МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

АВТОМАТИЧЕСКИЙ ПОИСК АНОМАЛИЙ В ДАННЫХ СИСТЕМ МОНИТОРИНГА С ПОМОЩЬЮ АЛГОРИТМОВ МАШИННОГО ОБУЧЕНИЯ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 411 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Гайворонского Алексея Евгеньевича

Научный руководитель
доцент _____ Б. А. Филиппов

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью настоящей работы является исследование и сравнение эффективности различных методов и моделей детекции аномалий в данных, собираемых системой мониторинга на основе Prometheus, развёрнутой в условиях Kubernetes кластера, а также реализация сервиса для автоматической детекции аномалий.

Основными задачами работы являются:

- разворачивание и настройка работоспособного Kubernetes кластера на локальной вычислительной машине;
- установка системы мониторинга на основе Prometheus;
- установка в кластер инструментов для хаос-тестирования;
- создание и проведение хаос-экспериментов для получения аномалий в данных;
- экспорт метрик из базы данных Prometheus, а также разметка аномальных значений;
- создание и обучение различных моделей для обнаружения аномалий в данных;
- получение предсказаний от различных моделей на основе тестовых данных и сравнительная оценка их работы;
- реализация сервиса для автоматической детекции аномалий с использованием моделей, исследованных ранее.

Актуальность работы обусловлена сильно растущим в последние годы рынком облачных решений и, как следствие, возрастающими потребностями мониторинга их состояния для повышения надёжности создаваемых систем. При этом классические методы мониторинга: алертинг на основе статичных правил и триггеров, визуальное наблюдение за графиками и показателями не дают должного результата в тех случаях, когда причина проблемы является слишком комплексной и сложной для отслеживания. Для таких случаев используются различные методы детекции аномалий, которые при правильном использовании дают отличные результаты там, где классические способы мониторинга бессильны, а автоматический поиск аномалий даёт возможность своевременно реагировать на неполадки в приложениях и инфраструктуре. Особенно это актуально в приложениях на микросервисой архитектуре, где на первый план выходит не надёжность физических компонентов, а устойчивость

сети и элементов системы к высоким нагрузкам. Поэтому для мониторинга в рамках исследования была взята система на основе Prometheus, развёрнутая в кластере Kubernetes, одновременно являющаяся также примером типичного микросервисного приложения.

1 Общее описание задачи

Описать поставленную задачу можно как исследование и сравнение различных моделей для обнаружения аномалий в данных, полученных из системы мониторинга на основе Prometheus, развёрнутой в Kubernetes кластере и построение сервиса для автоматического обнаружения аномалий с использованием исследуемых моделей.

План выполнения поставленной задачи можно сформулировать следующим образом: необходимо установить кластер Kubernetes, развернуть в нём систему мониторинга на основе Prometheus, собрать данные при нормальной работе кластера, развернуть и запустить систему для хаос-тестирования кластера, собрать данные с полученными в ходе хаос-экспериментов аномалиями, выгрузить данные из базы данных на локальную машину и разметить их (отметить аномальные участки), обучить различные модели для детекции аномалий на обучающем наборе данных, предсказать с помощью натренированных моделей аномальные участки на тестовых данных и сравнить предсказанные значения с эталонной разметкой. Затем следует построить веб-приложение, которое будет экспортировать данные из Prometheus, выявлять в них аномалии, и при наличии таковых подавать сигнал об этом в виде электронного письма.

2 Установка окружения для работы кластера

Для использования системы мониторинга, выбранной для этой работы, необходим установленный Kubernetes кластер. Для установки полноценного Kubernetes кластера нужно использовать выделенный сервер, но из-за ограничения ресурсов это не было реализовано в ходе работы.

Так как кластер планировалось использовать исключительно для тестовых действий, в качестве платформы для установки кластера был выбран пакет Microk8s.

На момент создания окружения для работы над проектом Microk8s стабильно работал только на операционной системе Linux. Для того, чтобы установить данный инструмент, имея в распоряжении машины только с операционной системой Windows, требовалось использовать виртуальную машину на базе ОС Linux.

Для этого был установлен бесплатный инструмент VMware Workstation 16 Player — программный инструмент, позволяющий использовать дополнительную операционную систему в качестве виртуальной машины.

