**ОТЧЁТ ПО ПРИНЖ**

**Написан Ларином Антоном Сергеевичем**

**Группа 11-210**

**Глава 1 Генерация Текста**

**Описание кода**

Код предназначен для генерации текста с использованием предобученной языковой модели GPT-2 из библиотеки transformers. Модель берёт текстовую подсказку (prompt) и генерирует продолжение текста на её основе.

### ****Подробный анализ****

1. **Импорт библиотек**
   * GPT2LMHeadModel и GPT2Tokenizer импортируются из библиотеки transformers.  
     Эти классы используются для работы с моделью GPT-2 и её токенайзером.
2. **Инициализация модели и токенайзера**
   * Используется модель GPT-2 (model\_name = "gpt2"), которая загружается из предобученного хранилища с помощью метода from\_pretrained().
   * Токенайзер также инициализируется с помощью from\_pretrained().
3. **Установка pad\_token**
   * Так как GPT-2 изначально не поддерживает pad\_token, его необходимо задать вручную, чтобы обеспечить корректную обработку текста. В данном случае используется токен конца строки (eos\_token) в качестве заполнителя.
4. **Подготовка текстового ввода**
   * Подсказка "Science and technology of the future" преобразуется в тензор идентификаторов токенов с помощью tokenizer.encode().
   * Создаётся attention\_mask, которая маскирует пустые токены (pad\_token).
5. **Генерация текста**  
   Метод model.generate() используется для генерации продолжения текста с заданными параметрами:
   * max\_length=500: Ограничение на длину сгенерированного текста.
   * temperature=0.7: Параметр температуры для регулировки случайности при выборе следующего токена.
   * top\_p=0.9: Используется метод nucleus sampling для выбора токенов, чья совокупная вероятность превышает 90%.
   * do\_sample=True: Включение случайного семплирования токенов.
   * pad\_token\_id: Указание идентификатора заполнителя для корректной генерации.
6. **Декодирование текста**
   * Результат генерации (в формате токенов) декодируется обратно в текст с помощью tokenizer.decode().
   * Специальные токены (<pad>, <eos>) исключаются из декодированного текста (skip\_special\_tokens=True).
7. **Вывод результата**
   * Сгенерированный текст выводится в консоль.

**Ключевые особенности**

* **Универсальность**: Код работает с любой предобученной моделью GPT-2 благодаря использованию метода from\_pretrained().
* **Настраиваемость**: Гибкие параметры для генерации текста (например, длина, температура, топ-P).
* **Совместимость**: Учитываются ограничения модели GPT-2, такие как отсутствие pad\_token по умолчанию.

**Глава 2 Звук в текст**

### ****Описание****

Код выполняет задачу распознавания речи в реальном времени, записывая аудио с микрофона и преобразуя его в текст с помощью Google Web Speech API через библиотеку speech\_recognition.

### ****Анализ работы****

#### **1. Импорт библиотеки**

* **speech\_recognition** (sr): популярная библиотека Python для распознавания речи, поддерживающая различные API, включая Google Web Speech API.

#### **2. Инициализация распознавателя и микрофона**

recognizer = sr.Recognizer()

microphone = sr.Microphone()

* **Recognizer**: основной объект для управления процессом распознавания.
* **Microphone**: объект для взаимодействия с микрофоном устройства.

#### **3. Функция для распознавания речи**

def recognize\_speech\_from\_microphone():

* Основной функционал для записи и преобразования речи в текст.

#### **4. Пошаговая логика функции**

1. **Настройка микрофона**

recognizer.adjust\_for\_ambient\_noise(source)

* + Устанавливает порог шума для фильтрации фоновых звуков.
  + Рекомендуется говорить в тишине для повышения точности.

1. **Запись аудио**

python

Копировать код

audio\_data = recognizer.listen(source)

* + Захватывает аудио с микрофона и сохраняет в формате AudioData.

1. **Распознавание речи**

text = recognizer.recognize\_google(audio\_data, language="ru-RU")

* + Использует Google Web Speech API для преобразования аудио в текст.
  + Указан язык распознавания: **русский** (ru-RU).

