Санкт-Петербургский государственный университет

Факультет Прикадной Математики - Процессов Управления
Кафедра технологии программирования

Буланина Екатерина Дмитриевна

Извлечение данных из учебного контента

Выпускная квалификационная работа бакалавра

Научный руководитель:
к. ф.-м. н., доцент Добрынин В. Ю.

Рецензент:
разработчик ЗАО «БИОКАД» Чуриков Н. С.

Заведующий кафедрой:
к. т. н., доцент Блеканов И. С.

Санкт-Петербург
2019

SAINT-PETERSBURG STATE UNIVERSITY

Applied Mathematics and Control Processes Faculty
Programming Technology Chair

Ekaterina Bulanina

Educational data mining

Graduation Thesis

Scientific supervisor:
Vladimir Dobrynin

Reviewer:
Nikita Churikov

Chairhead:
Ivan Blekanov

Saint-Petersburg
2019

Оглавление

Введение 4

1. Постановка задачи 5

2. Обзор 6
   2.1. Терминология 6
   2.2. Литература 6

3. Архитектура 8

4. Извлечение и обработка данных 9
   4.1. Формат данных 9
   4.2. Извлечение данных 10
   4.3. Предобработка 10

5. Построение модели 13
   5.1. Входные данные и необходимые метрики 13
   5.2. Предсказание тем 14

6. Построение и расширение графа 18
   6.1. Обозначения 18
   6.2. Построение графа 18
   6.3. Расширение графа 19

7. Апробация 20
   7.1. Построение графа тем 20
   7.2. Привязка заданий к темам 20

Заключение 22

Список литературы 23

Введение

За последние десятилетия электронная промышленность достигла колоссальных масштабов, а компьютеры и информационные технологии повсеместно вошли в быт людей. Человечество использует компьютеры для автоматизации всех сфер деятельности: производство и сельское хозяйство, логистика и транспорт, биология и медицина, а также образование.

Если говорить об образовании, то в настоящее время существует множество систем управления обучением (Learning Management System), одним из примеров которых в частности является известный каждому студенту нашего университета BlackBoard [3]. Такие системы управления образованием предоставляют своим ученикам возможность удалённого обучения онлайн и контроля своих знаний при помощи интерактивных заданий.

LMS отслеживают различную информацию о своих пользователях: расписание и список курсов, которые посещает студент; оценки и баллы, полученные в результате выполнения электронных заданий. Всё это — колоссальные объемы данных, которые потенциально можно использовать для улучшения образовательного процесса. Образовательные данные зачастую детализированны и точны, а их объем очень велик. Анализ данных в сфере образования включает в себя исследования, методы и инструменты, предназначенные для автоматического извлечения данных, связанных с учебной деятельностью людей и создаваемых в образовательных учреждениях.

К примеру, анализ данных в LMS может выявить связь между этапами обучения, к которым студент получил доступ в течение курса, и их итоговой оценкой курса. Аналогично, анализ данных стенограммы студента может выявить связь между оценкой студента по какому-то курсу и его решением изменить свою специализацию. Такая информация дает представление о процессе обучения, что позволяет учащимся и учителям принимать обоснованные решения о том, как взаимодействовать, предоставлять и управлять образовательными ресурсами.

1. Постановка задачи

Целью данной работы является автоматическая разметка существующего текстового учебного материала для его дальнейшего переиспользования.

Для достижения этой цели были поставлены следующие задачи:

1. Провести анализ данных, предоставленных ООО «ИнтеллиДжей Лабс» [8]
    - Векторизовать решения заданий
    - Построить множества тем для каждого задания с учетом их решений
2. Реализовать алгоритм автоматического построения графа тем по учебному контенту [10]
    - Построение начальной версии графа по данным из Wikidata
    - Построение финальной версии графа по начальной в соответствии с решениями заданий
    - Провести апробацию

2. Обзор

2.1. Терминология

В качестве основного источника для изучения методов машинного обучения и анализа данных я использовала книгу К. Маннинга «Введение в информационный поиск» [2]. В этом учебнике рассматривается современный подход ко всем аспектам проектирования и внедрения систем сбора, индексирования и поиска документов, методы оценки систем и использование методов машинного обучения в наборах текстов. В частности, в этой книге описываются алгоритмы стемминга и лемматизации, которые используются в данной работе.

