Рубежный контроль №2

Ишков Денис Олегович, ИУ5-24М, 2021г.

Тема: Методы обработки текстов

Решение задачи классификации текстов.

Необходимо решить задачу классификации текстов на основе любого выбранного Вами датасета (кроме примера, который рассматривался в лекции). Классификация может быть бинарной или многоклассовой. Целевой признак из выбранного Вами датасета может иметь любой физический смысл, примером является задача анализа тональности текста.

Необходимо сформировать два варианта векторизации признаков - на основе CountVectorizer и на основе TfidfVectorizer.

В качестве классификаторов необходимо использовать два классификатора по варианту для Вашей группы:

Группа: ИУ5-24М

Классификатор 1: KNeighborsClassifier

Классификатор 2: Complement Naive Bayes (CNB)

Для каждого метода необходимо оценить качество классификации. Сделайте вывод о том, какой вариант векторизации признаков в паре с каким классификатором показал лучшее качество.

Датасет

Бинарная классификация текста

https://www.kaggle.com/blackmoon/russian-language-toxic-comments (https://www.kaggle.com/blackmoon/russian-language-toxic-comments)

In [1]:

```
# загрузка датасета
!pip install wldhx.yadisk-direct
!curl -L $(yadisk-direct https://disk.yandex.ru/d/wedARfrtMn-Y-Q) -o labeled.csv
```

```
Requirement already satisfied: wldhx.yadisk-direct in /opt/conda/lib/pytho
n3.7/site-packages (0.0.6)
Requirement already satisfied: requests in /opt/conda/lib/python3.7/site-p
ackages (from wldhx.yadisk-direct) (2.25.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /opt/conda/lib/pyt
hon3.7/site-packages (from requests->wldhx.yadisk-direct) (1.26.4)
Requirement already satisfied: chardet<5,>=3.0.2 in /opt/conda/lib/python
3.7/site-packages (from requests->wldhx.yadisk-direct) (4.0.0)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python
3.7/site-packages (from requests->wldhx.yadisk-direct) (2020.12.5)
Requirement already satisfied: idna<3,>=2.5 in /opt/conda/lib/python3.7/si
te-packages (from requests->wldhx.yadisk-direct) (2.10)
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Cu
rrent
                                 Dload  Upload   Total   Spent    Left  Sp
eed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
0
100 4560k  100 4560k    0     0  3071k      0  0:00:01  0:00:01 --:--:-- 1
2.6M
```

Импорт нужных библиотек

In [2]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
import nltk
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import SnowballStemmer
nltk.download('punkt')
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import ComplementNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.metrics import precision_score, recall_score, precision_recall_curve, clas
sification_report
from matplotlib import pyplot as plt
from sklearn.metrics import plot_precision_recall_curve
import numpy as np
from sklearn.model_selection import GridSearchCV
```

```
[nltk_data] Downloading package punkt to /usr/share/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Анализ и обработка выбросов в данных

In [3]:

```
df = pd.read_csv("labeled.csv", sep=",")

df.describe()
```

Out[3]:

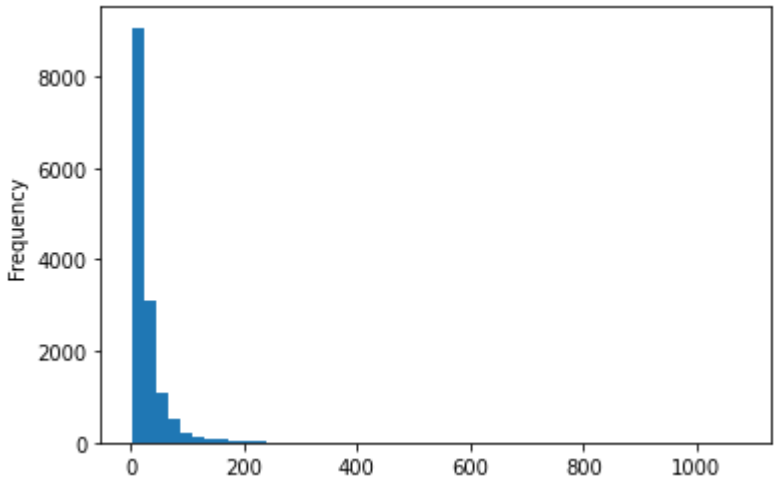
	toxic
count	14412.000000
mean	0.334860
std	0.471958
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	1.000000

