Федеральное государственное автономное
образовательное учреждение высшего образования
«Пермский государственный национальный исследовательский
университет» (ПГНИУ)
Региональный институт непрерывного образования (РИНО
ПГНИУ)
Цифровая кафедра

Выпускная аттестационная (квалификационная) работа
по курсу профессиональной переподготовки «Анализ данных»

Анализ тональности финансовых новостей с использованием трансформеров и категорной композиционной модели

Разработчики проекта:
Вихляев Егор Сергеевич,
Матыгуллин Булат Фаридович,
Остроумов Фрол Романович,
Устюгов Владислав Юрьевич.

Пермь, 2023

Оглавление

Паспорт проекта

Название проекта: Анализ тональности финансовых новостей с использованием трансформера BERT, категорной модели DisCoCat и градиентного бустинга.

Сведения об авторах: Вихляев Егор Сергеевич, Матыгуллин Булат Фаридович, Остроумов Фрол Романович, Устюгов Владислав Юрьевич.

Цель: построить ансамбль моделей BERT и DisCoCat с последующей конкатенацией их результатов для подачи в мета-модель градиентного бустинга.

Задачи:

1. Выполнить анализ проблемы, обосновать ее актуальность.
2. Осуществить загрузку данных и исследовательский анализ данных.
3. Осуществить предпоготовку данных для каждой из моделей.
4. Построить каждую из моделей (BERT, DisCoCat) по отдельности.
5. Произвести ансамблирование моделей, сконкатенировав их выходы для дальнейшей отправки в мета-модель градиентного бустинга.
6. Выполнить интерпретацию полученных результатов и сделать выводы.

Краткое описание проекта: Требуется произвести классификацию финансовых новостей по их тональности. Дать интерпретацию полученным результатам.

Ожидаемые результаты (гипотеза): Построенный ансамбль должен скомпенсировать обоюдные недостатки моделей, аппроксимировав итоговый результат с помощью новой архитектуры нейронных сетей и предоставив метрики лучшие, чем каждая модель по отдельности.

Содержание проекта

1.1 Анализ проблемы исследования

Компьтеры умеют понимать смысл слов, но не умеют достаточно хорошо понимать смысл предложений и более длинных текстов.

В 2010 году вышла статья [5], в которой авторы в значительной степени опираются на принцип композиционности — идею о том, что смысл предложения может быть определен смыслами отдельных слов вместе с грамматическими правилами их сочетания. То есть, если компьютер может понимать смысл отдельных слов и грамматические правила, то единственное, в чем ему нужно помочь, так это в умении объединять их в осмысленное целое. И это то место, где в силу вступает теория категорий!

Авторы, используя функториальную семантику, моделируют естественный язык как (моноидальный) функтор между компактными замкнутыми категориями:

$$\text{синтаксис} \to \text{семантика},$$

или проще:

$$\text{грамматика} \to \text{смысл слов}.$$

Этот функтор присваивает слову тип грамматики, а моноидальные структуры дают возможность объединить значения этих слов в предложение, смысл которого определен через принцип композиционности.

1.2 Исходные данные

В настоящей работе анализируется список заголовков финансовых новостей и их тональность, где 0 - негативная, 1 - нейтральная, 2 - позитивная.

Список колонок анализируемого набора данных:

- **label** - тональность заголовка.

- **text** - текст заголовка.

Необходимо провести анализ тональности текста. Эта задача сводится к трехклассовой классификации текста.

Выдвинем гипотезу исследования: построенный ансамбль должен скомпенсировать обоюдные недостатки моделей, аппроксимировав итоговый результат с помощью новой архитектуры нейронных сетей и предоставив метрики лучшие, чем каждая модель по отдельности.

1.3 Реализация проекта

1.3.1 Подготовка данных

На первом этапе была выполнена загрузка данных для анализа и обучения модели. Основные шаги:

1. **Загрузка данных**: Использовался метод `read_csv` библиотеки `pandas` для чтения файла с исходными данными. Данные представляли собой таблицу, где каждая строка — это текстовая запись с соответствующей меткой класса.

```
df = pd.read_csv('../data/raw/low_complexity.csv')
```

2. **Проверка и удаление дубликатов**: Для выявления повторяющихся строк был использован метод `duplicated`. Найденные дубликаты были удалены с помощью `drop_duplicates`, что позволило снизить избыточность данных и улучшить качество обучения.

```
duplicates = df.duplicated().sum()
df = df.drop_duplicates()
```

3. **Обработка пропущенных значений**: Анализ отсутствующих данных (`isnull().sum()`) помог определеить, что в датасете отсутствовали пустые строки.

```
missing_values = df.isnull().sum()
```

4. **Визуализация распределения классов**: Для оценки дисбаланса классов построен график распределения меток с помощью `seaborn.countplot`. Это помогло понять, какие классы представлены в данных и в каких пропорциях (см. Рис 1.1).

```
sns.countplot(x='label', data=df)
plt.title('Class distribution')
plt.show()
```

