МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

РАБОТА ПРОВЕРЕНА

Рецензент
Ст. преподаватель кафедры ММОМ
ФГБОУ ВО «ЮУрГГПУ», к.ф.-м.н.

_____ А.М. Шарафутдинова

“___”_____ 2020 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой,
д.ф.-м.н., профессор

_____ Л.Б. Соколинский

“___”_____ 2020 г.

Разработка приложения для подбора моделей машинного обучения на основе подхода AutoML

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2020.308-064.ВКР

Научный руководитель,
к.ф.-м.н., доцент кафедры СП
_____ С.А. Иванов

Автор работы,
студент группы КЭ-403
_____ М.А. Щукин

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
“___”_____ 2020 г.

Челябинск-2020

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
03.02.2020

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-403
Щукину Максиму Александровичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 24.04.2020 № 627)
Разработка приложения для подбора моделей машинного обучения на основе подхода AutoML.
2. **Срок сдачи студентом законченной работы:** 06.06.2020.
3. **Исходные данные к работе**
3.1. Google. Machine Learning Crash Course | Google Developers // Google.Com. 2018. С. 1–49.
3.2. Guyon I. и др. Analysis of the AutoML Challenge Series 2015–2018. 2019.
4. **Перечень подлежащих разработке вопросов**
4.1. Обзор научной литературы и существующих решений по данной теме.
4.2. Определение требований к приложению.
4.3. Проектирование архитектуры приложения.
4.4. Реализация приложения.
4.5. Тестирование приложения.
5. **Дата выдачи задания:** 03.02.2020.

Научный руководитель,
к.ф.-м.н., доцент кафедры СП С.А. Иванов

Задание принял к исполнению М.А. Щукин

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Основные определения

Машинное обучение (англ. Machine learning, ML) – это дисциплина, изучающая алгоритмы, которые обучаются на данных [3].

Автоматизированное машинное обучение (англ. Automated machine learning, AutoML) – это область машинного обучения, задача которой автоматизировать стандартные процессы, возникающие в ходе применения машинного обучения и анализа данных, к задачам реальной жизни [8].

Гиперпараметры (англ. Hyperparameter) – параметры алгоритма, которые не могут быть изучены во время процесса тренировки и значения которых устанавливаются заранее [3].

Оптимизация гиперпараметров (англ. Hyperparameter optimization, HPO) – задача поиска оптимальных значений гиперпараметров алгоритма машинного обучения [1].

Признак (англ. feature) – это некая обособленная измеримая характеристика или свойство некого явления [3].

Графический пользовательский интерфейс (англ. graphical user interface, GUI) – способ размещения информации на экране компьютера, который легко воспринимается, так как в нем используются иконки, меню и мышь, а не только текст [22].

Актуальность темы исследования

Машинное обучение достигло значительных успехов в интернет рекламе, рекомендательных системах, анализе финансовых рынков, компьютерном зрении, вычислительной лингвистике, биоинформатике и во многих других областях. Тем не менее, его успех в значительной мере зависит от людей-экспертов в машинном обучении, так как эксперты в той или иной степени вовлечены во все этапы машинного обучения. Люди всё ещё принимают критически важные решения в таких задачах как, преобразование жизненной задачи в задачу машинного обучения, сбор данных, форматирование и предобработка данных, конструирование признаков, выбор

или проектирование архитектуры модели, оптимизация гиперпараметров, оценка эффективности модели, развертывание подобранного решения [5].

Добиться приемлемых результатов обучения – наукоемкая и трудозатратная задача. Ввиду необходимости упростить применение технологий машинного обучения, а также уменьшить потребность в специалистах, автоматизированное машинное обучение (AutoML) на сегодняшний день стало одной из важнейших тем, представляющей как промышленный, так и академический интерес. Также AutoML может сделать машинное обучение еще более доступным для повсеместного использования в жизни [21].

AutoML может повысить производительность работы при одновременной экономии значительного количества времени и денег, поскольку специалистов по машинному обучению находить сложно и дорого. В последнее время резко возрос коммерческий интерес к AutoML, и в настоящее время несколько крупных технологических компаний разрабатывают свои собственные системы AutoML [8].

Свои AutoML решения имеют: Google (Cloud AutoML), Microsoft (Azure Machine Learning), Amazon (SageMaker Autopilot).

По данным сайта Google Trends, поисковые запросы «AutoML» и «Automated machine learning» с начала 2017 начали испытывать заметный прирост интереса. Данные поисковые запросы на текущий момент переживают пик интереса по всему миру, в особенности в КНР, а также Сингапуре, Южной Корее, Тайване и Гонконге, т.н. четырех странах-азиатских тиграх.

Однако идея автоматизации процесса машинного обучения не нова. Попытки решить проблему подбора моделей машинного обучения проводились еще в 1975 году [15].

Также существует множество свободных AutoML решений с открытым исходным кодом.

В ряде задач методы AutoML уже превосходят специалистов по машинному обучению [8].

Цель и задачи исследования

Целью данной работы является разработка приложения для подбора моделей машинного обучения на основе подхода AutoML.

Для достижения поставленной цели необходимо решить следующие задачи, перечисленные ниже.

1. Выполнить обзор научной литературы и существующих решений по данной теме.

2. Определить требования к приложению.

3. Выполнить проектирование архитектуры приложения.

4. Реализовать приложение.

5. Выполнить тестирование приложения.

Структура и объем работы

Работа состоит из введения, четырех глав, заключения, списка литературы и приложения.

В первой главе выполняется обзор аналогов разрабатываемого приложения, а также существующих решений данной задачи путем обзора научной литературы.

Во второй главе описаны функциональные и нефункциональные требования, представлены диаграммы вариантов использования и деятельности, описана архитектура системы.

В третьей главе приводится реализация приложения на основе составленного списка требований.

В четвертой главе приводятся протоколы функционального тестирования, а также результаты тестирования системы на некоторых популярных наборах данных.

В заключении приводятся основные результаты работы и рассматриваются направления дальнейших исследований.

В приложении приводится код некоторых элементов системы.

Объем работы составляет 46 страниц, объем списка литературы – 39 источников.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

AutoML стремится к автоматизации следующих стандартных процессов машинного обучения: сбор данных, очистка данных, обработка отсутствующих данных, отбор и трансформация признаков, выбор модели, объяснение полученных результатов, выделение вычислительных ресурсов, оптимизация гиперпараметров, детектирование аномалий [8].

1.1. Обзор аналогов и существующих решений

На данный момент существует множество проектов в области AutoML и выполнить обзор всех существующих решений рамках данной работы невозможно. Поэтому рассмотрим некоторые наиболее популярные решения с открытым исходным кодом.

Auto-WEKA [23]

Auto-WEKA – одна из первых AutoML систем. Поиск производится среди 28 алгоритмов классификации, а также среди их гиперпараметров. Для оценки точности обученной модели используется алгоритм перекрестной проверки по k блокам. Auto-Weka впервые использовала метод байесовской оптимизации к алгоритмам машинного обучения с большим количеством параметров [11]. Распространяется под лицензией GPLv3.

Auto-WEKA базируется на WEKA – ПО, содержащем инструменты визуализации и ML-алгоритмы, а также GUI для быстрого доступа к ним [9]. Примерно 11.7 млн скачиваний на сайте sourceforge.net [24].

Auto-sklearn [25]

Подобный подход был использован в библиотеке Auto-sklearn. Auto-sklearn также использует байесовскую оптимизацию, но содержит меньше алгоритмов классификации и гиперпараметров [11].

Auto-sklearn использует алгоритмы из библиотеки Scikit-learn – python библиотеки, объединяющей широкий спектр современных алгоритмов машинного обучения (например, алгоритмов классификации и предварительной обработки данных). В библиотеке делается акцент на простоту ис-

пользования, производительность, документацию и согласованность API. Scikit-learn распространяется по упрощенной лицензии BSD, поощряя использование в академических и коммерческих целях [13].

Из этой библиотеки в Auto-sklearn используются 15 алгоритмов классификации, 14 методов обработки признаков (например PCA) и 4 метода предобработки данных (например обработка отсутствующих данных, кодирование категориальных переменных). Auto-sklearn распространяется под той же лицензией что и scikit-learn.

1.2. Обработка данных

В задачах классификации данные представляются в виде таблиц, состоящих из строк и столбцов. Строка – это кортеж столбцов. Столбец – это определенный признак прогнозируемого явления либо его класс. Столбец содержит значение некого типа, обычно числового или символьного. Также эти значения можно наиболее простым образом классифицировать как:

1) целевые;
2) числовые;
3) категориальные;
4) не связанные с целевыми [26].

Чтобы отнести значения столбца к одному из этих классов, порой необходимо вмешательство человека, разбирающегося в предметной области, к которой принадлежат данные. Например, без знаний предметной области столбец может быть отнесен к числовому или категориальному типу. От выбора класса зависит, как будут дальше обрабатываться данные. Данный этап в некоторой степени возможно автоматизировать, однако для повышения точности прогноза и уменьшения сложности AutoML системы, данный этап следует выполнять вручную. Например, загружать в систему данные вместе с файлом, описывающим классы столбцов.

Категориальные данные, как правило, нужно приводить к определенному виду, так как часто они представлены в виде строк, а многие ал-

горитмы принимают на вход только числа. Для этого используются различные алгоритмы кодирования, например: One Hot Encoding, Binary Encoding, Frequency Encoding, Label Encoding и другие.

Множество алгоритмов кодирования реализовано в python библиотеке category_encoders [27]. Данная библиотека предоставляет простой scikit-learn подобный интерфейс.

Иногда в наборе данных отсутствуют определенные значения. Так как не все алгоритмы могут обрабатывать отсутствующие данные, необходимо обрабатывать подобные строки. Самое простое решение в такой ситуации – это удаление строки, в которой отсутствуют данные.

Предобработка данных перед подачей их алгоритму позволяет достичь более высокой точности [10]. Для более удобного выполнения предобработки scikit-learn предоставляет pipeline (конвейер), позволяющий объединить алгоритм классификации и предобработки (например StandardScaler или PCA) в единую модель.

1.3. Выбор модели машинного обучения

Выбор модели машинного обучения один из основных этапов AutoML. В данной работе рассматривается выбор модели для задачи бинарной классификации.