Для ознакомления с областью применения методов машинного обучения и интеллектуального анализа данных в образовательной сфере была использована книга «Handbook of Educational Data Mining» [7]. Данная книга содержит исчерпывающий обзор основных методов анализа образовательных данных, а также более двадцати реальных случаев применения этих методов на практике.

2.2. Литература

В большинстве случаев знания студентов оцениваются в соответствии с какой-то измерительной шкалой, например от 2 до 5 баллов за экзамен или от 0 до 100 баллов за тест. Авторы работы «Knowledge Spaces and Learning Spaces» [6] дают представление о том, как дать оценку умениям и навыкам студентов иным образом. Основная идея заключается в том, чтобы воспринимать полученные человеком знания как набор освоенных им «концепций», принимая при этом во внимание предыдущие результаты. Знания при этом представляются нелинейно в виде графа, в котором эти темы или концепции расположены с учётом трудности их понимания и взаимосвязи друг с другом. Так, в вершинах графа находятся концепции, а ребра графа показывают порядок их изучения. Работа представляет большой интерес, потому что в ней формально описываются способы оценки полученных знаний при

помощи распределения вероятностей правильных ответов на тестовые вопросы. Подобная методика могла быть использована в адаптивных онлайн-курсах, один из которых используется в моей работе.

Вдохновляющим примером разметки учебного материала послужила статья «Toward the Automatic Labeling of Course Questions for Ensuring their Alignment with Learning Outcomes» [17], опубликованная в 2017 году на конференции, посвященной анализу данных в сфере образования. В статье была рассмотрена задача классификации экзаменационных вопросов в соответствии с таксономией Блума [16]. Данная работа интересна тем, что решает задачу разметки контента при помощи метода векторизации TF-IDF, который также будет применяться в ходе моей работы.

В работе «An Automatic Knowledge Graph Creation Framework from Natural Language Text» [9] описывается универсальный подход для расширения графа тем по произвольному тексту на естественном языке. Для этого из текста извлекаются «тройки», элементы которых представляют из себя два объекта и отношение между ними. Затем авторами работы презентуется векторная метрика оценивания схожести таких «троек». Расширения графа новыми темами осуществляется с помощью алгоритма кластеризации таких троек и определения степени соответствия их какому-либо существующему элементу графа.

3. Архитектура

Работа над задачей была организована согласно принципам методологии CRISP-DM (CRoss Industry Standard Process for Data Mining) [15] и представляла из себя три этапа: сбор и исследование данных, подготовка данных и моделирование. Моделирование включает в себя построение модели машинного обучения для привязки задач к темам и создание графа тем по результатам.

Для реализации поставленной задачи использовался язык программирования Python и библиотека алгоритмов машинного обучения Scikit-Learn [14], а также язык описания графов DOT [4].

Исходный код разделён на три независимых части: обработка данных, моделирование и построение графа. Обработка данных происходит в два этапа: приведение к единому виду и экстракция необходимых для моделирования данных, затем подготовка данных. Для моделирования использовалось несколько алгоритма машинного обучения для сравнения полученных результатов и определения наиболее точного метода.

<тут картинка>

4. Извлечение и обработка данных

4.1. Формат данных

Каждый курс на платформе Stepik состоит из *степов* — шагов, которые могут представлять из себя просмотр видеоматериалов, решение проверочных заданий или работу с текстом. *Адаптивные* курсы подбирают задания индивидуально для каждого пользователя платформы, основываясь на уровне знаний, личной оценке той или иной задачи пользователем, а также затраченном времени. Курс «Adaptive Python» [1], для которого необходимо построить граф тем, представляет из себя адаптивный тренажёр для обучению языку программирования Python [11].

Для извлечения необходимых данных курса «Adaptive Python» было использовано Stepik REST API [1] — программный интерфейс платформы Stepik, позволяющий автоматически обращаться к её ресурсам. Компанией JetBrains были предоставлены данные о степах и попытках (как удачных, так и неудачных) решения задач этих степов пользователями платформы.