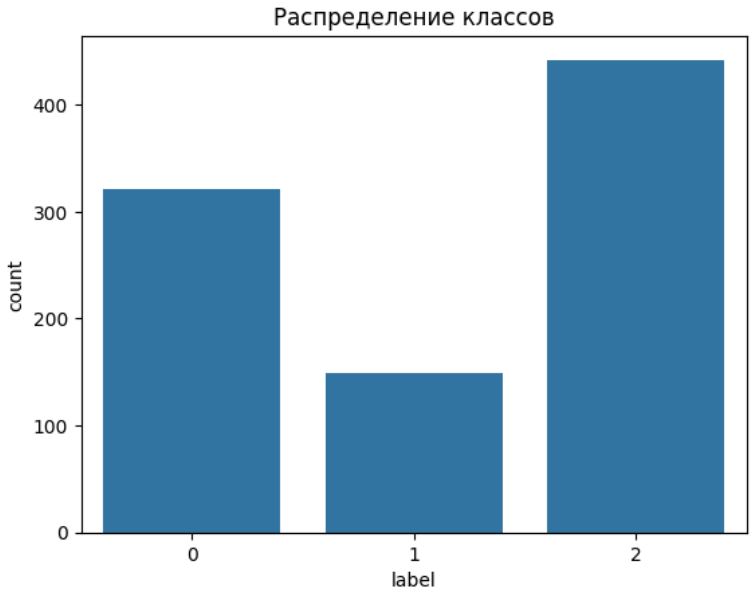

Рис. 1.1: Проверка балансировки классов

5. **Балансировка классов**: Проблема дисбаланса решалась добавлением недостающих данных меньшинственного класса с использованием метода `sample` и объединением с исходным датасетом (см. Рис 1.2).

```
class_counts = df['label'].value_counts()
min_class = class_counts.idxmin()
max_class = class_counts.idxmax()
difference = class_counts[max_class]
- class_counts[min_class]

min_class_samples = df[df['label'] ==
min_class].sample(difference, replace=True)
df = pd.concat([df, min_class_samples])

sns.countplot(x='label', data=df)
plt.show()
```

1.3.2 Предобработка текста для BERT

Для подготовки текстовых данных к обучению модели выполнены следующие шаги:

1. **Токенизация текста**: Использован предобученный токенайзер модели BERT (`BertTokenizer`), который преобразует текст в последо-

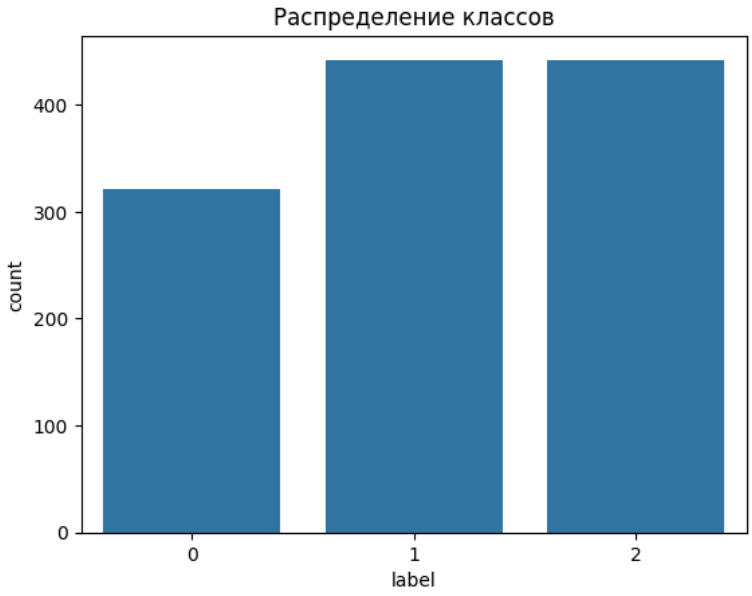

Рис. 1.2: Классы после балансировки

вательность токенов. Этот процесс включает разбиение текста на отдельные слова или фразы, а также их преобразование в числовое представление.

```
tokenizer = BertTokenizer.from_pretrained
('bert-base-uncased')

train_encodings = tokenizer
(list(X_train), truncation=True,
padding=True, return_tensors='pt')
val_encodings = tokenizer
(list(X_val), truncation=True,
padding=True, return_tensors='pt')
```

2. **Создание кастомного датасета**: Определен класс `SentimentDataset` для преобразования токенизированных данных и меток в формат, пригодный для использования в тренировочном процессе.

```
class SentimentDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key:torch.tensor(val[idx],
```

```
            dtype=torch.long)
                    for key, val in self.encodings.items()}
            item['labels'] = torch.tensor(self.labels[idx],
            dtype=torch.long)
            return item

        def __len__(self):
            return len(self.labels)

    train_dataset = SentimentDataset
    (train_encodings, y_train.tolist())
    val_dataset = SentimentDataset
    (val_encodings, y_val.tolist())
```

1.3.3 Обучение модели BERT

BERT (Bidirectional Encoder Representations from Transformers) — это модель глубокого обучения, основанная на архитектуре трансформеров. Она предназначена для обработки текста с учётом контекста как слева, так и справа от каждого слова, что делает её особенно эффективной для задач обработки естественного языка (NLP).