В качестве операционной системы была выбрана Ubuntu 20.04.2 LTS — бесплатный дистрибутив GNU/Linux, основанный на Debian GNU/Linux. В качестве виртуальной машины операционная система была установлена из свободно распространяемого iso образа [1]. Далее был установлен пакет Microk8s с помощью пакетной системы `snap`.

С помощью Microk8s был развёрнут локальный Kubernetes кластер, состоящий из одного узла. Также был использован аддон `microk8s dashboard` для управления кластером через графический интерфейс. Управление ресурсами и объектами кластера через командную строку осуществлялось через `kubectl` — инструмент командной строки для управления кластерами Kubernetes. В данном случае был использован kubectl, поставляемый вместе с Microk8s. Эта версия инструмента может быть вызвана через команду `microk8s kubectl` [2].

Несмотря на использование специфичных технологий для развёртки Kubernetes кластера, методы и программные решения, описанные в данной работе, могут применяться к любому Kubernetes кластеру: развёрнутому локально или в облаке.

3 Установка системы мониторинга

В данной работе используется Kubernetes — открытое программное обеспечение для оркестровки контейнеризированных приложений — автоматизации их развёртывания, масштабирования и координации в условиях кластера. Разработан компанией Google [3]. Kubernetes был выбран благодаря широкому распространению в области управления контейнеризированными решениями.

Исследуемое приложение для мониторинга, благодаря контейнеризации, основано на микросервисной архитектуре, т.е. каждый компонент приложения помещается в отдельный контейнер, слабо связанный с окружением и другими контейнерами преимущественно по HTTP протоколу. Такой элемент цельного приложения называется микросервисом.

Объекты Kubernetes, также именуемые ресурсами, имеют декларативное описание в формате YAML. При этом создание примитивных ресурсов также можно производить императивным путём с помощью `kubectl` [4].

В качестве исследуемой базовой системы мониторинга был выбран Prometheus — бесплатное программное обеспечение, сочетающее в себе систему сбора метрик, базу данных временных рядов, систему алертинга — отслеживания метрик в реальном времени по определённым правилам для оповещения о данных, представляющих интерес или опасность для стабильной работы отслеживаемого приложения. Также Prometheus предоставляет простой веб-интерфейс для выполнения запросов к базе данных и отслеживания оповещений, а также конфигураций.

Prometheus работает по принципу активного сбора данных с приложений по HTTP протоколу. Для этого наблюдаемое приложение должно отдавать метрики в особом формате по адресу, известному для системы мониторинга. На практике для наиболее распространённых случаев сбора различных метрик не модифицируют само приложение или инфраструктуру, о которой необходимо собирать информацию, а используют так называемые экспортеры — приложения (часто — микросервисы), которые собирают необходимую информацию и выставляют её в виде метрик, доступных Prometheus для чтения, по определённому эндпоинту — URL-адресу с определённым URL-путём. Затем Прометей конфигурируется таким образом, чтобы считывать информацию с данного эндпоинта с определённым временным интревалом и параметрами, также передающимся в составе адреса. Такая конфигурация называется таргетом.

Для управления Prometheus в среде Kubernetes был использован Prometheus Operator. Оператор обеспечивает собственное развертывание Kubernetes и управление Prometheus и соответствующими компонентами мониторинга. Цель проекта — упростить и автоматизировать настройку стека мониторинга на основе Prometheus для кластеров Kubernetes.

Оператор Prometheus включает, помимо прочего, следующие функции: настраиваемые ресурсы Kubernetes для развертывания и управления Prometheus, Alertmanager и связанными компонентами; упрощенная конфигурация развертывания — вся настройка Prometheus происходит через особый Prometheus CR (Custom Resourse) — декларативное описание характеристик создаваемого ресурса; целевая конфигурация Prometheus: автоматическое создание целевых конфигураций мониторинга (таргетов) при помощи Service Monitor CR [5].

Для целей исследования, помимо стандартных настроек, через Prometheus CR были сделаны следующие настройки для использования Прометея: при помощи установки параметра `retention: 90d` срок хранения метрик во внутренней базе данных увеличен с 15 до 90 дней; параметр `replicas` изменён с 2 до 1 для использования только одного экземпляра Prometheus, работающего одновременно, для экономии ресурсов виртуальной машины, в частности — дискового пространства для исключения дублирования хранимых метрик. Для исследования в рамках данной работы были установлены следующие экспортеры: blackbox-exporter, kube-state-metrics, node-exporter.