В предоставленном наборе данных каждому степу соответствовал набор *тем* — некоторых «тэгов», описывающих содержательную часть задания (к примеру, «Целые числа» или «ООП»). Помимо этого, для каждой темы был указан её идентификатор в базе знаний «Wikidata» [18]. В данных о попытках решения для каждого степа помимо кода решения на языке Python было также указано, являлось ли решение верным. Данные были представлены в формате рабочей книги Microsoft Excel (.xlsx) [2] и текстовом формате представления таблицы Comma-Separated Values (.csv) [3]. Для обработки этих форматов были использованы библиотеки openpyxl [20] и pandas [21].

[1] REpresentational State Transfer Application Programming Interface

[2] формат данных программы для работы с электронными таблицами

[3] значения колонок таблицы разделены запятой

4.2. Извлечение данных

Полученные данные содержали несущественные для решения поставленной задачи значения и были представлены в разных форматах, поэтому для удобства было принято решение в дальнейшем использовать формат представления JSON [4].

Формулировки заданий курса было решено исключить из рассмотрения, поскольку они зачастую слишком абстрактны и от студента курса требуется самостоятельно составить требования к решению. Более полезным для привязки того или иного степа курса к набору тем будет анализ кода решения, полученного платформой от пользователем. Поэтому из предоставленных данных были извлечены попытки решения пользователей того или иного задания, а также существующий список тем, соответствующих заданию.

4.3. Предобработка

Решения пользователей представляют из себя тексты кода на языке Python. Для построения модели было решено взять только верные попытки решения задач. Коды этих попыток были протестированы внутренней системой Stepik и удовлетворяют условиям заданий. К тому же, среди неверных попыток оказывается слишком много пустых или синтаксически некорректных блоков кода.

Поскольку степы адаптивного курса представляют из себя классические задачи по программированию, коды правильных решений в большинстве своём по структуре оказываются очень похожи. Однако они всё же отличаются названиями переменных, а также наличием или отсутствием в них комментариев, которые могут быть самыми разнообразными. В ходе экспериментов было обнаружено, что различное именование переменных отрицательно влияет на результат моделирования: одну и ту же целочисленную переменную-счетчик разные студенты могут назвать x, a, i, k, n; переменную для размера массива называют

[4] JavaScript Object Notation — основанный на языке программирования JavaScript формат обмена данными, преимуществом которого является удобство чтения как человеком, так и компьютером

size, *length*, *count*.

Поэтому было принято решение предобработать коды таким образом, чтобы очистить их от комментариев, а также переименовать все переменные единым образом. Процесс предобработки состоит из трех этапов: синтаксический анализ, унификация переменных и pretty-printing.

Синтаксический анализ

На первом этапе для исходного кода строится его синтаксическое дерево с помощью парсера из библиотеки ast [19].

Унификация переменных

Далее для каждого синтаксического дерева запускается процесс унификации переменных. Он состоит из двух рекурсивных обходов дерева, которые представляют из себя реализации стандартного для деревьев шаблона проектирования «Посетитель» [5]. Сначала первый посетитель VariablesCollector обходит дерево, собирает все имена переменных и назначает каждому имени свой номер. Затем второй посетитель VariablesRenamer переименовывает все переменные, назначая им имена i_1, i_2 и т.д. в соответствии с номером.

Pretty-printing

Pretty-printing [12] — это форматирование исходного кода в соответствии с некоторым заданным стилем. Модели, использованные в работе, оперируют с текстами, а не деревьями, поэтому по итогу предобработки мы должны снова получить текст программы. Как и в случае с разными названиями переменных, модели показывают худшие результаты на текстах с разными стилями оформления (например, с разной табуляцией, отступами и переносами или отсутствием переносов строк). Поэтому каждое синтаксическое дерево печатается в некотором едином стиле. Эта печать реализуется с помощью pretty-printer — ещё одного

посетителя, который обходит все вершины и печатает текст для каждой из них.

5. Построение модели

5.1. Входные данные и необходимые метрики

Для обучение модели классификации по нескольким меткам (Multi-label classification [5]) извлечённые данные были разделены на тренировочные и тестовые. Для этого данные были перемешаны с помощью генератора случайных чисел, затем 70% данных были отмечены как тренировочные, а 30% будут затем использоваться для проверки обученной модели.

Рассмотрим метрики для определения точности классификации.

