Отчет по рубежному контролю №2

```
Ф.И.О.: Козлов Егор

Группа: ИУ5-22М

Тема: Методы обработки текстов. Решение задачи классификации текстов.
```

Шаг 1: Подготовка среды и загрузка данных

Сначала установим необходимые библиотеки, загрузим датасет "Natural Language Processing with Disaster Tweets" с Kaggle и ознакомимся с данными.

```
# Установка Kaggle API
!pip install kaggle

# Импорт основных библиотек
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer,
TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Настройка для загрузки API ключа Kaggle
from google.colab import files
print('Пожалуйста, загрузите ваш файл kaggle.json')
files.upload()

!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

# Загрузка датасета
!kaggle competitions download -c nlp-getting-started

# Распаковка архивов
!unzip nlp-getting-started.zip
```

```
Downloading nlp-getting-started.zip to /content
   0% 0.00/593k [00:00<?, ?B/s]
 100% 593k/593k [00:00<00:00, 723MB/s]
Archive:  nlp-getting-started.zip
```

```
  inflating: sample_submission.csv
  inflating: test.csv
  inflating: train.csv
```

```
# Загрузка данных в DataFrame
df_train = pd.read_csv('train.csv')

# Вывод основной информации
print("Размер обучающего набора данных:", df_train.shape)
print("\nПервые 5 строк данных:")
print(df_train.head())

print("\nИнформация о данных:")
df_train.info()

print("\nРаспределение классов (0 - не катастрофа, 1 - катастрофа):")
print(df_train['target'].value_counts())
```

```
Размер обучающего набора данных: (7613, 5)

Первые 5 строк данных:
   id keyword location
text  \
0   1     NaN      NaN  Our Deeds are the Reason of this #earthquake
M...
1   4     NaN      NaN             Forest fire near La Ronge Sask.
Canada
2   5     NaN      NaN  All residents asked to 'shelter in place'
are ...
3   6     NaN      NaN  13,000 people receive #wildfires evacuation
or...
4   7     NaN      NaN  Just got sent this photo from Ruby #Alaska
as ...

   target
0       1
1       1
2       1
3       1
4       1

Информация о данных:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7613 entries, 0 to 7612
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        7613 non-null   int64
 1   keyword   7552 non-null   object
 2   location  5080 non-null   object
```

```
 3   text      7613 non-null   object
 4   target    7613 non-null   int64
dtypes: int64(2), object(3)
memory usage: 297.5+ KB

Распределение классов (0 - не катастрофа, 1 - катастрофа):
target
0    4342
1    3271
Name: count, dtype: int64
```

Шаг 2: Подготовка данных для обучения

Разделим данные на признаки (текст твита) и целевую переменную, а затем разобьем их на обучающую и тестовую выборки для корректной оценки моделей.

```
# Определение признаков (X) и целевой переменной (y)
X = df_train['text']
y = df_train['target']

# Разделение данных на обучающую и тестовую выборки (80% на обучение,
20% на тест)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

print(f"Размер обучающей выборки: {X_train.shape[0]} записей")
print(f"Размер тестовой выборки: {X_test.shape[0]} записей")

Размер обучающей выборки: 6090 записей
Размер тестовой выборки: 1523 записей
```

Шаг 3: Эксперимент с CountVectorizer

На этом шаге мы преобразуем текстовые данные в числовые векторы с помощью CountVectorizer и обучим на них два классификатора.

```
# Инициализация и настройка CountVectorizer
count_vectorizer = CountVectorizer(stop_words='english', max_df=0.9,
min_df=2)

# Обучение векторизатора на обучающих данных и преобразование выборок
X_train_count = count_vectorizer.fit_transform(X_train)
X_test_count = count_vectorizer.transform(X_test)

print(f"Размер словаря (количество уникальных токенов):
{len(count_vectorizer.get_feature_names_out())}")
print(f"Форма матрицы для обучающей выборки: {X_train_count.shape}")
```

```
Размер словаря (количество уникальных токенов): 5467
Форма матрицы для обучающей выборки: (6090, 5467)
```

```
# Инициализация и обучение модели
rf_clf_count = RandomForestClassifier(n_estimators=100,
random_state=42, n_jobs=-1)
rf_clf_count.fit(X_train_count, y_train)

# Предсказание на тестовой выборке
y_pred_rf_count = rf_clf_count.predict(X_test_count)

# Оценка качества
print("Результаты для RandomForestClassifier + CountVectorizer")
print(f"Accuracy: {accuracy_score(y_test, y_pred_rf_count):.4f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred_rf_count))
```

```
Результаты для RandomForestClassifier + CountVectorizer
Accuracy: 0.7879

Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.88      0.83       869
           1       0.81      0.67      0.73       654

    accuracy                           0.79      1523
   macro avg       0.79      0.77      0.78      1523
weighted avg       0.79      0.79      0.78      1523
```

```
# Инициализация и обучение модели
lr_clf_count = LogisticRegression(solver='liblinear', random_state=42)
lr_clf_count.fit(X_train_count, y_train)

# Предсказание на тестовой выборке
y_pred_lr_count = lr_clf_count.predict(X_test_count)

# Оценка качества
print("Результаты для LogisticRegression + CountVectorizer")
print(f"Accuracy: {accuracy_score(y_test, y_pred_lr_count):.4f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred_lr_count))
```

```
Результаты для LogisticRegression + CountVectorizer
Accuracy: 0.8076

Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.87      0.84       869
```

```
           1       0.81      0.73      0.76       654

    accuracy                           0.81      1523
   macro avg       0.81      0.80      0.80      1523
weighted avg       0.81      0.81      0.81      1523
```