Для визуального отображения данных, например в виде графиков или счётчиков, было выбрано веб-приложение Grafana, которое позволяет создавать настраиваемые доски с панелями для отображения информации по запросу из базы данных.

Также в качестве дополнительных сервисов были установлены alertmanager и prometheus-adapter. Alertmanager обрабатывает предупреждения, отправленные клиентскими приложениями, такими как сервер Prometheus. Alertmanager и prometheus-adapter не были использованы в данной по прямому назначению и были установлены в качестве дополнительной нагрузки на кластер, а также выполняли роль тестовых приложений. Alertmanager был установлен в высокодоступном (High Availability) режиме, поэтому сервис был продублирован два раза.

При установке компоненты системы мониторинга были установлены автоматически в так называемые поды (англ. pod — модуль) — абстрактные ресурсы Kubernetes, которые могут содержать один или несколько связанных между собой контейнеров. При этом контейнеризированные приложения внутри пода делят между собой его ресурсы — хранилище, сеть, порты и т.п.

Для установки приложений в Kubernetes используются развёртывания (англ. deployment), которые содержат необходимую информацию для создания пода с приложением. Разновидностью развёртывания является daemonset, который отличается тем, что разворачивает поды с экземплярами приложений на каждом из доступных узлов, при том что deployment по умолчанию использует для этого только один узел.

Также для успешного функционирования описанных выше компонентов системы требуется создать ресурс сервис (service) для каждого из них. Сервис позволяет поду взаимодействовать с внешним миром и другими подами путём присваивания ему внутреннего или внешнего IP адреса и использования определённых портов контейнерами внутри него. Помимо сервисов, для правильной работы всех компонентов системы требуются другие ресурсы, такие как роли (для получения разрешения от кластера на использование потенциально уязвимых ресурсов), сервис-аккаунты, сервис-мониторы и т.д.

Для развёртывания вышеперечисленных ресурсов в кластере необходимо описание их свойств в декларативной форме в формате YAML и последующее применение их к Kubernetes через команду `kubectl apply` или через веб-интерфейс кластера. Из-за большого количества файлов с описанием ресурсов было решено использовать пакетный менеджер Helm.

Таким образом, для развёртывания системы мониторинга, согласно документации Helm, были созданы шаблоны необходимых файлов и описаны параметры установки в файле `values.yaml`. Шаблоны, а также файл с параметрами и прочие вспомогательные файлы, помещённые в определённые директории, выстроенные в правильной иерархии, называются helm-чартом [6]. Основа для использованных шаблонов была взята из открытого репозитория kube-prometheus-stack [7].

4 Развёртывание и запуск хаос-тестирования

Хаос-тестирование — это дисциплина проведения экспериментов над программной системой в производственной среде с целью создания уверенности в способности системы противостоять неожиданным условиям. Чаще всего такое тестирование применяется в условиях кластера для тестирования устойчивости микросервисных приложений, а также инфраструктуры кластера в целом.

В данной работе инструменты хаос-тестирования были использованы для искусственного создания аномалий в данных мониторинга. В качестве программного обеспечения для создания хаос-экспериментов был выбран Chaos Mesh — решения для chaos engineering, созданное специально для использования в условиях кластера Kubernetes [8].

Chaos Mesh позволяет создать эксперименты для тестирования, отслеживать ход их выполнения и управлять ими, в том числе и с использованием встроенного веб-интерфейса. Для целей исследования были созданы эксперименты нескольких типов: pod kill, pod failure, stress test, network loss.

5 Экспорт и разметка данных

Prometheus поддерживает экспорт данных в целях резервного копирования. С помощью определённой команды можно получить доступ к контейнеру из пространства вне кластера через указанный порт, осуществляя так называемый «проброс портов» — закрепление номера порта `localhost` за портом соответствующего пода. Затем, пройдя по определённому URL-адресу, вызывается API Prometheus, после чего в файловой системе контейнера создаётся резервная копия данных, которую её можно выгрузить на локальную машину.

В ходе исследования было решено отказаться от JSON в пользу формата данных CSV. Чтобы выгрузить данные и сразу конвертировать их в выбранный формат, был использован скрипт, написанный на языке Python, который обращался Prometheus с помощью специального языка запросов PromQL, а затем записывал результат в стандартный канал вывода в формате CSV [9].

