ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ КИБЕРНЕТИЧЕСКИХ СИСТЕМ

Кафедра
«Криптология и кибербезопасность»

ОТЧЕТ
О НАУЧНО-ИССЛЕДОВАТЕЛЬКОЙ РАБОТЕ

«Применение природных алгоритмов при фильтрации нежелательных рассылок»

Исполнитель:

студент гр. Б18-505 Козловская А. А.

подпись, дата

Научный руководитель: Куприяшин М. А.

подпись, дата

Зам. зав. каф. № 42: Когос К.Г.

подпись, дата

Москва — 2021

РЕФЕРАТ

Пояснительная записка содержит 34 страницы, 2 рисунка и 16 источников.

Ключевые слова: СПАМ, МАШИННОЕ ОБУЧЕНИЕ, НАИВНЫЙ БАЙЕСОВСКИЙ КЛАССИФИКАТОР, МАШИНЫ ОПОРНЫХ ВЕКТОРОВ, ОПТИМИЗАЦИЯ, ПРИРОДНЫЙ АЛГОРИТМ, ГИПЕРПАРАМЕТР, ОЦЕНКА ПРОИЗВОДИТЕЛЬНОСТИ МОДЕЛИ.

Целью работы является разработка архитектуры системы для оценки эффективности природных алгоритмов в задаче классификации спама.

Объект исследования – классификаторы спама на основе машинного обучения.

Предмет исследования – оптимизация гиперпараметров алгоритмов машинного обучения.

Методы исследования – анализ существующих подходов к оптимизации гиперпараметров, синтез системы подбора гиперпараметров с применением природных алгоритмов.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящем отчете применяются следующие термины с соответствующими определениями, обозначениями и сокращениями:

Спам – нежелательные сообщения, массово рассылаемые по электронной почте

МО – машинное обучение

MTA – агент передачи сообщений, программное обеспечение, которое передает сообщения электронной почты с одного компьютера на другой с помощью SMTP (Message Transfer Agent)

IA – интеллектуальный агент, в области искусственного интеллекта относится к автономному лицу (человеку, устройству, программе), которое действует, направляя свою деятельность к достижению целей, через восприятие окружающей среды с помощью наблюдения через датчики и последующие интеллектуальные механизмы (Intellectual Agent)

SI – роевой интеллект, коллективное поведение децентрализованных, самоорганизованных систем, естественных или искусственных (Swarm Intelligence)

ВВЕДЕНИЕ

Электронная почта упростила способы общения как для многих организаций, так и для частных лиц. Однако этот способ общения часто используется спамерами для получения мошеннической выгоды путем рассылки нежелательных электронных писем. [1] По данным Лаборатории Касперского за 2020 год, доля спама в почтовом трафике превышает 50%, причем большая его часть (21,27%) пришла из России [2]. На этом фоне более быстрое и точное детектирование спама остается актуальной задачей.

Для решения задачи фильтрации нежелательных рассылок часто применяются алгоритмы машинного обучения. Качество их работы во многом зависит от правильно подобранных параметров, которые необходимо подбирать экспериментальным путем. Зачастую это осуществляется методами перебора, что не всегда дает желаемую производительность. В связи с этим применяются алгоритмы оптимизации, в том числе и природные.

В течение сотен лет эволюции многие живые организмы развили особые способности, не дающие им погибнуть. Успех биологических организмов вдохновил исследователей, занимающихся задачами оптимизации, на создание алгоритмов, копирующих поведение природы. За последние годы сообщество таких исследователей значительно выросло, достигнув большого разнообразия в том, что касается их источников вдохновения [3].

Современными исследованиями в области природных алгоритмов занимаются такие учёные, как Peter J. Bentley, Derviş Karaboğa, Xin-She Yang.

Данная работа, в соответствии с федеральным государственным образовательным стандартом высшего профессионального образования по направлению подготовки 090900 «Информационная безопасность», отвечает следующим задачам экспериментально-исследовательской деятельности: сбор, изучение научно-технической информации, отечественного и зарубежного опыта по тематике исследования; проведение экспериментов по заданной методике, об-

работка и анализ результатов.

Все представленные в работе результаты получены автором лично.

В первом разделе проводится обзор различных подходов к фильтрации спама.

Во втором разделе фильтрация спама рассматривается как задача машинного обучения, приводятся некоторые алгоритмы машинного обучения, используемые для ее решения.

В третьем разделе проводится обзор возможностей природных алгоритмов, описывается метод применения природных алгоритмов для оптимизации алгоритмов машинного обучения.

В четвертом разделе рассмотрены вопросы, касающиеся оценки производительности моделей машинного обучения.

В пятом разделе приведен план эксперимента, описана система для проведения эксперимента.

В шестом разделе описаны использованные в данной работе природные алгоритмы.

В седьмом разделе определены инструменты для программной реализации системы.

Восьмой раздел содержит описание реализованной системы.

1 Классификация методов фильтрации спама в электронной почте

Одна из причин, по которой спам трудно фильтровать, заключается в его динамическом характере. Характеристики спама, такие как темы, повторяющиеся термины и т. д., в электронной почте быстро меняются со временем, поскольку появляются новые способы обхода спам-фильтров. Эти способы включают: обфускацию слов, спам в виде изображений, рассылку спама по электронной почте с взломанных компьютеров и другие. Понимание природы и эволюции спама может помочь в разработке мер противодействия. Нет универсального метода защиты от спама, однако наилучший подход должен обладать механизмом для определения эволюции характеристик спама. Среди всех традиционных подходов огромного успеха в борьбе со спамом удалось достичь благодаря фильтрации на основе содержимого, в частности, системам на основе машинного обучения. Они обучаются и адаптируются к новым угрозам, реагируя на меры противодействия спамеров.

Согласно работе [4], классификация методов фильтрации спама включает в себя подходы на основе репутации, на основе текстового содержимого и на основе машинного обучения.

1.1 На основе репутации (Reputation-based)

Фильтрация на основе репутации включает в себя анализ источника сообщения, анализ социальных сетей, анализ трафика и анализ протокола.