Основные компоненты работы BERT

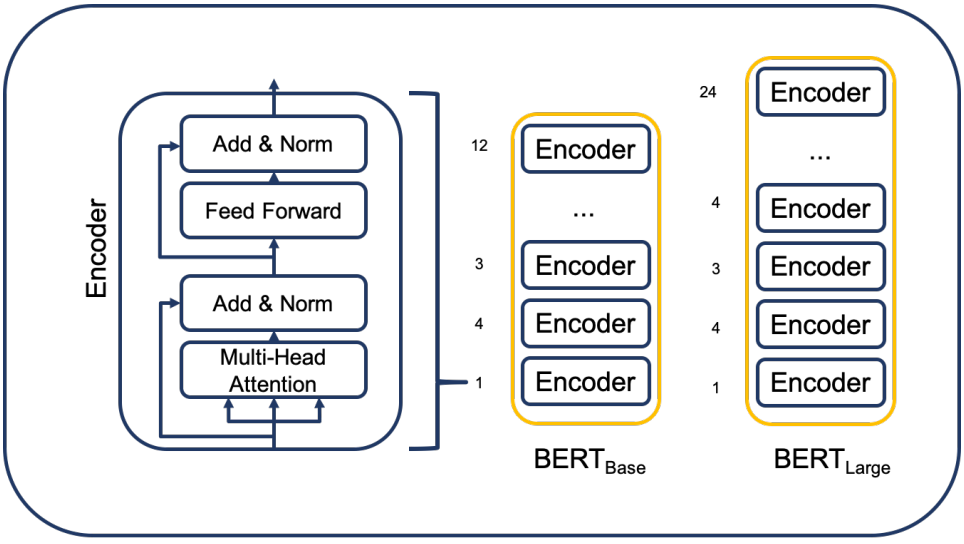

Рис. 1.3: Архитектура BERT

1. **Архитектура трансформеров**

- BERT использует энкодеры из трансформеров. Энкодер состоит из слоев самовнимания и полносвязных слоев.
- Самовнимание (Self-Attention) позволяет модели анализировать взаимосвязи между всеми словами в предложении, что помогает учитывать контекст.

2. **Двунаправленность**

   - В отличие от многих других моделей (например, GPT), которые обрабатывают текст слева направо или справа налево, BERT анализирует текст двунаправленно.
   - Это позволяет модели учитывать контекст не только перед словом, но и после него, что критично для понимания смысла предложения.

3. **Предобучение**

   - BERT обучается в два этапа:
     (a) **Masked Language Modeling (MLM)**:
       – На этапе обучения случайные слова в предложении заменяются маской (`[MASK]`).
       – Задача модели — предсказать замаскированные слова, используя контекст.
     (b) **Next Sentence Prediction (NSP)**:
       – Модель обучается определять, является ли второе предложение логическим продолжением первого.
       – Это полезно для задач, где важна связь между предложениями, например, в диалогах или понимании текста.

4. **Токенизация**

   - BERT использует токенайзер WordPiece, который разбивает текст на подслова (subwords).
   - Это позволяет эффективно обрабатывать редкие или новые слова, разделяя их на знакомые части.

5. **Использование эмбеддингов**

   - Для каждого токена модель создает эмбеддинг, который включает:
     – **Эмбеддинг слова** (Word Embedding).
     – **Эмбеддинг позиции** (Position Embedding), чтобы учитывать порядок слов.

– **Сегментный эмбеддинг** (Segment Embedding) для разделения предложений в задачах NSP.

6. **Выход модели**

   - На выходе модель формирует эмбеддинги для каждого токена и специального токена `[CLS]`.
   - Эмбеддинг `[CLS]` используется для задач классификации текста, таких как определение тональности или тематического анализа.

Применение BERT

- **Классификация текста**: Определение темы, тональности или типа текста.
- **Извлечение сущностей**: Нахождение имен собственных, дат и других сущностей в тексте.
- **Вопрос-ответ**: Модель может находить ответ на заданный вопрос в тексте.
- **Перевод текста**: Хотя BERT не специализирован для перевода, его можно использовать как часть более сложных систем.

BERT стал основой для многих более сложных моделей, таких как RoBERTa, ALBERT и DistilBERT, которые оптимизируют его производительность для различных задач и ресурсов.