Традиционно, выбор алгоритма классификации и его гиперпараметров делается специалистом на основе его личного опыта методом проб и ошибок [21].

Согласно теореме «no free lunch», не существует универсального алгоритма машинного обучения, который по точности превосходит другие алгоритмы на всех возможных задачах машинного обучения, связанных с прогнозированием [18]. Помимо точности прогноза, следует учитывать память, занимаемую моделью и время, за которое выдается прогноз. Эти характеристики не одинаковы для различных моделей и могут быть огра-

ниченны для поставленной задачи. Следовательно, необходимо выбрать алгоритм, наиболее подходящий для заданной проблемы.

Существует большое количество алгоритмов классификации. Однако проводить поиск оптимальной модели одновременно на всех из них не имеет смысла. Увеличение пространства поиска за счет добавления нового алгоритма классификации ведет к увеличению вычислительной нагрузки. Поэтому добавление нового алгоритма к пространству поиска должно быть обосновано [10].

Опыты, проведенные создателями auto-sklearn на 13 наборах данных с использованием 15 алгоритмов из библиотеки scikit-learn, продемонстрировали, что random forest, extremely randomized trees, AdaBoost и gradient boosting показали наибольшую эффективность, а SVM (linear и kernel) показали пиковую точность, но на некоторых наборах данных. Также было обнаружено, что некоторые алгоритмы классификации не могут достичь таких результатов: kNN, passive aggressive, decision tree, Gaussian NB, LDA и QDA. Они статистически значимо уступают предыдущей группе моделей на большинстве наборов данных. Также подтвердилось, что одна модель не может превосходить все остальные на всей совокупности наборов данных [7].

Опыты, проведенные создателями hyperopt-sklearn на трех известных наборах данных (MNIST, 20-Newsgroups, Convex Shapes) с использованием 24 алгоритмов из библиотеки scikit-learn, показали, что по точности SVM всегда были среди лучших моделей, а параметры самых точных SVM были очень разными. Например, на наборе данных MNIST и Convex среди лучших никогда не было SVM с сигмоидальным или линейным ядром. Однако на наборе данных 20 newsgroups SVM с линейными и сигмоидальными ядрами были часто самыми точными [10].

Другое исследование, проведенное с использованием 17 семейств алгоритмов классификации, состоящих из 179 отдельных алгоритмов, на 121 наборе данных, показало, что random forest и SVM являются наиболее уни-

версальными семействами алгоритмов. Также хорошие результаты продемонстрировали: ELM, decision tree (C5.0), MLP, AdaBoost, bagging использующий SVM (основанные на libSVM). Остальные реализации decision trees, остальные bagging и boosting ансамблей, kNN и GLM статистически значимо уступают предыдущей группе моделей на большинстве наборов данных [6].

Из чего можно сделать вывод, что поиск нужно проводить на небольшом числе наиболее эффективных алгоритмов, однако можно оставить пользователю возможность выбора.

1.4. Оценка точности модели

Оценка точности обученной модели – это этап машинного обучения, позволяющий сказать, насколько алгоритм подходит для прогнозирования данного явления, а также сравнить его с другими алгоритмами.

Наиболее простой алгоритм расчета точности – это деление числа выданных правильных прогнозов на общее число выданных прогнозов. Однако данный метод присваивает всем прогнозируемым классам одинаковый вес, поэтому он не корректно оценивает точность в задачах, где один класс встречается чаще другого. В таких случаях лучше использовать F-меру, так как она учитывает соотношение истинно положительных, ложно отрицательных и ложных позитивных прогнозов.

Следующий вопрос, как и на каких данных проводить оценку точности модели т.н. валидацию. Поскольку нельзя объективно оценить точность алгоритма на данных, на которых он обучался, вследствие такого явления как переобучение, можно использовать методы:

1) k-fold cross-validation;

2) holdout.

Данные методы основаны на делении данных на тренировочную и тестовую выборки в определенной пропорции.

Первый метод дает наиболее объективную оценку, особенно при большом значении k. Но поскольку процесс обучения и прогнозирования повторяется k раз, сильно увеличивается время оценки точности.

Второй метод оценки – простейший способ валидации. Он, наоборот, выполняется быстро, но не позволяет судить о точности алгоритма на всей выборке [3].

Существуют и другие методы валидации, они схожи с перечисленными выше методами.

1.5. Оптимизация гиперпараметров

Точность некоторых алгоритмов классификации (например SVM) в решающей степени зависит от того, проводится ли оптимизация гиперпараметров или нет [7].

Даже небольшое количество итераций алгоритма оптимизации на пространстве поиска может дать заметный прирост в точности [15].

Существует множество алгоритмов для решения данной задачи: перебор по решетке, случайный поиск, байесовская оптимизация, оптимизация с вычислением градиента, оптимизация с использованием эволюционных алгоритмов и другие.

Для задачи HPO также существует теорема «no free lunch», доказывающая невозможность существования алгоритма оптимизации, который был бы эффективнее любого другого алгоритма на всем множестве задач [19].

Байесовская оптимизация, основанная на Гауссовском процессе, демонстрирует эффективность на задачах с малым количеством числовых гиперпараметров [17].

Байесовская оптимизация, основанная на древовидных алгоритмах (например, TPE), демонстрирует эффективность на задачах с большим количеством, структурированных гиперпараметров [4].

Случайный поиск более эффективен, чем поиск по решетке в задаче HPO на некоторых алгоритмах и наборах данных. Также каждая итерация случайного поиска выполняется асинхронно [2].

В разных AutoML системах используются разные алгоритмы HPO.

1.6. Пространство поиска

Еще одна важная задача – это определение наиболее оптимальных гиперпараметров и их границ.

Пространство поиска может быть описано с помощью:

1) порядковых, непрерывных или категориальных гиперпараметров;

2) разных уровней шага поиска (равномерных, логарифмических);

3) условий (когда стоит выбор между двумя алгоритмами и после выбора имеют значение гиперпараметры только оставшегося алгоритма).

Иногда необходимая точность может быть получена на первом запуске алгоритма с параметрами по умолчанию. А иногда нет, например, SVM реализованный в библиотеке scikit-learn имеет гиперпараметр kernel принимающий значения linear, rbf, poly, sigmoid. Проверка данного гиперпараметра заметно влияет на точность алгоритма. Поэтому эти четыре значения должны обязательно проверяться, если SVM включен в поиск [10].

Некоторые исследования показывают, что для нейронных сетей, AdaBoost, SVM, random forest, kNN, XGBoost существуют диапазоны и даже конкретные значения гиперпараметров, на которых алгоритмы будут показывать наиболее высокие показатели точности [2, 14, 16]. Следовательно, не все гиперпараметры алгоритма классификации одинаково важны при поиске.

Многие AutoML системы вместо того, чтобы подбирать гиперпараметры для каждого алгоритма по отдельности, комбинируют выбор алгоритма и процесс оптимизации его гиперпараметров в единую задачу. Такая задача называется CASH [8].

2. ПРОЕКТИРОВАНИЕ

2.1. Функциональные требования

В ходе проектирования были определены следующие функциональные требования. Приложение должно:

1) позволять пользователю загружать набор данных и его описание;

2) позволять пользователю указывать требования к искомой модели;

3) позволять пользователю изменять настройки поиска;

4) выполнять поиск модели для задачи бинарной классификации;

5) сохранять модели, удовлетворяющие условиям на диск.

2.2. Нефункциональные требования

Были определены следующие нефункциональные требования, перечисленные ниже.

1. Приложение должно быть реализовано на языке программирования Python версии 3.6 или новее.

2. Приложение должно предоставлять GUI.

3. Алгоритмы классификации должны быть взяты из ML библиотек, предоставляющих scikit-learn подобный интерфейс.

4. Приложение должно поддерживать загрузку набора данных и его описания в текстовом формате CSV.

2.3. Варианты использования системы

На основе перечисленных выше требований была разработана диаграмма вариантов использования, представленная на рисунке 1.

В системе один актер – *Пользователь*, он взаимодействует с приложением через GUI. Для него были определены следующие прецеденты использования, перечисленные ниже.

1. *Подобрать модель* – запустить процесс подбора моделей машинного обучения.