1.1.1 Анализ источника сообщения

Анализ источника сообщения включает в себя:

— Blacklisting – создание и поддержание на уровне пользователя или сервера списка адресов электронной почты или IP-адресов сервера, с которого, как установлено, исходит спам. Сообщение автоматически блокируется на этапе подключения SMTP;

— Whitelisting – в противоположность черному списку, список предварительно утвержденных или доверенных контактов, доменов или IP-адресов,

которые могут общаться с пользователем почты.

1.1.2 Анализ социальных сетей

Социальные сети очень полезны для определения надежности отправителей, поэтому в подходах к фильтрации спама стали использоваться взаимодействия в социальных сетях. Например, можно анализировать поля заголовка электронной почты, чтобы построить граф социальной сети пользователя, а затем классифицировали сообщения электронной почты на основе «коэффициента кластеризации» подкомпонента графа;

1.1.3 Анализ трафика

Анализ трафика включает в себя следующие подходы:

— Анализ объема писем – такой фильтр использует алгоритм, который проверяет, сколько электронной почты получено от определенного хоста во время последних подключений. Если полученное количество писем превышает определенный порог, оно классифицируется как спам. Этот фильтр смог правильно классифицировать все допустимые электронные письма для достаточно высокого порога. Недостатком этого фильтра является то, что при его использовании много ложных срабатываний [5];

— SMTP Flow – анализ пути SMTP работает путем изучения «спамовости» или легитимности IP-адресов путем изучения истории электронной почты, доставленной через этот IP-адрес. Анализ SMTP-трафика при использовании в сочетании с традиционными фильтрами действительно повышает точность фильтров.

1.1.4 Анализ протокола

Анализ протокола включает в себя:

— C/R (challenge-response) systems – это спам-фильтр, который автоматически отправляет ответ с запросом (предполагаемому) отправителю входящего электронного письма. В этом ответе предполагаемого отправителя просят выполнить некоторые действия, чтобы гарантировать доставку исходного сообщения, которое в противном случае не было бы доставлено. Действие, кото-

рое нужно выполнить, может варьироваться от простого вопроса до *CAPTCHA* («Полностью автоматизированный общедоступный тест Тьюринга, позволяющий отличить компьютеры от людей»). Отправитель обязан правильно ответить в своем ответе; иначе его сообщение будет удалено или помещено в папку для спама. Хотя этот метод эффективен для перехвата спама из автоматических систем или бот-сетей, он приводит к нежелательной задержке в процессе доставки. Системы C/R – противоречивые решения, и их часто критикуют из-за неудобств, связанных с накладными расходами на связь. Кроме того, могут быть заблокированы легальные электронные письма из автоматических списков рассылки, так как они не справятся с задачей;

— SMTP Flow – когда SMTP-клиент подключается в первый раз, сервер-получатель может проверить, заблокирован ли IP-адрес отправителя или его адрес электронной почты или предварительно одобрен. Может быть и так, что их нет ни в черном, ни в белом списке. В этом случае сообщение временно отклоняется, и получатель отвечает сообщением о временной ошибке SMTP. Затем MTA-получатель записывает идентичность недавних попыток, и его базы данных обновляются информацией о новых клиентах; в соответствии с требованиями SMTP RFC, клиент пытается повторить попытку позже. Следующая попытка может быть принята для законных отправителей. Этот метод предполагает, что спамеры не тратят время на постановку в очередь или повторные попытки своих сообщений, и те, кто это делает, вероятно, попадут в черный список в публичных черных списках (DNSBLS) во время двух попыток. Хотя этот прием кажется очень эффективным, уклониться от него также очень просто. Спамеры могут использовать зомби для повторной попытки спамера.

1.2 На основе текстового содержимого

Фильтрация на основе текстового содержимого включает в себя эвристические и fingerprint-based фильтры.

1.2.1 Эвристические фильтры

Первоначально спам-фильтры следовали подходу «инженерии знаний» и основывались на закодированных правилах или эвристиках. Эвристический фильтр на основе содержимого анализирует содержимое сообщения и классифицирует его как спам или обычное письмо, основываясь на появлении в нем «спамных» слов, таких как «виагра» или «лотерея». Они были разработаны на основе знания закономерностей, наблюдаемых в сообщениях.

Недостатком эвристических фильтров является то, что разработка эффективного набора правил – трудоемкое дело, более того, правила необходимо постоянно обновлять, чтобы идти в ногу с новейшими тенденциями в области спама. Спамеры начали использовать «обфускацию» контента, маскируя определенные термины, которые очень распространены в спам-сообщениях. Более того, написание правил, основанных на регулярных выражениях, затруднено и подвержено ошибкам. Несмотря на эти ограничения, решение для фильтрации на основе правил пользовалось успехом с 2004 года до конца прошлого десятилетия.

1.2.2 Fingerprint-based фильтры

К данной категории относятся следующие подходы:

— Honeypots (приманки) – это сервер-ловушка или система, настроенная исключительно для сбора спама или информации о злоумышленниках. Он также используется для идентификации сборщиков адресов электронной почты с помощью специально созданных адресов электронной почты и для обнаружения ретрансляторов электронной почты. Большинство ловушек-приманок – это бездействующие учетные записи электронной почты, логика которых заключается в том, что, если мертвый почтовый ящик не может согласиться на получение электронной почты, любой, кто отправляет это письмо, должен быть спамером;

— Подход на основе сотрудничества – спамеры обычно рассылают спам

огромному количеству получателей. Вполне вероятно, что такой же спам был получен кем-то другим. Совместная фильтрация спама – это распределенный подход к фильтрации спама, при котором все сообщество работает вместе, имея общие знания о спаме. Подход на основе сотрудничества не учитывает содержание электронных писем; скорее, для этого требуется накопление любой идентифицирующей информации, касающейся спам-сообщений, например – тема, отправитель, результат вычисления математической функции над телом электронного письма и т. д. Спам-сообщения имеют цифровые следы, которые делятся с сообществом ранние приемники. Затем пользователи сообщества используют эти отпечатки пальцев спама для идентификации электронных писем со спамом. Однако такие схемы страдают от проблем с масштабируемостью и некоторых лежащих в основе неявных предположений;

— Подход на основе сигнатур – множество антивирусных продуктов работают на основе сигнатур. Хэши ранее идентифицированных спам-сообщений хранятся в базе данных на уровне MTA. Все входящие сообщения электронной почты проверяются по этим хэшам. Поскольку сигнатуры точно соответствуют шаблонам, эта схема может обнаруживать известный спам с очень высокой степенью уверенности. Однако серьезным недостатком является то, что неизвестный или вновь созданный спам сможет пройти через этот фильтр, не будучи обнаруженным. Базы сигнатур необходимо постоянно обновлять. Кроме того, спамеры могут вводить случайную строку в спам-сообщения для генерации различных хэшей.

