GeekBrains, ML in Business

Lesson 2 Homework

Ссылки:

http://www.machinelearning.ru/wiki/images/d/d5/Voron17survey-artm.pdf

https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation

Импорт библиотек

In [45]:
```
import itertools
import re

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score, roc_auc_score, precision_score, classification_report, precision_recall_curve, confusion_mat

from gensim.corpora.dictionary import Dictionary
from gensim.models import LdaModel, TfidfModel

import pymorphy2
from nltk.corpus import stopwords
from razdel import tokenize


%matplotlib inline
```

Функции для заданий

In [2]:
```
def clean_text(text):
    '''
    Очистка текста
    '''
    if not isinstance(text, str):
        text = str(text)
    text = text.lower()
    text = text.strip('\n').strip('\r').strip('\t')
    text = re.sub("-\s\r\n\|-\s\r\n|\r\n", '', str(text))
    text = re.sub("[0-9]|[-—.,:;_%©«»?*!@#№$^•·&()]|[+=]|[[]|[]]|[/]|", '', text)
    text = re.sub(r"\r\n\t|\n|\\s|\r\t|\\n", ' ', text)
    text = re.sub(r'[\xad]|[\s+]', ' ', text.strip())
    return text


def lemmatization(text):
    '''
    Лемматизация
        [0] если зашел тип не `str` делаем его `str`
        [1] токенизация предложения через razdel
        [2] проверка есть ли в начале слова '-'
        [3] проверка токена с одного символа
        [4] проверка есть ли данное слово в кэше
        [5] лемматизация слова
        [6] проверка на стоп-слова
    '''
    # [0]
    if not isinstance(text, str):
        text = str(text)
    # [1]
    tokens = list(tokenize(text))
    words = [_.text for _ in tokens]
    words_lem = []
    for w in words:
        if w[0] == '-': # [2]
            w = w[1:]
        if len(w)>1: # [3]
            if w in cache: # [4]
                words_lem.append(cache[w])
            else: # [5]
                temp_cach = cache[w] = morph.parse(w)[0].normal_form
                words_lem.append(temp_cach)
    words_lem_without_stopwords=[i for i in words_lem if not i in stopword_ru] # [6]
    return words_lem_without_stopwords
```

In [3]:
```
def get_lda_vector(text):
    unseen_doc = common_dictionary.doc2bow(text)
    lda_tuple = lda[unseen_doc]
    not_null_topics = dict(zip([i[0] for i in lda_tuple], [i[1] for i in lda_tuple]))
```

```
    output_vector = []
    for i in range(25):
        if i not in not_null_topics:
            output_vector.append(0)
        else:
            output_vector.append(not_null_topics[i])
    return np.array(output_vector)
```

In [4]:
```
def get_user_embedding(user_articles_list):
    user_articles_list = eval(user_articles_list)
    user_vector = np.array([doc_dict[doc_id] for doc_id in user_articles_list])
    user_vector = np.mean(user_vector, 0)
    return user_vector
```

In [5]:
```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Построение модели

Считываем данные.

In [6]:
```
news = pd.read_csv("articles.csv")
users = pd.read_csv("users_articles.csv")
churn = pd.read_csv("users_churn.csv")
```

In [7]:
```
stopword_ru = stopwords.words('russian')
with open('stopwords.txt') as f:
    additional_stopwords = [w.strip() for w in f.readlines() if w]
stopword_ru += additional_stopwords
```

Чистим данные.

In [8]:
```
news['title'] = news['title'].apply(lambda x: clean_text(x), 1)

cache = {}
morph = pymorphy2.MorphAnalyzer()
news['title'] = news['title'].apply(lambda x: lemmatization(x), 1)
```

