DOI: 10.15514/ISPRAS-2023-35(5)-13

Извлечение именованных сущностей из рецензий к исходному коду

[1,2] *В.В. Качанов,* ORCID: 0000-0002-9371-6483 <*vkachanov@ispras.ru*>
[1,3] *А.С. Хитрова*, ORCID: 0009-0003-1723-5199 <*akhitrova@ispras.ru*>
[1] *С.И. Марков,* ORCID: 0000-0002-6687-4937 <*markov@ispras.ru*>

[1] *Институт системного программирования РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25.
[2] *Московский физико-технический институт (НИУ),*
141701, Россия, Долгопрудный, Институтский пер., д. 9.
[3] *Московский государственный университет имени М.В. Ломоносова,*
119991, Россия, Москва, Ленинские горы, д. 1.

Аннотация. В данной статье рассматривается задача извлечения именованных сущностей из рецензий исходного кода. В работе приводится сравнительный анализ существующих подходов и предлагаются собственные методы для улучшения качества решения задачи. Предложенные и реализованные улучшения включают в себя: методы борьбы с дисбалансом данных, улучшения токенизации входных данных, использование больших массивов неразмеченных данных и применение дополнительных бинарных классификаторов. Для оценки качества собран и размечен вручную новый набор из 3000 пользовательских рецензий. Показано, что предложенные улучшения позволяют значительно увеличить показатели метрик качества, вычисляемых как на уровне токенов (+22%), так и на уровне сущностей целиком (+13%).

Ключевые слова: машинное обучение; извлечение именованных сущностей; набор данных.

Для цитирования: Качанов В.В., Хитрова А.С., Марков С.И. Извлечение именованных сущностей из рецензий к исходному коду. Труды ИСП РАН, том 35, вып. 5, 2023 г., стр. 193–214. DOI: 10.15514/ISPRAS–2023–35(5)–13.

Named Entity Recognition for Code Review Comments

[1,2] *V.V. Kachanov* ORCID: 0000-0002-9371-6483 <*vkachanov@ispras.ru*>
[1,3] *A.S. Khitrova* ORCID: 0009-0003-1723-5199 <*akhitrova@ispras.ru*>
[1] *S.I. Markov* ORCID: 0000-0002-6687-4937 <*markov@ispras.ru*>

[1] *Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.
[2] *Moscow Institute of Physics and Technology,*
9 Institutskiy per., Dolgoprudny, Moscow Region, 141701, Russia.
[3] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia.

Abstract. This paper addresses the problem of named entities recognition from source code reviews. The paper provides a comparative analysis of existing approaches and proposes its own methods to improve the quality of problem solving. Proposed and implemented improvements include: methods to deal with data imbalances, improved tokenization of input data, the use of large arrays of unlabeled data, and the use of additional binary

classifiers. To assess quality, a new set of 3,000 user code reviews was collected and manually labeled. It is shown that the proposed improvements can significantly increase the performance measured by quality metrics, calculated both at the token level (+22%) and at the entire entity level (+13%).

Keywords: machine learning, named entity recognition, dataset.

For citation: Kachanov V.V., Khitrova A.S., Markov S.I. Named entity recognition for code review comments. Trudy ISP RAN/Proc. ISP RAS, vol. 35, issue 5, 2023. pp. 193-214 (in Russian). DOI: 10.15514/ISPRAS-2023-35(5)-13.

1. Введение

Извлечение именованных сущностей (named entity recognition, NER) – важная задача в области обработки естественного языка (natural language processing, NLP). Это комплексная задача, состоящая из двух:

1) определить, является ли слово или словосочетание именованной сущностью, и
2) определить класс, к которому данная сущность относится.

Набор классов не является фиксированным и зависит от постановки конкретной задачи: зачастую ищут имена, названия городов, названия компаний и организаций. Однако NER можно использовать и для более узкоспециализированных областей, например, рецензии и комментарии к исходному коду (source code review). В этом случае набор классов будет специфичным и больший интерес будут представлять слова, связанные с исходным кодом, на который был написан комментарий. Распознавание таких слов поможет лучше понять смысл рецензии, что, в свою очередь, можно использовать в задачах классификации, кластеризации рецензий, семантического сравнения и поиска.

В ходе решения задачи извлечения именованных сущностей необходимо разбить текст на последовательность смысловых единиц – токенов, которые далее и будут классифицированы. Обычно токенами являются отдельные слова и знаки пунктуации. Таким образом, именованная сущность может состоять из нескольких токенов так же, как название организации может содержать несколько слов. Токенизация, то есть процесс разделения на токены, может быть сложнее, чем получение последовательности слов, разделённых пробельными символами, и зависеть от области, которой принадлежит этот текст. Для отрывков исходного кода правила разделения на смысловые единицы могут зависеть от синтаксиса языка программирования.

Существует множество различных подходов к решению задачи NER. Наиболее простыми являются подходы, основанные на использовании словарей, или проверки текста на наличие каких-либо шаблонов. Возможно использование статистических моделей классификации последовательностей, таких как скрытые модели Маркова или условно случайные поля. Так как NER является задачей классификации, возможно использование метода опорных векторов. Но наилучшие результаты дают подходы с использованием нейронных сетей. Это главным образом связано со способностью нейронных сетей (например, рекуррентных или трансформеров) эффективно использовать контекст. Кроме того, использование моделей векторного представления слов позволяет хранить больше информации о токенах в тексте.

Основой реализованных классификаторов является модель семейства BERT [1] (Bidirectional Encoder Representations from Transformers) – предобученная языковая модель, основанная на архитектуре Трансформер (Transformer). Благодаря своей архитектуре BERT способен улавливать контекстную информацию. Эта модель может быть использована для решения множества различных задач обработки естественного языка, в том числе и в рамках задачи распознавания именованных сущностей. В данной работе модели семейства BERT использовались для обучения классификатора токенов на новом наборе данных.

Одной из близких работ в данном направлении является SoftNER [2], в которой авторы исследовали и разрабатывали классификатор именованных сущностей для сообщений из

форума StackOverflow[1]. К числу основных результатов работы относится применение промежуточного векторного представления для токена от различных моделей как частей одного классификатора. Такой подход был взят за основу для дальнейших исследований как наилучший из представленных.

В ходе работы также оценена эффективность всех реализованных методов для нового набора данных и проведен анализ ошибок классификаторов.

Основным вкладом этой работы являются:

- собранный и размеченный на 15 классов набор из 3000 рецензий к исходному коду;
- предложенные методы улучшения работы классификаторов;
- обученные классификаторы, распознающие 31 тип сущностей (в BIO нотации), связанных с разработкой программного обеспечения.

2. Методы извлечения именованных сущностей

Задача классификации токенов в тексте довольно распространенная и изученная. За время исследований было предложено множество подходов к решению задачи. В качестве признаков для описания слов использовались различные характеристики, такие как: форма, лемма, область окружающих слов в скользящем окне, справочная информация, статистические данные, использование заглавных букв, пунктуация и так далее. Применение искусственных нейронных сетей не только расширило список численных характеристик слов и их частей, но и повысило качество определения и классификации слов в тексте [3].

2.1 Скрытые марковские модели

В работе [4] скрытые марковские модели использовались для распознавания и классификации имен, дат, времени и числовых величин в наборах данных MUC-6, MUC-7 и трансляциях новостей. Разработанная модель при обучающей выборке в 100,000 слов научилась классифицировать с точностью в 94%.

Здесь задача NER рассматривается как задача присвоения каждому слову одной из предложенных меток либо специальной метки NOT-A-NAME, чтобы обозначить, что слово не принадлежит ни одной из меток. Далее используется вероятностная модель для расчета возможности появления слов в контексте биграмм.

В более формальной постановке задача состоит в поиске наиболее вероятной последовательности имен классов (NC) при заданной последовательности слов (W): $\max P(NC|W)$.