Шаг 4: Эксперимент с TfidfVectorizer

Теперь повторим тот же процесс, но с использованием TfidfVectorizer, который учитывает не только частоту слов, но и их важность.

```
# Инициализация и настройка TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer(stop_words='english', max_df=0.9,
min_df=2)

# Обучение векторизатора на обучающих данных и преобразование выборок
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

print(f"Размер словаря (количество уникальных токенов):
{len(tfidf_vectorizer.get_feature_names_out())}")
print(f"Форма матрицы для обучающей выборки: {X_train_tfidf.shape}")
```

```
Размер словаря (количество уникальных токенов): 5467
Форма матрицы для обучающей выборки: (6090, 5467)
```

```
# Инициализация и обучение модели
rf_clf_tfidf = RandomForestClassifier(n_estimators=100,
random_state=42, n_jobs=-1)
rf_clf_tfidf.fit(X_train_tfidf, y_train)

# Предсказание на тестовой выборке
y_pred_rf_tfidf = rf_clf_tfidf.predict(X_test_tfidf)

# Оценка качества
print("Результаты для RandomForestClassifier + TfidfVectorizer")
print(f"Accuracy: {accuracy_score(y_test, y_pred_rf_tfidf):.4f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred_rf_tfidf))
```

```
Результаты для RandomForestClassifier + TfidfVectorizer
Accuracy: 0.7925

Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.89      0.83       869
           1       0.82      0.67      0.73       654
```

```
    accuracy                           0.79      1523
   macro avg       0.80      0.78      0.78      1523
weighted avg       0.80      0.79      0.79      1523
```

```
# Инициализация и обучение модели
lr_clf_tfidf = LogisticRegression(solver='liblinear', random_state=42)
lr_clf_tfidf.fit(X_train_tfidf, y_train)

# Предсказание на тестовой выборке
y_pred_lr_tfidf = lr_clf_tfidf.predict(X_test_tfidf)

# Оценка качества
print("Результаты для LogisticRegression + TfidfVectorizer")
print(f"Accuracy: {accuracy_score(y_test, y_pred_lr_tfidf):.4f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred_lr_tfidf))
```

```
Результаты для LogisticRegression + TfidfVectorizer
Accuracy: 0.8148

Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.90      0.85       869
           1       0.84      0.70      0.77       654

    accuracy                           0.81      1523
   macro avg       0.82      0.80      0.81      1523
weighted avg       0.82      0.81      0.81      1523
```

Шаг 5: Сводка результатов и выводы

Сведем все полученные метрики в одну таблицу для удобного сравнения. Будем сравнивать по Accuracy и F1-score (macro avg), так как F1-score является хорошей сбалансированной метрикой.

```
# Создание DataFrame с результатами
results_data = {
    'Vectorizer': ['CountVectorizer', 'CountVectorizer',
'TfidfVectorizer', 'TfidfVectorizer'],
    'Classifier': ['RandomForest', 'LogisticRegression',
'RandomForest', 'LogisticRegression'],
    'Accuracy': [
        accuracy_score(y_test, y_pred_rf_count),
        accuracy_score(y_test, y_pred_lr_count),
        accuracy_score(y_test, y_pred_rf_tfidf),
        accuracy_score(y_test, y_pred_lr_tfidf)
    ],
```

```
    'F1-score (macro)': [
        classification_report(y_test, y_pred_rf_count,
output_dict=True)['macro avg']['f1-score'],
        classification_report(y_test, y_pred_lr_count,
output_dict=True)['macro avg']['f1-score'],
        classification_report(y_test, y_pred_rf_tfidf,
output_dict=True)['macro avg']['f1-score'],
        classification_report(y_test, y_pred_lr_tfidf,
output_dict=True)['macro avg']['f1-score']
    ]
}

results_df = pd.DataFrame(results_data)
results_df = results_df.round(4) # Округляем для наглядности

print("Сводная таблица результатов:")
print(results_df)
```

```
Сводная таблица результатов:
        Vectorizer          Classifier  Accuracy  F1-score (macro)
0  CountVectorizer        RandomForest    0.7879            0.7775
1  CountVectorizer  LogisticRegression    0.8076            0.8009
2  TfidfVectorizer        RandomForest    0.7925            0.7818
3  TfidfVectorizer  LogisticRegression    0.8148            0.8063
```

Вывод:

На основе полученных результатов можно сделать следующие заключения:

```
Сравнение векторизаторов: В целом, TfidfVectorizer показал себя
немного лучше, чем CountVectorizer для обеих моделей. Это ожидаемо,
так как TF-IDF не просто считает слова, а взвешивает их по важности,
что часто дает прирост в качестве на задачах классификации текстов.

Сравнение классификаторов: Логистическая регрессия
(LogisticRegression) стабильно превзошла случайный лес
(RandomForestClassifier) на обоих типах векторов. Линейные модели,
такие как логистическая регрессия, часто показывают очень сильные
результаты на разреженных данных, которые получаются после
векторизации текстов.

Лучшая комбинация: TfidfVectorizer в паре с LogisticRegression
показала наилучшее качество классификации. Эта комбинация достигла
самой высокой точности (Accuracy = 0.8148) и лучшего значения F1-score
(F1-score (macro) = 0.8063). Это говорит о том, что данная модель
лучше всего обобщает данные и делает наиболее сбалансированные
предсказания для обоих классов.
```