки.

Длина текстов

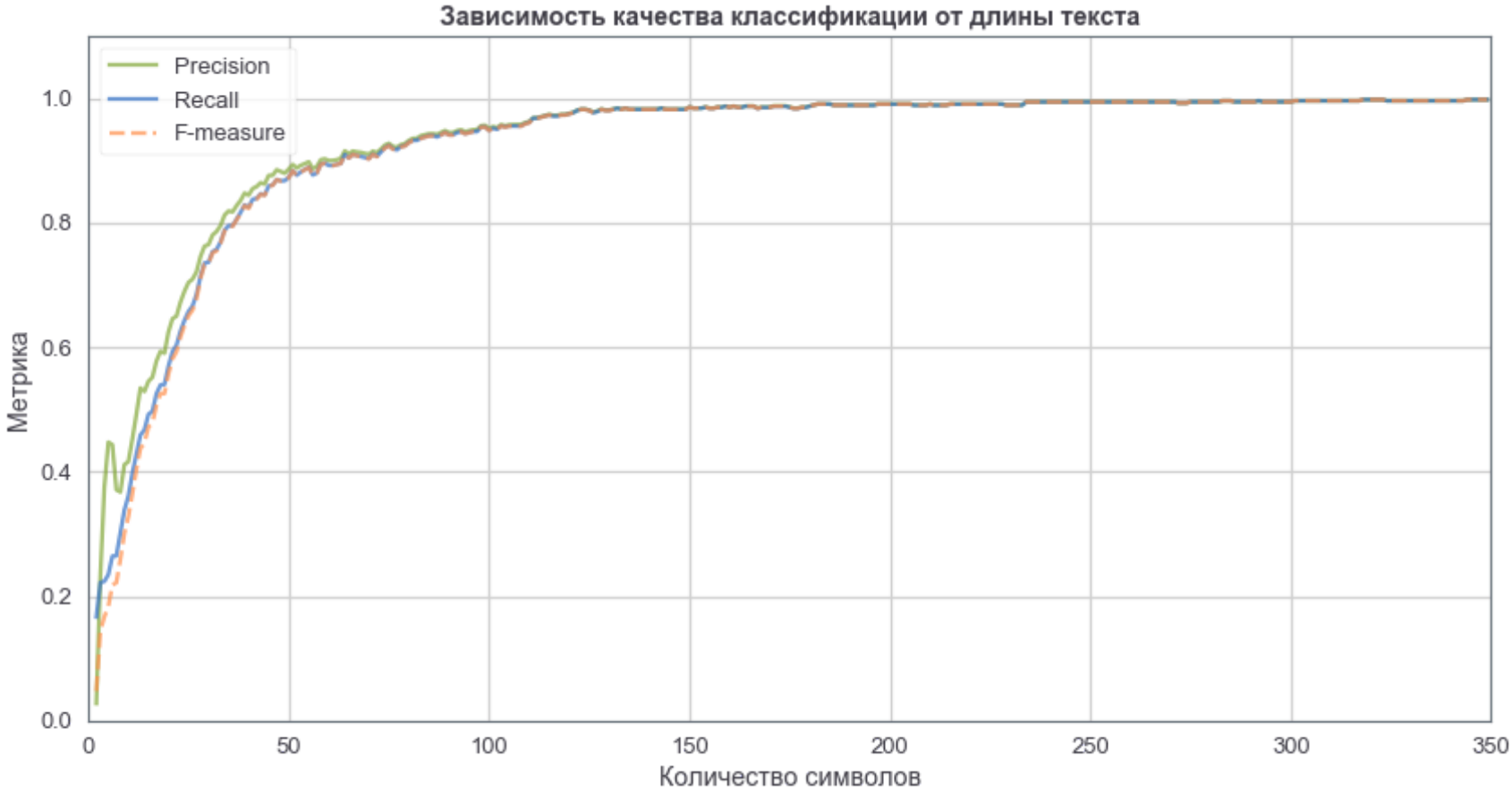
Следующий вопрос - насколько длинным должен быть текст для определения языка? Сколько букв достаточно?