1.3 Фильтрация на основе машинного обучения

Алгоритмы машинного обучения достигли наибольшего успеха среди всех предыдущих методов. Некоторые из них будут рассмотрены в следующем разделе.

2 Фильтрация спама на основе машинного обучения

Алгоритмы машинного обучения достигли наибольшего успеха среди всех перечисленных ранее методов, использованных в задаче фильтрации спама. Сегодня самые успешные спам-фильтры используют машинное обучение [4].

2.1 Фильтрация спама как задача машинного обучения

Автоматическая классификация электронной почты использует статистические подходы или методы машинного обучения и направлена на построение модели или классификатора специально для задачи фильтрации спама из потока почты пользователей.

Задача фильтрация спама в контексте машинного обучения – это задача классификации, в которой легитимные сообщения электронной почты рассматриваются как отрицательные экземпляры, а спам – как положительные. Для построения модели или классификатора требуется набор предварительно классифицированных документов. Процесс построения модели называется обучением. Примеры, которые система использует для обучения, называются обучающим набором *(training set)*. Каждый обучающий пример называется обучающим образцом *(training instance)*.

Классификация является типичной задачей обучения с учителем. При обучении с учителем обучающие данные, включают желательные решения, называемые метками. Алгоритм сначала тренируется на объектах из обучающей выборки, для которых заранее известны метки классов. Затем для каждого объекта из тестового набора данных *(test set)* обученный алгоритм предсказывает метку класса.

2.2 Модели машинного обучения, используемые для фильтрации спама

Успех алгоритмов машинного обучения в области фильтрации спама отчасти связан с тем, что обучить и построить классификатор для сообщений электронной почты, получаемых отдельными пользователями, легче, чем со-

здавать и настраивать набор правил фильтрации. Кроме того, спам-фильтры на основе машинного обучения переобучаются при использовании и сводят к минимуму ручные усилия.

2.2.1 Наивный Байесовский классификатор (NBC)

Классификатор был впервые разработан в 1990-х годах, и наиболее известен своим применением как один из первых методов фильтрации спама [6]. При Байесовском подходе максимизируется апостериорная вероятность класса.

В естественном языке вероятность появления определенного слова в письме сильно зависит от контекста. Байесовский классификатор представляет письмо как набор слов, вероятности появления которых условно не зависят друг от друга. Такой подход также называется моделью ”Мешок слов”. Исходя из предположения о независимости, условная вероятность принадлежности письма к категории спама аппроксимируется произведением условных вероятностей всех входящих в него слов.

Теорема Байеса применительно к задаче классификации спама имеет вид:

$$P(Class|WORD) = \frac{P(WORD|Class) \times P(Class)}{P(WORD)} \tag{1}$$

где:

— «$Class$» – либо «$Spam$», либо «Ham»;

— $WORD$ – это $(word_1, word_2, \ldots word_n)$;

— $P(Class|WORD)$ – условная вероятность того, что письмо принадлежит классу $Class$;

— $P(WORD|Class)$ – вероятность обнаружить письмо среди всех писем класса $Class$;

— $P(Class)$ – полная вероятность встретить письмо класса $Class$ в корпусе писем;

— $P(WORD)$ – безусловная вероятность письма в наборе писем;

Предположение о независимости: $P(WORD|Spam) \approx P(word_1|Spam) \ldots$

$P(word_n|Spam) = \prod_{i=1}^{n} P(word_i|Spam)$.

Если $Class = Spam$, уравнение (1) примет вид:

$$P(Spam|WORD) = \frac{\prod_{i=1}^{n} P(word_i|Spam) \times P(Spam)}{P(word_1, \ldots, word_n)} \quad (2)$$

Существует три типа наивных байесовских классификаторов: мультиномиальные, гауссовские и Бернулли. Для идентификации спама в электронной почте был выбран полиномиальный наивный байесовский алгоритм, поскольку он связан с текстом и превосходит по своим характеристикам гауссовский алгоритм и алгоритм Бернулли [1].

Несмотря на свои явно чрезмерно упрощенные предположения, наивные байесовские классификаторы довольно хорошо работают во многих реальных ситуациях, в том числе, при фильтрации спама. Им также требуется небольшой объем обучающих данных для оценки необходимых параметров [7].

2.2.2 Мультиномиальный наивный Байесовский классификатор (MNB)

В задачах классификации текста данные обычно представлены как счетчики векторов слов. Распределение параметризуется векторами $\theta_y = (\theta_{y1}, \ldots, \theta_{yn})$ для каждого класса y, где n - количество функций (в классификации текста - размер словаря) и θ_{yi} это вероятность $P(x_i \mid y)$ признака i из выборки, принадлежащей к классу y. Параметры θ_y оценивается сглаженной версией максимального правдоподобия, то есть подсчетом относительной частоты:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n} \quad (3)$$

где:

— $N_{yi} = \sum_{x \in T} x_i$ – это количество раз, которое появляется признак i в образце класса y

— $N_y = \sum_{i=1}^{n} N_{yi}$ – общее количество всех признаков для класса y

Сглаживающий параметр $\alpha \geq 0$ учитывает особенности, отсутствующие в обучающих выборках, и предотвращает нулевые вероятности в дальнейших

вычислениях. Параметр $\alpha = 1$ называется сглаживанием Лапласа, а $\alpha < 1$ называется сглаживанием Лидстоуна [7].

2.2.3 Машины опорных векторов (SVM)

В отличие от наивного Байеса, SVM – не вероятностный алгоритм.