```
<ipython-input-2-ced9971dbd37>:10: FutureWarning: Possible nested set at position 39
  text = re.sub("[0-9]|[-—.,:;_%©«»?*!@#№$^•·&()]|[+=]|[[]|[]]|[/]|", '', text)
```

Получаем эмбеддинги документов по словам и обучаем LDA модель.

In [9]:
```
#сформируем список наших текстов, разбив еще и на пробелы
texts = news['title'].tolist()
# Create a corpus from a list of texts
common_dictionary = Dictionary(news['title'].values)
common_corpus = [common_dictionary.doc2bow(text) for text in texts]
# Train the model on the corpus.
lda = LdaModel(common_corpus, num_topics=25, id2word=common_dictionary)#, passes=10)
```

Получаем эмбеддинги документов по темам.

In [10]:
```
topic_matrix = pd.DataFrame([get_lda_vector(text) for text in news['title'].values])
topic_matrix.columns = ['topic_{}'.format(i) for i in range(25)]
topic_matrix['doc_id'] = news['doc_id'].values
topic_matrix = topic_matrix[['doc_id']+['topic_{}'.format(i) for i in range(25)]]
doc_dict = dict(zip(topic_matrix['doc_id'].values, topic_matrix[['topic_{}'.format(i) for i in range(25)]].values))
```

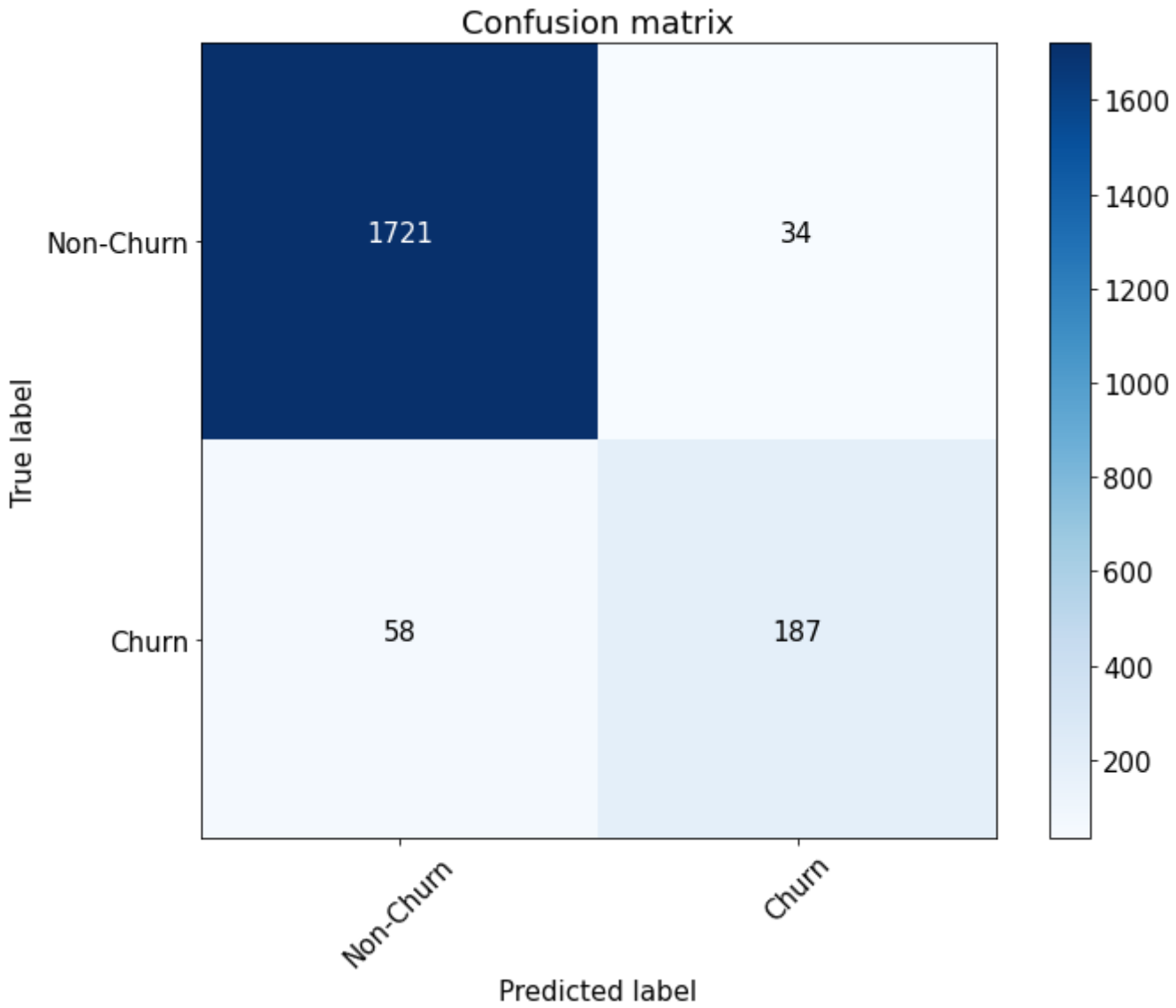
Функция для получения эмбеддингов пользователей и последующего обучения модели.

In [34]:
```
def train_embeddings_and_model(users, churn, user_embedding_fnc):
    # Получаем эмбеддинги пользователей по темам.
    user_embeddings = pd.DataFrame([i for i in users['articles'].apply(user_embedding_fnc, 1)])
    user_embeddings.columns = ['topic_{}'.format(i) for i in range(25)]
    user_embeddings['uid'] = users['uid'].values
    user_embeddings = user_embeddings[['uid']+['topic_{}'.format(i) for i in range(25)]]

    # Обучаем логистическую регрессию на данных оттока и эмбеддингах пользователей.
    X = pd.merge(user_embeddings, churn, 'left')
    X_train, X_test, y_train, y_test = train_test_split(X[['topic_{}'.format(i) for i in range(25)]],
                                                        X['churn'],
                                                        random_state=0)
    # logreg = LogisticRegression()
    logreg = RandomForestClassifier()
    logreg.fit(X_train, y_train)