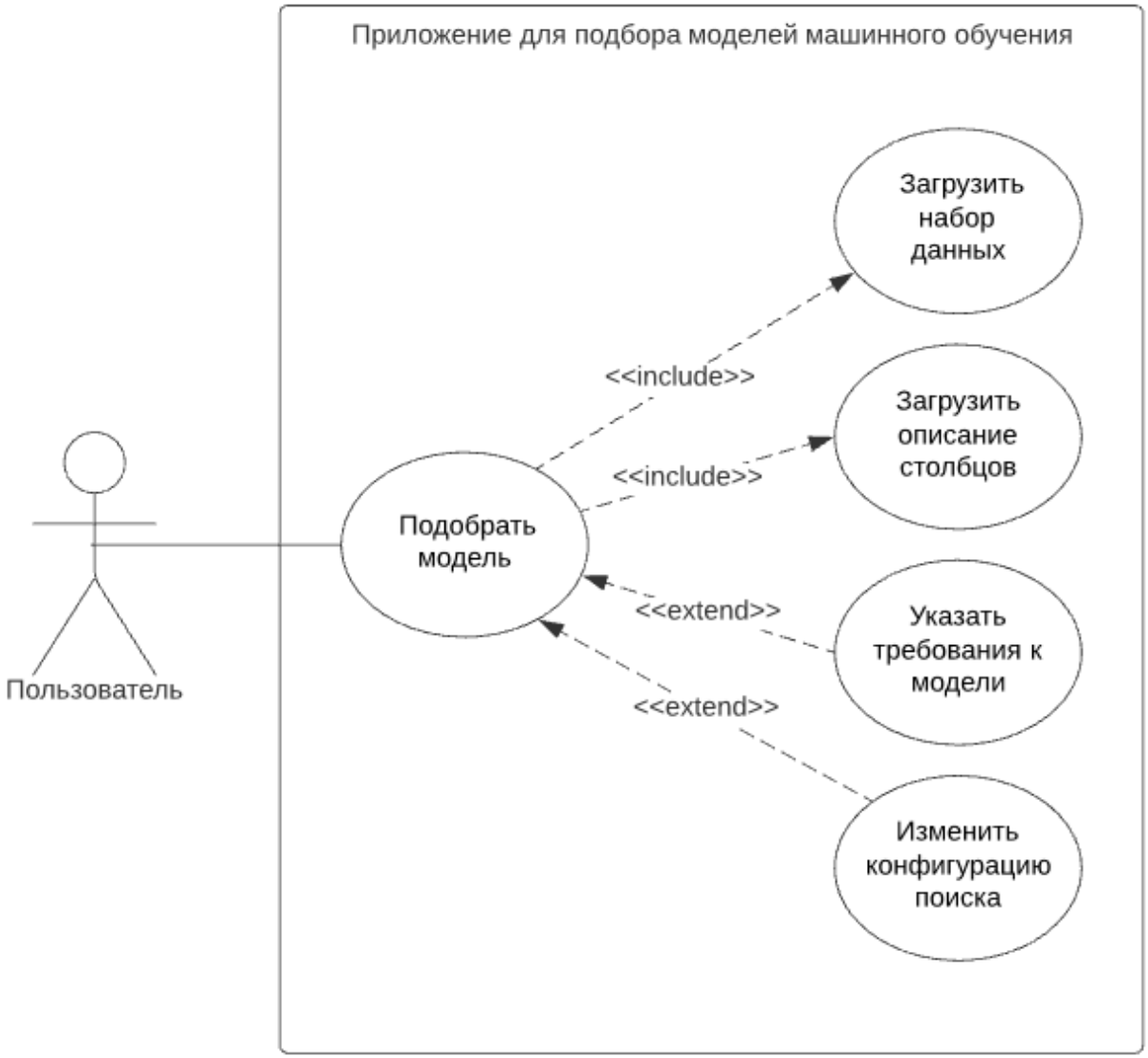

Рис. 1. Диаграмма вариантов использования приложения

2. *Загрузить набор данных* – в диалоговом окне выбора файла указать набор данных, для которого будет выполняться поиск моделей.

3. *Загрузить описание столбцов* – в диалоговом окне выбора файла указать файл, содержащий описание столбцов набора данных.

4. *Указать требования к модели* – задать условия, которым должна удовлетворять искомая модель. Опциональный прецедент, так как требования по умолчанию заданы.

5. *Изменить конфигурацию поиска* – изменить параметры поиска модели. Опциональный прецедент, так как конфигурация по умолчанию задана.

2.4. Диаграмма деятельности

Для демонстрации взаимодействия пользователя с системой была разработана диаграмма деятельности; она представлена на рисунке 2.

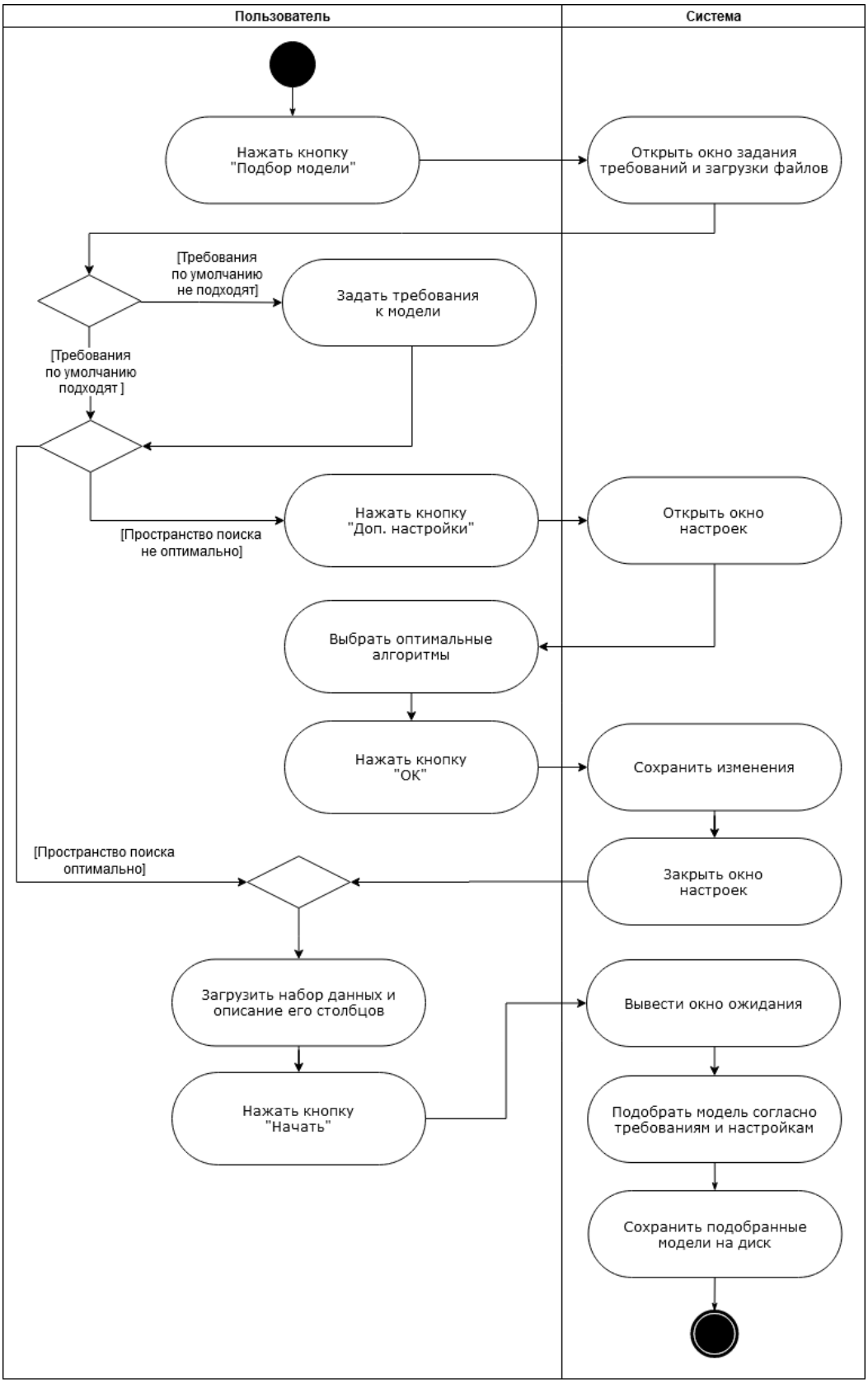

Рис. 2. Диаграмма деятельности

В приведенной диаграмме пользователь первоначально находится в главном окне приложения – главном меню. Во время процесса подбора модели параллельно в окне ожидания работает видимый пользователю таймер с оставшемся до конца поиска временем. По завершении поиска модели пользователь может вернуться в главное меню, нажав на ставшую активной кнопку «Меню», и запустить подбор повторно с теми же требованиями и настройками или с другими. Результаты поиска сохраняются в поддиректории согласно настройкам.

2.5. Архитектура системы

Прежде чем приступить к реализации системы, необходимо описать структуру ее компонентов. На рисунке 3 представлена диаграмма компонентов системы. Она показывает основные компоненты системы и их зависимости.

Система состоит из двух модулей:

1) модуль подбора моделей;

2) модуль пользовательского интерфейса.

Модуль подбора моделей

Модуль представляет собой python package, решающий задачу CASH для указанного набора данных. Модуль состоит из компонентов, перечисленных ниже.

1. *Компонент подбора моделей* определяет точку входа и программный интерфейс модуля для приема входных: набора данных и его описания, требований к искомой модели, а также настроек процесса поиска. Компонент проводит подбор моделей из компонента описания моделей согласно указанным требованиям и настройкам, после этого подобранные модели сохраняются на диск с помощью библиотеки сериализации.

2. *Библиотека сериализации* – сторонний пакет, предоставляющий средства для перевода python объекта в последовательность битов и обратно.