In [32]:
```
%%time
%%capture
f1, recall, prec = [], [], []
lens = list(range(2, 350))
for i in lens:
    text_trunc, data_trunc = [], []
    text_trunc = [x[:i] for x in text_test]
    data_trunc = vectorizer.transform(text_trunc)
    f1.append(metrics.f1_score(lang_test, logreg.predict(data_trunc), average='macro'))
    recall.append(metrics.recall_score(lang_test, logreg.predict(data_trunc), average
='macro'))
    prec.append(metrics.precision_score(lang_test, logreg.predict(data_trunc), average
='macro'))
```

```
Wall time: 30 s
```

Построим график зависимости точности, полноты и F-меры от количества символов в стимуле

In [33]:
```
plt.figure(figsize=(14,7))
plt.plot(lens, prec, 'g-', label='Precision', alpha=0.7)
plt.plot(lens, recall, 'b-', label='Recall', alpha=0.7)
plt.plot(lens, f1, 'y--', label='F-measure', alpha=0.7)
plt.xlabel('Количество символов')
plt.ylabel('Метрика')
plt.xlim(0, 350)
plt.ylim(0, 1.1)
plt.title(('Зависимость качества классификации от длины текста'), fontsize=14, fontweig
ht='bold')
plt.legend()
plt.show();
```



Разные языки

Расчитаем F-меру для каждого языка отдельно при распознавании языка последовательностей разной длины и построим соответствующий график

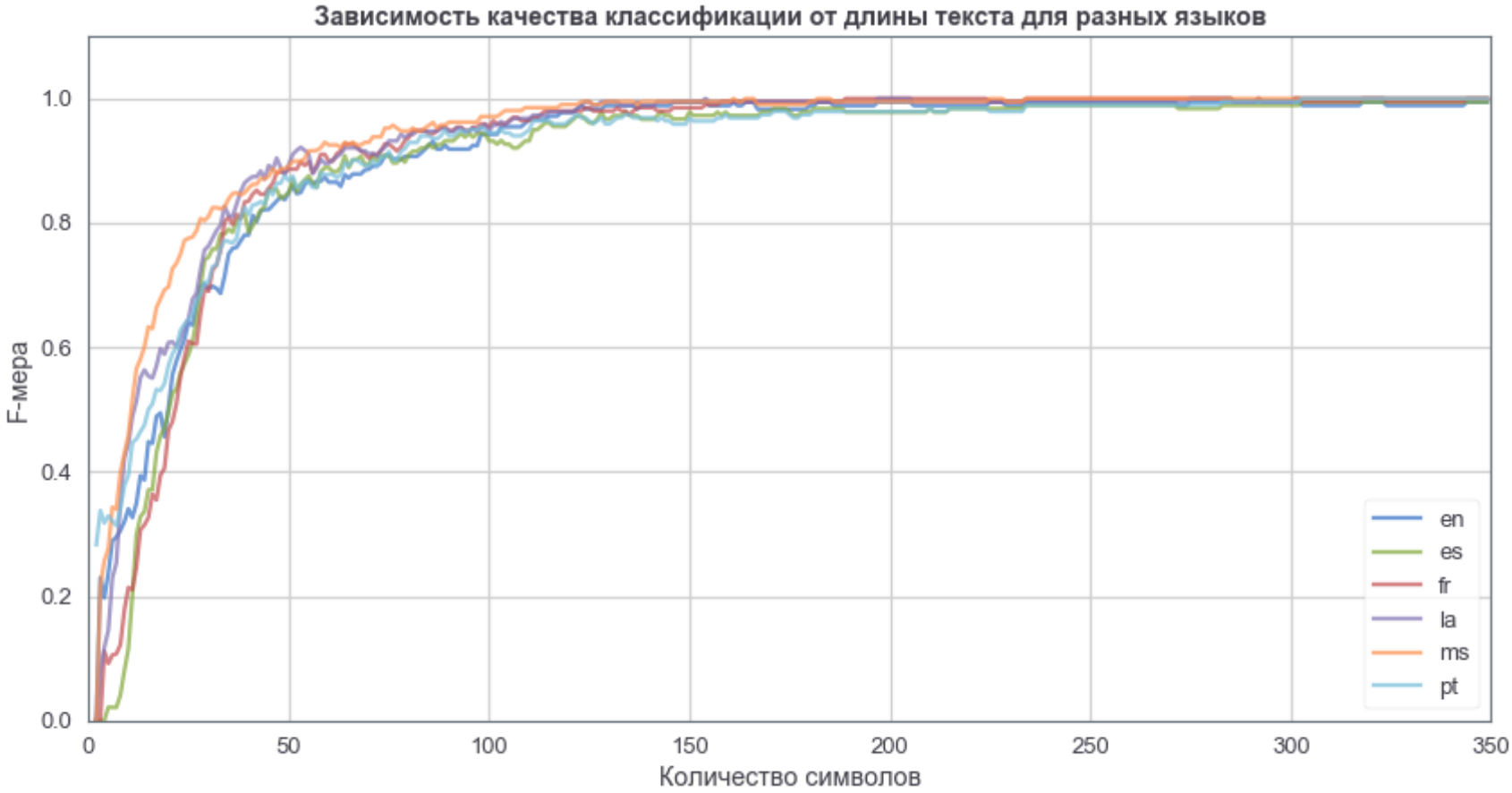
In [34]:
```
%%time
%%capture
f1_langs = []
```

```
lens = list(range(2, 350))
for i in lens:
    text_trunc, data_trunc = [], []
    text_trunc = [x[:i] for x in text_test]
    data_trunc = vectorizer.transform(text_trunc)
    f1_langs.append(metrics.f1_score(lang_test, logreg.predict(data_trunc), average=Non
e))
```

```
Wall time: 27.8 s
```