Основная часть работы связана с настройкой и обучением модели на подготовленных данных:

1. **Загрузка предобученной модели BERT**: Предобученния модель `bert-base-uncased` была настроена для выполнения задачи классификации. Параметр `num_labels` указывает на количество классов.

```
model_bert = BertForSequenceClassification.
from_pretrained('bert-base-uncased', num_labels=3)
```

2. **Определение параметров обучения**: Использовался класс `TrainingArguments` для задания гиперпараметров, таких как число эпох, размер батча, скорость обучения и директория для сохранения результатов.

```
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=8,
```

```
    per_device_eval_batch_size=8,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
)
```

3. **Запуск обучения**: Для обучения использовался класс `Trainer`, обеспечивающий автоматизацию процесса тренировки, валидации и логирования. После завершения обучения модель сохранялась для последующего использования.

```
trainer_bert = Trainer(
    model=model_bert,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
)

trainer_bert.train()
```

1.3.4 Оценка модели BERT

Для оценки качества модели выполнены следующие действия:

1. **Получение предсказаний**: С помощью метода `predict` класса `Trainer` предсказаны метки на валидационном наборе данных.

```
val_predictions = trainer_bert.predict
(val_dataset)
val_preds = np.argmax
(val_predictions.predictions, axis=1)
```

2. **Вычисление метрик**: Использован отчет о классификации (`classification_report`) из библиотеки `sklearn`, чтобы рассчитать такие метрики, как точность (accuracy), полнота (recall), F1-метрика и другие (см. Рис 1.4).

```
print("Accuracy:", accuracy_score(y_val, val_preds))
print(classification_report(y_val, val_preds))
```

3. **Визуализация результатов**: Построены графики метрик и матрица ошибок с использованием `matplotlib` и `seaborn`. Это позволило выявить области, где модель допускает ошибки, и оценить её производительность.

   Модель показала значение `Accuracy = 0.87`.

Метрики на валидационном датасете:
Accuracy: 0.8713692946058091

	precision	recall	f1-score	support
0	0.92	0.81	0.86	69
1	0.84	0.89	0.86	93
2	0.88	0.90	0.89	79
accuracy			0.87	241
macro avg	0.88	0.87	0.87	241
weighted avg	0.87	0.87	0.87	241

Рис. 1.4: Classification report

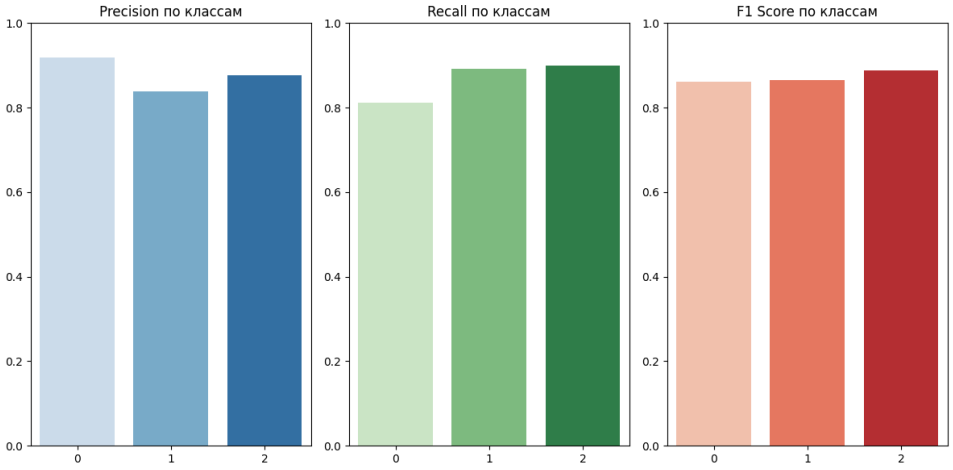

Рис. 1.5: Precision, Recall и F1 Score по классам

1.3.5 Предобработка текста для DisCoCat

Для подготовки текстовых данных к обучению модели DisCoCat нужно было применить иной подход. Данные приводятся в вид с двумя столбцами: `label` и `text`. Затем компилируются в формат `.txt`, где разделитель между `label` и `text` является просто пробелом.

1.3.6 Работа с категорной композиционной моделью дистрибутивности DisCoCat

Для расширения исследования применен другой подход, основанный на теории категорий DisCoCat. Сделано это для того, чтобы скомпенсиро-