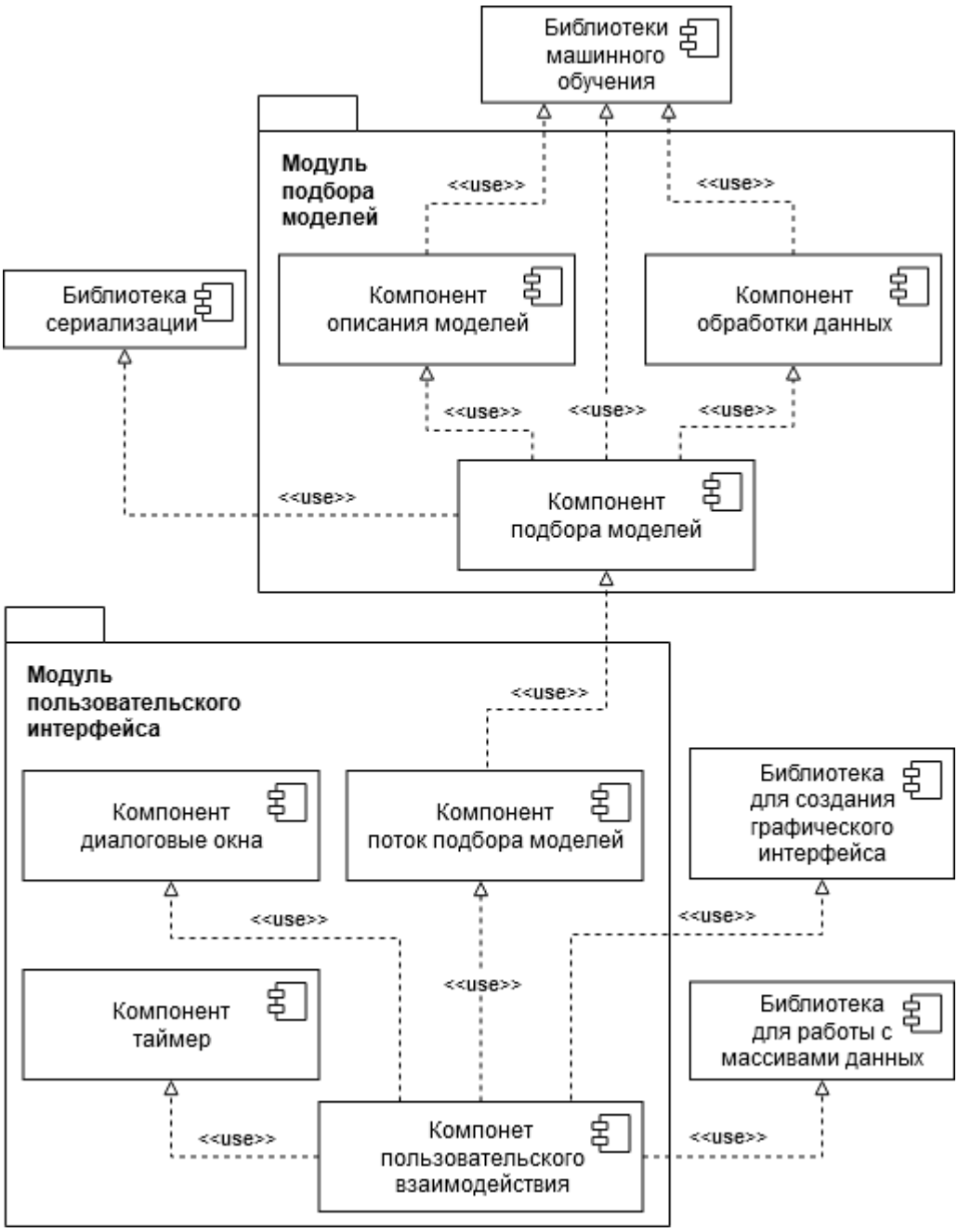

Рис. 3. Диаграмма компонентов

3. *Компонент описания моделей* содержит алгоритмы классификации, их оптимальный диапазон и значения гиперпараметров.

4. *Компонент обработки данных* обрабатывает поступивший на вход набор данных. Удаляет строки с пропущенными данными и выполняет кодирование категориальных столбцов.

5. *Библиотеки машинного обучения* сторонние пакеты, предоставляющие готовые решения для множества задач машинного обучения: алго-

ритмы классификации, функции оценки точности классификатора, алгоритмы для обработки данных, алгоритмы оптимизации гиперпараметров и функции для описания пространства поиска.

Модуль пользовательского интерфейса

Модуль представляет собой python модуль, который содержит компоненты, позволяющие пользователю взаимодействовать с модулем подбора моделей посредством графического интерфейса. Модуль состоит из компонентов, перечисленных ниже.

1. *Компонент пользовательского взаимодействия* определяет точку входа в программу, обеспечивает взаимодействие пользователя с модулем подбора моделей.

2. *Компонент таймер* отвечает за отображение времени, оставшегося до завершения поиска.

3. *Компонент диалоговые окна* хранит реализацию диалоговых окон приложения.

4. *Компонент поток подбора моделей* исполняет модуль подбора моделей в отдельном потоке.

5. *Библиотека для создания графического интерфейса* сторонний пакет, предоставляющий средства для создания пользовательского интерфейса.

6. *Библиотека для работы с массивами данных* сторонний пакет, предоставляющий средства для работы с массивами данных.

3. РЕАЛИЗАЦИЯ

Все модули приложения были написаны на языке программирования Python версии 3.6.8. Написание и отладка кода проводилась в интегрированной среде разработки Spyder 3.3.6 [28].

Большинство классовых методов не принимают на вход никаких данных, а обращаются к полям объекта класса, используя ключевое слово self.

Реализованное приложение содержит 3200 строк кода с учетом пустых разделительных строк.

3.1. Модуль подбора моделей

Модуль подбора моделей подбирает алгоритмы классификации и их гиперпараметры в соответствии с заданными требованиями. Результаты подбора сохраняются на диск. Модуль представляет из себя python модуль, он работает независимо и может использоваться как импортируемая библиотека.

Для модуля подбора моделей были разработаны классы, представленные на рисунке 4.

Классы Classification Algorithms

Classification Algorithms – это классы, предназначенные для хранения программной реализации и пространства поиска алгоритмов классификации.

Программная реализации алгоритма – это класс, импортированный из библиотеки машинного обучения. Классы, реализующие алгоритмы AdaBoost, SVM, random forest, bagging (SVC базовый алгоритм), extremely randomized trees, MLP, histogram gradient boosting, decision tree, ridge, SGD, k-nearest neighbors, passive-aggressive, nearest centroid, logistic regression, gaussian process, LDA, QDA, label spreading, Bernoulli naive Bayes, Gaussian naive Bayes, perceptron, импортируются из библиотеки Scikit-Learn [29], gradient boosting из XGBoost [30], ELM из Python-ELM [31], Factorization

machine и Polynomial Network из polylearn [32], DBN из deep-belief-network [33].

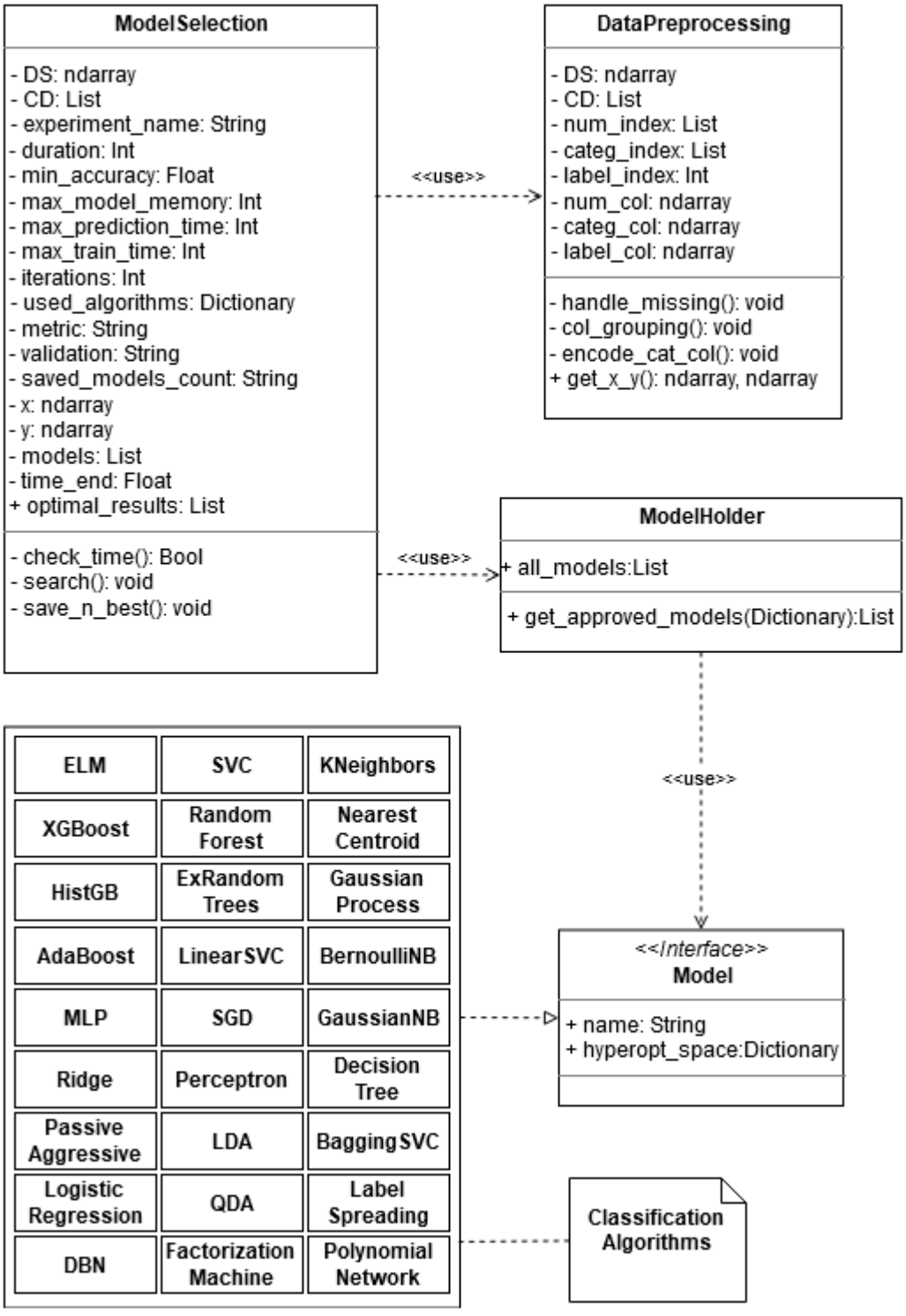

Рис. 4. Диаграмма классов модуля подбора моделей

Все импортированные классы имеют scikit-learn совместимый интерфейс, то есть реализуют публичные методы *fit* и *predict*.

Метод *fit* принимает на вход два значения *X* и *y* типа numpy array, где *X* – наборы признаков, а *y* – целевые значения. Данный метод обучает модель на входных данных.

Метод *predict* принимает на вход только *X*. Данный метод выполняет прогнозирование целевого значения на основе входных данных. Обучение и прогнозирование выполняется согласно внутренней инкапсулированной логике модели.

Пространство поиска – это значения гиперпараметров алгоритма, среди которых будет выполняться поиск. Пространство поиска задается с помощью библиотеки Hyperopt [34].

Классы *Classification Algorithms* предоставляют единый интерфейс. Каждый класс имеет два публичных поля:

1) *name* – строка, содержащая название алгоритма;

2) *hyperopt_space* – python словарь, хранящий алгоритм и его пространство поиска под ключами *model* и *param* соответственно. Также словарь хранит информацию о том следует ли использовать StandardScaler.

При составлении пространства поиска учитывались данные из исследований, приведенных в первой главе, собственных экспериментов, документации scikit-learn, других открытых и свободных AutoML проектов.

Пример *Classification Algorithms* класса для алгоритма SVM приведен в приложении А.

Класс ModelHolder

Класс *ModelHolder* используется для хранения и получения доступа к вышеописанным классам. Класс имеет одно поле и один метод.

Публичное поле *all_models* – python список, хранящий все классы *Classification Algorithms*.