In [35]:
```
f1_res = {}
for lang in langs:
    f1_res[lang] = []
    for i in f1_langs:
        f1_res[lang].append(i[langs.index(lang)])
```

In [36]:
```
plt.figure(figsize=(14,7))
plt.xlabel('Количество символов')
plt.ylabel('F-мера')
plt.xlim(0, 350)
plt.ylim(0, 1.1)

for lang in langs:
    plt.plot(lens, f1_res[lang], label=lang, alpha=0.7)

plt.title(('Зависимость качества классификации от длины текста для разных языков'),
          fontsize=14, fontweight='bold')
plt.legend()
plt.show();
```



Из графика видно, что лучше всего распознается малайский язык, что можно счесть знакомерным, так как это единственный неевропейский язык в выборке. Несколько хуже распознаются, как и ожидалось, испанский и португальский языки, очень похожие между собой.

Разные длины н-грамм

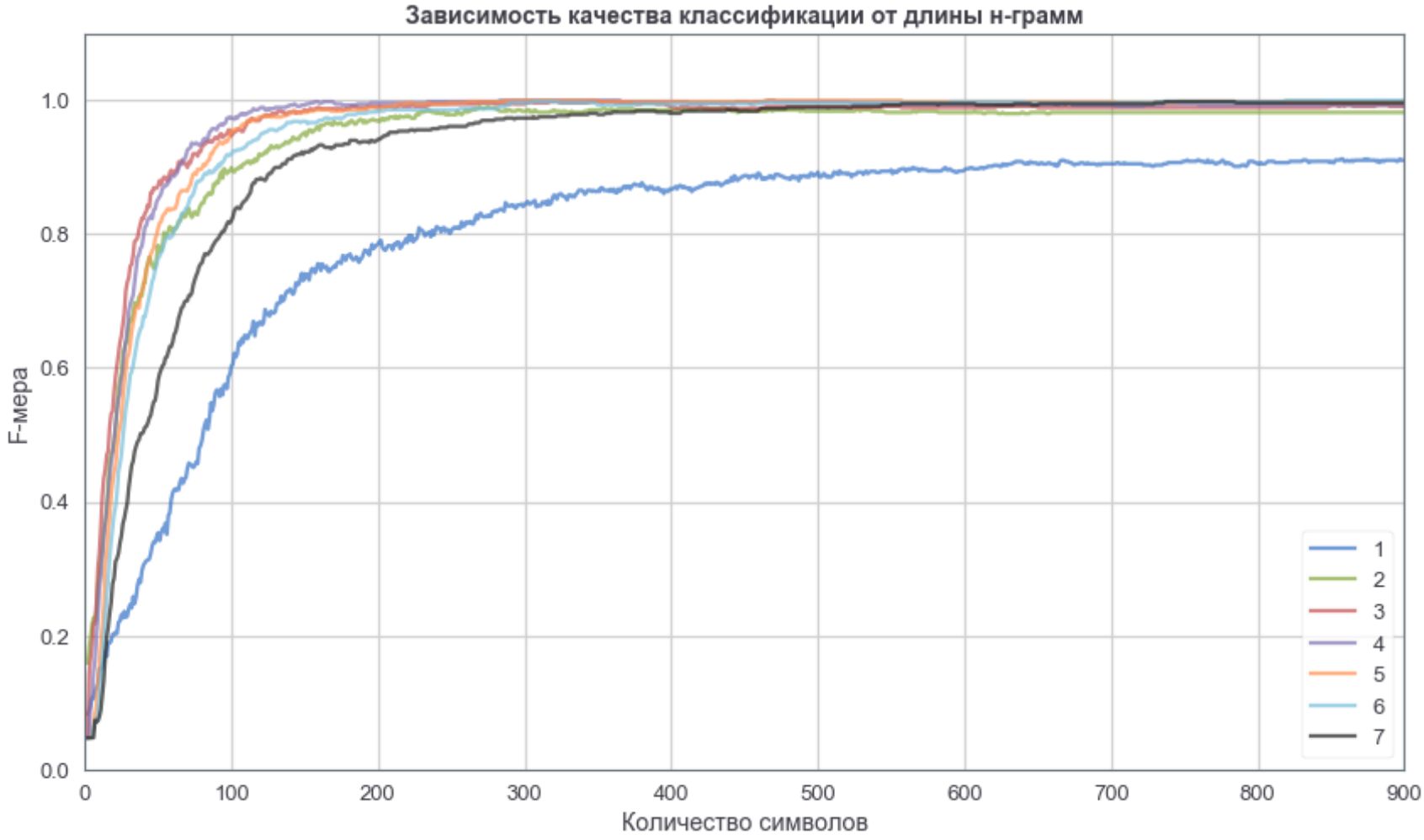
Базовое решение было основано на триграммах. Теперь посмотрим, как повлияет на точность определения языка изменение длины н-грамм.

In [37]:
```
%%time
ns = [1, 2, 3, 4, 5, 6, 7]
lens = list(range(2, 900))

f1_res = {}
for n in tqdm(ns):
    vectorizer_n = CountVectorizer(ngram_range=(n,n),
                                   lowercase=True,
                                   analyzer = 'char',
                                   min_df = 10,
                                   max_df = 1.0)
    data_train_ = vectorizer_n.fit_transform(text_train)
    logreg.fit(data_train_, lang_train)
    f1_res[n] = []
    for i in lens:
        text_trunc, data_trunc = [], []