Из-за специфики данного скрипта и языка PromQL использование одной команды позволяет получить только один временной ряд. Но так как для обучения и тестирования моделей для детекции аномалий требуются, как правило, сотни временных рядов из нескольких метрик, то был написан скрипт на языке Shell, последовательно вызывающий предыдущую программу с различными PromQL запросами для выгрузки данных с определённой агрегацией и за необходимый промежуток времени.

Выбранные метрики отражают загрузку центрального процессора основного контейнера в поде `alertmanager-main-0`, использование памяти этим же контейнером, а также общее число ошибок с кодом `404` в Kubernetes API соответственно. Хаос-тестирование, в частности эксперименты stress test и network loss, воздействовали на кластер так, что в моменты их воздействия в вышеприведённых метриках наблюдался резкий рост значений.

Разметка данных производилась в полуавтоматическом режиме. Для первоначальной разметки был написан скрипт, который помечал ряд данных как аномальный, если рост использования процессора в этот момент превосходил некий установленный порог. Меткой являлось дополнительное поле в CSV файле, именуемое `anomaly`. Значение 0 данного поля указывало на отсутствие аномалии в этой точке, тогда как 1 помечало ряд как аномальный. После работы программы разметка просматривалась и корректировалась вручную.

6 Обучение, применение и сравнение моделей детекции аномалий

Далее в ходе работы размеченные данные использовались для обучения и оценки моделей обнаружения аномалий. Были выбраны такие методы классификации аномальных данных, как Z-оценка, метод опорных векторов [10, 11], смесь нормальных распределений [12, 13] и градиентный бустинг [14, 15].

В начале алгоритм считывает из CSV файлов обучающие тестовые данные, разделяя содержимое таблиц на массив значений и массив меток для каждого набора данных.

Вычисление Z-оценки представлено классом `ZScoreModel`. Метод класса `z_score` вычисляет количество стандартных отклонений для точки данных, подающейся на вход — её z-оценку [16]. Метод класса `is_anomaly` возвращает булево значение `true`, если z-оценка точки выше порога, обозначенного константой `MAX_Z_SCORE`. Метод класса `anomaly_prediction` проходит по всем значениям из массива, поступающего на вход функции, и формирует массив предсказанных меток на основе вычисленных ранее значений.

В основной программе создаётся объект `ZScoreModel`, а затем вызывается функция `anomaly_prediction`. По возвращённым из метода значениям и эталонной разметке строится ROC-кривая, а также вычисляется AUC значение, которые будут отображены в отдельном окне после вычисления оценок всех моделей [17].

Оставшиеся модели были взяты из библиотеки `sklearn`. Также были использованы взятые из неё функции для вычисления TPR и FPR значений для построения ROC-кривой и функция вычисления AUC для этой кривой.

В результате с помощью библиотеки `matplotlib` строится график, наглядно показывающий эффективность каждого метода. Интерпретировать его следует следующим образом: чем больше площадь фигуры, образованной ROC-кривой для определённого метода детекции аномалий (т.е. AUC), тем больше его точность на тестовых данных [18].

7 Сервис для автоматической детекции аномалий

На основе исследованных моделей для детекции аномалий в ходе работы был создан сервис для автоматического распознавания аномалий и сигнализирования в случае их обнаружения. Смысл работы такого приложения в том, выявлять аномалии в данных, получаемых из Prometheus в реальном времени, и подавать сигнал при их обнаружении, в частности отсылать письмо по электронной почте и отражать текущий статус на веб-странице. Сервис был реализован на языке Python. Так как целью было создание легковесного и простого сервиса, для основы веб-приложения был взят фреймворк Flask [19].

Содержимое основного модуля программы начинается с инициализации объекта `Flask` — главного объекта веб-приложения, а также переменных `current_data`, которая хранит данные, собранные для выявления аномалий, и `enabled_models`, хранящую объекты моделей машинного обучения, которые должны быть задействованы для обнаружения аномалий. Следом создаётся словарь `anomalies_detected` для хранения статуса детекции в будущем.