In [4]:

```
df.comment.str.split(' ').apply(len).plot(kind='hist', bins=50)
```

Out[4]:

```
<AxesSubplot:ylabel='Frequency'>
```



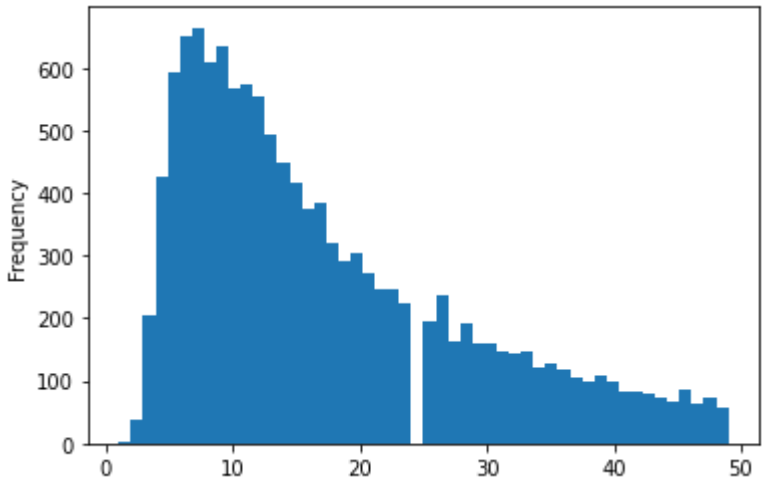
Как видно из гистограммы, количество слов сообщений в данных распределено по экспоненциальному закону. Уберём "хвост"

In [5]:

```
df = df[df.comment.str.split(' ').apply(len) < 50].copy()
df.comment.str.split(' ').apply(len).plot(kind='hist', bins=50)
```

Out[5]:

```
<AxesSubplot:ylabel='Frequency'>
```



In [6]:

```
df["toxic"] = df["toxic"].apply(int)
df["toxic"].value_counts()
```

Out[6]:

```
0    8102
1    4419
Name: toxic, dtype: int64
```

Разделим данные на обучающую и тестовую выборки

In [7]:

```
train_df, test_df = train_test_split(df, test_size=500, random_state=69, stratify=df.to
xic)
```

In [8]:

```
test_df["toxic"].value_counts(), train_df["toxic"].value_counts()
```

Out[8]:

```
(0    324
 1    176
 Name: toxic, dtype: int64,
 0    7778
 1    4243
 Name: toxic, dtype: int64)
```

Предобработка текста

In [9]:

```
sentence_example = df.iloc[-1]["comment"]
tokens = word_tokenize(sentence_example, language="russian")
tokens_without_punctuation = [i for i in tokens if i not in string.punctuation]
russian_stop_words = stopwords.words("russian")
tokens_without_stop_words_and_punctuation = [i for i in tokens_without_punctuation if i
not in russian_stop_words]
snowball = SnowballStemmer(language="russian")
stemmed_tokens = [snowball.stem(i) for i in tokens_without_stop_words_and_punctuation]
```