        text_trunc = [x[:i] for x in text_test]
        data_trunc = vectorizer_n.transform(text_trunc)
        f1_res[n].append(metrics.f1_score(lang_test, logreg.predict(data_trunc), averag
e='macro'))
```

```
  0%|                                                                                   | 0/7 [00:00<?, ?it/s]c:\user
s\mytas\appdata\local\programs\python\python36\lib\site-packages\sklearn\metrics\classification.py:1135: UndefinedMetricWarnin
g: F-score is ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
100%|████████████████████████████████████████████████████████████████████████████| 7/7 [18:26<00:00, 158.07s/it]

Wall time: 18min 26s
```

In [38]:
```
plt.figure(figsize=(14,8))
plt.xlabel('Количество символов')
plt.ylabel('F-мера')
```

```
plt.xlim(0, 900)
plt.ylim(0, 1.1)

for n in ns:
    plt.plot(lens, f1_res[n], label=n, alpha=0.7)

plt.title(('Зависимость качества классификации от длины н-грамм'),
          fontsize=14, fontweight='bold')
plt.legend()
plt.show();
```



In [39]:
```
max(f1_res[1])
```

```
0.91247681117492696
```

Видим, что классификаторы, построенные на 3-, 4- и 5-граммах достигают максимальной точности определения языка при примерно одинаковой длине тектов, тогда как классификатор на биграммах достигает такой точности гораздо позже, а для классификатора на униграммах максимальное качество

классификации составляет *F=0,91*. При этом видим, что дальнейшее увеличение окна не ведет к улучшению качества классификации, а использование н-грамм с большим n наоборот снижает его.

