### Рубежный контроль №2

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

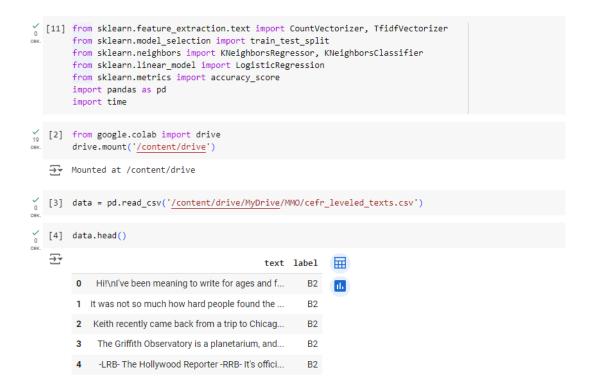
Необходимо сформировать два варианта векторизации признаков – на основе CountVectorizer и на основе TfidfVectorizer.

В качестве классификаторов необходимо использовать два классификатора по варианту группы: KNeighborsClassifieru LogisticRegression.

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

#### Ход выполнения работы

# Импорт библиотек и загрузка набора данных



#### Разделение выборки на обучающую и тестовую

```
[5] data.info()
   </pre
        RangeIndex: 1494 entries, 0 to 1493
        Data columns (total 2 columns):

# Column Non-Null Count Dtype
        0 text 1494 non-null
1 label 1494 non-null
                                       object
                                       object
        dtypes: object(2)
        memory usage: 23.5+ KB
\sqrt{\phantom{a}} [6] prop_mask = data.isna()
        props = prop_mask.sum()
        props

→ text

        label
                 0
        dtype: int64
[7] X, Y = data['text'], data['label']
        X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 42)
        time_arr = []
  [8] count_vect = CountVectorizer()
        {\tt X\_train\_counts = count\_vect.fit\_transform(X\_train)}
        X_{\text{test\_counts}} = \text{count\_vect.transform}(X_{\text{test}})
  [9] tfidf_vect = TfidfVectorizer()
        X_{train} fidf = tfidf_vect.fit_transform(X_{train})
        X_test_tfidf = tfidf_vect.transform(X_test)
```

# Обучение для CountVectorizer

```
√ [13] gbc = KNeighborsClassifier()
        start time = time.time()
        \mathsf{gbc.fit}(\mathsf{X\_train\_counts},\ \mathsf{y\_train})
        train_time = time.time() - start_time
        time_arr.append(train_time)
        pred_gbc_counts = gbc.predict(X_test_counts)
        print("Точность CountVectorizer и KNeighborsClassifier: ", accuracy_score(y_test, pred_gbc_counts))
   → Точность CountVectorizer и KNeighborsClassifier: 0.47491638795986624
   lr = LogisticRegression(max_iter = 1000)
        start_time = time.time()
        lr.fit(X\_train\_counts,\ y\_train)
        train_time = time.time() - start_time
        {\tt time\_arr.append(train\_time)}
        pred_lr_counts = lr.predict(X test counts)
       print("Точность CountVectorizer и Logistic Regression: ", accuracy_score(y_test, pred_lr_counts))

→ Точность CountVectorizer и Logistic Regression: 0.6053511705685619

14 [15] gbc = KNeighborsClassifier()
        start time = time.time()
        gbc.fit(X_train_tfidf, y_train)
        train_time = time.time() - start_time
        time_arr.append(train_time)
        pred_gbc_tfidf = gbc.predict(X_test_tfidf)
        print("Точность TfidfVectorizer и KNeighborsClassifier: ", accuracy_score(y_test, pred_gbc_tfidf))
   → Точность TfidfVectorizer и KNeighborsClassifier: 0.4882943143812709
5 [16] lr = LogisticRegression(max_iter = 1000)
        start_time = time.time()
        lr.fit(X\_train\_tfidf, y\_train)
        train_time = time.time() - start_time
        time arr.append(train time)
       pred_lr_tfidf = lr.predict(X_test_tfidf)
       print("Точность TfidfVectorizer и Logistic Regression: ", accuracy_score(y_test, pred_lr_tfidf))

→ Точность TfidfVectorizer и Logistic Regression: 0.5986622073578596
```