2.2 Условно случайные поля

Условно случайные поля (conditional random field, CRF) были введены авторами работы [5] как инструмент статистического моделирования для задачи распознавания шаблонов. В работе [6] был предложен вариант использования CRF для извлечения именованных сущностей.

$$P(s|o) = \frac{\exp(\sum_{t=1}^{T} \beta_k f_k(s_{t-1}, s_t, o, t))}{Z},$$

где $o = < o_1, o_2, \dots, o_T >$ – входная последовательность (слов), $s = < s_1, s_2, \dots, s_T >$ – последовательность состояний (меток, соответствующих входной последовательности), Z – некоторый нормализующий коэффициент $f_k(s_{t-1}, s_t, o, t)$ – произвольная функция-признак, β_k – обучаемый вес признаков. Значения переходов между двумя состояниями могут быть вычислены с помощью алгоритмов динамического программирования. В работе была

[1] https://stackoverflow.com/questions

показана эффективность предложенного решения, на соревновании CoNLL 2003 получив 84.04% для английского и 68.11% для немецкого языков.

2.3 Метод опорных векторов

Метод опорных векторов был представлен Кортесом и Вапником [7]. В основе алгоритма лежит идея разделяющей линейной гиперплоскости, которая старается максимизировать расстояние от этой плоскости до ближайших точек обоих классов. В работе [8] описывается применение метода опорных векторов к задаче извлечения именованных сущностей. Суть заключается в создании 8 классификаторов, по одному на класс. В качестве вектора признаков используется более 1200 бинарных меток про каждое слово. При описании каждого слова также используется его контекст размера 7 (3 слова до и 3 после). Имея результаты классификации 8 моделей, метка для слова определяется исходя из коэффициентов уверенности моделей. Если ни одна из них не уверена, что слово принадлежит соответствующему классу, то ставилась метка "O".

2.4 Нейронные сети

С распространением методов машинного обучения и глубоких нейронных сетей для задач обработки естественного языка они начали применяться и для задачи NER. Так как рекуррентные нейронные сети работают с последовательностью данных, они хорошо подходят для обработки естественного языка. Более того, разработанные позже Long Short-term Memory (LSTM) сети умеют "запоминать" длинные зависимости в тексте, улучшая понимание текста целиком. В работе [9] было показано, что решения на основе BiLSTM-CRF показали лучшие результаты на исследуемых выборках (в наборе CoNLL 2003 для английского – 90.94% F1-score, для немецкого – 78.76% F1-score, CoNLL 2002 для испанского – 86.75% F1-score). Применение искусственных нейронных сетей в задаче NER не ограничивается только классической предметной областью с обнаружением сущностей классов ORG, PERSON, DATETIME, но также используется в области медицины [10] для поиска названий болезней, симптомов и фармацевтических препаратов. Моделей на архитектуре Трансформер с момента их появления активно используются во многих задачах, связанных с обработкой естественного языка. Эта архитектура основана на механизме внимания [11], что позволяет отлавливать зависимости между удалёнными словами в предложении. Более того, оказалось, что для извлечения именованных сущностей в большинстве областей модели, основанные на этой архитектуре, превосходят другие [12]. В работе [13] описано применение BERT моделей для решения задачи извлечения именованных сущностей из текстов о кибербезопасности на русском языке. Авторы предлагают метод аугментации данных для получения большего количества примеров именованных сущностей. Одним из способов улучшения результатов работы классификатора является обучение большой языковой модели на специфическом наборе текстов. Наиболее приближенной к нашей задаче можно считать классификацию именованных сущностей, связанных с программным обеспечением в вопрос-ответах со StackOverflow, рассмотренную в статье [2]. Основой предлагаемого решения являются глубокие нейронные сети и обучение с учителем. В работе рассматриваются несколько моделей векторного представления слов (ELMo, GloVe, BERT). Также предлагаются 2 метода повышения качества классификации: путем предобучения модели, кодирующей слова в векторы; использование дополнительных моделей, предоставляющих векторные представления слов. Наилучшей рассмотренной конфигурацией оказалось использование предобученного на большом наборе неразмеченных сообщений со StackOverflow кодировщика BERT (BERTOverflow) и использование промежуточных векторов из двух моделей в качестве дополнительных признаков для

основного классификатора. К достоинствам работы также можно отнести создание открытого вручную размеченного набора данных.

3. Набор размеченных рецензий к исходному коду

В данном разделе описан процесс создания набора данных CodeReviewCommentsNER, который доступен публично на Zenodo[2].

Существует не так много решений для извлечения именованных сущностей для рецензий исходного кода или похожих предметных областей [14, 15]. Единственным доступным набором данных, связанным с разработкой программного обеспечения с достаточно детальной разметкой, является набор SoftNER [2]. К положительным аспектам можно отнести схожую предметную область, а именно комментарии пользователей из форума StackOverflow. Также безусловным плюсом является наличие меток, связанных с исходным кодом: Class, Variable, Function, Value, Data Type.

К недостаткам набора данных SoftNER можно отнести правила токенизации. Например, единым токеном с меткой "Code_Block" является «`size(S.vertices)+1`» и аналогично «`as.numeric(df$EventName)`» с меткой Library_Function, хотя с нашей точки зрения оба этих выражения можно разбить на более мелкие вызовы методов, использование переменных и константных значений.

Также не все предложенные классы являются четко сформулированными, актуальными и достаточно часто используемыми в рецензиях к исходному коду.

Так, класс "IN LINE CODE" всегда будет пересекаться с другими классами, связанными с кодом, и его можно скорее использовать в случае иерархической классификации как обобщающий.

Кроме того, предложенный набор данных имеет распределение сущностей не схожее с тем, что было получено из рецензий к исходному коду. Например, из табл. 1 видно, что в предложенном наборе количество имен переменных в 13 раз меньше, чем в исследованных нами рецензиях.

Табл. 1. Процентное соотношение важных классов в наборе данных
Table 1. Percentage of important classes in the dataset

Набор данных	**CodeReviewCommentsNER**	**SoftNER**
Variable	2.7%	0.2%
Value	1.8%	0.8%
Function	1.5%	0.5%
External_Tool/Application	0.3%	1.1%

Таким образом, из-за отличия предметной области и разногласий в разметке данных, было необходимо собрать собственный набор данных.

Для анализа и разметки было собрано 3000 рецензий к исходному коду из различных проектов с открытым исходным кодом (из Github[3] проектов на Java, Python, C/C++, из Gerrit систем Android Open Source Project[4], Tizen Gerrit[5]). 2577 комментариев были взяты случайным образом из общей базы в 420 тысяч комментариев. 433 комментария подбирались по вхождению ключевых слов для увеличения количества представителей некоторых классов из этой же общей базы.

[2] https://zenodo.org/doi/10.5281/zenodo.10060889
[3] https://github.com/
[4] https://android-review.googlesource.com/
[5] https://review.tizen.org/gerrit/

3.1 Описание классов

В качестве начального набора классов были рассмотрены 20, предложенных авторами работы SoftNER: CLASS, VARIABLE, IN LINE CODE, FUNCTION, LIBRARY, VALUE, DATA TYPE, HTML XML TAG, APPLICATION, UI ELEMENT, LANGUAGE, DATA STRUCTURE, ALGORITHM, FILE TYPE, FILE NAME, VERSION, DEVICE, OS, WEBSITE, USER NAME.

Как было описано выше, такой набор меток требует переработки для рецензий исходного кода.

В классе APPLICATION был расширен смысл и преобразован в External_Tool. Классы UI ELEMENT, LANGUAGE, DATA STRUCTURE, ALGORITHM, VERSION, DEVICE и USER NAME было решено убрать. Добавлен класс Error_Name.