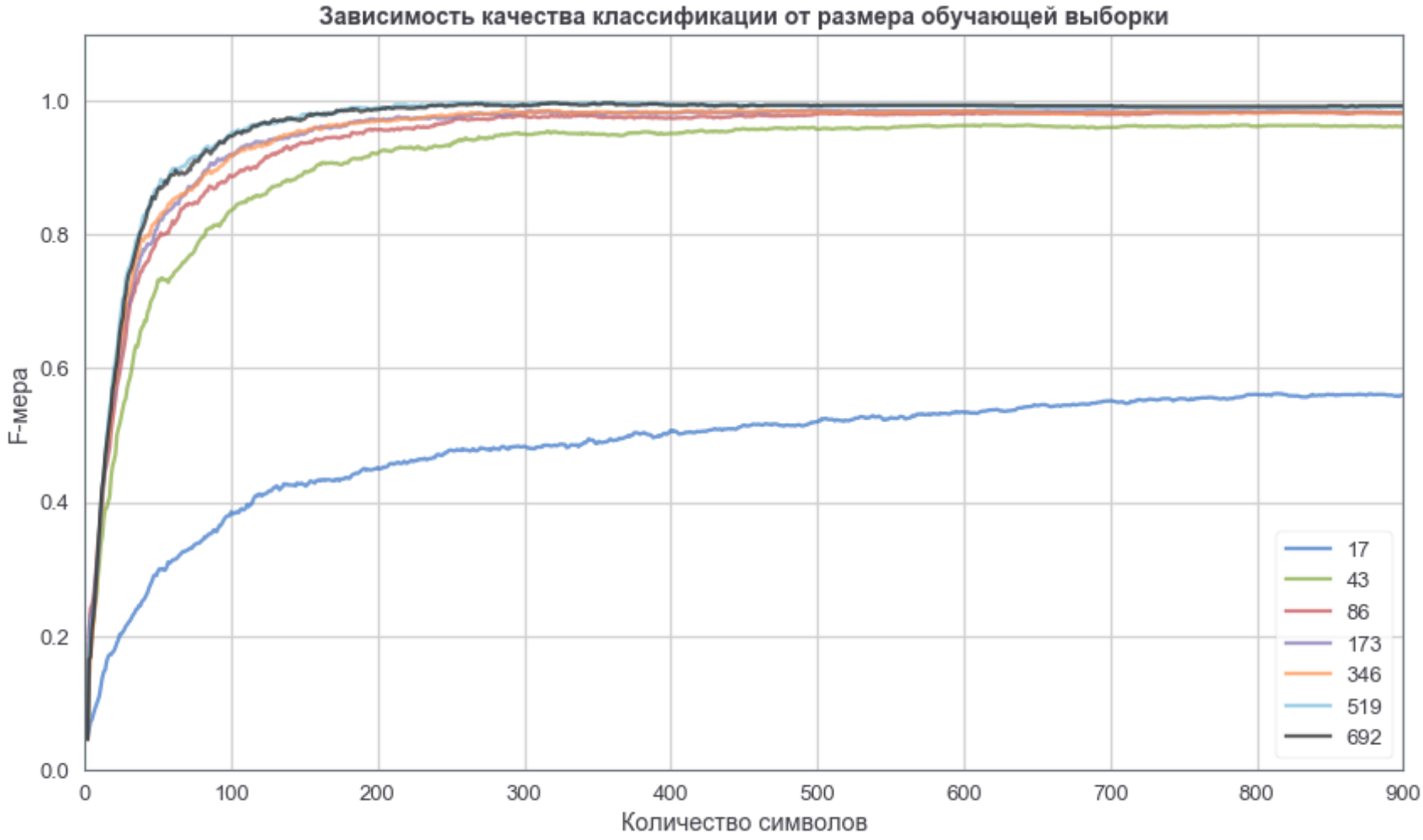
Обучающее множество разного размера

In [56]:
```
%%time

ss = [0.01, 0.025, 0.05, 0.1, 0.2, 0.3, 0.4]
lens = list(range(2, 900))
vectorizer_s = CountVectorizer(ngram_range=(3,3),
                               lowercase=True,
                               analyzer = 'char',
                               min_df = 10,
                               max_df = 1.0)

f1_res = {}
for s in tqdm(ss):
    text_train_, text_test_, lang_train_, lang_test_ = train_test_split(data.content,
                                                                        data.lang,
                                                                        test_size=(1-s),
                                                                        random_state=1)
    data_train_ = vectorizer_s.fit_transform(text_train_)
    logreg.fit(data_train_, lang_train_)
    f1_res[s] = []
    for i in lens:
        text_trunc, data_trunc = [], []
        text_trunc = [x[:i] for x in text_test_]
        data_trunc = vectorizer_s.transform(text_trunc)
        f1_res[s].append(metrics.f1_score(lang_test_, logreg.predict(data_trunc), avera
ge='macro'))
```

```
  0%|                                                                                                | 0/7 [00:00<?, ?it/s]c:\user