    # Рассчитаем F1-score, ROC-AUC, построим матрицу ошибок
    preds = logreg.predict_proba(X_test)[:, 1]
    # print(pd.DataFrame({'test': y_test, 'pred': preds}))
    precision, recall, thresholds = precision_recall_curve(y_test, preds)
    # print(precision, recall)
    f1 = (2 * precision * recall) / (precision + recall)
    ix = np.argmax(f1)

    precision_score = precision[ix]
    recall_score = recall[ix]
    f1_score = f1[ix]
    print(f'F1 score: {f1_score}')

    roc_auc = roc_auc_score(y_test, preds)
    print(f'ROC-AUC score: {roc_auc}')

    font = {'size' : 15}
    plt.rc('font', **font)
    cnf_matrix = confusion_matrix(y_test, preds>thresholds[ix])
    plt.figure(figsize=(10, 8))
    plot_confusion_matrix(cnf_matrix,
                          classes=['Non-Churn', 'Churn'],
                          title='Confusion matrix')
    plt.show()

    return {'precision': precision_score,
            'recall': recall_score,
            'f1_score': f1_score,
            'roc_auc': roc_auc}
```

Находим эмбеддинги и обучаем модель, сохраняем метрики.

In [35]:
```
metrics_mean = train_embeddings_and_model(users, churn, get_user_embedding)
metrics_df = pd.DataFrame({key: [value] for key, value in metrics_mean.items()})
```

```
F1 score: 0.8085106382978723
ROC-AUC score: 0.974589220303506
```



In [36]:
```
metrics_df
```

Out[36]:

	precision	recall	f1_score	roc_auc
0	0.844444	0.77551	0.808511	0.974589

Задание 1

Самостоятельно разобраться с тем, что такое tfidf (документация https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html и еще - https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction).

Задание 2

Модифицировать код функции get_user_embedding таким образом, чтобы считалось не среднее (как в примере np.mean), а медиана.

Решение Задания 2

Меняем .mean() на .median().

In [37]:
```
def get_user_embedding_median(user_articles_list):
    user_articles_list = eval(user_articles_list)
    user_vector = np.array([doc_dict[doc_id] for doc_id in user_articles_list])
    # .mean() на .median()
    user_vector = np.median(user_vector, 0)
    return user_vector
```

Задание 3

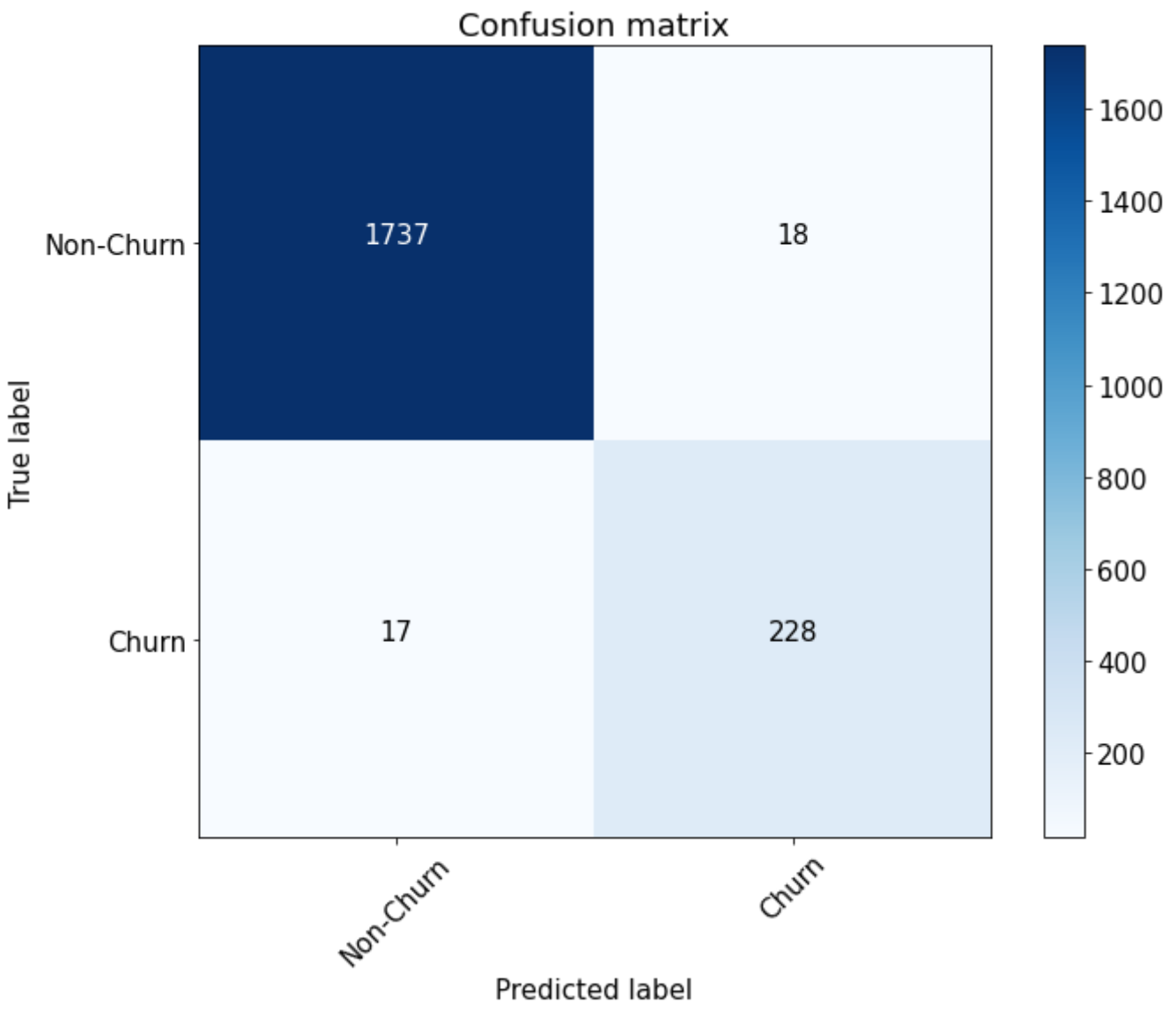
Применить такое преобразование к данным, обучить модель прогнозирования оттока и посчитать метрики качества и сохранить их: roc auc, precision/recall/f_score (для 3 последних - подобрать оптимальный порог с помощью precision_recall_curve, как это делалось на уроке).

Решение Задания 3

Сохраняем метрики для user embeddings через .median().

In [38]:
```
metrics_median = train_embeddings_and_model(users, churn, get_user_embedding_median)
metrics_df = metrics_df.append(metrics_median, ignore_index=True)
```