Итоговый набор из 15 классов представлен в табл. 2.

Разметку первых 100 комментариев проводили три специалиста, средний коэффициент согласия каппа Коэна составил 0.82, что указывает на почти полное согласие размечающих. После анализа разногласий по некоторым классам в инструкцию для разметки были внесены дополнения и примеры согласованных именованных сущностей. Дальнейшая разметка проводилась без перекрёстного анализа, и произведена частичная кросс-валидация, показавшая высокий коэффициент согласия. Основным типом несогласованностей был пропуск именованных сущностей, что составило менее 0,1% от общего числа токенов. Для итогового набора размеченных данных был проведен повторный просмотр комментариев и исправлены пропущенные метки.

3.2 Токенизация рецензий

Одной из отличительных черт наших данных является особенность разделения текста на токены. Разбивать просто по пробелам не является верным подходом, так как исходный код часто содержит множество смысловых объектов, которые разделены различными знаками – точками, скобками и т.д., а правила написания программного кода, в свою очередь, зависят от правил синтаксиса конкретного языка. Таким образом возникает необходимость создания правил получения списка токенов из текста, которые будут учитывать все эти отличительные черты текста рецензии, но при этом не создавать излишне большого количества токенов.

С помощью регулярных выражений был создан метод токенизации, удовлетворяющий данным требованиям. Основными принципами являются:

- отделение буквенно-цифровых символов от не буквенно-цифровых;
- отделение кавычек от остальных символов;
- отделение точек и запятых от буквенно-цифровых символов;
- отделение скобок от нескобочных символов.

Благодаря относительно небольшому количеству правил удалось сохранить высокую скорость токенизации, и при этом получить разбиение, которое хорошо соответствует ручной разметке и подходит для разбиения отрывков кода вне зависимости от использованного языка программирования.

В модель передаются данные уже после предварительной токенизации. Далее эти токены подразбиваются на ещё меньшие токены, которые могут быть представлены моделью BERT для получения ее векторного представления.

3.3 Характеристики данных

Всего в 3000 рецензий содержалось 94328 токенов. При разделении на обучающую и тестовую выборки соблюдалось правило, что в тестовую попадает 10% + 1 пример каждого класса, чтобы избежать ситуации, когда примеров малочисленных классов нет в тестовом наборе. Некоторая общая статистика приведена в табл. 3.

Табл. 2. Список меток-классов с описанием
Table 2. List of class labels with description

Имя класса	**Описание**	**Примеры**
Variable	имена переменных и объектов	logicColumn, Position, mount_point_count, MS_BIND
Function	имена функций, методов и процедур, не захватывая символы "()"	_handle_fromlist, EXPECT_THAT, getInput, ForEach
Class	имена классов, структур	DisplayVk, VirtualHost, ImagePipeline, ImporterMesh, UniformLocation
Value	константные строки, включая ", числа, булевы значения, значения «null», «None»	"1px solid grey", "<classpathentry ...>", "0 0,1,2,3 11 19 AUG ? 2018", 0.7, 30, 1, 2, null
File_Name	название файла с расширением, полный путь к файлу/директории	testing/tf.py, wine_data.csv, /data/app/lib/x86/libgame.so
File_Type	упоминание расширения файла, не учитывая полного имени файла	GIFs, webp, .jar, binary
Keyword	ключевые слова используемые в языках программирования	PUBLIC, ifndef, unsigned, class, interface, else, select, yield
Data_Type	название типов данных, включая пользовательские типы и структуры, если используются в качестве типа	unsigned char, Float, ConstBytes, TypeName, Vector, LandmarkPoint, UInt64
Library_Package	упоминания подгружаемых библиотек/модулей	com.google.gerrit.httpd.rpc.change.ListChangesServlet, log4j, mlflow.projects.backend.local._create_virtualenv
Error_Name	имя класса ошибки или ее название	NoManifestException, IOException, InvalidOperation, ImportError, Unimplemented, Exceptions, NPD, ANR, OOM, out of memory
HTML_XML_Tag	html/xml тэг с символами <>	<classpathentry kind="output">, <body>, <c1>, </style>, <svg height="20" width="20">
Operating_System	название операционных систем	android, Chrome OS, Linux, Unix,SerenityOS, Win 8, bsd, arch, selinux
Programming_ Language	имена языков программирования	C++17, xml, cpp,c++, R, pythonic
External_Tool	имена сторонних приложений, которые можно запустить как самодостаточный инструмент	Qt, travis, alibaba-dubbo, pylint, JVM, dockerd, isoltest

Website	ссылки на веб-ресурсы	https://www.example.com/, https://www.example.com/somepage\#heading2, https://example.com/file.png

Табл. 3. Общая характеристика набора данных
Table 3. General characteristics of the data set

	Общий	**Обучение**	**Тест**
Кол-во примеров	3000	2543	457
Кол-во токенов	94328	80838	13490
Кол-во токенов с сущностями	15420	13429	1991
Среднее кол-во токенов сущностей на 1 пример	5.1	5.3	4.4
Кол-во сущностей	8514	7241	1285
Среднее кол-во сущностей на 1 пример	2.8	2.8	2.8

Структура по типам сущностей представлена в табл. 4. Исходя из данных в таблице, можно сделать два замечания: токенов сущностей значительно меньше, чем токенов без какой-либо сущности; некоторых сущностей не так много, но они длинные и состоят из большого количества токенов (Website, File_Name).

Табл. 4. Структура данных по сущностям
Table 4. Entity data structure

Тип сущности	**Количество токенов**	**Количество сущностей**
O	77079	77079
Variable	5089	2525
Website	3669	161
Function	3655	1429
Value	1548	823
Class	1328	666
File_Name	1169	174
HTML_XML_Tag	1027	287
Library_Package	955	278
Keyword	848	815
Data_Type	741	514
External_Tool	347	259
Error_Name	272	131
File_Type	241	154
Operating_System	193	160
Programming_Language	172	138

4. Базовое решение

4.1 Базовые модели

В качестве базового решения были выбраны модели семейства Bidirectional Encoder Representations from Transformer (BERT), а именно RoBERTa [16] и CodeBERT [17]. Первая хорошо зарекомендовала себя в разных задачах обработки естественного языка (natural

language processing, NLP). CodeBERT – модель, основанная на RoBERTa, но обученная на примерах исходного кода с документацией, для построения семантической связи между исходным кодом и описанием этого кода на естественном языке.

Кроме того, как было указано в работе SoftNER, предобученные модели на текстах из нужной области улучшают итоговые результаты. Так как полученная авторами модель BertOverflow находится в открытом доступе, она также использовалась в экспериментах.

Для предобучения на рецензиях к исходному коду были взяты около 420 тыс. уникальных комментариев с различных проектов с открытым исходным кодом. Обучение проводилось на задаче Masked Language Model (MLM), когда часть токенов маскируются специализированным токеном, а модель просят восстановить пропущенный токен.

CodeBERT – это уже предобученная модель на своем наборе данных, поэтому мы обучим её с нуля на своем наборе сообщений (PretrainCodeBert).

Таким образом есть 4 базовые модели, с которыми мы будем проводить дальнейшие эксперименты: roberta-base, codebert-base, BertOverflow, PretrainCodeBert.

Для всех классификаторов, использующих только одну BERT-подобную модель, использовалась следующая архитектура:

- BERT токенизатор: превращает входящую последовательность в токены словаря модели и их индексы, понятные для BERT encoder;
- BERT кодировщик: для каждого входящего элемента возвращает векторное представление, состоящее из 768 чисел;
- Слои классификации: набор слоев нейронных сетей в различной конфигурации, решающих задачу классификации на N классов по входному вектору признаков.

Стандартной функцией потерь для задачи многоклассовой классификации является CrossEntropy. В работе использовали оптимизатор Adam с фиксированным learning_rate=5e-6.