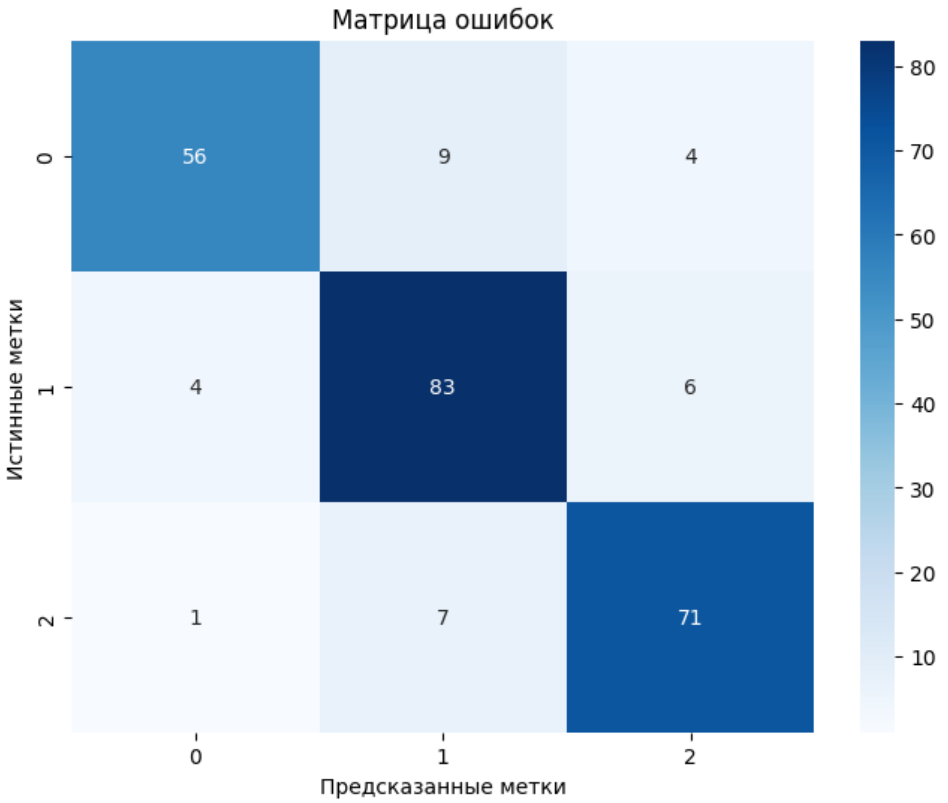

Рис. 1.6: Матрица ошибок

вать неумение моделей понимать смысл предложений и текста.

Мы можем смоделировать грамматику алгебраически через прегруппу, конструкцию, введенную математиком Йоахимом Ламбеком в ранние 1990-е годы. Затем нам предстоит определить категорию семантики.

Идея Джона Фёрса [1] заключается в том, что слова в схожих контекстах будут иметь схожие значения. В лингвистике это называют дистрибутивной гипотезой. Можно предположить, что яблоко больше похоже на банан, чем на щенка, так как яблоки и бананы часто встречаются рядом с такими словами как сладкий, закуска, зеленый, есть и т. д. В то же время, щенок чаще встречается рядом с такими словами, как домашнее животное, милый, пушистый, кора и т.д.

Итак, мы можем представить значение слова через вектор. Обычно это называют distributional model of meaning.

К сожалению, дистрибутивная модель не работает для предложений. Одно и то же предложение редко встречается дважды в документе, поэтому мы не можем следовать описанной процедуре. Именно здесь нам поможет теория категорий!

В свете принципа композиции, значения предложений должны опре-

деляться значениями индивидуальных слов и грамматических правил для их комбинации. Мы можем сопоставить значения слов с грамматическими типами через отображение из синтаксиса (грамматики) в семантику (значения слов) посредством функтора

$$F : \text{PregX} \rightarrow \text{FVect},$$

где X — конечное множество базовых грамматических типов. Фактически, F является сильным моноидальным функтором, тем, который соблюдает компактную закрытую структуру. Это суть категорной композиционной модели дистрибутивности Кука.

В библиотеке `lambeq`, конвертация предложения в квантовую схему происходит согласно следующему пайплайну (см. Рис 1.7).

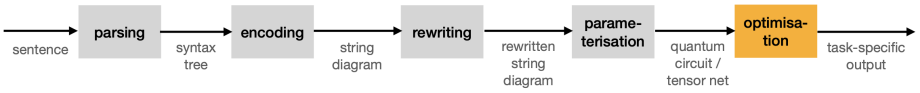

Рис. 1.7: DisCoCat-пайплайн в библиотеке lambeq

- Синтаксическое дерево для предложения создаётся с помощью статистического CCG-парсера. `lambeq` оснащён подробным API, который значительно упрощает этот процесс, и поддерживает несколько современных парсеров.

- Внутренне дерево парсинга преобразуется в строковую диаграмму. Это абстрактное представление предложения, отражающее взаимосвязи между словами в соответствии с выбранной композиторной моделью, независимо от каких-либо решений на уровне реализации.

- Строковую диаграмму можно упростить или преобразовать, применяя правила переписывания. Например, такие правила могут использоваться для удаления специфических взаимодействий между словами, которые могут считаться избыточными для решаемой задачи, или для упрощения вычислений, чтобы они лучше подходили для реализации на квантовом процессоре.