```
F1 score: 0.9308943089430893
ROC-AUC score: 0.9943415314843886
```



In [39]:
```
metrics_df
```

Out[39]:

	precision	recall	f1_score	roc_auc
0	0.844444	0.775510	0.808511	0.974589

	precision	recall	f1_score	roc_auc
1	0.927126	0.934694	0.930894	0.994342

Задание 4

Повторить п.2, но используя уже не медиану, а max.

Решение Задания 4

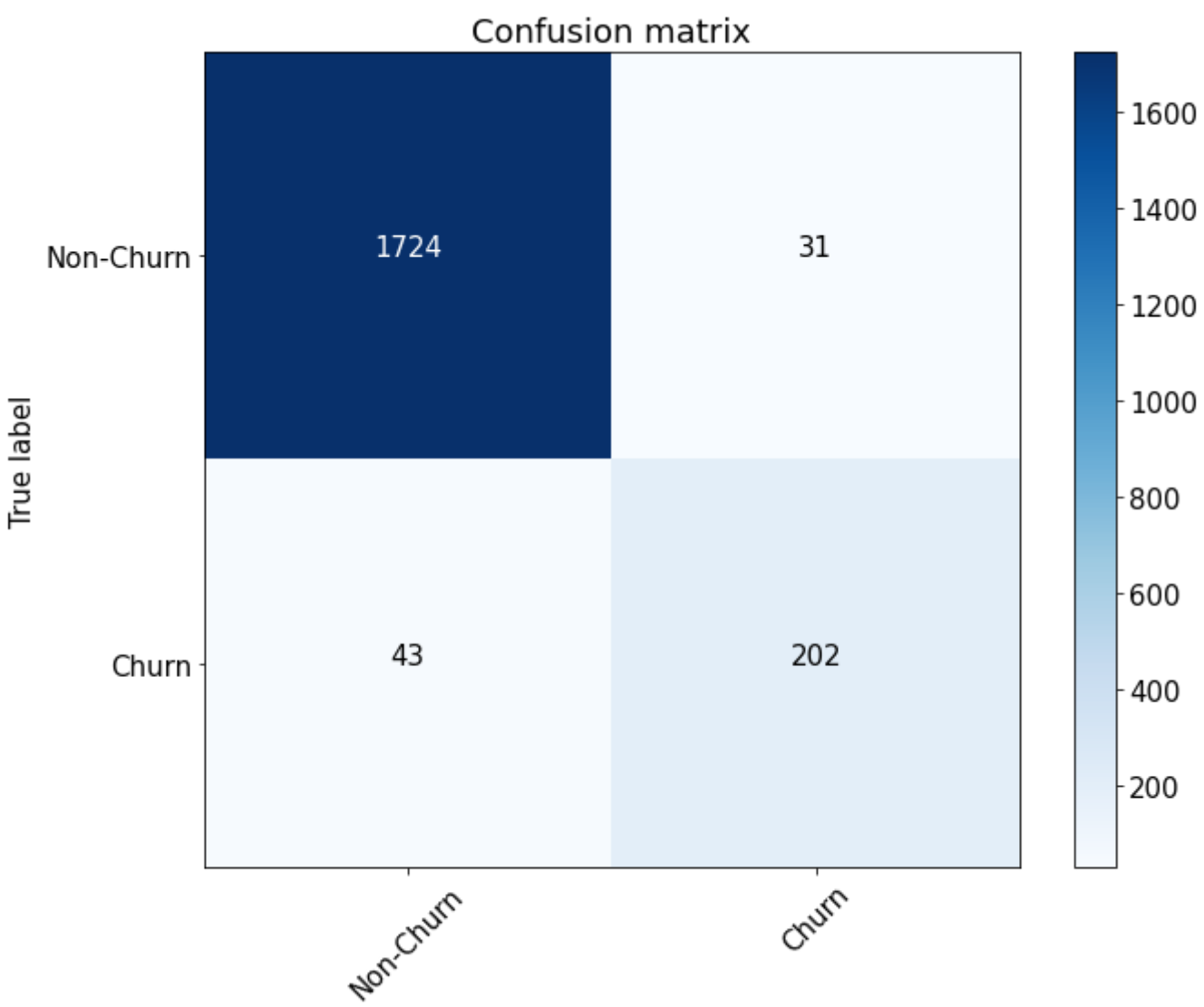
Меняем .mean() на .max().

In [40]:
```
def get_user_embedding_max(user_articles_list):
    user_articles_list = eval(user_articles_list)
    user_vector = np.array([doc_dict[doc_id] for doc_id in user_articles_list])
    # .mean() на .max()
    user_vector = np.max(user_vector, 0)
    return user_vector
```

Сохраняем метрики для user embeddings через .max().

In [42]:
```
metrics_max = train_embeddings_and_model(users, churn, get_user_embedding_max)
metrics_df = metrics_df.append(metrics_max, ignore_index=True)
```