Следующим шагом загружается информация из конфигурационного файла `config.yaml` и сохраняется в оперативной памяти в виде словаря, при этом некоторые значения заносятся в отдельные переменные для удобства использования. Затем из CSV файла выгружается информация для обучения моделей детекции. На основании этих данных обучаются модели для детекции аномалий. При этом объект, содержащий очередную модель, инициализируется и добавляется в список `enabled_models` только тогда, когда соответствующий параметр конфигурации имеет значение `true`.

Каждый класс для взаимодействия с моделями определённого вида построен по схожему принципу: он содержит конструктор, который принимает на вход тренировочные значения и метки, а затем сразу создаёт и обучает модель. Также каждый класс имеет метод `predict`, используя который можно получить оценку значений, передающихся в качестве аргумента, на предмет наличия аномалий в тех или иных точках данных. Метод возвращает массив меток, предсказанных моделью детекции аномалий.

Класс, позволяющий использовать метод z-оценки, взят практически без изменений из аналогичного класса в программе по оценке моделей детекции. Остальные классы модуля `models.py` не различаются по внутренней структуре.

После инициализации моделей детекции с помощью Python-библиотеки `threading` в отдельном потоке запускается метод `scrape_metrics`. Это сделано для того, чтобы функция выполнялась во время работы приложения постоянно и независимо от главного потока, так как она содержит вызов метода `time.sleep`, который в противном случае приостанавливал бы работу всего приложения [20].

Сама функция `scrape_metrics` представляет собой «бесконечный» цикл, в процессе выполнения которого с помощью библиотеки `requests` вызывается API Prometheus, возвращающее информацию о данных по результатам выполнения запроса к БД, после чего формируется кортеж, помещающийся в список `current_data`.Затем запускается функция `prediction` для обновления статусов детекции аномалий. После этого поток приостанавливает работу на время, указанное в конфигурации.

Функция `prediction` отвечает за формирование словаря, который содержит следующие поля: булево поле `anomaly` принимает значение `True`, если в текущих данных обнаружена аномалия; поле `time` отражает время последнего обновления данного списка; поле `models` содержит список моделей с их статусами и именами. Также в этой функции находится логика по отправке электронного письма при обнаружении аномалии.

Функция `create_email` формирует электронное письмо по заданному шаблону и вызывает функцию `send_email`, которая, используя библиотеку `smtplib`, отсылает по заданным в email-конфигурации реквизитам сформированное на предыдущем шаге письмо [21].

Стандартным методом проверки статуса детекции аномалий в данном сервисе является веб-страница, отражающая последние результаты работы моделей. Страница была создана с использованием технологии Bootstrap 4 для упрощения работы со стилями [22], Jinja2 в качестве шаблонизатора, поставляющегося вместе с Flask, для переноса данных из Python на HTML страницу. Загрузка страницы осуществляется с помощью так называемого роута — специальной сущности Flask, которая закрепляет за определённым URL какую-либо логику, по результатам выполнения которой возвращается HTML страница.

ЗАКЛЮЧЕНИЕ

В ходе работы был развёрнут Kubernetes кластер на локальной виртуальной машине, установлена система мониторинга на основе Prometheus, развёрнута система хаос-тестирования и проведены хаос-эксперименты для получения аномалий в данных мониторинга. Также были написаны программы для экпорта метрик из базы данных Prometheus, разметки аномальных значений на них. Были созданы и обучены различные модели для детекции аномалий и проведён сравнительный анализ их работы. На основе данных моделей было создано веб-приложение для автоматического обнаружения аномалий в данных в режиме реального времени, а также сигнализации об их появлении в данных.

В результате сравнения различных моделей распознавания аномалий было выяснено, что на данных, подобных тем, что взяты в ходе работы, лучшую эффективность в обнаружении аномалий показывает модель машинного обучения, основанная на градиентном бустинге.

Также результатом работы является образец легковесного сервиса для автоматического обнаружения аномалий в данных, экспортируемых из Prometheus. Данное решение можно использовать для систем мониторинга, развёрнутых на любом Kubernetes кластере.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Ubuntu as a VMware Guest [Электронный ресурс]. — URL: https://help.ubuntu.com/community/VMware/Workstation (Дата обращения 01.05.2021). Загл. с экр. Яз. англ.

2 MicroK8s — Introduction to MicroK8s [Электронный ресурс]. — URL: https://microk8s.io/docs (Дата обращения 15.05.2021). Загл. с экр. Яз. англ.