1. **Обработка ошибок**
   * **UnknownValueError**: Если речь не может быть распознана.
   * **RequestError**: Если произошла ошибка соединения с API.

#### **5. Запуск функции**

recognize\_speech\_from\_microphone()

* Функция вызывается, когда код исполняется.

### ****Ключевые особенности****

1. **Лёгкость использования**  
   Библиотека speech\_recognition обеспечивает высокоуровневый интерфейс для распознавания речи.
2. **Реализация с микрофоном**  
   Код взаимодействует с микрофоном устройства, автоматически калибрует его и записывает звук.
3. **Google Web Speech API**  
   Используется облачная служба для точного распознавания речи.

**Глава 3 Тональность текста**

### ****Описание****

Код выполняет задачу анализа тональности текстов с использованием предварительно обученной модели **DistilBERT**, доступной через библиотеку **Hugging Face Transformers**. Модель определяет, является ли тональность текста **положительной** (POSITIVE) или **отрицательной** (NEGATIVE), и возвращает соответствующую метку с вероятностью.

### ****Анализ работы****

#### **1. Импорт библиотек**

from transformers import AutoTokenizer, AutoModelForSequenceClassification

import torch

* **transformers**: для загрузки модели и токенизатора.
* **torch**: для выполнения вычислений с тензорами (предсказание, вероятности).

#### **2. Загрузка модели и токенизатора**

MODEL\_NAME = "distilbert-base-uncased-finetuned-sst-2-english"

tokenizer = AutoTokenizer.from\_pretrained(MODEL\_NAME)

model = AutoModelForSequenceClassification.from\_pretrained(MODEL\_NAME)

* **MODEL\_NAME**: название модели. Используется модель, обученная для анализа тональности на основе корпуса **SST-2** (Stanford Sentiment Treebank).
* **AutoTokenizer**: преобразует текст в числовое представление (токены).
* **AutoModelForSequenceClassification**: модель, обученная на задаче классификации последовательностей.

#### **3. Функция для анализа тональности**

def analyze\_sentiment(texts):

* Вход: строка или список строк.
* Выход: список словарей с метками тональности (label) и вероятностями (score).

##### Основные шаги:

1. **Проверка и преобразование входных данных**:  
   Если на вход подана строка, она преобразуется в список.
2. **Токенизация текста**:

inputs = tokenizer(text, return\_tensors="pt", truncation=True, padding=True)

* + **truncation=True**: обрезает слишком длинные тексты.
  + **padding=True**: добавляет заполнение для выравнивания длины последовательностей.

1. **Прогноз модели**:

outputs = model(\*\*inputs)

probabilities = torch.nn.functional.softmax(outputs.logits, dim=-1)

* + Выход модели (logits) преобразуется в вероятности с помощью **Softmax**.

1. **Определение метки и вероятности**:

label = "POSITIVE" if torch.argmax(probabilities) == 1 else "NEGATIVE"

score = probabilities[0][torch.argmax(probabilities)].item()

* + Метка определяется по индексу с максимальной вероятностью.
  + Вероятность извлекается для выбранного класса.

1. **Формирование результата**:

results.append({

"text": text,

"label": label,

"score": round(score, 4)

})

* + Каждый результат сохраняется в виде словаря.

#### **4. Пример использования**

sample\_texts = [

"I absolutely love this product! It's amazing!",

"Milady, shall we go for a walk?",

"I will do anything to kill you"

]

* Три текста с разными оттенками эмоциональной окраски.

sentiments = analyze\_sentiment(sample\_texts)

for result in sentiments:

print(f"Text: {result['text']}")

print(f"Sentiment: {result['label']}")

print(f"Score: {result['score']}")

print("-" \* 40)

* Результаты выводятся с меткой тональности и вероятностью.

### ****Ключевые особенности****

1. **Использование DistilBERT**: компактная и производительная модель.
2. **Удобный интерфейс**: входная строка или список строк, результат в формате JSON.
3. **Простота использования**: легко адаптируется к другим моделям анализа тональности.

[PrEng/Анализ Тональности 2.py at main · WarponoidSEX/PrEng](https://github.com/WarponoidSEX/PrEng/blob/main/%D0%90%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7%20%D0%A2%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D0%B8%202.py)