Гиперплоскость – пространство размерностью на единицу меньше размерности исходного пространства. Основная цель машины опорных векторов в задаче бинарной классификации – найти уравнение разделяющей гиперплоскости $w_1x_1+w_2x_2+\ldots+w_nx_n+w_0 = 0$ в пространстве R^n, которая бы разделила два класса неким оптимальным образом.

Возможные значения меток классов $Y = \{-1, +1\}$. Объект – вектор в пространстве R^n с N признаками $x = (x_1, x_2, \ldots, x_n)$. Алгоритм при обучении должен построить функцию $F(x) = y$, аргументом x которой является объект из пространства R^n, а результатом – метка класса y.

Любая гиперплоскость может быть задана в виде $\langle w, x\rangle + b$. Метод опорных векторов строит классифицирующую функцию

$$F(x) = sign(\langle w, x\rangle + b) \tag{4}$$

где:

— $\langle , \rangle$ – скалярное произведение;

— w – нормальный вектор к разделяющей гиперплоскости;

— b – вспомогательный параметр;

Объекты, для которых $F(x) = 1$, оказываются по одну сторону гиперплоскости, то есть попадают в один класс, а объекты с $F(x) = -1$ – по другую, соответственно, попадают в другой класс.

w и b выбираются таким образом, чтобы максимизировать расстояние до каждого класса. Другими словами, алгоритм максимизирует отступ (англ. *margin*) между гиперплоскостью и объектами классов, расположенными к ней ближе всего. Такие объекты называются опорными векторами *(support vectors)*.

Расстояние до каждого класса равно $\frac{1}{\|w\|}$. Проблема нахождения максимума $\frac{1}{\|w\|}$ эквивалентна проблеме нахождения минимума $\|w\|^2$. Задача оптимизации:

$$\begin{cases} arg \min\limits_{w,b} \|w\|^2 \\ y_i(\langle w, x_i \rangle + b) \geqslant 1, \ i = 1, \ldots, m \end{cases} \tag{5}$$

решается с помощью множителей Лагранжа.

В случае линейной неразделимости, когда данные нельзя разделить гиперплоскостью, поступают все элементы обучающей выборки вкладываются в пространство X более высокой размерности с помощью специального отображения $\varphi : R^n \to X$, которое выбирается таким образом, чтобы выборка была линейно разделима в X.

Классифицирующая функция F принимает вид $F(x) = sign(\langle w, \varphi(x) \rangle + b)$. Ядро *(kernel function)* классификатора: $k(x, x') = \langle \varphi(x), \varphi(x') \rangle$. Ядром может служить любая положительно определенная симметричная функция двух переменных. В разных алгоритмах SVM используются разные типы функций ядра. Например, линейная, нелинейная, полиномиальная, радиальная базисная функция (RBF) и сигмоид [8].

Машины опорных векторов (SVM) считаются одним из лучших алгоритмов обучения с учителем. Они обеспечивают превосходную производительность обобщения, требуют меньше примеров для обучения и могут обрабатывать многомерные данные [9].

2.2.4 Стохастический градиентный спуск

На практике часто применяются машины опорных векторов, представляющие собой линейную модель со «стохастическим градиентным спуском (SGD)».

Строго говоря, SGD – это просто метод оптимизации и не соответствует конкретному семейству моделей машинного обучения [1].

Этот алгоритм обеспечивает более точные результаты, чем сам SVM (ал-

горитм SVC в scikit-learn [10]). Недостатком работы с SVC является то, что он не может обрабатывать большой набор данных, тогда как SGD обеспечивает эффективность и разнообразие возможностей для настройки [9].

2.3 Выбор алгоритма МО для эксперимента

Наивный байесовский классификатор использовался во многих приложениях обработки информации, включая обработку естественного языка, поиск информации и т. д. Данный метод особенно подходит для решения задач с большим объемом входных данных.

Однако на данный момент спамеры умеют легко обходить фильтр Байеса, просто дописывая в конец письма много «легитимных» слов [4]. Этот метод обхода получил название «Отравление Байеса», а для фильтрации спама стали применять другие алгоритмы. Но наивный Байес навсегда остался в учебниках как очень простой, красивый и один из первых приносящих практическую пользу.

Поэтому в эксперименте предпочтительнее использовать модель машины опорных векторов со «стохастическим градиентным спуском». К тому же, большее количество параметров, которые можно оптимизировать, предоставляют бóльшие возможности для получения модели, обеспечивающей наилучшую производительность, измеренную на проверочном наборе.

3 Обзор природных алгоритмов

В последние годы в литературе описано большое количество алгоритмов, основанных на природе и биологии. Это семейство алгоритмов симулирует различные биологические процессы, наблюдаемые в природе, чтобы решать сложные задачи оптимизации [11]. Каждый естественный процесс можно считать адаптируемым и имитируемым для создания нового метаэвристического подхода, но с различными возможностями достижения глобальных оптимальных решений для задач оптимизации [3].

3.1 Задача оптимизации

Без преувеличения можно сказать, что в оптимизация используется повсюду: от инженерного проектирования до бизнес-планирования и маршрутизации Интернета до планирования праздников. Почти во всех этих действиях мы пытаемся достичь определенных целей или оптимизировать что-то, например прибыль, качество и время. Поскольку в реальных приложениях ресурсы, время и деньги всегда ограничены, мы должны найти решения для оптимального использования этих ценных ресурсов при различных ограничениях. Математическая оптимизация или программирование - это изучение таких проблем планирования и проектирования с использованием математических инструментов. В настоящее время компьютерное моделирование становится незаменимым инструментом для решения таких задач оптимизации с помощью различных эффективных алгоритмов поиска.