In [10]:

```
print(f"Исходный текст: {sentence_example}")
print("-----------------")
print(f"Токены: {tokens}")
print("-----------------")
print(f"Токены без пунктуации: {tokens_without_punctuation}")
print("-----------------")
print(f"Токены без пунктуации и стоп слов: {tokens_without_stop_words_and_punctuation}"
)
print("-----------------")
print(f"Токены после стемминга: {stemmed_tokens}")
print("-----------------")
```

```
Исходный текст: До сих пор пересматриваю его видео. Орамбо кстати на своем
канале пилит похожий контент, но качеством похуже, там же и Шуран не редко
светится, храню хрупкую надежду что когда-то он вернется, такая годнота ве
дь.

-----------------
Токены: ['До', 'сих', 'пор', 'пересматриваю', 'его', 'видео', '.', 'Орамб
о', 'кстати', 'на', 'своем', 'канале', 'пилит', 'похожий', 'контент', ',',
'но', 'качеством', 'похуже', ',', 'там', 'же', 'и', 'Шуран', 'не', 'редк
о', 'светится', ',', 'храню', 'хрупкую', 'надежду', 'что', 'когда-то', 'о
н', 'вернется', ',', 'такая', 'годнота', 'ведь', '.']
-----------------
Токены без пунктуации: ['До', 'сих', 'пор', 'пересматриваю', 'его', 'виде
о', 'Орамбо', 'кстати', 'на', 'своем', 'канале', 'пилит', 'похожий', 'конт
ент', 'но', 'качеством', 'похуже', 'там', 'же', 'и', 'Шуран', 'не', 'редк
о', 'светится', 'храню', 'хрупкую', 'надежду', 'что', 'когда-то', 'он', 'в
ернется', 'такая', 'годнота', 'ведь']
-----------------
Токены без пунктуации и стоп слов: ['До', 'сих', 'пор', 'пересматриваю',
'видео', 'Орамбо', 'кстати', 'своем', 'канале', 'пилит', 'похожий', 'конте
нт', 'качеством', 'похуже', 'Шуран', 'редко', 'светится', 'храню', 'хрупку
ю', 'надежду', 'когда-то', 'вернется', 'такая', 'годнота']
-----------------
Токены после стемминга: ['до', 'сих', 'пор', 'пересматрива', 'виде', 'орам
б', 'кстат', 'сво', 'канал', 'пил', 'похож', 'контент', 'качеств', 'поху
ж', 'шура', 'редк', 'свет', 'хран', 'хрупк', 'надежд', 'когда-т', 'верне
т', 'так', 'годнот']
-----------------
```

In [11]:

```
snowball = SnowballStemmer(language="russian")
russian_stop_words = stopwords.words("russian")

def tokenize_sentence(sentence: str, remove_stop_words: bool = True):
    tokens = word_tokenize(sentence, language="russian")
    tokens = [i for i in tokens if i not in string.punctuation]
    if remove_stop_words:
        tokens = [i for i in tokens if i not in russian_stop_words]
    tokens = [snowball.stem(i) for i in tokens]
    return tokens

tokenize_sentence(sentence_example)
```

Out[11]:

```
['до',
 'сих',
 'пор',
 'пересматрива',
 'виде',
 'орамб',
 'кстат',
 'сво',
 'канал',
 'пил',
 'похож',
 'контент',
 'качеств',
 'похуж',
 'шура',
 'редк',
 'свет',
 'хран',
 'хрупк',
 'надежд',
 'когда-т',
 'вернет',
 'так',
 'годнот']
```