4.2 Вспомогательное разбиение токенов

Токенизация, проходящая для получения векторного представления для BERT, состоит из двух этапов: предварительная токенизация (претокенизация, pre-tokenization) и токенизация с помощью предобученного токенизатора BERT модели. Без использования претокенизации встроенный токенизатор BERT не может разбить длинные токены на составные части, которых нет в словаре модели BERT. Этап претокенизации нужен, чтобы получить более осмысленные разбиения в дальнейшем. Так с помощью заданных правил он разбивает названия функций, в которых был использован snake_case или camelCase, на отдельные слова, что позволяет разделить длинные токены на составные части.

Без претокенизации получим «mAssistants» -> [‘m’, ‘Ass’, ‘ist’, ‘ants’].

С претокенизацией «mAssistants» -> [‘m’, ‘Assistants’] -> [‘m’, ‘Assist’, ‘ants’] – остается информация о слове «Assist».

Эта разница возникает из-за того, что при обучении языковой модели, особенно на текстах областей, не связанных с исходным кодом, она не встречала подобные длинные токены. С другой стороны, каждое слово подобного составного токена по отдельности может существовать в словаре. В случае RoBERTa токены часто кодируются с ведущим пробелом, а значит токен без ведущего пробела имеет больший шанс быть интерпретированным неверно, так как его нет в словаре. Таким образом претокенизация позволяет справиться с данной проблемой, проводя первичное разбиение на подходящие подтокены. Далее предобученыный токенайзер BERT разбивает до тех подтокенов, которые есть в словаре BERT, для дальнейшего использования.

4.3 Дисбаланс классов

Одной из проблем данной задачи является сильный дисбаланс классов. Из табл. 4 видно, что токенов класса "O" более 77%, тогда как "Error_Name" – чуть больше 0.1%. Существует несколько видов борьбы с подобными явлениями в данных, а именно: искусственное увеличение числа примеров малочисленных классов (upsampling), уменьшение примеров многочисленных классов (downsampling), применение взвешенных функций потерь для более чувствительного обучения модели на примерах нужного типа.

Upsampling был применен в процессе сбора примеров комментариев для CodeReviewCommentsNER, когда по итогам разметки его части стало понятно, что некоторых классов мало. Для этого из общего набора рецензий при помощи регулярных выражений были собраны более подходящие примеры.

Downsampling не очень подходит, так как основной перевес имеет класс «O», а этого практически невозможно избежать в силу особенностей текстов и выбранных классов. Для остальных классов абсолютное количество было и так небольшим и применять к ним downsampling не имело смысла.

В качестве функции потерь, поддерживающей возможность передавать веса классам, был выбран FocalLoss [18]. Данная функция потерь была разработана с целью устранения дисбаланса классов в задачах компьютерного зрения:

$$FL(p_t) = -\alpha_t(1-p_t)^{\gamma}\log(p_t),\ p_t = \begin{cases} p, & \text{если } y = 1 \\ 1-p, & \text{иначе} \end{cases}$$

где $\alpha \in [0,1]$, хотя на практике α можно инвертировать и брать из промежутка $[1,\infty)$, $\gamma \in [0,5]$.

Введение такого взвешивающего (weighting) параметра α – общепринятый метод борьбы с дисбалансом классов. Этот параметр выравнивает важность многочисленных и малочисленных примеров, однако не делает различий между простыми и сложными примерами. Для этого вводится параметр γ, который помогает уменьшить вес простых примеров.

Эмпирическим методом были подобраны константы $\gamma = 2, \alpha = \begin{cases} 1, \text{для класса «}O\text{»} \\ 10, \text{иначе} \end{cases}$.

5. Вспомогательные бинарные модели

В работе SoftNER были предложены две дополнительные модели, которые обучались бинарной классификации на «O» и «Entity», после чего предпоследний линейный слой классификатора использовался как часть векторного представления токенов в основной модели. Одна из моделей основана на архитектуре BERT (Segmenter), вторая – контекстно-независимая модель, оценивающая каждое слово отдельно на основе частотных характеристик (Recognizer). Использование таких двух разных моделей по заверениям авторов должно помочь лучше классифицировать слова, которые встречаются как в контексте обычного общения, так и в участках исходного кода.

5.1 Контекстно-независимая модель

Как было сказано выше, одна из моделей контекстно-независимая, то есть обрабатывает каждое слово вне зависимости от контекста, в котором оно находится.

Она составная и содержит в себе три части:

- частотный словарь, построенный по набору данных GigaWord[6], содержащему обычный текст;

[6] https://catalog.ldc.upenn.edu/LDC2011T07

- частотный словарь, построенный по набору вопросов и ответов StackOverflow;
- модель Fasttext[7], обученная по принципу обучения без учителя на значительном наборе рецензий к исходному коду.

Далее частотные характеристики словарей превращаются в векторы методом гауссовской дискретизации [19] и конкатенируются с выходным вектором Fasttext. После чего полученный вектор проходит через пару полносвязных линейных слоев искусственной нейронной сети. Детальная схема представлена на рис. 1.

Так как эта модель также зависит от словаря и разбиения на токены, то для разбиения текста использовался токенизатор из раздела 3.2.

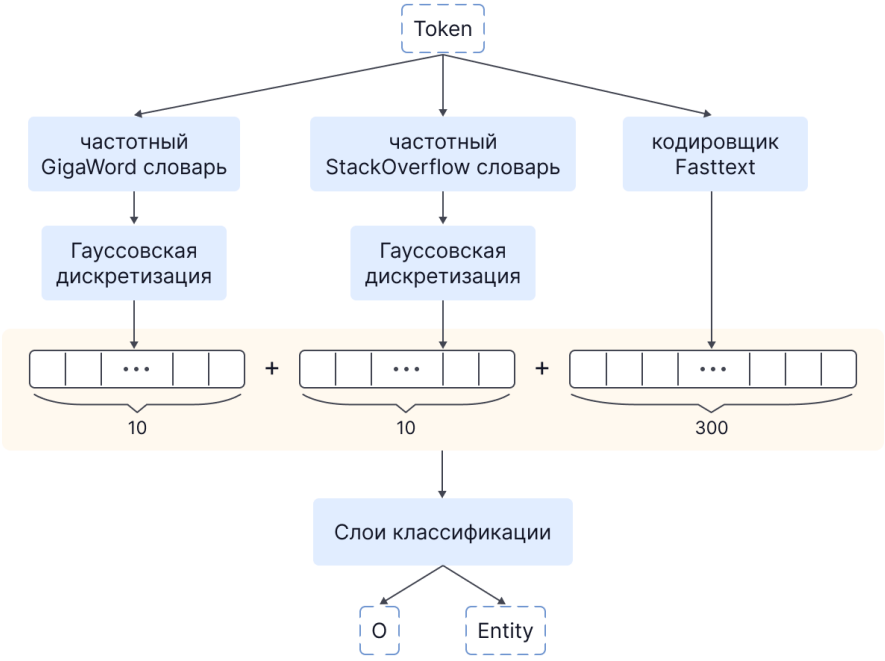

Рис. 1. Архитектура контекстно-независимой модели.
Fig. 1. Context-independent model architecture.

5.2 Контекстно-зависимая модель

Контекстно-зависимая модель построена на архитектуре BERT. Как и для базового решения многоклассовой классификации, здесь имеют место эксперименты с базовой моделью. Нами были использованы те же 4 базовые модели, описанные в разделе 4.1. Структура бинарного классификатора повторяет структуру многоклассового.

5.3 Методы взаимодействия моделей

Как уже было сказано выше, в работе SoftNER дополнительные модели применялись для предоставления дополнительной части векторного представления токена, как показано на рис. 2. В случае такой конкатенации двух векторов по 768 чисел получится слишком резкий

[7] https://fasttext.cc/

перепад размерности векторов с 1536 в 31, поэтому можно добавить промежуточный слой размера 500.