```
F1 score: 0.8466257668711656
ROC-AUC score: 0.9843630443630443
```



In [43]:
```
metrics_df
```

Out[43]:

	precision	recall	f1_score	roc_auc
0	0.844444	0.775510	0.808511	0.974589
1	0.927126	0.934694	0.930894	0.994342
2	0.848361	0.844898	0.846626	0.984363

Задание 5*

Воспользовавшись полученными знаниями из п.1, повторить пункт 2, но уже взвешивая новости по tfidf.

(Подсказка 1 - нужно получить веса-коэффициенты для каждого документа. Не все документы одинаково информативны и несут какой-то положительный сигнал.)

(Подсказка 2 - нужен именно idf, как вес.)

Сформировать на выходе единую таблицу, сравнивающую качество 3 разных метода получения эмбедингов пользователей: mean, median, max, idf_mean по метрикам roc_auc, precision, recall, f_score. Сделать самостоятельные выводы и предположения о том, почему тот или ной

способ оказался эффективнее остальных.

Решение Задания 5*

Считаем tf-idf значения для каждого слова в каждом документе.

In [48]:
```
tfidf_model = TfidfModel(common_corpus)
common_corpus_tfidf = [tfidf_model[doc] for doc in common_corpus]
```

Тренируем LDA на tf-idf корпусе.

In [49]:
```
lda_tfidf = LdaModel(common_corpus_tfidf, num_topics=25, id2word=common_dictionary)#, passes=10)
```

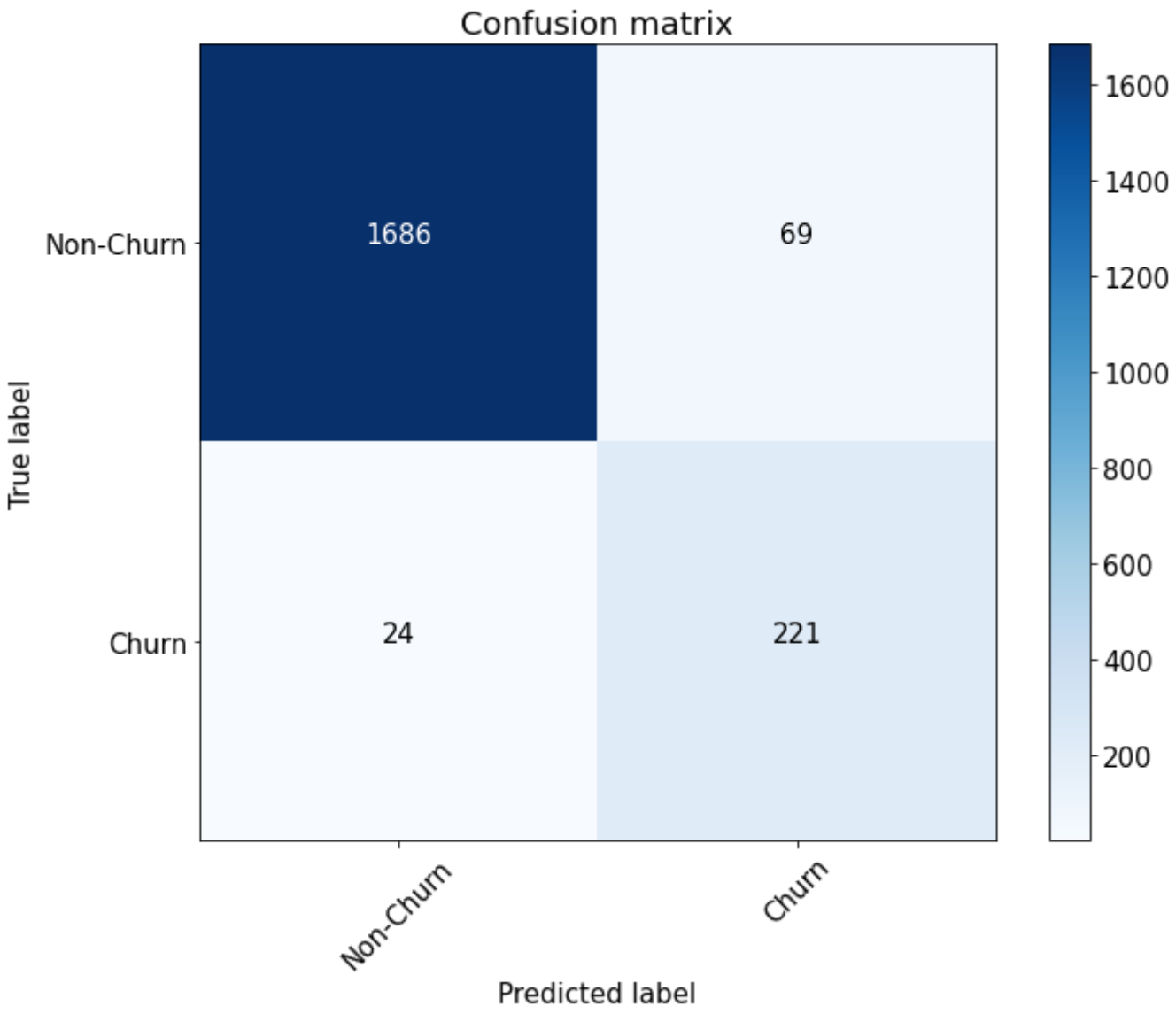
Получаем эмбеддинги документов по темам, с учетом подсчета для нового документа значений tf-idf.

In [50]:
```
def get_lda_vector_tfidf(text):
    unseen_doc = tfidf_model[common_dictionary.doc2bow(text)]
    lda_tuple = lda_tfidf[unseen_doc]
    not_null_topics = dict(zip([i[0] for i in lda_tuple], [i[1] for i in lda_tuple]))