С математической точки зрения, большинство задач оптимизации можно описать в общем виде:

$$\underset{X \in R^n}{minimize}\ f_i(X), (i = 1, 2, \ldots, M), \tag{6}$$

$$subject\ to\ h_j(X) = 0, (j = 1, 2, \ldots, J), \tag{7}$$

$$g_k(X) \leq 0, (k = 1, 2, \ldots, K) \tag{8}$$

Где $f_i(X), h_j(X), g_k(X)$ – функции порождающего вектора

$$X = (x1, x2, \ldots, x_n)^T \tag{9}$$

Здесь компоненты x_i переменной x называются переменными решения. Функции $f_i(x)$, где $i = 1, 2, \ldots, M$, называются *целевыми функциями* или функциями стоимости. Пространство, охватываемое переменные решения, называется *пространством поиска* R^n, а пространство, образованное значениями целевых функций, называются *пространством решений* или пространством ответов. Равенство (7) с h_j и неравенство (8) с g_k называются *ограничениями*.

Проблемы оптимизации обычно описывают в терминах локальной или глобальной оптимизации:

— *Локальная оптимизация*, при которой алгоритм ищет точку, которая является только локально оптимальной, что означает, что она минимизирует целевую функцию среди возможных точек, которые находятся рядом с ней. Методы локальной оптимизации широко используются в приложениях, где есть смысл найти хорошую, если не самую лучшую точку [12];

— *Глобальная оптимизация*, при которой алгоритм ищет глобальный оптимум, используя механизмы для поиска в более крупных частях пространства поиска. Глобальная оптимизация используется для задач с небольшим количеством переменных, где время вычислений не критично, а ценность поиска истинного глобального решения очень высока [12].

3.2 Оптимизация гиперпараметров в алгоритмах МО

Гиперпараметр модели – это параметр, который является внешним по отношению к модели и значение которого невозможно оценить по данным. Мы не можем знать наилучшее значение гиперпараметра модели для той или иной проблемы. Часто гиперпараметры можно установить с помощью эвристики.

К основным подходам к оптимизации гиперпараметров относятся:

— *Поиск по сетке* представляет собой простой перебор по заданному вручную подмножеству гиперпараметрического пространства алгоритма обучения;

— *Случайный поиск* заменяет перебор всех комбинаций их случайным выбором;

— *Градиентный спуск* – для конкретных алгоритмов МО (например, машины опорных векторов) можно вычислить градиент относительно гиперпараметров, а затем оптимизировать гиперпараметры с помощью градиентного спуска;

— Идея *байесовской оптимизации* состоит в построении вероятностной модели целевой функции и использовании ее для выбора наиболее подходящих гиперпараметров;

— *Эвристический подход* подразумевает применение различных эвристическикх и метаэвристических алгоритмов оптимизации.

3.3 Алгоритмы оптимизации

Также алгоритмы оптимизации можно разделить на *детерминированные* и *стохастические*. В отличие от детерминированных моделей, которые дают одинаковые точные результаты для определенного набора входных данных, стохастические модели представляют данные и предсказывают результаты, учитывающие определенные уровни непредсказуемости или случайности.

Стохастические алгоритмы бывают *эвристические* и *метаэвристические*. Можно сказать, что эвристика использует зависящую от проблемы информацию для поиска «достаточно хорошего» решения конкретной проблемы, в то время как метаэвристика представляет собой общие алгоритмические идеи, которые можно применять к широкому кругу проблем.

Стоит отметить, что в литературе не существует согласованных определений эвристики и метаэвристики. Некоторые используют термины «эвристика» и «метаэвристика» как синонимы. Однако в последнее время все стохастиче-

ские алгоритмы с рандомизацией и локальным поиском называются метаэвристическими [11].

3.4 Классификация природных алгоритмов

Классифицировать природные алгоритмы можно по разным признакам, однако наиболее часто этим признаком служит биологических источник вдохновения [3].

3.4.1 Эволюционные алгоритмы

В эту категорию входят алгоритмы, основанные на популяциях, а также на принципах естественной эволюции. К ним относятся как классические алгоритмы эволюционных вычислений, такие как генетический алгоритм (GA), стратегии эволюции (ES) и дифференциальная эволюция (DE), таки менее известные алгоритмы, основанные на воспроизведении различных биологических организмов, таких как пчелиные матки и сорняки.

3.4.2 Алгоритмы на основе роевого интеллекта

Роевой интеллект (SI) – это уже устоявшийся термин. Его можно определить как коллективное поведение децентрализованных, самоорганизованных систем в естественной или искусственной среде. Выражение было предложено в контексте роботизированных систем, но с годами обобщено, чтобы обозначить возникновение коллективного разума из группы простых агентов, управляемых простыми поведенческими правилами. Таким образом, биологические метаэвристики, относящиеся к роевому интеллекту, основаны на коллективном поведении сообществ животных, таких как колонии насекомых или стаи птиц, в которых коллективный разум позволяет решать задачи оптимизации.

Разделение на подкатегории по среде может быть следующим:

— Летающие животные

Эта категория включает в себя метаэвристику, основанную на концепции роевого интеллекта, в которой траектория агентов определяется полетом, как у птиц, летучих мышей или других летающих животных. Самыми известными алгоритмами в этой подкатегории являются метод роя частиц (PSO) [2] и

алгоритм пчелиной колонии (ABC);

— Наземные животные

Метаэвристика в этой категории основана на поисках пищи или передвижениях наземных животных. Наиболее известным подходом в этой категории является муравьиный алгоритм (ACO), который воспроизводит механизм, используемый муравьями для определения местоположения источников пищи и информирования об их существовании своим собратьям в колонии. В эту категорию также входят другие популярные алгоритмы, такие как алгоритм стаи серых волков (GWO), алгоритм льва (LOA), которые имитируют методы охоты, используемые этими животными;

— Водные животные

Этот тип метаэвристических алгоритмов основан на поведении водных животных. Сюда относятся такие популярные алгоритмы, такие как стадо криля (KH), алгоритм китов (WOA), а также алгоритмы, основанные на эхолокации, используемой дельфинами для обнаружения рыб, такие как оптимизация партнеров дельфинов (DPO) и эхолокация дельфинов (DEO);

— Микроорганизмы

Метаэвристика, основанная на микроорганизмах, связана с поиском пищи бактериями. Колония бактерий совершает движение в поисках пищи. Другие типы метаэвристик, которые могут быть частью этой категории, связаны с вирусами, которые обычно воспроизводят процесс заражения клетки вирусом. Наиболее известным алгоритмом этой категории является алгоритм оптимизации сбора бактерий (BFOA).