Другой вариант использования дополнительной модели предложен в [20]. В работе рассмотрены два метода: Classify-Verify и Classify-Trust. В первом методе если общая модель предсказала какой-то ответ, то это предсказание сравнивается с предсказаниями специализированной модели и принимается решение об итоговом классе. Если же общая модель предсказала «Отсутствие ответа» (что в нашем случае можно интерпретировать как класс «O»), то «Отсутствие ответа» и будет итоговым ответом. Во втором методе Classify-Trust если общая модель предсказывает какой-то ответ, то автоматически используется ответ специализированной модели, и аналогично первому методу – в случае отсутствия ответа от общей модели – такой ответ является итоговым.

В нашей конфигурации «общей» моделью можно назвать модель бинарной классификации, а «специализированной» – многоклассовую. Метод Classify-Trust (схема реализации представлена на рис. 3) применяется прямо по предложенному методу: если бинарная модель предсказывает «O» – итоговый ответ «O», иначе смотрим на предсказания многоклассовой модели. Метод Classify-Verify сложнее в интерпретации, так как не ясно каким образом сравнивать предсказания бинарной и многоклассовой моделей.

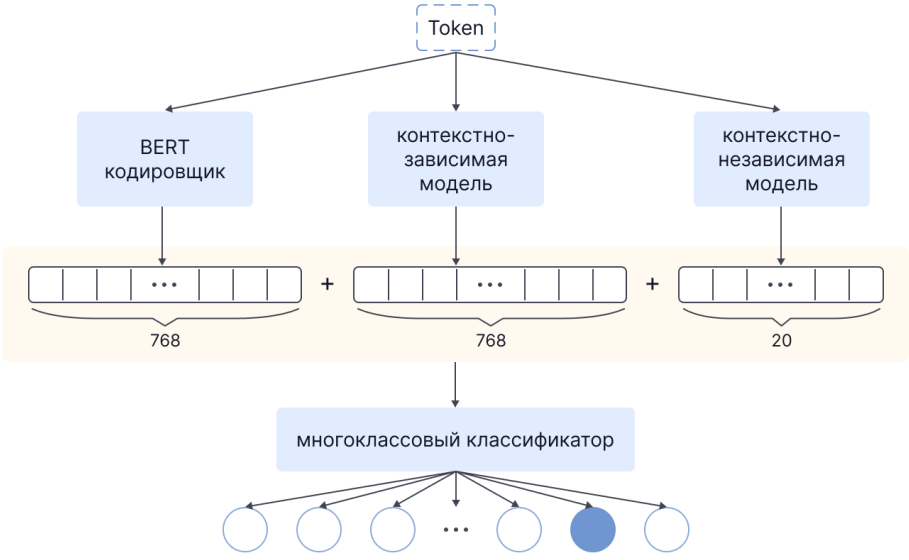

Рис. 2. Схема объединения векторных представлений методом конкатенации.
Fig. 2. Scheme for combining vector representations using the concatenation method.

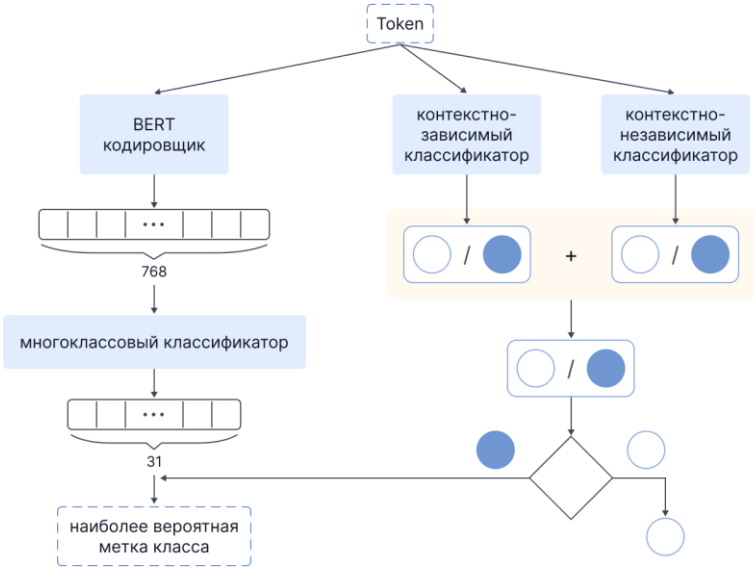

Рис. 3. Применение вспомогательных моделей методом Classify-Trust.
Fig. 3. Using auxiliary models using the Classify-Trust method.

6. Метрики

Базовыми метриками в задаче NER являются, как и в задачах классификации в целом, точность (precision), полнота (recall), f-мера (f1-score). Они вычисляются по каждому классу отдельно и усредняются по одной из стратегий: macro, micro, weighted. Для нас самой подходящей стратегией является macro, так как по ней общее среднее значение считается как среднее по всем классам, независимо от количества элементов в каждом классе отдельно. Однако эти метрики считаются по каждому токену, а как было показано в разделе 3.2, токенизаторы иногда разбивают исходные слова довольно сильно, тем самым порождая большое количество не совсем значимых токенов. Тем более, что с точки зрения конечного пользователя важнее видеть, что классификатор полностью выделил всю именованную сущность, а не только некоторые подтокены из нее.

6.1 BIO нотации

BIO (сокращенно от beginning – «начало», inside – «внутри», outside – «снаружи») нотации – это один из общепринятых форматов маркирования токенов в задаче фрагментирования текстов. Префикс B- перед тегом указывает, что тег является началом фрагмента, I- перед тегом – что тег находится внутри фрагмента. Тег O указывает, что токен не принадлежит ни одному из классов. Существует несколько разновидностей подобных нотаций: BIO/IOB, IOB2, BIOES, BILOU, которые отличаются наличием дополнительных префиксов к тегам или различными правилами разметки. Так по правилам BIO нотации префикс B- ставится токену только тогда, когда за ним токен следует I- тег того же класса. При разметке данных были использованы правила IOB2, где префикс B- ставится всегда в первом токене новой сущности.

6.2 Метрики по сущностям

На International Workshop on Semantic Evaluation (SemEval) 2013 года [21] были введены 4 способа вычислять precision/reacll/f1:

- точное совпадение границ и типа;
- точное совпадение границ, независимо от типа;
- частичное совпадение границ, независимо от типа;
- есть некоторое пересечение между предсказанным и правильным ответом.

Нас заинтересовал Strict, как самый строгий и точный метод, и Type, в котором проверяется, что есть хоть какое-то совпадение с правильным ответом. Type засчитывается в случае любого пересечения по позициям правильного ответа и предсказанного, но только если тип сущности совпадает. Strict же равен 0.0 всегда, кроме случая полного совпадения, предсказанного с правильным ответом.

7. Эксперименты и результаты

В этом разделе будут описаны результаты тестирования моделей в различных конфигурациях на нашем наборе данных CodeReviewCommentsNER.

Для оценки качества моделей многоклассовой классификации мы использовали метрики Type F1 и Strict F1, описанные в разделе 6.2, а также обычный Token F1 – F1-мера по токенам (а так как используется IOB2 нотация, то классов не 16, а 31). Для всех метрик использовалось macro усреднение. Модели бинарной классификации оценивались метриками Точность, Полнота, F-мера по токенам.

Эксперименты проводились на нашем размеченном наборе данных с фиксированным разбиением на обучающую и тестовую выборки.

Измерения проводились на рабочей станции с операционной системой Ubuntu Server 20.04 LTS, процессором Intel® Core™ i7-6700 CPU @ 3.40GHz, 32GB RAM, и графическим ускорителем NVIDIA TITAN Xp с 12GB памяти. Для конфигурации Python использовался виртуальное окружение с Python 3.9.16, torch==2.0.1, transformers==4.27.4.

