```
# Импортируем все необходимые библиотеки

import pandas as pd
from transformers import AutoTokenizer, DistilBertModel,
DataCollatorWithPadding
from torch.utils.data import Dataset, DataLoader
import torch
from tqdm import tqdm
from sqlalchemy import create_engine
from dotenv import load_dotenv
import os
from sklearn.decomposition import TruncatedSVD
from sklearn.cluster import KMeans
import gc
import warnings

warnings.filterwarnings("ignore")

# Загружаем переменные окружения из .env файла
load_dotenv()

# Создаем URL для SQLAlchemy
SQLALCHEMY_DATABASE_URL = (
    f"postgresql://"
    f"{os.getenv('POSTGRES_USER')}:{os.getenv('POSTGRES_PASSWORD')}@"
    f"{os.getenv('POSTGRES_HOST')}:{os.getenv('POSTGRES_PORT')}/"
    f"{os.getenv('POSTGRES_DATABASE')}"
)
engine = create_engine(SQLALCHEMY_DATABASE_URL)
connection = engine.connect().execution_options(stream_results=True)
```

Выгрузим таблицу с постами из базы данных и посторим на основе их эмбеддинги

1. Таблица `post_text_df`

```
# Структура таблицы:
posts_info = {
    "post_id": "Уникальный ID поста (int64)",
    "text": "Текст поста (str)",
    "topic": "Категория (str: 'politics', 'entertainment', ...)",
}

# Посты и топики

posts_info = pd.read_sql("""SELECT * FROM post_text_df""",
con=connection)

posts_info.head()
```

```
   post_id                                               text
topic
0        1  UK economy facing major risks\n\nThe UK manufa...
business
1        2  Aids and climate top Davos agenda\n\nClimate c...
business
2        3  Asian quake hits European shares\n\nShares in ...
business
3        4  India power shares jump on debut\n\nShares in ...
business
4        5  Lacroix label bought by US firm\n\nLuxury good...
business
```

Advanced Feature Engineering для таблицы `post_text_df`

План генерации признаков

Построение эмбеддингов постов с использованием DistilBERT, TruncatedSVD и KMeans

Для анализа и кластеризации текстовых данных постов мы реализуем многоступенчатый подход, который объединяет современные методы обработки естественного языка (NLP) и снижения размерности:

1. **Генерация эмбеддингов с помощью DistilBERT**
   Мы используем предобученную модель **DistilBERT**, которая является легковесной версией BERT. Она преобразует текстовые данные в семантически значимые векторные представления (эмбеддинги), сохраняя контекст и смысл каждого поста. Это ключевой этап, так как DistilBERT эффективно учитывает сложные зависимости в тексте.

2. **Снижение размерности с помощью TruncatedSVD**
   Эмбеддинги, полученные из DistilBERT, имеют высокую размерность (обычно 768 или более). Для упрощения дальнейшего анализа и повышения производительности мы применяем метод **TruncatedSVD** (сингулярное разложение с усечением). Этот метод позволяет сократить размерность данных до управляемого уровня, сохраняя при этом основные паттерны и структуру.

3. **Кластеризация с помощью KMeans**
   После снижения размерности мы используем алгоритм **KMeans** для группировки постов на основе их эмбеддингов. Кластеризация помогает выявить скрытые категории или темы в данных, что может быть полезно для рекомендательных систем, анализа трендов или категоризации контента.

Преимущества подхода

- **Семантическая значимость**: DistilBERT обеспечивает глубокое понимание текста, учитывая контекст и смысл.
- **Эффективность**: TruncatedSVD снижает размерность данных, что делает процесс кластеризации быстрее и менее ресурсоемким.
- **Интерпретируемость**: KMeans позволяет разделить посты на четко определенные группы, которые можно интерпретировать и использовать для практических задач.

Ожидаемые результаты

В результате мы получим:

- **Кластеры постов**, где каждый кластер соответствует определенной теме или категории.
- Возможность **автоматически классифицировать новые посты** на основе их близости к существующим кластерам.
- Более глубокое понимание структуры данных, что может быть использовано для улучшения качества рекомендаций или анализа пользовательских интересов.

```
model_name = "distilbert-base-cased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = DistilBertModel.from_pretrained(model_name)
```


```
{"model_id":"aa2bf181ad60443d939bed32b605cc14","version_major":2,"vers
ion_minor":0}

{"model_id":"3a6fd7a887a347288b537e3af7b00546","version_major":2,"vers
ion_minor":0}

{"model_id":"0fb52014255c4f5fb3c27bf7ccc2cc6e","version_major":2,"vers
ion_minor":0}

{"model_id":"4aaa48040e32457fa38487d6048fa673","version_major":2,"vers
ion_minor":0}

{"model_id":"60cbb0c7262d4f9a8187b52a4083f307","version_major":2,"vers
ion_minor":0}
```


```
# Создадим датасет для постов


class PostDataset(Dataset):
    def __init__(self, texts, tokenizer):
        super().__init__()

        self.texts = tokenizer.batch_encode_plus(
```

```
            texts,
            add_special_tokens=True,
            return_token_type_ids=False,
            return_tensors="pt",
            truncation=True,
            padding=True,
        )
        self.tokenizer = tokenizer

    def __getitem__(self, idx):
        return {
            "input_ids": self.texts["input_ids"][idx],
            "attention_mask": self.texts["attention_mask"][idx],
        }

    def __len__(self):
        return len(self.texts["input_ids"])


dataset = PostDataset(posts_info["text"].values.tolist(), tokenizer)
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
loader = DataLoader(
    dataset, batch_size=32, collate_fn=data_collator, pin_memory=True,
shuffle=False
)

# Переключимся на GPU

device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")
model = model.to(device)
print(device)
print(torch.cuda.get_device_name())
```

```
cuda:0
Tesla P100-PCIE-16GB
```

```
# Функция для создания эмбеддингов


@torch.inference_mode()
def get_embeddings_labels(model, loader):
    model.eval()

    total_embeddings = []

    for batch in tqdm(loader):
        batch = {key: batch[key].to(device) for key in
["attention_mask", "input_ids"]}

        embeddings = model(**batch)["last_hidden_state"][:, 0, :]
```

```
        total_embeddings.append(embeddings.cpu())

    return torch.cat(total_embeddings, dim=0)

embeddings = get_embeddings_labels(model, loader).numpy()
```

```
100%|██████████| 220/220 [00:54<00:00,  4.01it/s]
```

```
# Для каждого поста получили эмбеддинг, размерностью 768

embeddings.shape
```

```
(7023, 768)
```

```
# Снизим размерность эмбеддингов
centered = embeddings - embeddings.mean()

# TruncatedSVD
svd = TruncatedSVD(n_components=100, random_state=42)
svd_decomp = svd.fit_transform(centered)

### Кластеризация по семантике
kmeans = KMeans(n_clusters=15, random_state=42)
cluster = kmeans.fit_predict(svd_decomp).reshape(-1, 1)
posts_info["TextCluster"] = cluster

# Генерация названий столбцов динамически
num_clusters = 15  # Количество кластеров
dists_columns = [
    f"DistanceTo{cluster}thCluster" for cluster in range(1,
num_clusters + 1)
]

# Создание DataFrame с автоматически сгенерированными названиями
столбцов
dists_df = pd.DataFrame(data=kmeans.transform(svd_decomp),
columns=dists_columns)

# Проверка первых строк
dists_df.head()
```

	DistanceTo1thCluster	DistanceTo2thCluster	DistanceTo3thCluster	\
0	3.803364	3.641078	3.503630	
1	3.610889	3.381778	3.169918	
2	3.647825	3.428573	3.177275	
3	3.133031	3.879746	3.885824	
4	3.528582	3.251029	3.060982	