При разделении на подкатегории по вдохновляющему поведению каждый алгоритм классифицируется как принадлежащий к одному из следующих поведенческих паттернов:

— Движение

Алгоритм относится к подкатегории движения, если биологическое вдохновение в основном состоит в том, как животное перемещается в окружающей

среде;

— Собирательство

В алгоритмах на основе собирательства поведение животного определяет механизм, используемый для получения пищи.

3.4.3 Алгоритмы на основе физики / химии

Алгоритмы этой категории характеризуются тем, что они имитируют поведение физических или химических явлений, таких как гравитационные силы, электромагнетизм, электрические заряды и движение воды, а также химические реакции и движение частиц газа. В этой категории мы можем найти некоторые хорошо известные алгоритмы, разработанные в прошлом веке, такие как алгоритм имитации отжига (SA) или алгоритм гравитационного поиска (GSA). Другие алгоритмы, такие как гармонический поиск (HS), относятся к процессу сочинения музыки, человеческому изобретению, которое имеет больше общего с другими физическими алгоритмами в том, что касается использования звуковых волн, чем с алгоритмами, основанными на социальном поведении человека.

3.4.4 Алгоритмы, основанные на социальном поведении человека

Алгоритмы, попадающие в эту категорию, вдохновлены человеческими социальными концепциями, такими как принятие решений и идеями, связанными с конкуренцией идеологий внутри общества, таких как алгоритм идеологии (IA) или политическими концепциями, такими как алгоритм империалистической колонии (ICA). В эту категорию также входят алгоритмы, имитирующие спортивные соревнования, такие как алгоритм футбольной лиги (SLC). Процессы мозгового штурма также заложили вдохновляющие основы нескольких алгоритмов, таких как алгоритм оптимизации мозгового штурма (BSO.2) и глобальный передовой алгоритм оптимизации мозгового штурма (GBSO).

3.4.5 Алгоритмы на основе растений

Эта категория объединяет все алгоритмы оптимизации, поиск которых основан на растениях. В этом случае, в отличие от методов в категории роевого

интеллекта, нет связи между агентами. Один из самых известных – это алгоритм оптимизации леса (FOA.1), вдохновленный процессом воспроизводства растений.

3.4.6 Алгоритмы со смешанными источниками вдохновения

В эту категорию входят алгоритмы, которые не подходят ни к одной из предыдущих, то есть в ней можно обнаружить алгоритмы с различными характеристиками, такими как оптимизация пар Инь-Ян (YYOP). Хотя эта категория неоднородна и не демонстрирует единообразия среди алгоритмов, которые представляет, ее включение в классификацию служит примером очень разных источников вдохновения, существующих в литературе. С появлением новых алгоритмов эта группа должна дать начало новым категориям.

4 Природные алгоритмы в настройке гиперпараметров

Как уже было сказано, оптимизация гиперпараметров – это поиск набора переменных, который позволит алгоритму достичь более качественных результатов. Гиперпараметры обычно оказывают значительное влияние на успех алгоритмов машинного обучения. Когда требуется охватить большое пространство поиска, чтобы не застрять в локальном минимуме, и одновременно ограничить количество обучаемых моделей до приемлемого количества, при оптимизации гиперпараметров прибегают использованию более сложных алгоритмов, чем поиск по сетке или случайный поиск. В частности, возможно применений природных алгоритмов.

4.1 Оценка производительности моделей

Для оценки производительности некоторой обученной модели можно используют определенные метрики.

4.1.1 Матрица ошибок

Обнаружение спама в электронных письмах можно оценить с помощью различных показателей эффективности. Матрица ошибок используется для визуализации работы алгоритмов и может быть определена как на Рисунке 1.

	HAM	SPAM
HAM	TN	FP
SPAM	FN	TP

Рисунок 1 – Матрица ошибок

где:

— TN = TrueNegative – Письма, распознанные как письма;

— FP = FalsePositive – Спам-письма, распознанные как письма;

— FN = FalseNegative – Письма, распознанные как спам-письма;

— TP = TruePositive – Спам-письма, распознанные как спам-письма.

4.1.2 Accuracy

Accuracy – это отношение количества правильных прогнозов к общему количеству входных выборок [13]:

$$Accuracy = \frac{(TN + TP)}{(TP + FN + FP + TN)} \tag{10}$$

4.1.3 Recall

Recall описывает, сколько писем было правильно отнесено к спаму из общего количества писем, распознанных как спам $(TP + FN)$ [13]:

$$Recall = \frac{TP}{(TP + FN)} \tag{11}$$

4.1.4 Precision

Измерение precision заключается в доли верно идентифицированного спама из всех спам-писем $(TP + FP)$ [13]. Определяется уравнением:

$$Precision = \frac{TP}{(TP + FP)} \tag{12}$$

4.1.5 F1-score

Оценка $F1$ может быть интерпретирована как средневзвешенное значение precision и recall, где оценка $F1$ достигает своего лучшего значения при 1 и худшего значения при 0 [13]. Формула для оценки $F1$: »»»»> Stashed changes

$$F1 = \frac{2 \times (precision \times recall)}{(precision + recall)} \tag{13}$$

4.2 Переобучение

Мы, люди, иногда склонны делать чрезмерные обобщения. К сожалению, машины могут оказаться в такой же ситуации. В контексте МО это называется *переобучением*.

Ограничение модели с целью ее упрощения и снижения риска переобучения называется *регуляризацией* [14].

Тестирование модели на одних и тех же данных при выборе параметров является методологической ошибкой: модель, которая будет просто повторять

метки образцов, которые она только что увидела, будет иметь идеальную оценку, но не сможет предсказать что-либо полезное на новых данных [15].

Чтобы решить эту проблему, еще одна часть набора данных может быть представлена как так называемый проверочный набор *(validation set)*: обучение продолжается на обучающем наборе, после чего выполняется оценка на проверочном наборе, и когда эксперимент кажется успешным, окончательную оценку можно провести на тестовом наборе [16].