Размер группы примеров (batch) был выбран 16 – максимальный размер, при котором процесс обучения помещается в память графического ускорителя. Процесс обучения на 20 эпох занимает около 15 минут для конфигураций с одной моделью и около 20 – при использовании дополнительных моделей. Запуск на тестовом наборе из 457 комментариев занимает 4-9 секунд в зависимости от конфигурации.

7.1 Базовые модели

Далее представлены результаты сравнительных запусков обучения 4 базовых моделей (из пункта 4.1) и влияние предложенных нами улучшений, описанных в пункте 4.2, 4.3.

Как видно из табл. 5, улучшения, предложенные в пунктах 4.1, 4.2 (применение новой функции потерь отмечено как "+ focal loss", а использование обоих улучшений – как "+ enhance"), повышают качество классификации. Наибольшее увеличение показателей предложенные улучшения дают самой базовой roberta-base, повышая Token F1 на 16.3%, а метрики по сущностям в среднем на 7.7%. На примере codebert-base видно, что предложенные улучшения могут снизить результаты по некоторым метрикам (Strict F1 −3.2%). Исходя из результатов экспериментов видно, что использование претокенизации не всегда положительно влияет на качество классификации. Такое поведение можно объяснить необходимостью прописывать собственные правила вспомогательного разбиения на токены для каждой модели. Также можно отметить, что применение предобученной модели на текстах из соответствующей предметной области также повышают результаты: PretrainedCodeBert имеет лучше результаты, чем codebert-base, а BertOverflow превосходит roberta-base.

Табл. 5. Результаты тестирования моделей многоклассовой классификации
Table 5. Results of multi-class classification models testing

Модель	Token F1	Type F1	Strict F1
roberta-base	0.5913	0.6741	0.6145
roberta-base + focal loss	0.6692 (+13.2%)	0.7111 (+5.5%)	0.6543 (+6.4%)
roberta-base + enhance	**0.6879 (+16.3%)**	**0.7253 (+7.6%)**	**0.6628 (+7.8%)**
BertOverflow	0.6483	0.6963	0.6428
BertOverflow + focal loss	0.6983 (+7.7%)	0.7108 (+2.1%)	0.6606 (+2.7%)
BertOverflow + enhance	**0.7091 (+9.3%)**	**0.7116 (+2.2%)**	**0.6617 (+2.9%)**
codebert-base	0.6212	0.7414	0.7192
codebert-base + focal loss	0.6836 (+10%)	**0.7467 (+0.7%)**	**0.7193 (+0%)**
codebert-base + enhance	**0.6943 (+11.7%)**	0.7464 (+0.6%)	0.6962 (-3.2%)
PretrainedCodeBert	0.6414	0.7528	0.6954
PretrainedCodeBert + focal loss	0.7219 (+12.5%)	**0.7699 (+2.3%)**	**0.7246 (+4.2%)**
PretrainedCodeBert + enhance	**0.7342 (+14.4%)**	0.7695 (+2.2%)	0.7235 (+4.0%)

7.2 Вспомогательные бинарные модели

В данной секции описаны результаты экспериментов с обучением дополнительных моделей бинарной классификации, описанных в разделе 5. Для обучения бинарной модели использовались только два класса [«O», «Entity»] без BIO нотации. Качество считалось по метрикам Точность, Полнота, F-мера по токенам. Использовались те же данные, что и для многоклассовой классификации с заменой меток всех сущностей на «Entity».

Табл. 6 содержит результаты тестирования бинарных моделей. Все классификаторы, основанные на архитектуре BERT, имеют примерно равные результаты, с небольшим отрывом лучше показал себя классификатор с PretrainedCodeBert. Контекстно-независимая модель Recognizer показала также неплохой результат в 0.8081 F-меры, однако она сильно отстает от контекстно-зависимых классификаторов.

Табл. 6. Результаты тестирования моделей бинарной классификации
Table 6. Results of binary classification models testing

Модель	F-мера	Точность	Полнота
Recognizer	0.8081	0.7883	0.8292
roberta-base	0.9496	0.9462	0.9531
BertOverflow	0.9404	0.9328	0.9486
CodeBert	0.9414	0.9325	0.9514
PretrainedCodeBert	**0.9541**	**0.9498**	**0.9585**

7.3 Ансамбли моделей

Далее показаны результаты экспериментов объединения дополнительных моделей с основной, описанные в разделе 5.3.

В табл. 7 представлены результаты тестирования различных конфигураций объединения моделей. В качестве основной модели использовалась roberta-base с улучшениями, описанными в 4.2, 4.3 (в табл. 7 обозначена как Base), как показавшая наибольшую чувствительность к улучшениям в экспериментах из раздела 7.1. Дополнительные модели: контестно-зависимая Segmenter с базовой моделью roberta-base (в табл. 7 – Seg) и контекстно-независимая Recognizer (в табл. 7 – Reco).

Табл. 7. Результаты тестирования конфигураций ансамблей моделей
Table 7. Results of testing of model ensembles configurations

Модель	Token F1	Type F1	Strict F1
Base	0.6879	0.7253	0.6628
Base + Reco -Emb	0.6872	0.7053	0.6426
Base + Seg -Emb	0.6948	0.7251	0.6783
Base + Seg -Emb 500_31	0.6773	0.7199	0.6782
Base + Seg+Reco -Emb	0.6998	0.7204	0.6825
Base + Seg+Reco -Emb 500_31	0.6791	0.7142	0.6662
Base + Seg -T	0.7127	0.7342	0.6873
Base + Reco -T	0.6073	0.6439	0.6032
Base + Seg||Reco -T	0.6227	0.6424	0.6054
Base + Seg&&Reco -T	**0.7211**	**0.7395**	**0.6954**

Запуски с пометкой «-Emb» обозначают конкатенацию векторных представлений, генерируемых дополнительными моделями, с векторным представлением от Base. Пометка «500_31» означает наличие дополнительного промежуточного линейного слоя. В результате конкатенации векторных представлений их размер на каждый токен составлял (768 + 768 =) 1536 для «Base + Seg –Emb» и (768 + 768 + 20 =) 1556 для «Base + Seg+Reco –Emb».

Как видно из табл. 7, использование двух дополнительных моделей дает больший прирост качества, чем использование этих моделей по отдельности. Хотя по метрике Type F1 больший результат имеет запуск с использованием только Segmenter. Также можно отметить, что применение дополнительного промежуточного линейного слоя размерности 500 только ухудшает результаты классификатора.

Запуски с пометкой «-T» обозначают объединение моделей методом Classify-Trust, в котором вспомогательные модели принимаются оракулами, отвечающими на вопрос: "является ли данный токен не сущностью (принадлежит данный токен классу «O»)?". В случае положительного ответа оракула, токен отмечается как «O», иначе смотрится предсказание многоклассовой классификации. В случае применения двух бинарных моделей-оракулов их результаты можно либо объединять (если хоть один из оракулов дает положительный ответ, в табл. 7 обозначен как ||), либо пересекать (только если оба оракула дают положительные ответы, в табл. 7 обозначен как &&).

Исходя их данных табл. 7 можно сделать вывод, что, применяя метод Classify-Trust, также лучше использовать обе вспомогательные модели, чем каждую их них по отдельности. Стоит также отметить, что запуски Reco и Seg||Reco значительно хуже остальных, так как получается большое количество ложных срабатываний оракулов и значительное количество токенов неверно отмечаются как «O». В отличии от Seg&&Reco, где наоборот метка «O» оракулом выдавалась более точно.

В итоге можно отметить, что в наших экспериментах метод Classify-Trust показывает себя лучше, чем конкатенация векторных представлений. Лучший результат показал запуск "Base + Seg&&Reco -T", использующий обе вспомогательные модели.