Accuracy

Простейшая метрика, представляющая из себя долю верно классифицированных объектов среди всего размера обучающей выборки.

$$\text{Accuracy} = \frac{\{\text{correct}\}}{\{\text{train sample}\}}$$

Precision and recall

Точность классификатора определяется как доля верно классифицированных объектов среди всех объектов, отнесённых классификатором к классу k:

$$\text{Precision}_k = \frac{|\{\text{relevant}\} \cap \{\text{retrieved}\}|}{|\{\text{retrieved}\}|}$$

Точность отвечает на вопрос: сколько из всех выбранных объектов действительно принадлежат классу k?

Полнота определяется как доля верно классифицированных объектов среди всех объектов класса k в тестовой выборке:

$$\text{Recall}_k = \frac{|\{\text{relevant}\} \cap \{\text{retrieved}\}|}{|\{\text{relevant}\}|}$$

[5] задача классификации, при которой каждому объекту сопоставляется один или несколько классов

Полнота отвечает на вопрос: сколько из всех действительно принадлежащих классу k объектов были выбраны?

F1-score

F1-мера определяется как среднее гармоническое между точностью и полнотой класса k:

$$\text{F1-score}_k = \frac{2 \cdot \text{Precision}_k \cdot \text{Recall}_k}{\text{Precision}_k + \text{Recall}_k}$$

Как видно, F1-мера стремится к нулю, если точность или полнота стремится к нулю. При этом одинаковый вес придаётся обоим этим метрикам.

Общая формула для оценки Fβ -score, которая придаёт больший вес полноте на порядок β:

$$\text{F}\beta_k = \frac{(1 + \beta^2) \cdot \text{Precision}_k \cdot \text{Recall}_k}{\beta^2 \cdot \text{Precision}_k + \text{Recall}_k}$$

Support

Метрика поддержки показывает для класса k количество объектов, ему принадлежащих. Данное значение служит коэффициентом для вычисления средних значений Precision и Recall по всем классам.

5.2. Предсказание тем

Для того, чтобы привязать уроки онлайн-курса к темам, используется векторная модель представления их решений. Предсказанные на данном этапе темы впоследствии будут соответствовать некоторому пути в графе тем.

TF-IDF представление

Для векторизации решений заданий курса используется TF-IDF мера. Введем необходимые обозначения и определения:

- *Документ* — текстовые данные, состоящие из набора лексем. В нашем случае документом является текстовое представление решения пользователя.

- *TF (term frequency)* — отношение числа вхождения определенной лексемы n_i к общему количеству лексем в документе:

$$\mathrm{TF} = \frac{n_i}{\sum_k n_k}$$

- *IDF (inverse document frequency)* — обратная документная частота, определяемая следующим образом:

$$\mathrm{IDF} = \log \frac{N}{df}$$

  N — количество документов в коллекции
  df — количество документов, содержащиз n_i.

- *TF-IDF* — мера, определяющая вес каждой лексемы в каждом документе:

$$\text{TF-IDF} = \mathrm{TF} \cdot \mathrm{IDF}$$

  Мера *TF-IDF* уменьшается, когда лексема встречается в большом количестве документов, и увеличивается, если лексема встречается часто в небольшом количестве документов.

Задача привязки заданий к темам рассматривается в работе как задача классификации по k категориям. Для этого используются представленные ниже классификаторы, результаты работы которых будут сопоставлены в соответствии с приведенными выше метриками.

One vs Rest

Задача классификации данных (в нашем случае заданий) по k категориям может быть сведена к k задачам бинарной классификации.

Для каждого из классов обучается свой классификатор, определяющий принадлежность либо не принадлежность классу. Так работает классификатор One vs Rest. Помимо вычислительной эффективности и простоты имплементации, преимуществом данной классификации является легкость интерпретации полученных данных.

K Neighbors

Алгоритм k-ближайших соседей — простейший алгоритм классификации и регрессии для задач обучения с учителем. Классификатор использует метрику (например, стандартную в евклидовом пространстве) для вычисления расстояния между объектами выборки, и определяет класс каждого из объектов как наиболее часто встречающийся среди k его ближайших по метрике соседей. Несмотря на недостатки алгоритма, например необходимость настройки параметров, было принято решение попробовать его как классический и наиболее простой вариант.