s\mytas\appdata\local\programs\python\python36\lib\site-packages\sklearn\metrics\classification.py:1135: UndefinedMetricWarnin
g: F-score is ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
100%|████████████████████████████████████████████████████████████████████████████████| 7/7 [41:27<00:00, 355.42s/it]

Wall time: 41min 27s
```

In [58]:

```
plt.figure(figsize=(14,8))
plt.xlabel('Количество символов')
plt.ylabel('F-мера')
plt.xlim(0, 900)
plt.ylim(0, 1.1)

for s in ss:
    plt.plot(lens, f1_res[s], label=round(len(data.lang)*s), alpha=0.7)

plt.title(('Зависимость качества классификации от размера обучающей выборки'),
          fontsize=14, fontweight='bold')
plt.legend()
plt.show();
```



Судя по полученному графику, исходная обучающая выборка, в которую вошли 1158 текстов, может считаться даже избыточной, так как уже при переходе от 519 текстов к 692 точность распознавания языка не повышается. Кроме того, значительно хуже работает только модель, обученная на 17 текстах, тогда как модель, обученная на 43 текстах дает результат, близкий к максимальному.

Разные классификаторы

Наконец, сравним различные методы классификации, а именно следующие: логистическая регрессия (для нескольких классов), метод опорных векторов, наивный байесовский классификатор и случайный лес.