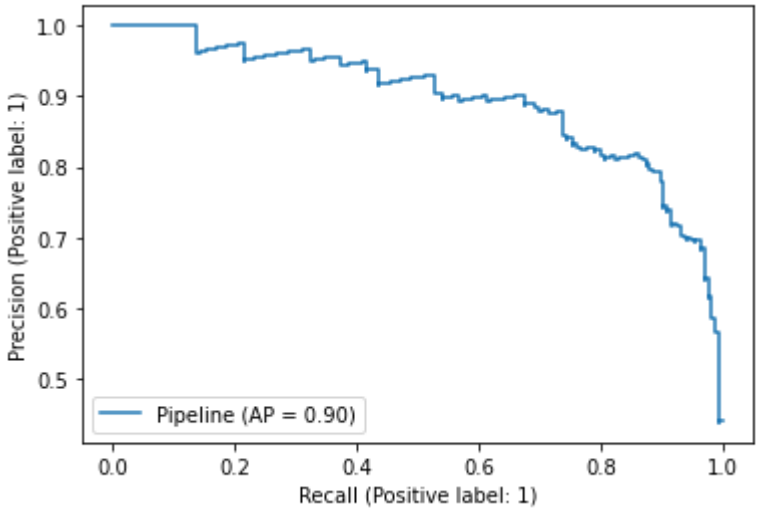
Классификатор 1: Complement Naive Bayes

CountVectorizer

In [15]:

```
grid_pipeline = Pipeline([
    ("vectorizer", CountVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop
_words=True))),
    ("model",
     GridSearchCV(
        ComplementNB(alpha=1.0, norm=False),
        param_grid={'alpha': [0.35+5e-3*i for i in range(20)]+\
                             [0.5+5e-3*i for i in range(60)]+\
                             [0.5, 0.75, 1., 10., 100., 1e3, 1e4, 1e5],
                    'norm': [True, False]},
        cv=5,
        verbose=0,
        scoring='roc_auc',
        )
    )
])

grid_pipeline.fit(train_df["comment"], train_df["toxic"])
print(grid_pipeline['model'].best_params_)
model_pipeline = Pipeline([
    ("vectorizer", CountVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop
_words=True))),
    ("model", ComplementNB(**grid_pipeline['model'].best_params_),)
]
)

model_pipeline.fit(train_df["comment"], train_df["toxic"])
prec_c_10, rec_c_10, thresholds_c_10 = precision_recall_curve(y_true=test_df["toxic"],
                                                              probas_pred=model_pipelin
e.predict_proba(test_df["comment"])[:, 1])
plot_precision_recall_curve(estimator=model_pipeline, X=test_df["comment"], y=test_df[
"toxic"])
print(classification_report(y_true=test_df["toxic"],
                            y_pred=model_pipeline.predict(test_df["comment"]),
                            digits=4))
```

```
{'alpha': 0.63, 'norm': False}
              precision    recall  f1-score   support

           0     0.9009    0.8981    0.8995       324
           1     0.8136    0.8182    0.8159       176

    accuracy                         0.8700       500
   macro avg     0.8572    0.8582    0.8577       500
weighted avg     0.8702    0.8700    0.8701       500
```

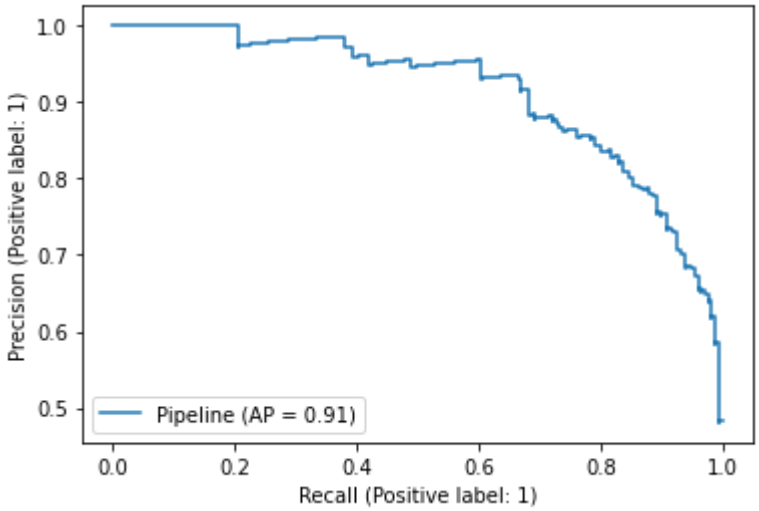


TfidfVectorizer

In [16]:

```
grid_pipeline = Pipeline([
    ("vectorizer", TfidfVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop
_words=True))),
    ("model",
     GridSearchCV(
        ComplementNB(alpha=1.0, norm=False),
        param_grid={'alpha': [0.3+5e-3*i for i in range(60)]+\
                             [0.5, 0.75, 1., 10., 100., 1e3, 1e4, 1e5],
                    'norm': [True, False]},
        cv=5,
        verbose=0,
        scoring='roc_auc',
        )
    )
])

grid_pipeline.fit(train_df["comment"], train_df["toxic"])
print(grid_pipeline['model'].best_params_)
model_pipeline = Pipeline([
    ("vectorizer", TfidfVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop
_words=True))),
    ("model", ComplementNB(**grid_pipeline['model'].best_params_),)
]
)

model_pipeline.fit(train_df["comment"], train_df["toxic"])
prec_c_10, rec_c_10, thresholds_c_10 = precision_recall_curve(y_true=test_df["toxic"],
                                                              probas_pred=model_pipelin
e.predict_proba(test_df["comment"])[:, 1])
plot_precision_recall_curve(estimator=model_pipeline, X=test_df["comment"], y=test_df[
"toxic"])
print(classification_report(y_true=test_df["toxic"],
                            y_pred=model_pipeline.predict(test_df["comment"]),
                            digits=4))
```

```
{'alpha': 0.37, 'norm': False}
              precision    recall  f1-score   support

           0     0.8801    0.9290    0.9039       324
           1     0.8544    0.7670    0.8084       176

    accuracy                         0.8720       500
   macro avg     0.8673    0.8480    0.8561       500
weighted avg     0.8711    0.8720    0.8703       500
```