3 Основы Kubernetes | Kubernetes [Электронный ресурс]. — URL: https://kubernetes.io/ru/docs/tutorials/kubernetes-basics (Дата обращения 10.05.2021). Загл. с экр. Яз. рус.

4 *Арундел, Д.* Kubernetes для DevOps: развертывание, запуск и масштабирование в облаке / Д. Арундел, Д. Домингус. — СПб: Питер, 2020.

5 Prometheus — Monitoring system and time series database [Электронный ресурс]. — URL: https://prometheus.io (Дата обращения 30.04.2021). Загл. с экр. Яз. англ.

6 Helm | Helm Install [Электронный ресурс]. — URL: https://helm.sh/docs/helm/helm_install (Дата обращения 03.05.2021). Загл. с экр. Яз. англ.

7 kube-prometheus-stack 15.4.4 — prometheus/prometheus-community [Электронный ресурс]. — URL: https://artifacthub.io/packages/helm/prometheus-community/kube-prometheus-stack/15.4.4 (Дата обращения 03.05.2021). Загл. с экр. Яз. англ.

8 A Powerful Chaos Engineering Platform for Kubernetes | Chaos Mesh® [Электронный ресурс]. — URL: https://chaos-mesh.org/ (Дата обращения 21.05.2021). Загл. с экр. Яз. англ.

9 Prometheus query results as CSV – Robust Perception | Prometheus Monitoring Experts [Электронный ресурс]. — URL: https://www.robustperception.io/prometheus-query-results-as-csv (Дата обращения 14.05.2021). Загл. с экр. Яз. англ.

10 Support Vector Machines: A Simple Tutorial [Электронный ресурс]. — URL: https://timofey.pro/static/pdfdocs/AI_023_SVM_tutorial_Alexey_Nefedov.pdf (Дата обращения 24.05.2021). Загл. с экр. Яз. англ.

11 Одноклассовый SVM — визуализация и поиск аномалий [Электронный ресурс]. — URL: https://ru.coursera.org/lecture/unsupervised-learning/odnoklassovyi-svm-EpMGE (Дата обращения 22.05.2021). Загл. с экр. Яз. рус.

12 *Xu, L.* On convergence properties of the em algorithm for gaussian mixtures / L. Xu, M. I. Jordan // *Neural Computation*. — 1996. — Vol. 1. — Pp. 129–151.

13 *Wan, H.* A novel gaussian mixture model for classification // 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC). — Bari, Italy: 2019.

14 Градиентный бустинг — просто о сложном [Электронный ресурс]. — URL: https://neurohive.io/ru/osnovy-data-science/gradientyj-busting (Дата обращения 24.05.2021). Загл. с экр. Яз. рус.

15 *Friedman, J.* Greedy function approximation: A gradient boosting machine / J. Friedman // *The Annals of Statistics*. — 2001. — Vol. 29. — Pp. 1189–1232.

16 *Мелник, М.* Основы прикладной статистики / М. Мелник. — Москва: Энергоатомиздат, 1983.

17 AUC ROC (площадь под кривой ошибок) | Анализ малых данных [Электронный ресурс]. — URL: https://dyakonov.org/2017/07/28 (Дата обращения 05.05.2021). Загл. с экр. Яз. рус.

18 *Fawcett, T.* An introduction to roc analysis / T. Fawcett // *Pattern Recognition Letters*. — 2006. — Vol. 27. — Pp. 861–874.

19 Welcome to Flask — Flask Documentation (2.0.x) [Электронный ресурс]. — URL: https://flask.palletsprojects.com/en/2.0.x (Дата обращения 22.05.2021). Загл. с экр. Яз. англ.

20 Многопоточность в Python [Электронный ресурс]. — URL: https://docs-python.ru/tutorial/mnogopotochnost-python (Дата обращения 21.05.2021). Загл. с экр. Яз. рус.

21 smtplib — SMTP protocol client — Python 3.8.10 documentation [Электронный ресурс]. — URL: https://docs.python.org/3/library/smtplib.html (Дата обращения 19.05.2021). Загл. с экр. Яз. англ.

22 Bootstrap. Документация на русском языке [Электронный ресурс]. — URL: https://bootstrap-4.ru (Дата обращения 26.05.2021). Загл. с экр. Яз. рус.