Публичный метод *get_approved_models* принимает на вход значение *used_algorithms* – python словарь, хранящий информацию о том, будет ли использоваться определенный алгоритм для поиска оптимальной модели. На основе информации из словаря метод выбирает подходящие модели из

списка *all_models*, а после возвращает их. Листинг класса *ModelHolder* представлен на рисунке 5.

```
class ModelHolder:
    def __init__(self):
        self.all_models=[
            Perceptron(), Ridge(), PassiveAggressive(),
            LogisticRegression(), LDA(), QDA(), LinearSVC(),
            SVM(), SGD(), KNeighbors(), NearestCentroid(),
            GaussianProcess(), BernoulliNB(), GaussianNB(),
            DecisionTree(), BaggingSVC(), RandomForest(),
            xRandTrees(), AdaBoost(), HistGB(), MLP(),
            LabelSpreading(), XGBoost(), ELM(), DBN(),
            FactorizationMachine(), PolynomialNetwork()
          ]

    def get_approved_models(self, used_algorithms):
        approved_models=[]
        used_names=[]
        for name in used_algorithms:
            if(used_algorithms.get(name)==True):
                used_names.append(name)
        for model in self.all_models:
            if(model.short_name in used_names):
                approved_models.append(model)
                print(model.short_name)
        return approved_models
```

Рис. 5. Листинг класса *ModelHolder*

Класс DataPreprocessing

Данный класс используется для обработки набора данных и его описания. При создании объекта класса обязательна передача набора данных и его описания в конструктор класса. Все поля класса приватные. Класс содержит пять полей:

1) *DS* – numpy array, хранящий набор данных;

2) *CD* – python список, содержащий описание столбцов набора данных;

3) *num_index* и *categ_index* – python списки, хранящие индексы столбцов числового и категориального класса соответственно;

4) *label_index* – целое число, отображающее индекс целевого столбца;

5) *num_col*, *categ_col* и *label_col* – массивы типа numpy array хранящие числовые, категориальные и целевые столбцы соответственно.

Приватный метод *handle_missing* вызывается конструктором для обработки отсутствующих значений в *DS*. Если в строке *DS* отсутствует хотя бы одно значение, то строка удаляется.

Приватный метод *col_grouping*, используя список *CD*, распределяет индексы столбцов по *num_index*, *categ_index* и *label_index* и на их основе формирует массивы столбцов *num_col*, *categ_col* и *label_col*.

Приватный метод *encode_cat_col* с помощью класса *OrdinalEncoder* (импортированного из библиотеки category_encoders [27]) выполняет кодирование всех категориальных столбцов.

Публичный метод *get_x_y* вызывает метод *encode_cat_col*, а после возвращает объединенные *num_col* и *categ_col* как *x* и *label_col* как *y*. Листинг класса *DataPreprocessing* приведен в приложении А.

Класс ModelSelection

Класс *ModelSelection*, согласно переданным в конструктор параметрам, выполняет подбор моделей и их гиперпараметров. Результаты подбора сериализуются, а затем сохраняются на диск. Класс не имеет публичных полей (кроме *optimal_results*) и методов.

Следующие поля обязательно должны быть переданы в конструктор:

1) *DS*, *CD*, *used_algorithms* – были описаны выше;

2) *experiment_name* – название текущего эксперимента, значение используется для названия папки, в которую будут сохранены результаты;

3) *duration* – временной бюджет на поиск моделей (в секундах);

4) *min_accuracy* – согласно метрике, минимальная необходимая точность прогноза;

5) *max_model_memory* – максимальное количество памяти, занимаемое моделью (в байтах);

6) *max_prediction_time* – максимальное время прогнозирования, выполняемое для одной строки с данными (в миллисекундах);

7) *max_train_time* – максимальное время обучения модели (в секундах);

8) *metric* – название алгоритма, с помощью которого считается точность;

9) *validation* – название метода валидации модели;

10) *saved_models_count* – количество сохраняемых моделей;

11) *iterations* – количество итераций, выполняемых алгоритмом TPE.

Также класс имеет следующие поля:

1) *x* и *y* принимают значения, возвращаемые методом *get_x_y* класса *DataPreprocessing*;

2) *models* принимает значение, возвращаемое методом *get_approved_models* класса *ModelHolder*;

3) *time_end* – время, по достижению которого процесс поиска должен быть остановлен;

4) *optimal_results* – python словарь, хранящий обученные модели, а также их характеристики: точность, занимаемую память, время обучения и время выполнения одного прогноза. Единственное публичное поле.

Метод *check_time* проверяет, не закончился ли временной бюджет на поиск, если закончился, возвращает False иначе True.

Метод *search* проводит поиск модели, используя функцию *fmin*, импортированную из библиотеки hyperopt. Для работы с данной функцией указываются: целевая функция (функция учитывающая все требования к моделям и процессу их поиска, возвращаемое значение которой будет минимизировать *fmin*), пространство поиска (создается на основе поля *hyperopt_space* моделей, хранящихся в списке *models*), алгоритм оптимизации (TPE) и количество его итераций (приравнивается значению *iterations*).

Информация о состоянии поиска отправляется в стандартный поток вывода. После завершения поиска модели, удовлетворяющие требованиям, сохраняются в *optimal_results*. Фрагмент листинга с реализацией целевой функции внутри метода *search* приведен в приложении А.

Метод *save_n_best* сериализует и сохраняет модели на диск с помощью функции *dump*, импортированной из библиотеки Joblib [35]. Данная функция – аналог функции *pickle* из стандартной библиотеки python, но она более эффективно работает с объектами, содержащими крупные numpy массивы (чем часто является модель после применения метода *fit*). Также данный метод сохраняет на диск информацию обо всех итерациях алгоритма оптимизации (точность, время обучения, время единичного прогноза, занимаемая оперативная память, модель и её гиперпараметры) в таблицу формата *.xlsx.

Количество сохраняемых моделей зависит от поля *saved_models_count*. Сохраненные на диск модели можно загрузить и использовать по собственному усмотрению с помощью функции *load* импортированной из библиотеки Joblib. Например, вызвать метод *get_params* для отображения информации об используемых гиперпараметрах алгоритма.

3.2. Модуль пользовательского интерфейса

Модуль пользовательского интерфейса служит для взаимодействия пользователя с модулем подбора моделей через графический интерфейс.

Данный модуль позволяет пользователям, не имеющим навыков в программировании, пользоваться системой подбора моделей машинного обучения.

Для создания GUI использовалась библиотека PyQt5 [36] и предоставляемый ей редактор – Qt Designer 5.13.

Для модуля подбора моделей были разработаны классы, представленные на рисунке 6.

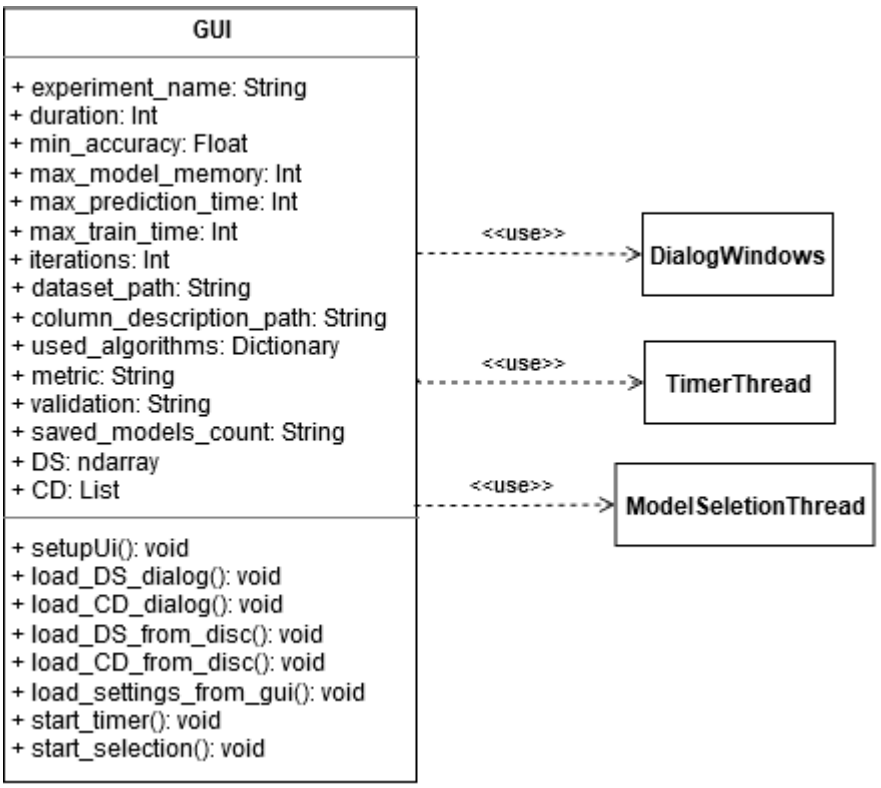

Рис. 6. Диаграмма классов модуля графического интерфейса

Класс DialogWindows

Класс предназначен для хранения диалоговых окон, таких как окно настроек и окно с предупреждением о неуказанных данных.

Классы TimerThread и ModelSelection

Классы используются для обеспечения работы таймера и для выполнения поиска соответственно. Классы создают новые потоки, чтобы сохранять интерактивность пользовательского интерфейса. Классы используют события и сигналы из библиотеки PyQt5. Листинги классов приведены в приложении Б.

Класс GUI

Метод *setupUi* отрисовывает графический интерфейс.

Методы *load_DS_dialog* и *load_CD_dialog* вызываются для отрисовки диалогового окна выбора файла, чтобы выбрать набор данных и его описание соответственно.

Методы *load_DS_from_disc* и *load_CD_from_disc* с помощью библиотеки для работы с массивами данных Numpy [37] загружают в систему данные в формате CSV.

Метод *load_settings_from_gui* загружает в систему данные, указанные в формах графического интерфейса.

Метод *start_timer* инициирует запуск таймера *TimerThread* в отдельном потоке.

Метод start_selection инициирует создание класса *ModelSelection* в отдельном потоке посредством класса *ModelSelectionThread*.

3.3. Графический пользовательский интерфейс

Реализованный проект представляет собой оконное приложение Windows. Окно его главного меню представлено на рисунке 7, на нем две кнопки: «Подбор модели» и «Выход».