Multinomial Naive Bayesian Classifier

Мультиномиальный наивный байесовский классификатор представляет из себя вероятностную модель с применением теоремы Байеса в предположении о независимости. Классификатор определяет искомый класс объекта как класс, имеющий максимальную *апостериорную* вероятность. Для избежания потери значимых разрядов при умножении зачастую используют сумму логарифмов вероятностей вследствие монотонности логарифмической функции и равенства $\log(x \cdot y) = \log(x) + \log(y)$.

Decision Tree

Алгоритм решающих деревьев — логический алгоритм классификации, задающийся бинарным деревом, используется во множестве задач регрессии и классификации. Каждому листу дерева соответствует класс, и при классификации того или иного объекта алгоритм проходит

весь путь от корня до некоторого листа, определяя тем самым класс. Для построения алгоритма необходимо выбрать признак разделения и минимальный порог (если необходим).

Random Forest

Случайный лес — ансамбль решающих деревьев, эффективно используемый для обработки данных с большой размерностью. Основными преимуществами данного метода машинного обучения являются высокая масштабируемость и параллелизуемость. Один из его недостатков — склонность к переобучению на зашумлённых данных.

6. Построение и расширение графа

6.1. Обозначения

Введем необходимые обозначения и определения:

- $\mathcal{T}$ — множество всех тем онлайн-курса.
- *Урок* — множество тем $\{t_i\}$ одного занятия онлайн-курса.
- $\mathcal{L}$ — множество всех уроков онлайн-курса.
- G — направленный граф тем; его вершинами являются элементы множества $\mathcal{T}$, а дуги соответствуют порядку изучения тем
- $count(t_i) = |\{l \in \mathcal{L} : t_i \in l\}|$ – количество уроков с этой темой
- $neighbors(t_i)$ – соседние темы t_i

6.2. Построение графа

Сначала необходимо получить первое приближение G_0 искомого графа G. Вершинами обоих графов являются исходные темы онлайн-курса. Для построения дуг используются данные из Wikidata: с помощью запросов на языке SPARQL [13] проверяется, как темы соотносятся между собой. Тема t_1 считается предком темы t_2, если для них имеет место отношение "instance of", "subclass of" или "part of".

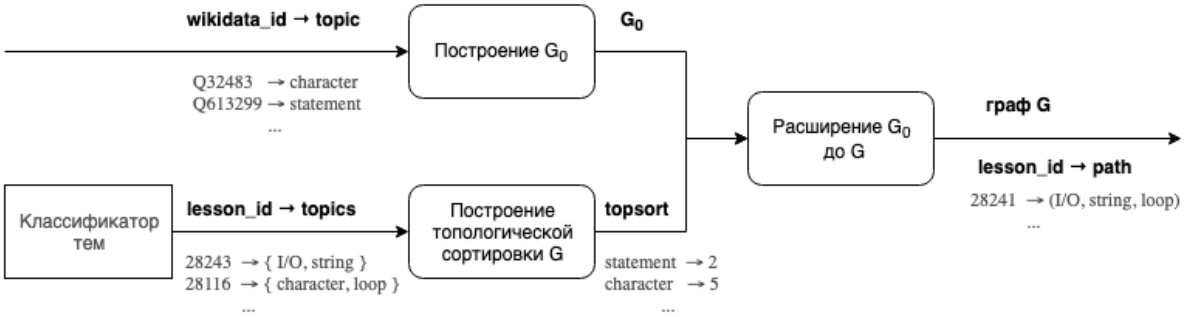
В рамках расширения графа тем предполагается, что каждый урок соответствует некоторому пути в искомом графе, то есть

$$\forall l \in \mathcal{L} \; \exists \; (t_1, t_2, ..., t_n) \in G : l = \{t_1, t_2, ..., t_n\}$$

Основная идея построения графа состоит в том, чтобы сначала найти топологическую сортировку расширенного графа, а затем по ней построить сам расширенный граф.