Реализованные методы были протестированы на наборе данных из статьи SoftNER, а точнее тех данных, которые находятся в открытом доступе[8]. Среди опубликованных данных содержится обучающая и тестовая выборки с разметкой на 28 классов в IOB2 нотации. Также есть обучающий набор с сокращенным набором меток классов, однако тестового набора с соответствующей разметкой нет. Табл. 8 содержит результаты тестирования. Видно, что по

[8] https://github.com/jeniyat/StackOverflowNER/tree/master/resources/annotated_ner_data/

всем трем метрикам наилучший результат показал классификатор со всем предложенными улучшениями.

Табл. 8. Результаты тестирования реализованных классификаторов на наборе SoftNER
Table 8. Results of testing the implemented classifiers on the SoftNER dataset

Модель	Token F1	Type F1	Strict F1
roberta-base	0.326	0.4806	0.3923
roberta-base + enhance	0.4971	0.5472	0.4666
roberta-base + enhance -Emb	0.4836	0.5607	0.494
roberta-base + enhance -T	**0.5066**	**0.5716**	**0.4984**

Лучшая конфигурация для объединения моделей "Base + Seg && Reco -T" была применена ко все базовым моделям. Результаты тестирования приведены в табл. 9. В каждой секции табл. 9 представлены:

a) результаты базовой модели с улучшениями,
b) классификатор с дополнительными моделями методом Classify-Trust (обозначено как «-T»),
c) случай идеального оракула методом Classify-Trust (обозначено «-T*»).

Для всех классификаторов запуски, обозначенные «-T», показывают лучшие результаты, чем базовые.

В силу архитектуры классификатора с методом Classify-Trust достаточно легко провести эксперимент с идеальным оракулом бинарной классификации (в табл. 9 запуски, отмеченные «-T*»), где вместо текущих классификаторов будет использоваться истинное значение метки токена. Этим экспериментом можно получить максимально возможный прирост показателей от применения метода Classify-Trust. По всем базовым моделям идеальный оракул добавляет от 5 до 8 пунктов F-меры.

В итоге, наилучший результат показал классификатор с базовой предобученной моделью PretrainedCodeBert, с улучшениями, предложенными в пунктах 4.2, 4.3, и со вспомогательными моделями, примененными методом Classify-Trust, получив результаты в Token F1 = 0.7347, Type F1 = 0.7703, Strict F1 = 0.7290.

Табл. 9. Результаты тестирования ансамблей моделей
Table 9. Results of ensembles of models testing

Модель	Token F1	Type F1	Strict F1
roberta-base	0.6879	0.7253	0.6628
roberta-base -T	**0.7211**	**0.7395**	**0.6954**
*roberta-base -T**	*0.7464*	*0.7805*	*0.7327*
BertOverflow	**0.7091**	0.7116	0.6617
BertOverflow -T	0.6965	**0.7467**	**0.6999**
*BertOverflow -T**	*0.7732*	*0.7930*	*0.7586*
codebert-base	0.6943	0.7464	0.6962
codebert-base -T	**0.7172**	**0.7685**	**0.7175**
*codebert-base -T**	*0.7517*	*0.8153*	*0.7629*
PretrainedCodeBert	0.7342	0.7695	0.7235
PretrainedCodeBert -T	**0.7347**	**0.7703**	**0.7290**
*PretrainedCodeBert -T**	*0.7866*	*0.8263*	*0.7917*

7.4 Ошибки извлечения именованных сущностей

На рис. 4 изображена матрица несоответствий классификации лучшей из полученных моделей (PretrainedCodeBert). Для удобства просмотра в результатах тестирования были объединены классы B-Tag и I-Tag, таким образом получили матрицу 16 x 16, вместо 31 x 31. В матрице приведены нормированные значения по истинным меткам (по горизонтали сумма должна равняться единице, с точностью до округления).

При ручном анализе примеров, на которых ошибается классификатор, можно выделить несколько групп ошибок.

Ошибки в подтокенах. Стоит заметить, что классификация происходит на уровне подтокенов, которые получаются после токенизатора RoBERTa, и разработанные улучшения в этом направлении (пункт 4.2) не всегда помогают. Иногда изначальные длинные сущности разбиваются на множество маленьких подтокенов, каждый из которых модели необходимо классифицировать. Для борьбы с этой проблемой есть пара подходов: а) поменять способ обучения так, что в подсчете функции потерь и при итоговой классификации будет учитываться только первый подтокен; б) соединять итоговые предсказания модели по всем подтокенам путем некоторого голосования для единой оценки всей сущности.

Ошибки с классом «O». Они являются самыми распространенными: из 924 некорректно классифицированных токена 523 ошибки связаны с классом «O». Из рис. 4 можно заметить, что 28% «Extenral_tool» классифицировались как «O». Например, в комментарии "*I ran `make lint` and `make pylint` before pushing this commit*." токен "make" не распознался как сущность. Или в примере *"* return just empty ColumnNothing otherwise*" токен "empty" распознался как Keyword, хотя таковым не является.

Ошибки классификации между классами. Самой понятной можно назвать пару Variable и Function (81 ошибочно классифицированный токен). Так, например, в комментарии "*So I think I've got the current fastest implementation using bytesBefore and the underlying SWAR indexOf, thanks to you! ;-)*" токены "bytes" и "Before" модель классифицировала как Function, но правильным является класс Variable. Или в примере "*I pushed a change to use `np.newaxis` whenever possible.*" токены "new" и "axis" определились как Function при верном ответе Variable. Ещё одной часто путающейся парой является Variable и Value (53 взаимных ошибки). Так как к Value мы относили строковые константы, которые зачастую обрамлены кавычками, то возникают следующие ситуации: в комментарии "*'propertyIdWithAreaIds' I think it is better to be more clear with the name.*" propertyIdWithAreaIds классифицировался как Value, но является именем переменной.

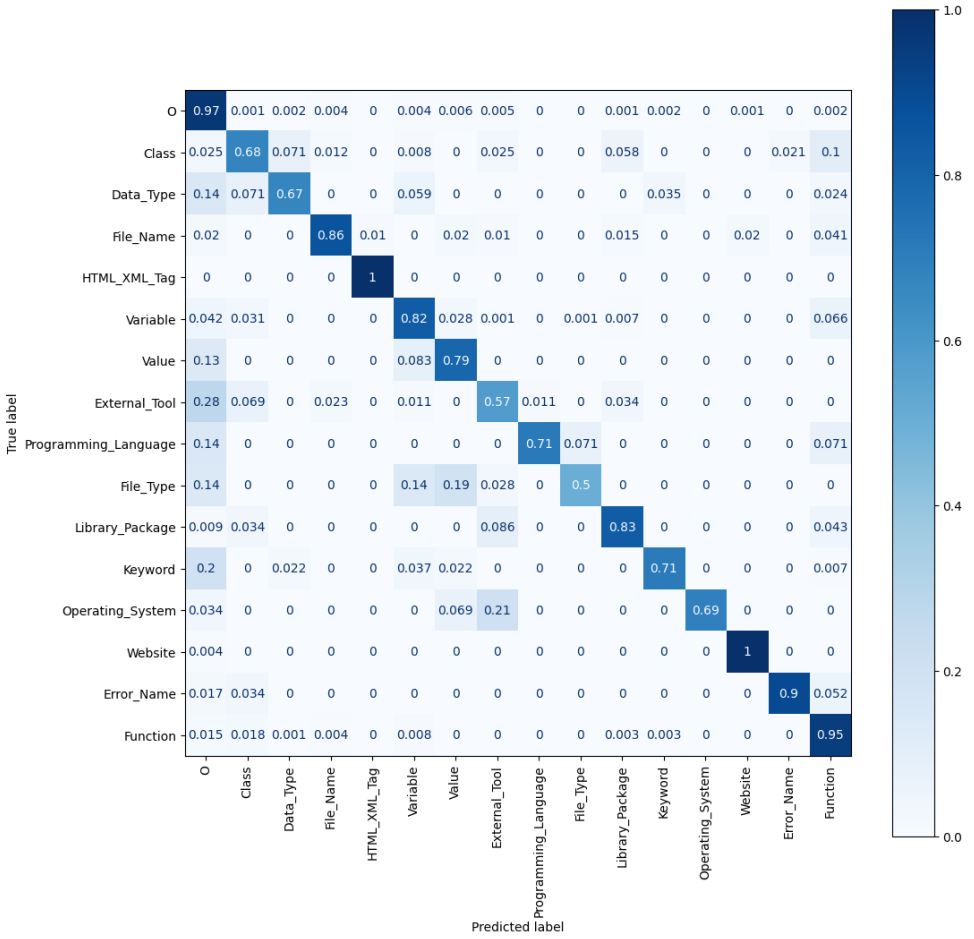

Рис. 4. Матрица несоответствия.
Fig. 4. Confusion matrix.