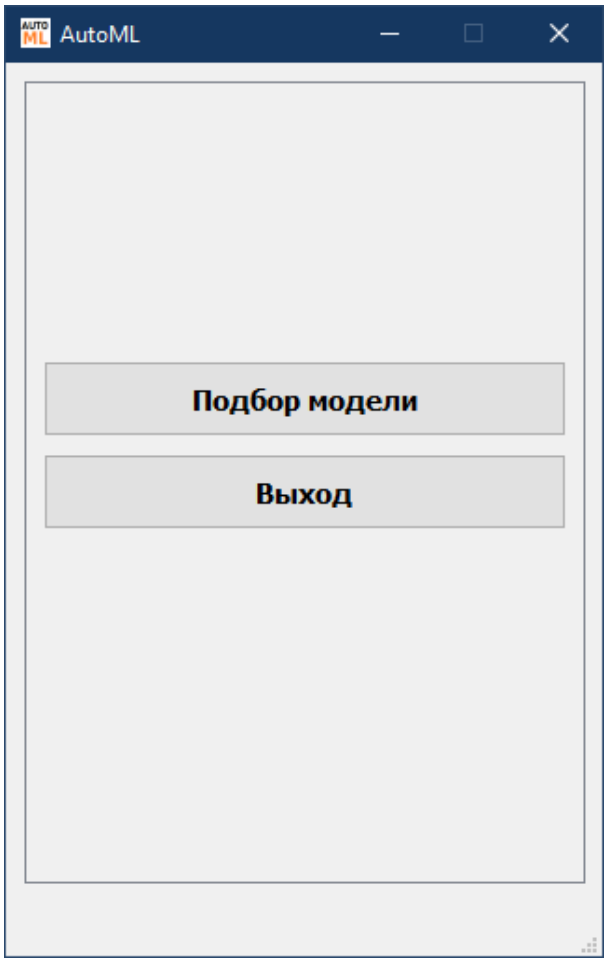

Рис. 7. Окно главного меню

Нажатие на кнопку «Подбор модели» открывает страницу задания требований и загрузки файлов, ее окно представлено на рисунке 8. Окно содержит поля для задания требований к моделям, общее время поиска и количество итераций алгоритма оптимизации. Также окно содержит кнопки «Загрузить», «Назад», «Доп. настройки», «Начать».

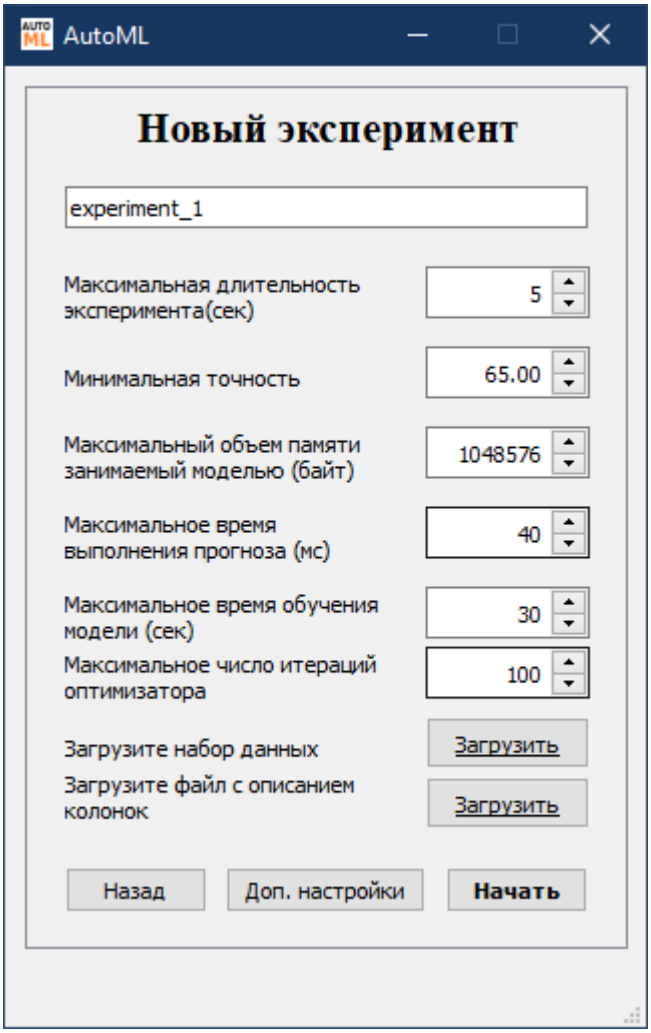

Рис. 8. Окно задания требований и загрузки файлов

Кнопки «Загрузить» открывают диалоговое окно выбора файла, в котором пользователь должен указать набор данных или его описания. Пользователь может указать файлы только в формате CSV. Если пользователь указал данные, то кнопка меняет название на «Готово» без нижнего подчеркивания.

Кнопка «Назад» возвращает пользователя на предыдущую страницу к окну главного меню.

Кнопка «Доп. настройки» создает модальное диалоговое окно, в котором пользователь может уточнить, среди каких моделей будет выполняться поиск, какая метрика точности будет использоваться, как будет выполняться валидация, а также указать количество сохраняемых на диск моделей. Основываясь на данных приведенных в первой главе, были выбраны 7 алгоритмов, среди которых по умолчанию проводится поиск. Также пользователь может протестировать алгоритмы из секции «Экспериментальные». Поиск на этих алгоритмах проводится в разы дольше. Внешний вид окна настройки поиска представлен на рисунке 9.

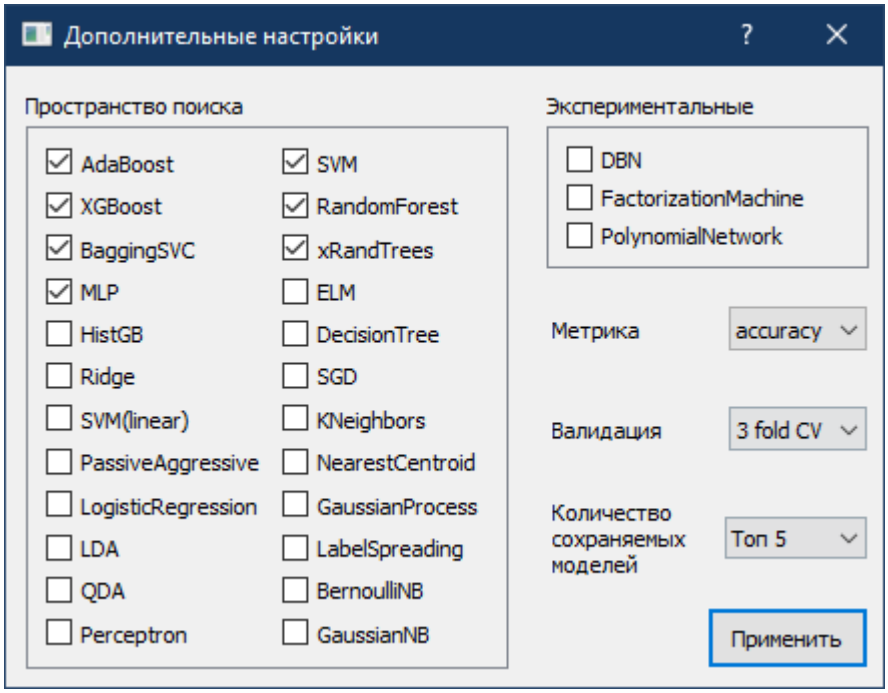

Рис. 9. Окно настройки поиска

Кнопка «Начать», при условии, что пользователь загрузил набор данных и его описание, запускает процесс поиска с помощью модуля подбора моделей и открывает следующую страницу с окном ожидания. Если набор данных и/или его описание не загружены, то выводится модальное диалоговое окно, сообщающее об этом пользователю. Если название эксперимента указанно пустым, или если в настройках не выбрано ни одного алгоритма, то выводится модальное диалоговое окно с предупреждением.

Окно ожидания содержит таймер, отсчитывающий время до конца поиска. После того, как значение таймера станет равно нулю, процесс поиска закончится (также процесс поиска закончится, если у оптимизатора закончится бюджет итераций). Внешний вид окна ожидания представлен на рисунке 10.

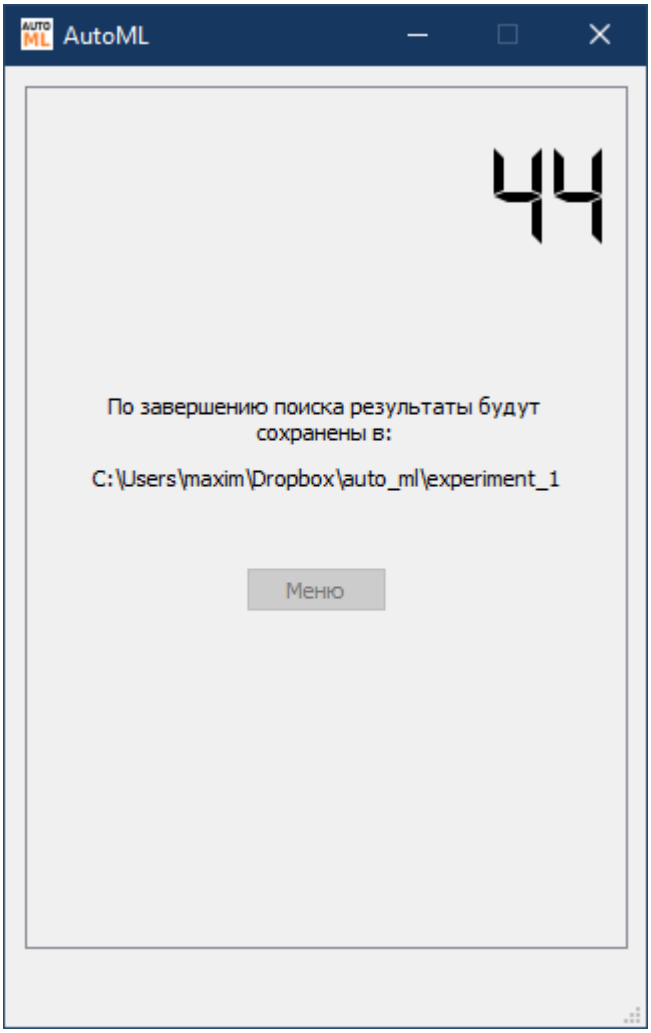

Рис. 10. Окно ожидания

После сохранения результатов подбора станет активной кнопка «Меню», а по указанному адресу пользователь сможет найти папку с названием эксперимента, в которой будут сохранены подобранные модели в формате *.joblib, а также полная статистика по результатам поиска в формате *.xlsx.