6.3. Расширение графа

Рис. 1: Схема работы алгоритма



1. Составить словарь $\forall t_i \in \mathcal{T}\ \{t_i : count(t_i)\}$

2. В каждом уроке отсортировать темы t_i по убыванию $count(t_i)$

3. Составить словарь $\forall t_i \in \mathcal{T}\ \{t_i : neighbors(t_i)\}$; в дальнейшем считаем, что $neighbors(t_i)$ являются предками t_i

4. Отсортировать уроки по убыванию количества тем в них

5. Пронумеровать уроки уровнями в соответствии с топологической сортировкой: $\forall t_i \in \mathcal{T} : level[t_i] \in [0..N]$

   Теперь мы знаем уровни вершин (т.е. длины путей от одной из корневых вершин). Осталось добавить в граф ребра в соответствии с уровнями и уроками (как упоминалось выше, каждый урок соответствует некоторому пути)

6. Проходим по всем уровням от 0 до последнего: для каждой темы добавляем ребра из его предков, если тема ещё недостижима из предка.

   Достижимость вершины проверяется с помощью поиска в глубину или алгоритма Флойда-Уоршалла.

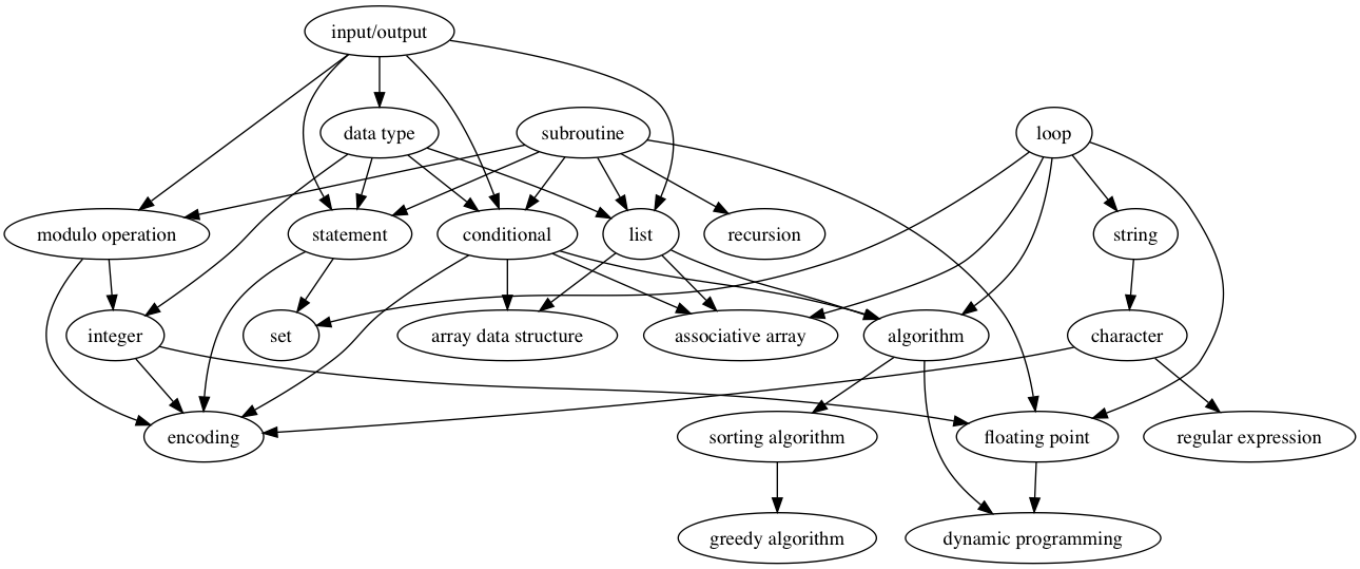
В результате выполнения алгоритма заданный граф G_0 расширяется новыми дугами.

7. Апробация

7.1. Построение графа тем

Был построен граф тем, каждой из вершин которого соответствует задание курса. Для наглядности исходный граф был значительно сокращён. Полноразмерный вариант графа, а также соответствие заданий и путей в нём, приведены вместе с исходным кодом проекта на ресурсе GitHub [6].