- Полученная строковая диаграмма может быть преобразована в конкретную квантовую схему (или в тензорную сеть для "классического"эксперимента) на основе выбранной параметризации и конкретных вариантов анзацев. lambeq содержит расширяемую иерархию классов с набором предопределённых анзацев, подходящих как для классических, так и для квантовых экспериментов.

- Теперь результат работы конвейера (квантовая схема или тензорная сеть) готов к использованию для обучения. Начиная с версии 0.2.0, `lambeq` предоставляет подробную иерархию классов моделей и обучающих средств, охватывающую все важные случаи использования в задачах обучения с учителем.

Итак, перейдем к имплементации нашей модели.

1. **Парсинг предложений.** Для начала парсим предложения нашего датасета с помощью `BobcatParser`.

```
from lambeq import BobcatParser

parser = BobcatParser(verbose='text')

raw_train_diagrams = parser.sentences2diagrams(train_data)
raw_test_diagrams = parser.sentences2diagrams(test_data)
```

2. **Определение грамматических типов**: Текстовые данные преобразованы в строковые диаграммы с использованием библиотеки `lambeq`.

```
from lambeq import RemoveCupsRewriter

remove_cups = RemoveCupsRewriter()

train_diagrams = [remove_cups(diagram) for
diagram in raw_train_diagrams]
test_diagrams = [remove_cups(diagram) for
diagram in raw_test_diagrams]

train_diagrams[0].draw()
```

3. **Создание квантовых схем**: Диаграммы трансформированы в квантовые схемы для дальнейшего использования в квантовой обработке (см. Рис 1.9).

```
from lambeq import AtomicType, IQPAnsatz

N = AtomicType.NOUN
S = AtomicType.SENTENCE
P = AtomicType.PREPOSITIONAL_PHRASE

ansatz = IQPAnsatz({N: 1, S: 1, P: 1}, n_layers=2)

train_circuits = [ansatz(diagram) for
```

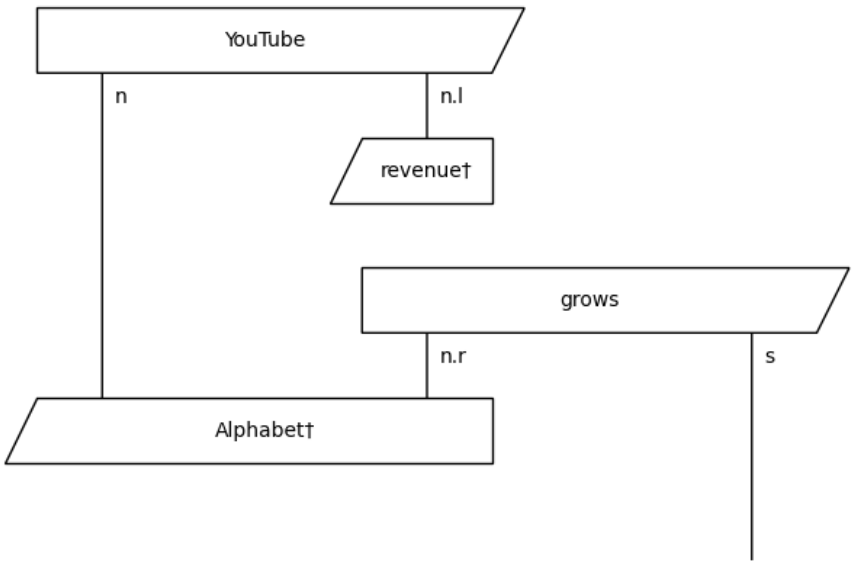

Рис. 1.8: Диаграмма грамматических типов для одного из предложений

```
diagram in train_diagrams]
test_circuits = [ansatz(diagram) for
diagram in test_diagrams]

train_circuits[0].draw(figsize=(8, 8))
```

4. **Конкатенируем схемы и инициализируем PennyLaneModel**. Для дальнейшей работы нам необходимо сконкатенировать схемы и инициализировать основную модель `PennyLaneModel`.

```
from lambeq import PennyLaneModel

all_circuits = train_circuits + test_circuits

model_discocat = PennyLaneModel.
from_diagrams(all_circuits)
model_discocat.initialise_weights()
```

5. **Создание датасета**. Для удобства обучения, создаем тренировочный и валидационный датасеты, состоящие их квантовых схем.

```
from lambeq import Dataset
```