4. ТЕСТИРОВАНИЕ

Тестирование проводилось на персональном компьютере с установленной операционной системой Windows 10 Pro и следующим аппаратным обеспечением: процессор Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz, ОЗУ 8 ГБ.

4.1. Функциональное тестирование

Для проверки реализации функциональных требований, заявленных во второй главе, было проведено тестирование. Ниже приводится протокол функционального тестирования.

Тест № 1.

Действие: пользователь не указал набор данных и/или его описание, а после нажал кнопку «Поиск».

Ожидаемый результат: выведено окно с предупреждением о необходимости загрузки обоих файлов.

Тест пройден? Да.

Тест № 2.

Действие: Пользователь очистил поле «Название эксперимента», загрузил набор данных с его описанием, а после нажал кнопку «Поиск».

Ожидаемый результат: выведено окно с предупреждением о необходимости заполнения поля.

Тест пройден? Да.

Тест № 3.

Действие: Пользователь в настройках снял галки со всех алгоритмов, загрузил набор данных с его описанием, а после нажал кнопку «Поиск».

Ожидаемый результат: выведено окно с предупреждением о необходимости указания хотя бы одного алгоритма классификации.

Тест пройден? Да.

Тест № 4.

Действие: пользователь загрузил набор данных и его описание, а после нажал кнопку «Поиск».

Ожидаемый результат: программа штатно начала поиск, по завершении работы результаты были сохранены на диск.

Тест пройден? Да.

Тест № 5.

Действие: после завершения поиска, пользователь вернулся в меню, изменил требования к модели, указал новые алгоритмы классификации в настройках, а после нажал кнопку «Поиск».

Ожидаемый результат: программа штатно начала новый поиск, по завершении работы результаты были сохранены на диск.

Тест пройден? Да.

Тест № 6.

Действие: пользователь загрузил набор данных и его описание, задал небольшой временной бюджет и большое число итераций. После этого нажал кнопку «Поиск».

Ожидаемый результат: программа штатно начала новый поиск, после того, как значение таймера стало равно 0, процесс подбора выполнил текущую итерацию, затем перешел в режим завершения работы, после этого результаты были сохранены на диск.

Тест пройден? Да.

Тест № 7.

Действие: пользователь указал заведомо невыполнимые требования к модели, загрузил набор данных с его описанием, а после нажал кнопку «Поиск».

Ожидаемый результат: программа штатно начала новый поиск, по завершении поиска в консоли было выведено сообщение о том, что не найдено ни одной модели, удовлетворяющей требованиям.

Тест пройден? Да.

4.2. Тестирование на публичных наборах данных

Для определения работоспособности и эффективности разработанной системы было проведено тестирование на публичных наборах данных. Источник наборов данных – публичный репозиторий UCI Machine Learning Repository [38]. Информация о максимальной точности на каждом из наборов данных получена с сайта проекта OpenML [39]. Первая строка набора данных, содержащая названия столбцов, автоматически пропускается системой.

Для каждого набора данных была создана таблица *column_description.csv*. Данная таблица содержит описание столбцов набора данных в виде вертикальной таблицы, где в каждой строке указан: номер строки, тип столбца (Label, Num, Categ, Auxiliary).

Для каждого набора данных тестирование проводилось со следующими конфигурациями:

1) валидация – перекрестная проверка на 10 подгруппах;

2) пространство поиска – AdaBoost, XGBoost, Bagging(SVC), MLP, HistGB, LinearSVC, SVM, RandomForest, xRandTrees.

Максимальная длительность поиска выбиралась примерно, с учетом размера набора данных и сложности задачи. Количество итераций оптимизатора равняется длительности поиска. Остальные конфигурации были оставлены без изменений со значениями по умолчанию.

Наборы данных выбирались исходя из их известности и количества проведенных экспериментов на сайте проекта OpenML. Количество прогнозируемых классов равно двум. Большинство выбранных наборов данных сравнительно небольшого размера. Самый большой набор данных – higgs. Он содержит 29 столбцов и 98050 строк. Всего в тестировании участвуют 15 публичных наборов данных.

Результаты тестирования представлены в таблице 1. Все названия наборов данных соответствует названиям в системе OpenML.

Табл. 1. Результаты тестирования на публичных наборах данных

№	Набор данных	Максимальная точность по статистике OpenML	Максимальная точность полученная при тестировании	Время затраченное на поиск (сек)	Лучшие подобранные модели
1.	blood-transfusion-service-center	0.8021	0.7984	300	RandomForest
2.	breast-w	0.9757	0.9713	300	MLP
3.	climate-model-simulation-crashes	0.9296	0.9222	200	RandomForest
4.	credit-g	0.786	0.784	300	XGBoost
5.	diabetes	0.7878	0.7799	300	LinearSVC
6.	higgs	0.7333	0.7248	500	HistGB
7.	monks-problems-1	1	1	200	AdaBoost HistGB RandomForest XGBoost
8.	monks-problems-2	1	0.9883	200	AdaBoost
9.	monks-problems-3	0.9892	0.9892	200	RandomForest XGBoost HistGB
10.	ozone-level-8hr	0.9503	0.9428	500	XGBoost
11.	qsar-biodeg	0.8872	0.8787	300	LinearSVC
12.	spambase	0.9626	0.9228	500	HistGB
13.	steel-plates-fault	1	0.8906	300	LinearSVC
14.	tic-tac-toe	1	0.9800	500	SVM
15.	wdbc	0.9842	0.9824	400	Bagging(SVC)

Также система была проверена на возможность распознавания небольших изображений. Для этого были использованы базы данных mnist_784 [12] и Fashion-MNIST [20] из проекта OpenML. Так как приложение проводит бинарную классификацию, из баз данных были удалены все классы, кроме двух. Временной бюджет был установлен в 4000 секунд для каждого набора данных. Для mnist_784 была получена точность

0.9948, а для Fashion-MNIST 0.9725, что говорит о работоспособности системы.

Из полученных результатов можно сделать вывод, что система способна автоматически подбирать модели машинного обучения высокого качества на небольших наборах данных за приемлемый срок.

ЗАКЛЮЧЕНИЕ

В рамках данной выпускной квалификационной работы было разработано приложение для подбора моделей машинного обучения с использованием подхода AutoML.

Были выполнены все поставленные задачи.

1. Выполнен обзор научной литературы и существующих решений по данной теме.

2. Выполнено определение требований к приложению.

3. Выполнено проектирование архитектуры приложения.

4. Выполнена реализация приложения.

5. Выполнено тестирование приложения.

Система доказала свою способность находить модели согласно указанным требованиям.

Дальнейшим развитием системы будет:

1) включение большего количества алгоритмов предварительной обработки в задачу CASH;

2) добавление иных алгоритмов классификации и оптимизации гиперпараметров;

3) больший упор на искусственные нейронные сети и NAS;

4) разработка модуля улучшающего точность за счет применения алгоритмов ансамблирования;

5) изучение методов мета обучения;

6) поддержка временных рядов;

7) упор на максимальное использование выделенных вычислительных мощностей;

8) сохранение состояния поиска с возможностью в дальнейшем продолжить поиск с этого состояния;

9) реализация поиска только по времени или только по максимальному числу итераций или в комбинации (сейчас только в комбинации).

ЛИТЕРАТУРА

1. Bergstra J. [и др.]. Algorithms for hyper-parameter optimization 2011.

2. Bergstra J., Bengio Y. Random search for hyper-parameter optimization // Journal of Machine Learning Research. 2012.

3. Bishop C.M. Machine Learning and Pattern Recogniton / C.M. Bishop, 2006.

4. Eggensperger K. [и др.]. Towards an Empirical Foundation for Assessing Bayesian Optimization of Hyperparameters // BayesOpt workshop (NIPS). 2013.

5. Escalante H.J. [и др.]. AutoML @ NeurIPS 2018 Challenge: Design and Results 2020.

6. Fernández-Delgado M. [и др.]. Do we need hundreds of classifiers to solve real world classification problems? // Journal of Machine Learning Research. 2014. (15). С. 3133–3181.

7. Feurer M. [и др.]. Efficient and robust automated machine learning 2015.

8. Guyon I. [и др.]. Analysis of the AutoML Challenge Series 2015–2018 2019.

9. Hall M. [и др.]. The WEKA data mining software: An update // ACM SIGKDD Explorations Newsletter. 2009.

10. Komer B., Bergstra J., Eliasmith C. Hyperopt-Sklearn: Automatic Hyperparameter Configuration for Scikit-Learn 2014.

11. Kotthoff L. [и др.]. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA // Journal of Machine Learning Research. 2017.

12. LeCun Y., Cortes C., Burges C. THE MNIST DATABASE of handwritten digits // The Courant Institute of Mathematical Sciences. 1998.

13. Pedregosa F. [и др.]. Scikit-learn: Machine learning in Python // Journal of Machine Learning Research. 2011.

14. Probst P., Boulesteix A.L., Bischl B. Tunability: Importance of

hyperparameters of machine learning algorithms // Journal of Machine Learning Research. 2019.

15. Rice J.R. The Algorithm Selection Problem // Advances in Computers. 1976.

16. Rijn J.N. Van, Hutter F. Hyperparameter importance across datasets 2018.

17. Snoek J., Larochelle H., Adams R.P. Practical Bayesian optimization of machine learning algorithms 2012.

18. Wolpert D.H. The Lack of a Priori Distinctions between Learning Algorithms // Neural Computation. 1996.

19. Wolpert D.H., Macready W.G. No free lunch theorems for optimization // IEEE Transactions on Evolutionary Computation. 1997.