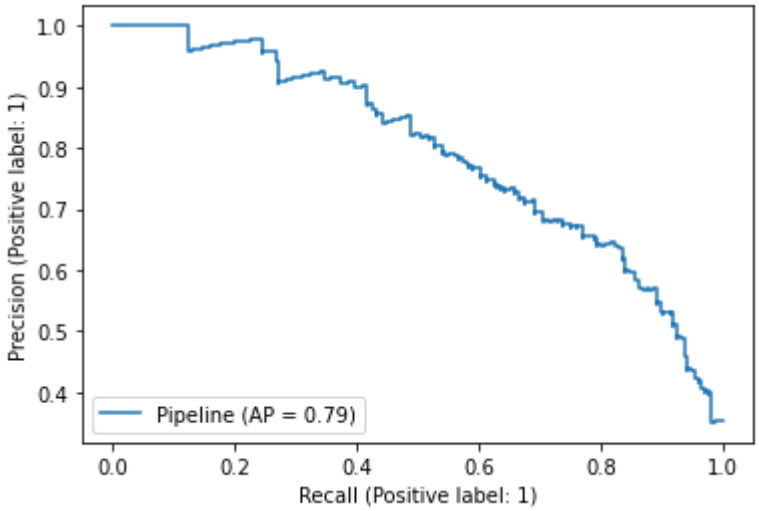
Классификатор 2: KNearestNeighbors

CountVectorizer

In [17]:

```
grid_pipeline = Pipeline([
    ("vectorizer", CountVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop
_words=True))),
    ("model",
     GridSearchCV(
        KNeighborsClassifier(),
        param_grid={'n_neighbors': [i for i in range(48, 96, 2)],
                    'weights': ['uniform', 'distance'],
                    'metric': ['euclidean', 'cosine',]},
        cv=5,
        verbose=1,
        scoring='roc_auc', #'f1'
        )
    )
])

grid_pipeline.fit(train_df["comment"], train_df["toxic"])
print(grid_pipeline['model'].best_params_)
model_pipeline = Pipeline([
    ("vectorizer", CountVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop
_words=True))),
    ("model", KNeighborsClassifier(**grid_pipeline['model'].best_params_),)
]
)

model_pipeline.fit(train_df["comment"], train_df["toxic"])
prec_c_10, rec_c_10, thresholds_c_10 = precision_recall_curve(y_true=test_df["toxic"],
                                                              probas_pred=model_pipelin
e.predict_proba(test_df["comment"])[:, 1])
plot_precision_recall_curve(estimator=model_pipeline, X=test_df["comment"], y=test_df[
"toxic"])
print(classification_report(y_true=test_df["toxic"],
                            y_pred=model_pipeline.predict(test_df["comment"]),
                            digits=4))
```

```
Fitting 5 folds for each of 96 candidates, totalling 480 fits
{'metric': 'cosine', 'n_neighbors': 90, 'weights': 'distance'}
              precision    recall  f1-score   support

           0     0.7413    0.9815    0.8446       324
           1     0.9155    0.3693    0.5263       176

    accuracy                         0.7660       500
   macro avg     0.8284    0.6754    0.6855       500
weighted avg     0.8026    0.7660    0.7326       500
```



Какие параметры лучшие?

In [18]:

```
cols = ['param_metric', 'param_n_neighbors', 'param_weights', 'mean_test_score']
pd.DataFrame(grid_pipeline['model'].cv_results_).sort_values(by='rank_test_score').loc
[:, cols].head(10)
```

Out[18]:

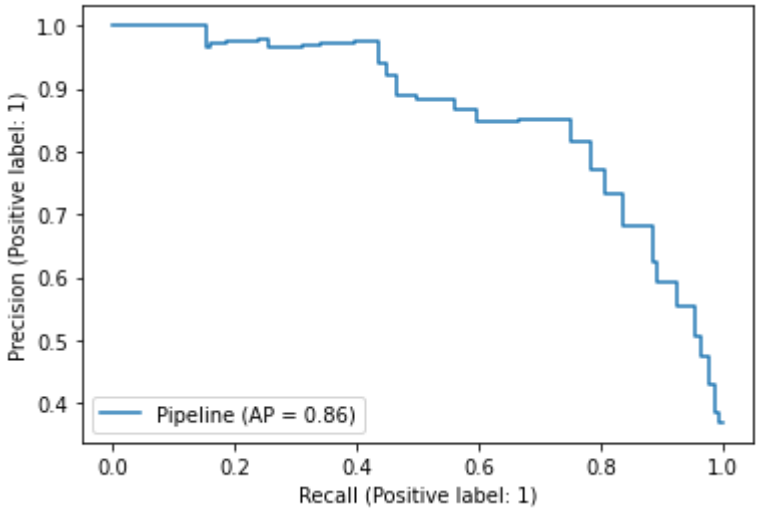
	param_metric	param_n_neighbors	param_weights	mean_test_score
91	cosine	90	distance	0.868720
93	cosine	92	distance	0.868559
89	cosine	88	distance	0.868515
95	cosine	94	distance	0.868301
85	cosine	84	distance	0.868067
87	cosine	86	distance	0.868009
83	cosine	82	distance	0.867868
90	cosine	90	uniform	0.867667
81	cosine	80	distance	0.867628
79	cosine	78	distance	0.867582

TFidfVectorizer

In [19]:

```
grid_pipeline = Pipeline([
    ("vectorizer", TfidfVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop
_words=True))),
    ("model",
     GridSearchCV(
        KNeighborsClassifier(),
        param_grid={'n_neighbors': [i for i in range(31, 64, 2)]+[44, 46],
                    'weights': ['uniform', 'distance'],
                    'metric': ['euclidean', 'cosine',]},
        cv=5,
        verbose=1,
        scoring='roc_auc', #'f1'
        )
    )
])

grid_pipeline.fit(train_df["comment"], train_df["toxic"])
print(grid_pipeline['model'].best_params_)
model_pipeline = Pipeline([
    ("vectorizer", TfidfVectorizer(tokenizer=lambda x: tokenize_sentence(x, remove_stop
_words=True))),
    ("model", KNeighborsClassifier(**grid_pipeline['model'].best_params_),)
]
)

model_pipeline.fit(train_df["comment"], train_df["toxic"])
prec_c_10, rec_c_10, thresholds_c_10 = precision_recall_curve(y_true=test_df["toxic"],
                                                              probas_pred=model_pipelin
e.predict_proba(test_df["comment"])[:, 1])
plot_precision_recall_curve(estimator=model_pipeline, X=test_df["comment"], y=test_df[
"toxic"])
print(classification_report(y_true=test_df["toxic"],
                            y_pred=model_pipeline.predict(test_df["comment"]),
                            digits=4))
```

```
Fitting 5 folds for each of 76 candidates, totalling 380 fits
{'metric': 'cosine', 'n_neighbors': 39, 'weights': 'uniform'}
              precision    recall  f1-score   support

           0     0.7805    0.9660    0.8634       324
           1     0.8889    0.5000    0.6400       176

    accuracy                         0.8020       500
   macro avg     0.8347    0.7330    0.7517       500
weighted avg     0.8187    0.8020    0.7848       500
```



Какие параметры лучшие?

In [20]:

```
cols = ['param_metric', 'param_n_neighbors', 'param_weights', 'mean_test_score']
pd.DataFrame(grid_pipeline['model'].cv_results_).sort_values(by='rank_test_score').loc
[:, cols].head(10)
```

Out[20]:

	param_metric	param_n_neighbors	param_weights	mean_test_score
46	cosine	39	uniform	0.918629
48	cosine	41	uniform	0.918623
50	cosine	43	uniform	0.918557
72	cosine	44	uniform	0.918119
44	cosine	37	uniform	0.917986
52	cosine	45	uniform	0.917931
56	cosine	49	uniform	0.917782
42	cosine	35	uniform	0.917675
74	cosine	46	uniform	0.917671
60	cosine	53	uniform	0.917582

Выводы

	ComplementNB	KNN Classifier
CountVectorizer	0.8701	0.7326
TfIdfVectorizer	0.8703	0.7848

Лучше всего по f1-мере показала себя связка TfIdfVectorizer + ComplementNB. Скорее всего, качество выше соседей, потому что модель специально заточена под несбалансированные выборки. В таком случае нужно было заранее выравнивать по классам данные для обучения соседей, чего не требовалось по заданию РК.