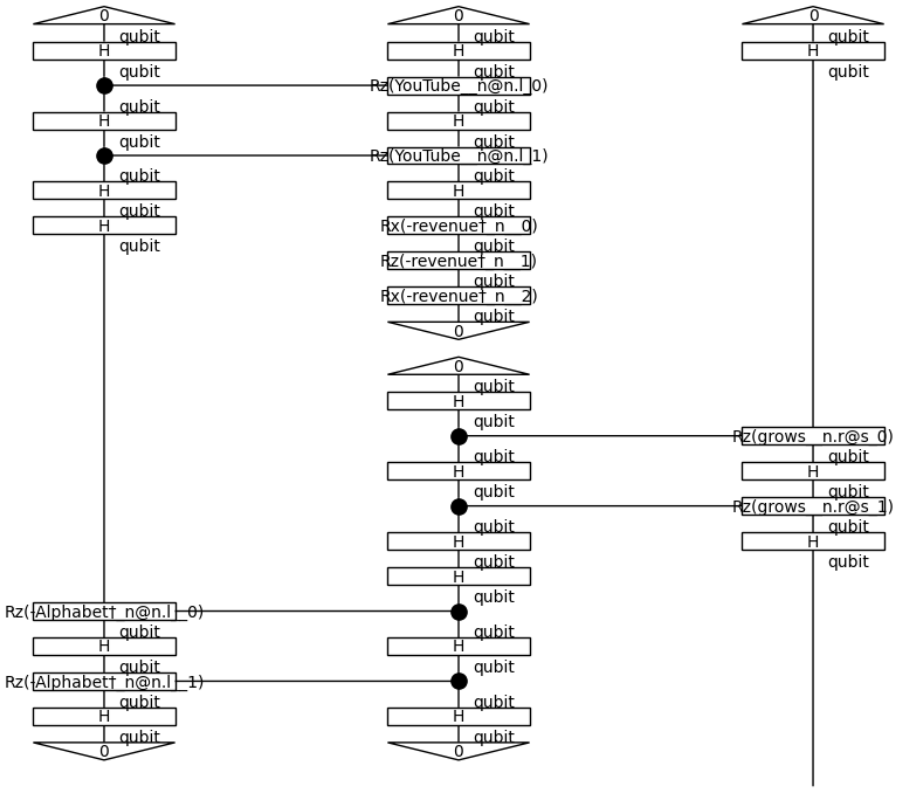

Рис. 1.9: Одна из квантовых схем

```
train_dataset = Dataset(train_circuits,
                        train_labels,
                        batch_size=BATCH_SIZE)

val_dataset = Dataset(test_circuits, test_labels)
```

6. **Определяем метрики**. Прописываем функции метрик для оценки работы модели.

```
def acc(y_hat, y):
    return (torch.argmax(y_hat, dim=1) ==
            torch.argmax(y, dim=1)).sum().item()/len(y)

def loss(y_hat, y):
```

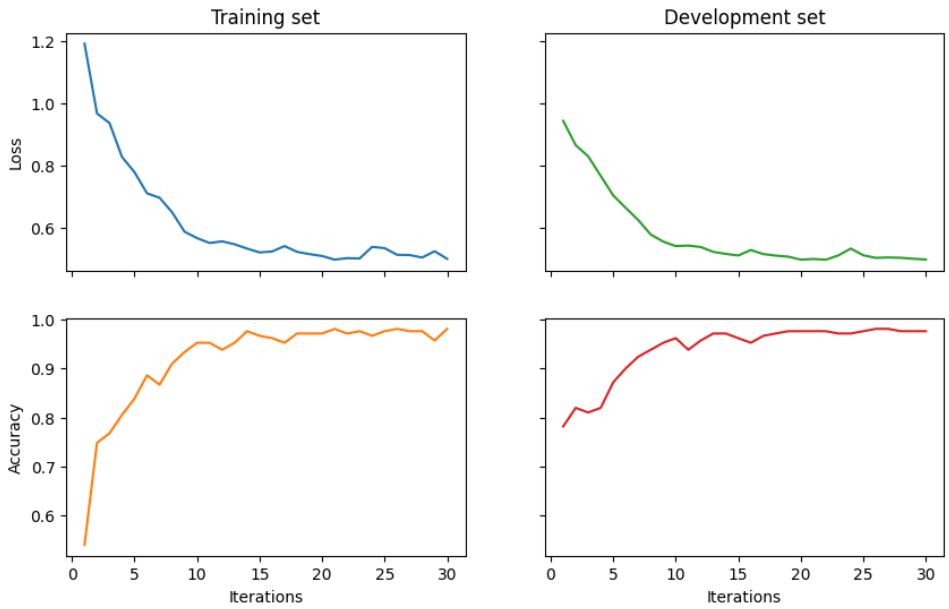

Рис. 1.10: Accuracy DisCoCat

```
        return torch.nn.functional.mse_loss(y_hat, y)
```

7. **Обучение альтернативной модели**: Для тренировки использован `PytorchTrainer`, который позволяет эффективно обучать модели с учетом квантовых данных (см. Рис

```
from lambeq import PytorchTrainer

trainer_discocat = PytorchTrainer(
        model=model_discocat,
        loss_function=loss,
        optimizer=torch.optim.Adam,
        learning_rate=LEARNING_RATE,
        epochs=EPOCHS,
        evaluate_functions={"acc": acc},
        evaluate_on_train=True,
        use_tensorboard=False,
        verbose='text',
        seed=SEED
    )

trainer_discocat.fit(train_dataset, val_dataset)
```

Модель показала значение `Accuracy = 0.97`.