    output_vector = []
    for i in range(25):
        if i not in not_null_topics:
            output_vector.append(0)
        else:
            output_vector.append(not_null_topics[i])
    return np.array(output_vector)
```

In [51]:
```
topic_matrix_tfidf = pd.DataFrame([get_lda_vector_tfidf(text) for text in news['title'].values])
topic_matrix_tfidf.columns = ['topic_{}'.format(i) for i in range(25)]
topic_matrix_tfidf['doc_id'] = news['doc_id'].values
topic_matrix_tfidf = topic_matrix_tfidf[['doc_id']+['topic_{}'.format(i) for i in range(25)]]
doc_dict = dict(zip(topic_matrix_tfidf['doc_id'].values, topic_matrix_tfidf[['topic_{}'.format(i) for i in range(25)]].values))
```

In [53]:
```
metrics_tfidf = train_embeddings_and_model(users, churn, get_user_embedding)
metrics_df = metrics_df.append(metrics_tfidf, ignore_index=True)
```

```
F1 score: 0.8268156424581007
ROC-AUC score: 0.9813826385254956
```



In [55]:
```
metrics_df.index = ['mean', 'median', 'max', 'idf_mean']
```

In [56]:
```
metrics_df
```

Out[56]:

	precision	recall	f1_score	roc_auc
mean	0.844444	0.775510	0.808511	0.974589
median	0.927126	0.934694	0.930894	0.994342

	precision	recall	f1_score	roc_auc