8. Заключение

В данной работе были исследованы методы повышения качества решения задачи извлечения именованных сущностей в применении к области разработки программного обеспечения. Проведено сравнение реализованных подходов на новом собранном и размеченном вручную наборе из 3000 рецензий, оставленных пользователями на изменения в исходном коде. Количество распознаваемых классов – 15. Предложенные и реализованные методы повышения качества классификации позволили поднять результаты по разным метрикам на 8-13 пунктов F-меры. Ручной запуск показал достаточно хорошее качество классификации для применения инструмента в прикладных задачах. Разработанный инструмент поиска и классификации сущностей позволит лучше решать задачи классификации/кластеризации и семантического поиска комментариев из предметной области.

Список литературы / References

[1]. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and

Short Papers). pp. 4171–4186. Association for Computational Linguistics, Minneapolis, Minnesota (Jun 2019). DOI:10.18653/v1/N19-1423.

[2]. Tabassum, J., Maddela, M., Xu, W., Ritter, A.: Code and named entity recognition in StackOverflow. In: Jurafsky, D., Chai, J., Schluter, N., Tetreault, J. (eds.) Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. pp. 4913–4926. Association for Computational Linguistics, Online (Jul2020). DOI:10.18653/v1/2020.acl-main.443.

[3]. Rahul Sharnagat, Named Entity Recognition: A Literature Survey, June 30, 2014 https://www.cfilt.iitb.ac.in/resources/surveys/rahul-ner-survey.pdf.

[4]. Bikel, D.M., Schwartz, R. & Weischedel, R.M. An Algorithm that Learns What's in a Name. Machine Learning 34, 211–231 (1999). DOI:10.1023/A:1007558221122.

[5]. John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In Proceedings of the Eighteenth International Conference on Machine Learning (ICML '01). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 282–289.

[6]. Andrew McCallum and Wei Li. 2003. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003 - Volume 4 (CONLL '03). Association for Computational Linguistics, USA, 188–191. DOI: 10.3115/1119176.1119206.

[7]. Cortes, C., Vapnik, V. Support-vector networks. Mach Learn 20, 273–297 (1995). DOI:10.1007/BF00994018.

[8]. McNamee, P., & Mayfield, J. (2002). Entity Extraction without Language-Specific Resources. Conference on Computational Natural Language Learning.

[9]. Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural Architectures for Named Entity Recognition. In Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 260–270, San Diego, California. Association for Computational Linguistics. DOI: 10.18653/v1/N16-1030.

[10]. Batbaatar E, Ryu KH. Ontology-Based Healthcare Named Entity Recognition from Twitter Messages Using a Recurrent Neural Network Approach. International Journal of Environmental Research and Public Health. 2019; 16(19):3628. DOI:10.3390/ijerph16193628.

[11]. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Curran Associates Inc., Red Hook, NY, USA, 6000–6010.

[12]. Cedric Lothritz, Kevin Allix, Lisa Veiber, Tegawendé F. Bissyandé, and Jacques Klein. 2020. Evaluating Pretrained Transformer-based Models on the Task of Fine-Grained Named Entity Recognition. In Proceedings of the 28th International Conference on Computational Linguistics, pages 3750–3760, Barcelona, Spain (Online). International Committee on Computational Linguistics. DOI: 10.18653/v1/2020.coling-main.334.

[13]. Tikhomirov, M., Loukachevitch, N., Sirotina, A., Dobrov, B. (2020). Using BERT and Augmentation in Named Entity Recognition for Cybersecurity Domain. In Natural Language Processing and Information Systems. NLDB 2020. Lecture Notes in Computer Science, vol 12089. Springer, Cham. https://doi.org/10.1007/978-3-030-51310-8_2.

[14]. Malik, G., Cevik, M., Bera, S., Yildirim, S., Parikh, D., & Basar, A. (2022). Software requirement specific entity extraction using transformer models. Proceedings of the Canadian Conference on Artificial Intelligence. DOI:10.21428/594757db.9e433d7c.

[15]. D. Ye, Z. Xing, C. Y. Foo, Z. Q. Ang, J. Li and N. Kapre, "Software-Specific Named Entity Recognition in Software Engineering Social Content," 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Osaka, Japan, 2016, pp. 90-101, DOI:10.1109/SANER.2016.10.

[16]. Liu Zhuang, Lin Wayne, Shi Ya, and Zhao Jun. 2021. A Robustly Optimized BERT Pre-training Approach with Post-training. In Proceedings of the 20th Chinese National Conference on Computational Linguistics, pages 1218–1227, Huhhot, China. Chinese Information Processing Society of China.

[17]. Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In Findings of the Association for Computational Linguistics: EMNLP 2020, pages 1536–1547, Online. Association for Computational Linguistics. DOI: 10.18653/v1/2020.findings-emnlp.139.

[18]. T. -Y. Lin, P. Goyal, R. Girshick, K. He and P. Dollár, "Focal Loss for Dense Object Detection," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 42, no. 2, pp. 318-327, 1 Feb. 2020, DOI:10.1109/TPAMI.2018.2858826.

[19]. Anderson, P.E., Reo, N.V., DelRaso, N.J. et al. Gaussian binning: a new kernel-based method for processing NMR spectroscopic data for metabolomics. Metabolomics 4, 261–272 (2008). DOI:10.1007/s11306-008-0117-3.

[20]. Charlie Xu, Solomon Barth, Zoe Solis. Applying Ensembling Methods to BERT to Boost Model Performance. Stanford University. Accessed at: 01.11.2023.

[21]. Naushad UzZaman, Hector Llorens, Leon Derczynski, James Allen, Marc Verhagen, and James Pustejovsky. 2013. SemEval-2013 Task 1: TempEval-3: Evaluating Time Expressions, Events, and Temporal Relations. In Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013), pages 1–9, Atlanta, Georgia, USA. Association for Computational Linguistics.

Информация об авторах / Information about authors

Владимир Владимирович КАЧАНОВ – аспирант. Сфера научных интересов: машинное обучение, программная инженерия.

Vladimir Vladimirovich KACHANOV – postgraduate student. Research interests: machine learning, software engineering.

Ариана Сергеевна ХИТРОВА – бакалавр. Сфера научных интересов: машинное обучение, обработка естественного языка.

Ariana Sergeevna KHITROVA – bachelor. Research interests: machine learning, natural language processing.

Сергей Игоревич МАРКОВ – специалист, старший научный сотрудник. Сфера научных интересов: статический анализ кода, динамический анализ кода, программная инженерия, машинное обучение.

Sergei Igorevich MARKOV – specialist, senior researcher. Research interests: static program analysis, dynamic program analysis, software engineering, machine learning.