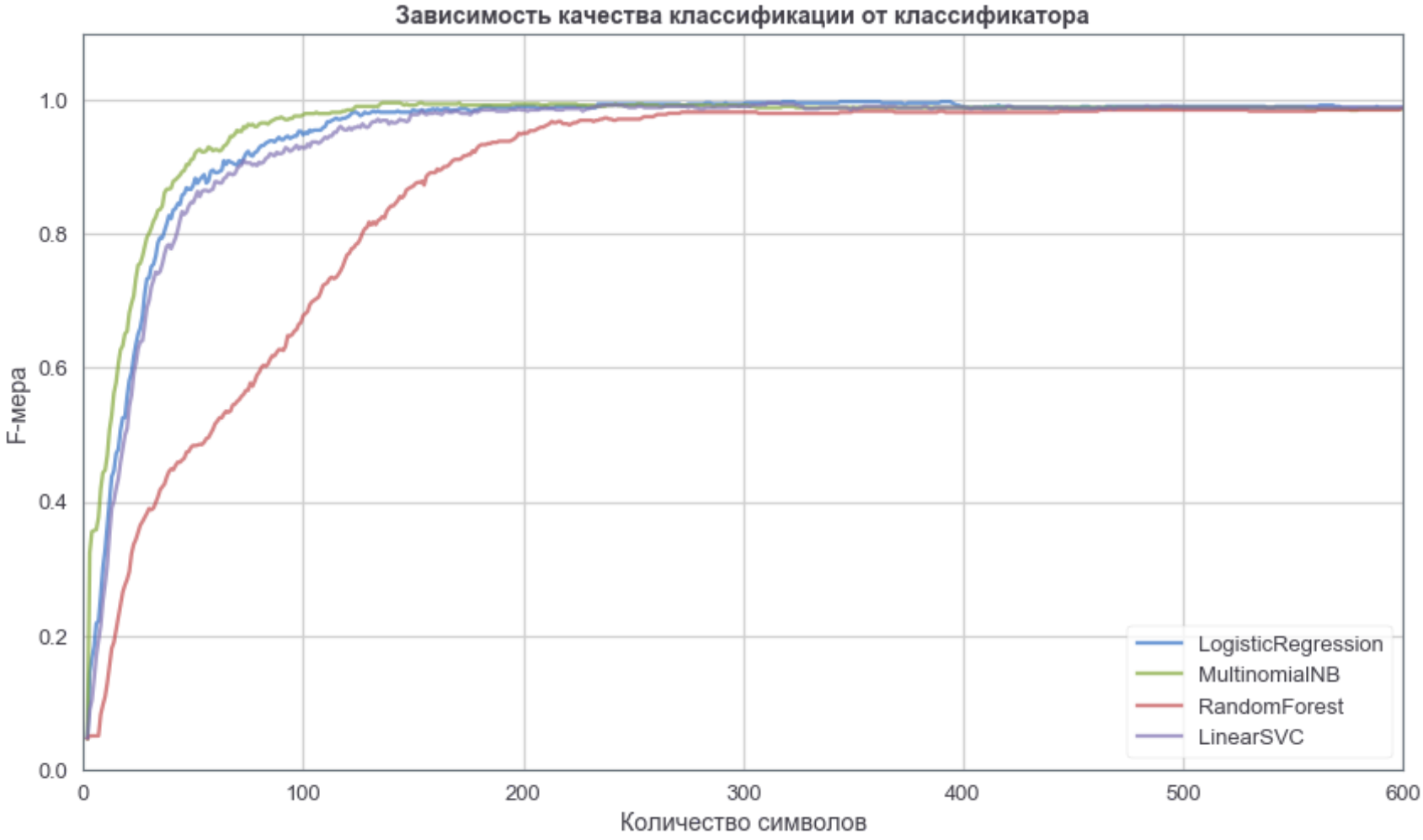
In [42]:
```
mnb = MultinomialNB()
forest = RandomForestClassifier(n_estimators = 100)
svc = LinearSVC()
```

In [53]:
```
%%time
%%capture
estimators = {'LogisticRegression': logreg,
              'MultinomialNB': mnb,
              'RandomForest': forest,
              'LinearSVC': svc}
lens = list(range(2, 500))

f1_res = {}
for k, v in tqdm(estimators.items()):
    v.fit(data_train, lang_train)
    f1_res[k] = []
    for i in lens:
        text_trunc, data_trunc = [], []
        text_trunc = [x[:i] for x in text_test]
        data_trunc = vectorizer.transform(text_trunc)
        f1_res[k].append(metrics.f1_score(lang_test, v.predict(data_trunc), average='ma
cro'))
```

```
Wall time: 5min 50s
```

In [54]:
```
plt.figure(figsize=(14,8))
```

```
plt.xlabel('Количество символов')
plt.ylabel('F-мера')
plt.xlim(0, 500)
plt.ylim(0, 1.1)

for k, v in estimators.items():
    plt.plot(lens, f1_res[k], label=k, alpha=0.7)

plt.title(('Зависимость качества классификации от классификатора'),
          fontsize=14, fontweight='bold')
plt.legend()
plt.show();
```



Из графика видно, что результаты полученные с помощью логистической регрессии, метода опорных векторов и байесовского классификатора, в целом сходны (с небольшим преимуществом байесовского классификатора), тогда как случайный лес на коротких фрагментах текстов работает значительно хуже. При увеличении длинны фрагментов все классификаторы дают одинаково точный результат.