Рис. 2: Сокращённый вариант построения графа тем

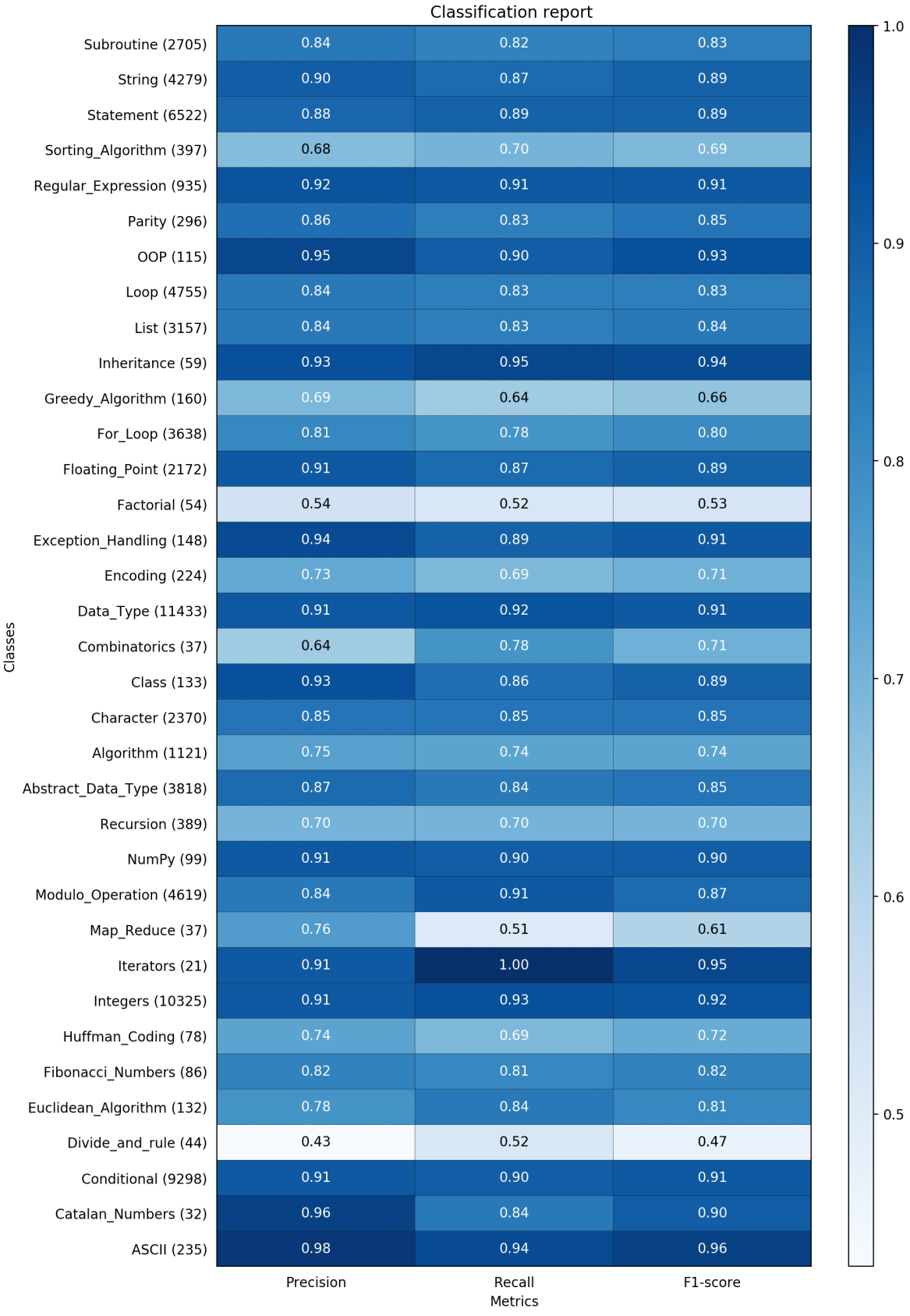


7.2. Привязка заданий к темам

В ходе апробации было проведено сравнение нескольких перечисленных методов классификации по темам. Для этого использовалась композиция метрик Precision, Recall и F1-score. Поскольку наилучшим образом справился с задачей метод Decision Tree, ниже на Рис. 3 приведена таблица значений метрик по каждому из классов, определенных при помощи данного классификатора.

[6] `https://github.com/shabatar/knowledge-spaces`

Рис. 3: Результат Decision Tree Classifier для данных.