max	0.848361	0.844898	0.846626	0.984363
idf_mean	0.760274	0.906122	0.826816	0.981383

Можем заметить, что получение user embeddings по медиане сработало значительно лучше, чем по максимальному значению или среднему (которые показали себя достаточно похоже).

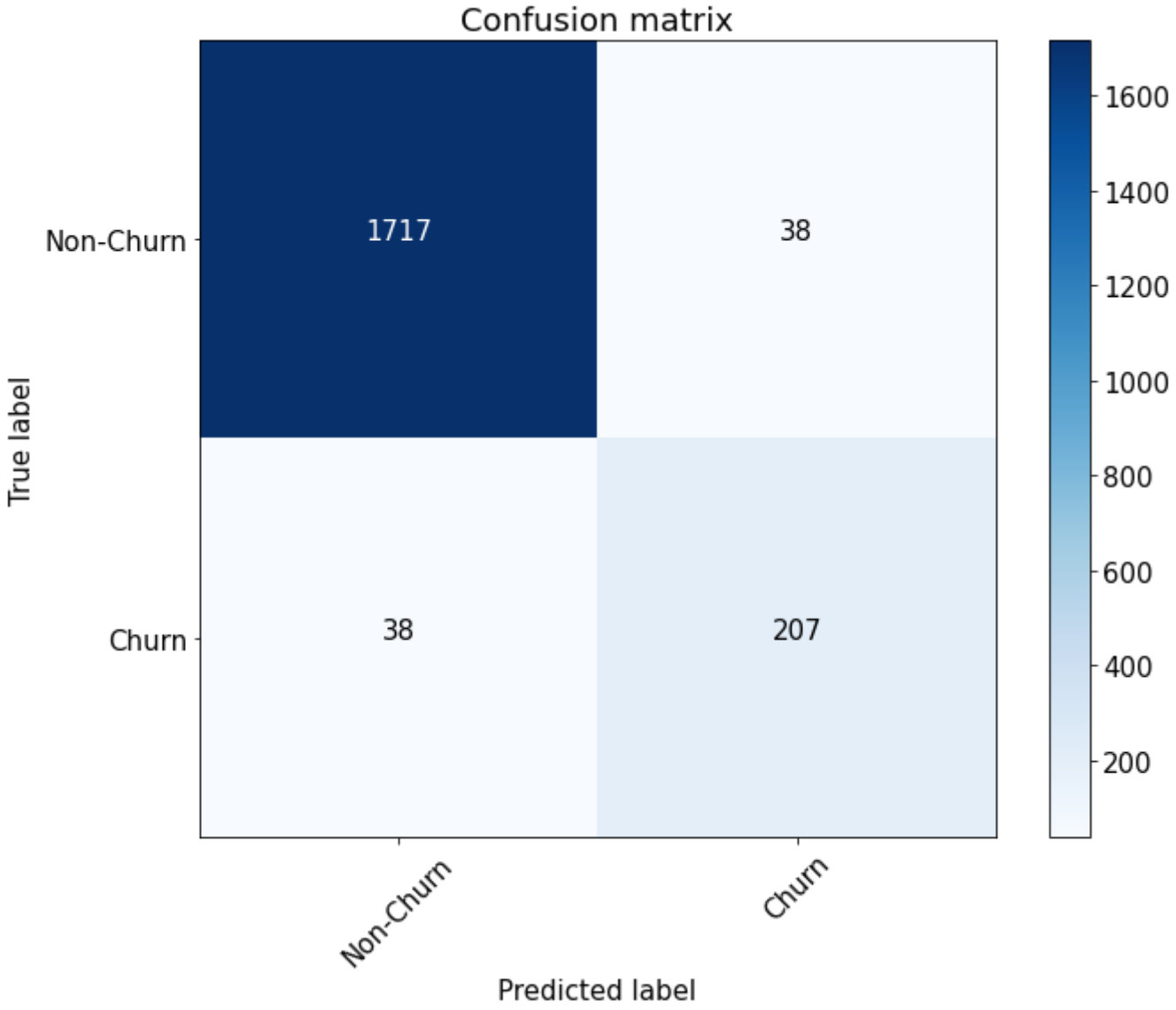
Получение embeddings по медиане работает лучше потому, что она более устойчива к выбросам. Например, пользователь мог случайно посмотреть всего 1 статью с сильной тематикок "политика", а остальные 20 статей по совершенно другим темам. В таком случае пользователю вряд ли будет интересна тематика "политика". Медиана отражает это лучше, чем среднее или максимальное значение по тематике.

Среднее по tf-idf сработало в некоторой степени противоположно bag-of-words среднему в плане precision и recall - оно начало реже делать ошибки False Negative, но чаще делать ошибки False Positive. В целом tf-idf помог улучшить значения F1-score и ROC-AUC, но они все еще хуже значений медианы и максимального значения для bag-of-words векторизации.

Найдем user embeddings по tf-idf, но с помощью медианы и максимального значения.

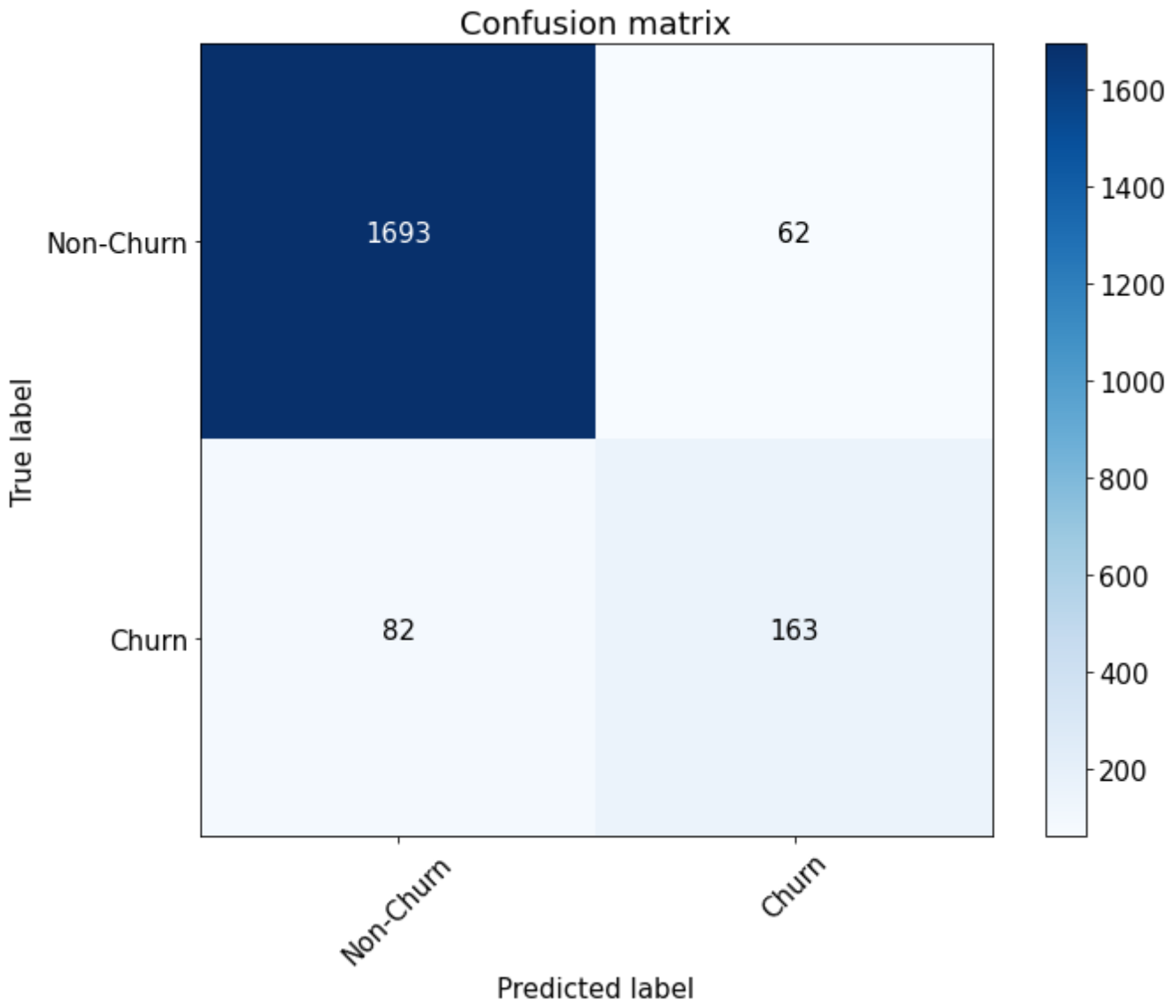
In [57]:
```