4.3 Переобучение

Мы, люди, иногда склонны делать чрезмерные обобщения. К сожалению, машины могут оказаться в такой же ситуации. В контексте МО это называется *переобучением*.

Ограничение модели с целью ее упрощения и снижения риска переобучения называется *регуляризацией* [14].

Тестирование модели на одних и тех же данных при выборе параметров является методологической ошибкой: модель, которая будет просто повторять метки образцов, которые она только что увидела, будет иметь идеальную оценку, но не сможет предсказать что-либо полезное на новых данных [15].

Чтобы решить эту проблему, еще одна часть набора данных может быть представлена как так называемый проверочный набор *(validation set)*: обучение продолжается на обучающем наборе, после чего выполняется оценка на проверочном наборе, и когда эксперимент кажется успешным, окончательную оценку можно провести на тестовом наборе [16].

Однако, разбивая доступные данные на три набора, мы резко сокращаем количество выборок, которые можно использовать для изучения модели, и результаты могут зависеть от конкретного случайного выбора для пары наборов.

Решением этой проблемы является процедура *перекрестной проверки* (сокращенно CV). Набор тестов по-прежнему должен храниться для окончательной оценки, но набор для проверки больше не нужен. В базовом подходе, называемом k-fold CV, обучающая выборка разбивается на k меньших наборов. Для

каждого набора выполняется:

- Модель обучается с использованием $k - 1$ наборов в качестве обучающих данных;

- Результирующая модель проверяется на оставшейся части данных (то есть используется в качестве тестового набора для вычисления показателя производительности).

Показатель производительности, полученный перекрестной проверкой, является средним из значений, вычисленных в цикле. Такой подход может быть дорогостоящим в вычислительном отношении, но не расходует слишком много данных. »»»»> fixed

5 Подготовка к проведению эксперимента

Для программной реализации системы оценки производительности алгоритмов оптимизации, необходимо спланировать эксперимент, разработать систему для проведения эксперимента, выбрать инструменты для ее реализации.

5.1 Планирование эксперимента

Цель эксперимента – выявить преимущества и недостатки тех или иных видов оптимизации гиперпараметров и их влияния на время обучения и производительность модели.

Входные данные – набор текстов писем электронной почты, для каждого из которых заранее известно, является это письмо спамом или нет.

Выходные данные – оценка нескольких обученных на входных данных классификаторов спама.

Для удобства проведения эксперимента ограничимся алгоритмами роевого интеллекта, которые работают по общему принципу, а метод роя частиц уже показал хорошие результаты в этой области [1].

План проведения эксперимента:

1. Подготовить данные для эксперимента;
2. Предварительно обработать данные. Разделить данные на тренировочный и тестовый наборы;
3. Настроить параметры алгоритма машинного обучения;
4. Реализовать алгоритм глобальной оптимизации функции на основе того или иного природного алгоритма;
5. Реализовать объектную функциию, подлежащую оптимизации алгоритмом роевого интеллекта. Эта функция должна вычислять оценку модели МО для каждой частицы с новыми параметрами, а возвращать среднее арифметическое от вектора этих параметров. Оценочной функцией может служить как одна из 4.1, так и нестандартная весовая функция;
6. Обучить модель с использованием полученных гиперпараметров;

7. Протестировать модель на тестовом наборе данных;

8. Оценить производительность модели, вычислив величины из 4.1 на тестовом наборе;

9. Повторить экспеприемент, получив те же параметры с использованием других подходов к оптимизации, описанных в 3.2;

10. Повторить эксперимент для другого природного алгоритма;

11. Провести сравнительный анализ получившихся оценок с целью выявления преимуществ и недостатков тех или иных видов оптимизации и их влияния на время обучения и производительность модели;

12. Объяснить полученные результаты эксперимента, сделать соответствующие выводы;

13. Обозначить направление дальнейших исследований.

5.2 Архитектура системы для проведения эксперимента

Механизм обнаружения спама должен иметь возможность работать с электронной почтой, включающей html-заголовки и другую лишнюю в контексте данной задачи информацию. В связи с этим набор входных данных подлежит предварительной обработке.

Данные также подлежат разделению на тренировочный и тестовый наборы.

Задаются начальные параметры алгоритма, желаемое значение оценочной функции (например, $Accuracy$), число частиц, максимальное количество итераций. Затем на тренировочном наборе данных проводится обучение модели с перекрестной проверкой на различных наборах гиперпараметров, предоставляемых оптимизационным алгоритмом. Процесс продолжается до тех пор, пока не будет достигнуто желаемое значение оценки алгоритма или максимальное число итераций.

Затем модель, обученная на лучших параметрах, тестируется на тестовом наборе данных, осуществляется оценка производительности полученной модели.

Рисунок 2 представляет процесс реализации описанной системы.

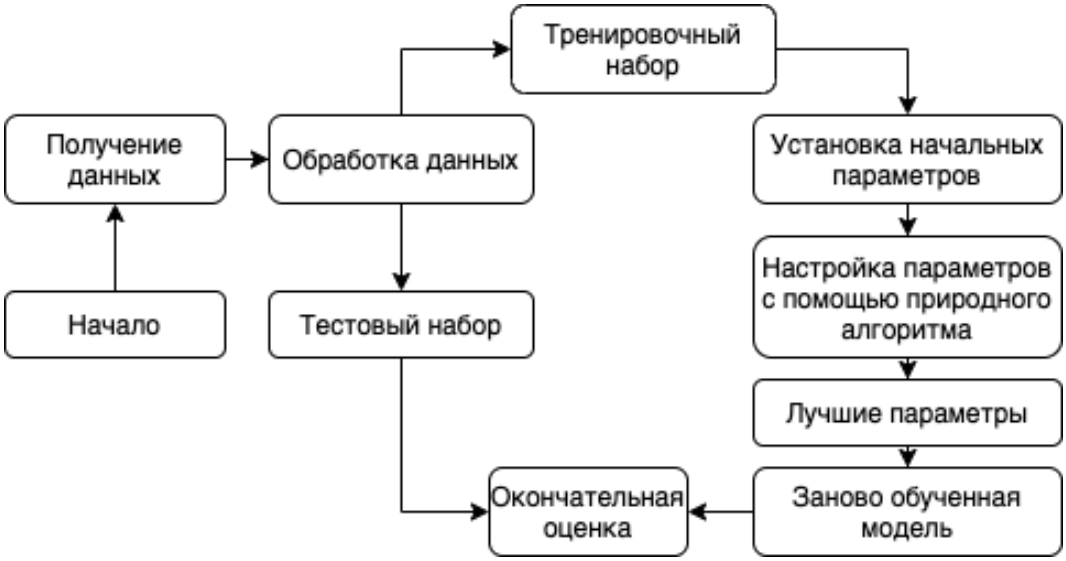

Рисунок 2 – Схема процесса обучения модели

5.3 Выбор инструментов для реализации системы

Для реализации системы выбран язык программирования Python версии 3.9 или выше, поскольку его использование обеспечивает наличие множества библиотек, предназначенных для машинного обучения.