Заключение

В ходе данной работы были получены следующие результаты:

1. Проведён анализ предметной области. Сделан обзор аналогичных существующих решений, используемых в них методов анализа данных и машинного обучения, а также метрик для оценки результатов их работы.

2. Реализован алгоритм многоклассовой классификации извлечённых данных. Он затем используется для привязки задач онлайн-курса к той или иной вершине в графе тем. Были протестированы варианты реализации алгоритма с различными классификаторами. В результате наилучшим образом себя показал метод решающего дерева.

3. Реализован метод автоматического построения графа тем по учебному контенту:

    - Используя результаты работы классификатора на предыдущем шаге, задачи курса привязываются к темам в графе.
    - Используя идею топологической сортировки, реализован алгоритм расширения полученного графа новыми вершинами.

Список литературы

[1] Adaptive Python. — https://stepik.org/course/568/. — Дата обращения: 07.04.2019.

[2] Alpaydin Ethem. Introduction to Machine Learning. Adaptive Computation and Machine Learning. — 3 edition. — Cambridge, MA : MIT Press, 2014. — ISBN: 978-0-262-02818-9.

[3] Blackboard Learn. — https://www.blackboard.com/about-us. — Дата обращения: 07.04.2019.

[4] DOT language. — http://www.graphviz.org/about/. — Дата обращения: 07.04.2019.

[5] Design Patterns: Elements of Reusable Object-oriented Software / Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. — Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1995. — ISBN: 0-201-63361-2.

[6] Doignon Jean-Paul, Falmagne Jean-Claude. Knowledge Spaces and Learning Spaces. — 2015. — 11.

[7] Handbook of educational data mining / C. Romero, S. Ventura, M. Pechenizkiy, R.S.J. Baker, de. Chapman and Hall/CRC data mining and knowledge discovery series. — CRC Press, 2011. — ISBN: 978-1-4398-045-7-5.

[8] IntelliJ Labs. — https://www.jetbrains.com/company/. — Дата обращения: 07.04.2019.

[9] KERTKEIDKACHORN Natthawut, ICHISE Ryutaro. An Automatic Knowledge Graph Creation Framework from Natural Language Text // IEICE Transactions on Information and Systems. — 2018. — 01. — Vol. E101.D. — P. 90–98.

[10] Knowledge Graphs: In Theory and Practice / Nitish Aggarwal, Saeedeh Shekarpour, Sumit Bhatia, Amit Sheth. — 2017. — 11.

[11] Python. — https://www.python.org/about/. — Дата обращения: 07.04.2019.

[12] Rubin L. F. Syntax-Directed Pretty Printing—A First Step Towards a Syntax-Directed Editor // IEEE Transactions on Software Engineering. — 1983. — March. — Vol. SE-9, no. 2. — P. 119–127.

[13] SPARQL Query Language. — https://www.w3.org/TR/rdf-sparql-query/. — Дата обращения: 07.04.2019.

[14] Scikit-Learn. — http://scikit-learn.org/. — Дата обращения: 07.04.2019.

[15] Shearer C. The CRISP-DM model: the new blueprint for data mining // J Data Warehouse. — 2000. — 01. — Vol. 5. — P. 13–22.

[16] Taxonomy of educational objectives. The classification of educational goals. Handbook 1: Cognitive domain / B. S. Bloom, M. B. Engelhart, E. J. Furst et al. — New York : Longmans Green, 1956.

[17] Supraja S., Hartman Kevin, Tatinati Sivanagaraja, Khong Andy W. H. Toward the Automatic Labeling of Course Questions for Ensuring their Alignment with Learning Outcomes. — 2017. — Access mode: http://educationaldatamining.org/EDM2017/proc_files/papers/paper_44.pdf.

[18] Wikidata Knowledge Base. — https://www.wikidata.org/wiki/Wikidata:Introduction. — Дата обращения: 07.04.2019.

[19] ast Python library. — https://docs.python.org/3/library/ast.html. — Дата обращения: 07.04.2019.

[20] openpyxl Python library. — https://openpyxl.readthedocs.io/en/stable/. — Дата обращения: 07.04.2019.

[21] pandas Python library. — https://pandas.pydata.org/about.html. — Дата обращения: 07.04.2019.