	DistanceTo4thCluster	DistanceTo5thCluster	DistanceTo6thCluster	\
0	3.033075	2.276970	3.463066	
1	2.885389	2.292614	3.320669	

```
2               3.095651               3.101687               3.435107
3               3.391752               3.506583               3.885024
4               2.880629               3.147450               3.021023

   DistanceTo7thCluster  DistanceTo8thCluster  DistanceTo9thCluster  \
0               3.508869              3.406232              3.463735
1               3.285646              3.387437              3.375190
2               3.458061              3.546807              3.416218
3               4.156692              3.799252              3.836481
4               3.433394              3.007250              3.059660

   DistanceTo10thCluster  DistanceTo11thCluster  DistanceTo12thCluster
\
0               3.413809               2.417847               1.983550

1               3.364406               2.386165               2.246995

2               3.330880               2.493758               1.920925

3               3.627957               2.957552               2.575662

4               3.250440               2.337950               1.870461


   DistanceTo13thCluster  DistanceTo14thCluster  DistanceTo15thCluster

0               1.912621               3.475528               2.879717

1               1.478820               3.019533               2.602993

2               1.855948               3.039433               2.946122

3               2.617805               3.820382               3.488249

4               2.439339               2.901595               2.430091


posts_info = pd.concat((posts_info, dists_df), axis=1)
posts_info.head()

   post_id                                               text
topic  \
0        1  UK economy facing major risks\n\nThe UK manufa...
business
1        2  Aids and climate top Davos agenda\n\nClimate c...
business
2        3  Asian quake hits European shares\n\nShares in ...
business
3        4  India power shares jump on debut\n\nShares in ...
business
4        5  Lacroix label bought by US firm\n\nLuxury good...
```

```
business

   TextCluster  DistanceTo1thCluster  DistanceTo2thCluster  \
0           12              3.803364              3.641078
1           12              3.610889              3.381778
2           12              3.647825              3.428573
3           11              3.133031              3.879746
4           11              3.528582              3.251029

   DistanceTo3thCluster  DistanceTo4thCluster  DistanceTo5thCluster  \
0              3.503630              3.033075              2.276970
1              3.169918              2.885389              2.292614
2              3.177275              3.095651              3.101687
3              3.885824              3.391752              3.506583
4              3.060982              2.880629              3.147450

   DistanceTo6thCluster  DistanceTo7thCluster  DistanceTo8thCluster  \
0              3.463066              3.508869              3.406232
1              3.320669              3.285646              3.387437
2              3.435107              3.458061              3.546807
3              3.885024              4.156692              3.799252
4              3.021023              3.433394              3.007250

   DistanceTo9thCluster  DistanceTo10thCluster  DistanceTo11thCluster
\
0              3.463735               3.413809               2.417847

1              3.375190               3.364406               2.386165

2              3.416218               3.330880               2.493758

3              3.836481               3.627957               2.957552

4              3.059660               3.250440               2.337950


   DistanceTo12thCluster  DistanceTo13thCluster  DistanceTo14thCluster
\
0               1.983550               1.912621               3.475528

1               2.246995               1.478820               3.019533

2               1.920925               1.855948               3.039433

3               2.575662               2.617805               3.820382

4               1.870461               2.439339               2.901595


   DistanceTo15thCluster
0               2.879717
```

```
1                2.602993
2                2.946122
3                3.488249
4                2.430091
```

```
# Загружаем таблица со всеми признаками постов в базу данных

posts_info.to_sql(
    "posts_info_deep_features_ruslan_prashchurovich",
    con=engine,
    index=False,
    if_exists="replace",
)
```

```
23
```

```
# Функция для пакетной загрузки данных из SQL
def batch_load_sql(query: str, engine) -> pd.DataFrame:
    CHUNKSIZE = 200000
    conn = engine.connect().execution_options(stream_results=True)
    chunks = []
    for chunk_dataframe in pd.read_sql(query, conn,
chunksize=CHUNKSIZE):
        chunks.append(chunk_dataframe)
    conn.close()
    return pd.concat(chunks, ignore_index=True)


# Функция для загрузки признаков из базы данных
def load_posts_features(engine) -> pd.DataFrame:
    query = "SELECT * FROM
posts_info_deep_features_ruslan_prashchurovich"
    return batch_load_sql(query, engine)


# Проверка загрузки данных
post_features = load_posts_features(engine)
post_features.head()
```

```
   post_id                                               text
topic  \
0        1  UK economy facing major risks\n\nThe UK manufa...
business
1        2  Aids and climate top Davos agenda\n\nClimate c...
business
2        3  Asian quake hits European shares\n\nShares in ...
business
3        4  India power shares jump on debut\n\nShares in ...
business
4        5  Lacroix label bought by US firm\n\nLuxury good...
business
```

```
   TextCluster  DistanceTo1thCluster  DistanceTo2thCluster  \
0           12              3.803364              3.641078
1           12              3.610889              3.381778
2           12              3.647826              3.428573
3           11              3.133031              3.879746
4           11              3.528582              3.251030

   DistanceTo3thCluster  DistanceTo4thCluster  DistanceTo5thCluster  \
0              3.503630              3.033075              2.276970
1              3.169918              2.885389              2.292614
2              3.177275              3.095651              3.101687
3              3.885824              3.391752              3.506583
4              3.060982              2.880629              3.147450

   DistanceTo6thCluster  DistanceTo7thCluster  DistanceTo8thCluster  \
0              3.463066              3.508869              3.406232
1              3.320669              3.285646              3.387437
2              3.435107              3.458061              3.546807
3              3.885024              4.156692              3.799252
4              3.021023              3.433394              3.007250

   DistanceTo9thCluster  DistanceTo10thCluster  DistanceTo11thCluster
\
0              3.463735               3.413809               2.417847

1              3.375190               3.364406               2.386165

2              3.416218               3.330880               2.493758

3              3.836481               3.627957               2.957552

4              3.059660               3.250440               2.337950


   DistanceTo12thCluster  DistanceTo13thCluster  DistanceTo14thCluster
\
0               1.983550               1.912621               3.475528

1               2.246995               1.478820               3.019533

2               1.920925               1.855948               3.039433

3               2.575662               2.617805               3.820382

4               1.870461               2.439339               2.901595


   DistanceTo15thCluster
0               2.879717
1               2.602994
```

```
2                2.946122
3                3.488249
4                2.430091
```

```
# Почистим переменные

model.cpu()

del model
del tokenizer

del dataset
del loader

del embeddings
del centered
del svd
del svd_decomp

gc.collect()
```

```
8
```