20. Xiao H., Rasul K., Vollgraf R. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms 2017.

21. Yao Q. [и др.]. Taking Human out of Learning Applications: A Survey on Automated Machine Learning 2018.

22. Cambridge Free English Dictionary and Thesaurus [Электронный ресурс]. URL: https://dictionary.cambridge.org/dictionary/english/ (дата обращения: 05.05.2020).

23. Auto-WEKA [Электронный ресурс]. URL: https://www.cs.ubc.ca/labs/beta/Projects/autoweka/ (дата обращения: 01.05.2020).

24. Download Statistics: All Files [Электронный ресурс]. URL: https://sourceforge.net/projects/weka/files/stats/map?dates=2000-04-27+to+2020-05-01 (дата обращения: 01.05.2020).

25. automl/auto-sklearn: Automated Machine Learning with scikit-learn [Электронный ресурс]. URL: https://github.com/automl/auto-sklearn (дата обращения: 01.05.2020).

26. Columns description - CatBoost. Documentation [Электронный ресурс]. URL: https://catboost.ai/docs/concepts/input-data_column-

descfile.html (дата обращения: 14.05.2020).

27. scikit-learn-contrib/category_encoders: A library of sklearn compatible categorical variable encoders [Электронный ресурс]. URL: https://github.com/scikit-learn-contrib/category_encoders (дата обращения: 19.05.2020).

28. Spyder Website [Электронный ресурс]. URL: https://www.spyder-ide.org/ (дата обращения: 04.05.2020).

29. scikit-learn/scikit-learn: scikit-learn: machine learning in Python [Электронный ресурс]. URL: https://github.com/scikit-learn/scikit-learn (дата обращения: 17.05.2020).

30. dmlc/xgboost: Scalable, Portable and Distributed Gradient Boosting (GBDT, GBRT or GBM) Library, for Python, R, Java, Scala, C++ and more. Runs on single machine, Hadoop, Spark, Flink and DataFlow [Электронный ресурс]. URL: https://github.com/dmlc/xgboost (дата обращения: 17.05.2020).

31. masaponto/Python-ELM: Extreme learning machine implemented by python3 with scikit-learn interface [Электронный ресурс]. URL: https://github.com/masaponto/Python-ELM (дата обращения: 17.05.2020).

32. scikit-learn-contrib/polylearn: A library for factorization machines and polynomial networks for classification and regression in Python. [Электронный ресурс]. URL: https://github.com/scikit-learn-contrib/polylearn/ (дата обращения: 17.05.2020).

33. albertbup/deep-belief-network: A Python implementation of Deep Belief Networks built upon NumPy and TensorFlow with scikit-learn compatibility [Электронный ресурс]. URL: https://github.com/albertbup/deep-belief-network (дата обращения: 17.05.2020).

34. Hyperopt Documentation [Электронный ресурс]. URL: https://hyperopt.github.io/hyperopt/ (дата обращения: 17.05.2020).

35. joblib/joblib: Computing with Python functions. [Электронный ресурс]. URL: https://github.com/joblib/joblib (дата обращения: 17.05.2020).

36. Riverbank Computing | Introduction [Электронный ресурс]. URL:

https://www.riverbankcomputing.com/software/pyqt/intro (дата обращения: 22.05.2020).

37. numpy/numpy: The fundamental package for scientific computing with Python. [Электронный ресурс]. URL: https://github.com/numpy/numpy (дата обращения: 17.05.2020).

38. UCI Machine Learning Repository [Электронный ресурс]. URL: https://archive.ics.uci.edu/ml/index.php (дата обращения: 28.05.2020).

39. OpenML Home [Электронный ресурс]. URL: https://www.openml.org/ (дата обращения: 28.05.2020).

ПРИЛОЖЕНИЯ

Приложение А – Листинги модуля подбора моделей

```
from sklearn import svm
from hyperopt import hp

class SVM:
      def __init__(self, ):
           self.name = 'SVM'
           self.hyperopt_space = {
                 'scale':hp.choice('SVM_scaler',[True,False]),
                 'model':svm.SVC,
                 'param':hp.choice('SVM_kernel_type',[
                 {
                       'kernel':hp.choice('kernel',['rbf','sigmoid'] ),
                       'gamma': hp.pchoice('gamma',
                       [ (0.05,'scale'),
                         (0.05,'auto'),
                         (0.9, hp.loguniform('gamma',-10.4,2.08 ))
                       ]),
                       'C':hp.loguniform('C1', -3.46,10.4 ),
                       'degree':2
                 },
                 {
                       'kernel':'poly',
                       'gamma':'scale',
                       'C':hp.loguniform('C2', -3.46,10.4 ),
                       'degree':hp.choice('degree',range(2,5))
                 }
           ])}
```

Рис. А.1. Листинг класса *SVM*

```
import numpy as np
class DataPreprocessing:
    def __init__(self, DS, CD):
        self.__DS = DS
        self.__CD = CD
        self.__handle_missing()
        self.__col_grouping()

    def __handle_missing(self):
        self.__DS=self.__DS[np.all(np.isfinite(self.__DS), axis=1)]

    def __col_grouping(self):
        self.__num_index = []
        self.__categ_index = []
        self.__label_index = None
        for column in self.__CD:
            if(column[1]=='Num'):
                self.__num_index.append(column[0]-1)
            elif(column[1]=='Categ'):
                self.__categ_index.append(column[0]-1)
            elif(column[1]=='Label'):
                self.__label_index = column[0]-1
        self.__num_col   = self.__DS[:,self.__num_index]
        self.__categ_col = self.__DS[:,self.__categ_index]
        self.__label_col = self.__DS[:,self.__label_index]
    def __encode_cat_col(self):
        from category_encoders import OrdinalEncoder
        enc = OrdinalEncoder(return_df=False).fit(self.__categ_col)
        self.__categ_col = enc.transform(self.__categ_col)

    def get_x_y(self):
        if(len(self.__categ_index)!=0 ):
            self.__encode_cat_col()
            if(len(self.__num_index)!=0 ):
                x = np.hstack([self.__num_col,self.__categ_col])
            else:
                x=self.__categ_col
        else:
            x=self.__num_col
        y=self.__label_col
        return x,y
```

Рис. А.2. Листинг класса *DataPreprocessing*

```
def __search(self):
    def objective_func(args):
       # проверка оставшегося времени
        if(self.__check_time()== True):

            if args['name']=='SVM':
                clf = args['model'](
                    kernel = args['param']['kernel'],
                    gamma = args['param']['gamma'],
                    C = args['param']['C'],
                    degree = args['param']['degree']
                )
                 if(args['scale']==True):
                    clf = make_pipeline(StandardScaler(), clf)
            elif args['name']=='XGBoost':
                clf = args['model'](
                #"booster":["gbtree","gblinear","dart"]
                    n_estimators = args['param']['n_estimators'],
                    learning_rate = args['param']['learning_rate'],
                    subsample = args['param']['subsample'],
                    max_depth = args['param']['max_depth'],
                    min_child_weight = args['param']['min_child_weight'],
                    colsample_bytree = args['param']['colsample_bytree'],
                    reg_lambda = args['param']['reg_lambda']  ,
                    reg_alpha = args['param']['reg_alpha']  ,
                )
            # и так для каждого алгоритма

            if( self.__valtype=='CV' ):
                start_timer = perf_counter()
                cv_results = cross_val_score(clf, self.__x, self.__y,
                                cv=self.__kfold, scoring = self.__metric)
                mem=cv_results['memory_fited']
                pred_time=cv_results['inference_time'].max()
                accuracy = cv_results['test_score'].mean()
                end_timer = perf_counter()
                time_all = end_timer - start_timer
            loss=(-accuracy)

            if(self.__metric=='accuracy'):
                accuracy=accuracy*100

            # Model requirements check
            if(accuracy < self.__min_accuracy or
               mem > self.__max_model_memory or
               pred_time > self.__max_prediction_time or
               time_all > self.__max_train_time):
                status=STATUS_FAIL
                loss=999
            else:
                status=STATUS_OK

            return {
                    'loss':loss,
                    'status': status,
                    'accuracy': accuracy,
                    'model_memory': mem,
                    'prediction_time': pred_time,
                    'train_time': time_all,
                    'model':clf
                    }
```

Рис. А.3. Фрагмент листинга метода *search*

Приложение Б - Листинги модуля пользовательского интерфейса

```
import auto
import time
from PyQt5.QtCore import pyqtSlot, pyqtSignal, QThread

class TimerThread(QThread):

    signal_timer = pyqtSignal(int)
    signal_timer_finish = pyqtSignal()

    @pyqtSlot(int)
    def slot_timer_start(self,value):
        self.seconds=value
        print("Start value:",value)

    def run(self):
        count = self.seconds

        while count > 0:
            time.sleep(1)
            count -= 1
            self.signal_timer.emit(count)

        self.signal_timer_finish.emit()
```

Рис. Б.1. Листинг класса *TimerThread*

```
class ModelSeletionThread(QThread):

    signal_model_selection_finish = pyqtSignal()

    @pyqtSlot(numpy.ndarray,list,str,int,float,int,int,int,dict,str,
                                                              str,str,int)
    def slot_start_model_selection_(self,DS,CD,experiment_name,duration,
                                    min_accuracy,max_model_memory,
                                    max_prediction_time,max_train_time,
                                    used_algorithms,metric,validation,
                                    saved_models_count,iterations):
        self.DS=DS
        self.CD=CD
        self.experiment_name=experiment_name
        self.duration=duration
        self.min_accuracy=min_accuracy
        self.max_model_memory=max_model_memory
        self.max_prediction_time=max_prediction_time
        self.max_train_time=max_train_time
        self.used_algorithms=used_algorithms
        self.metric=metric
        self.validation=validation
        self.saved_models_count=saved_models_count
        self.iterations=iterations


    def run(self):
        auto.ModelSelection(self.DS, self.CD, self.experiment_name,
                self.duration, self.min_accuracy, self.max_model_memory,
                self.max_prediction_time, self.max_train_time,
                self.used_algorithms, self.metric, self.validation,
                self.saved_models_count,self.iterations)
        print("ModelSeletionThread finish")
        # отправка сигнала о завершении потока
        self.signal_model_selection_finish.emit()
```

Рис. Б.2. Листинг класса *ModelSeletionThread*