metrics_tfidf_median = train_embeddings_and_model(users, churn, get_user_embedding_median)
metrics_df = metrics_df.append(metrics_tfidf_median, ignore_index=True)
```

```
F1 score: 0.8531187122736418
ROC-AUC score: 0.985430548287691
```



In [60]:
```
metrics_tfidf_max = train_embeddings_and_model(users, churn, get_user_embedding_max)
metrics_df = metrics_df.append(metrics_tfidf_max, ignore_index=True)
```

```
F1 score: 0.6947368421052631
ROC-AUC score: 0.9353253096110239
```



In [61]:
```
metrics_df.index = ['mean', 'median', 'max', 'idf_mean', 'idf_median', 'idf_max']
```

In [62]:
```
metrics_df
```

Out[62]:

	precision	recall	f1_score	roc_auc
mean	0.844444	0.775510	0.808511	0.974589
median	0.927126	0.934694	0.930894	0.994342
max	0.848361	0.844898	0.846626	0.984363
idf_mean	0.760274	0.906122	0.826816	0.981383
idf_median	0.841270	0.865306	0.853119	0.985431
idf_max	0.717391	0.673469	0.694737	0.935325

Лучшим все так же остался метод нахождения user embeddings с помощью bag-of-words векторизации и медианы.

Метод с tf-idf векторизацией и медианой оказался вторым по F1-score и ROC-AUC. Возможно тексты в данном датасете слишком короткие, что ухудшает качество работы tf-idf.