1.3.7 Ансамблевое обучение

Для повышения качества предсказаний использовался ансамблевый метод стекинга, объединяющий несколько моделей. Архитектура ансамбля выглядит следующим образом:

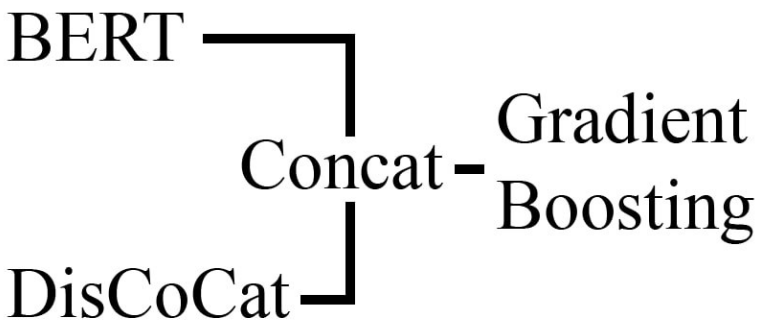

Рис. 1.11: Архитектура ансамбля

1. **Объединение предсказаний**: Выходы двух моделей BERT и DisCoCat конкатенировались для создания входных данных мета-модели.

```
ensemble_inputs = torch.cat((probs_tensor,
predicted_tensor), dim=-1)
```

2. **Обучение мета-модели**: Использован алгоритм градиентного бустинга (`GradientBoostingClassifier` из библиотеки `sklearn`) для обучения мета-модели на объединенных предсказаниях.

```
X = ensemble_inputs.detach().numpy()
meta_model = GradientBoostingClassifier()
meta_model.fit(X, y)
```

3. **Прогнозирование и оценка**: Ансамблевая модель использовалась для предсказания меток, а её точность оценивалась с помощью метрик, таких как точность (accuracy).

```
ensemble_prediction = meta_model.predict(X)
accuracy_ensemble = accuracy_score(y,
ensemble_prediction)

print("Accuracy:", accuracy_ensemble)
```

В результате, точность ансамбля составила `Accuracy = 0.99`.

Заключение

В рамках данной работы была разработана система анализа тональности финансовых новостей с использованием современных подходов обработки текста и ансамблевых методов. Основное внимание было уделено интеграции модели BERT, категорной модели DisCoCat и мета-модели на основе градиентного бустинга.

Проведённый эксперимент продемонстрировал высокую эффективность предложенного ансамблевого подхода, обеспечив метрику Accuracy = 0.99, что превосходит результаты отдельных моделей. Комбинирование различных архитектур позволило компенсировать их взаимные ограничения и усилить качество прогнозирования.

Результаты показывают, что предложенный подход может быть успешно применён в реальных задачах анализа тональности текстов, особенно в финансовой сфере, где точность классификации играет ключевую роль. В дальнейшем возможно расширение исследования за счёт оптимизации ансамблевых методов и использования более разнообразных наборов данных.

Литература

[1] Firth, J.R. (1957). "A synopsis of linguistic theory 1930-1955". Studies in Linguistic Analysis: 1–32. Reprinted in F.R. Palmer, ed. (1968). Selected Papers of J.R. Firth 1952-1959. London: Longman.

[2] Брэдли Т.-Д. What is Applied Category Theory? [Электронный ресурс] // arXiv.org. 2019. URL: https://arxiv.org/abs/1809.05923 (дата обращения: 10.10.2024).

[3] Девлин Дж., Чанг М.-В., Ли К., Туранини К. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [Электронный ресурс] // arXiv.org. 2019. URL: https://arxiv.org/abs/1810.04805 (дата обращения: 10.11.2024).

[4] У Я., Кук Б., Садрзаде С.С., Клар Б. A Categorical Compositional Distributional Modelling for the Language of Life [Электронный ресурс] // arXiv.org. 2019. URL: Аналогично https://arxiv.org/abs/1902.09303 (дата обращения: 11.11.2024)

[5] Садрзаде М., Кларк С., Кобурн Б. Mathematical Foundations for a Compositional Distributional Model of Meaning [Электронный ресурс] // arXiv.org. 2019. URL: https://arxiv.org/abs/1003.4394 (дата обращения: 14.11.2024).

[6] Карцакис Д., Коэкке Б., Лорензо Г., Садрзаде М., Йеун Р., Коэкке Б. lambeq: An Efficient High-Level Python Library for Quantum NLP [Электронный ресурс] // arXiv.org. 2019. URL: https://arxiv.org/abs/2110.04236 (дата обращения: 17.11.2024).