Реализации описанных в 2.2 алгоритмов машинного обучения можно найти в библиотеке Scikit-Learn. Scikit-Learn – библиотека машинного обучения, написанная на Python, довольно популярная среди ученых-вычислителей. Она может похвастаться простым в использовании интерфейсом и содержит множество инструментов для машинного обучения, в том числе для классификация, предварительной обработки, кластеризации и регрессии данных. Многие из этих алгоритмов могут быть полезны ”из коробки”, но чтобы получить лучшую производительность, гиперпараметры должны быть настроены под конкретную проблемную область, в которой алгоритм будет использоваться. Для такой настройки будут использоваться алгоритмы роевого интеллекта, такие как, например, Firefly, Ant Colony Optimization и Bee Colony Optimization.

6 Определение набора алгоритмов для проведения эксперимента

Для данной работы решено было использовать алгоритмы роевого интеллекта, опубликованные в 2020-2021 годах:

— AO – Aquila Optimizer;

— HGS – Hunger Games Search;

— SSA – Sparrow Search Algorithm;

— MRFO – Manta Ray Foraging Optimization.

6.1 Оптимизатор орла (Aquila optimizer)

В работе [17] представлен новый метод оптимизации на основе популяций, называемый оптимизатор орла, основанный на поведении орлов в природе во время охоты. Оптимизационные функции данного алгоритма обусловлены четырьмя процессами: выбор пространства поиска высоким взлетом с вертикальным наклоном, разведка в пределах расходящегося пространства поиска контурным полетом с короткой планирующей атакой, использование в пределах сходящегося пространства поиска низким полетом с атакой медленного спуска, а также бег и захват добычи.

6.2 Поиск голодных игр (Hunger Games Search)

Метод оптимизации на основе популяций, называемый поиском голодных игр, основан на поведенческом выборе животных в соответствии с чувством голода. Этот метод поиска использует простую концепцию голода как наиболее важную гомеостатическую мотивацию и причины поведения всех живых существ. Кроме того, почти все животные используют вычислительно-логическ правила (игры). Часто такие действия являются следствиями эволюции.

Особенностью данного метода, по словам авторов, является его динамический характер, простая структура и высокая производительность, гибкость и масштабируемость, а также тот факт, что метод был не только проверен на хорошо известном наборе тестовых функций, но и применен к нескольким инженерным задачам [18].

6.3 Алгоритм поиска воробьев (Sparrow Search Algorithm)

Алгоритм поиска воробьев представляет собой новый подход к оптимизации на основе роевого интеллекта. Метод вдохновлен поведением воробьев в поисках пищи и в борьбе с хищниками. Авторы алгоритма утверждают, что предлагаемый метод может обеспечить высококонкурентные результаты. Более того, результаты двух практических инженерных задач также показывают, что алгоритм имеет высокую производительность в различных областях поиска [19].

6.4 Алгоритм оптимизации кормодобывания скатов манта (Manta Ray Foraging Optimization)

В основе природного алгоритма оптимизации кормодобывания скатами манта (MRFO) лежит разумное поведение скатов-мантов. Целью данного алгоритма, по утверждению авторов, является обеспечение альтернативного подхода к оптимизации для решения реальных инженерных проблем. Производительность алгоритма была оценена путем сравнения с другими современными оптимизаторами, с помощью функций оптимизации тестов и восьми реальных примеров инженерного проектирования. Результаты сравнения тестовых функций показывают, что данный подход намного превосходит своих конкурентов. Кроме того, реальные инженерные приложения демонстрируют достоинства этого алгоритма в решении сложных проблем с точки зрения затрат на вычисления и точности решения [20].

7 Определение инструментов и данных

Данный раздел описывает инструменты, используемые для программной реализации эксперимента, а также подготовку данных.

7.1 Оборудование и программное обеспечение

Проект реализован на ноутбуке с 16 ГБ оперативной памяти и процессором Intel Core i7 (2,60 ГГц). В качестве языка программирования был использован Python версии 3.9.7. А также его модули:

— mealpy – библиотека, предоставляющая реализации множества метаэвристических природных алгоритмов;

— nltk – библиотека для символьной и статистической обработки естественного языка для английского языка;

— numpy – библиотека, поддерживающая большие многомерные массивы и матрицы, а также большой набор высокоуровневых математических функций для работы с ними;

— pandas – библиотека для обработки и анализа данных, предлагающая, в частности, структуры данных и операции для управления таблицами и временными рядами;

— Scikit-learn – библиотека машинного обучения, включающая в себя различные алгоритмы классификации, регрессии и кластеризации.

7.2 Определение наборов данных

Для тестирования алгоритмов использовались общедоступные наборы данных, электронные письма в которых представлены строками. Список использованных наборов данных:

— Набор данных Ling-Spam (2893 писем) [21];

— Набор данных Spam-Assasin (9352 писем) [22];

— Набор данных Enron (33715 писем) [23].

С помощью программы, приведенной в Приложении, данные были приведены к единому простому виду, содержащему поля: содержимое письма (message) и

категория (label).

Категория, в свою очередь, может принимать два значения: обычные письма (ham) и спам (spam).

Соотношение спама и обычных писем для набора данных Ling-Spam представлено на 3, для Spam-assasin – на 4, для Enron – на 5.

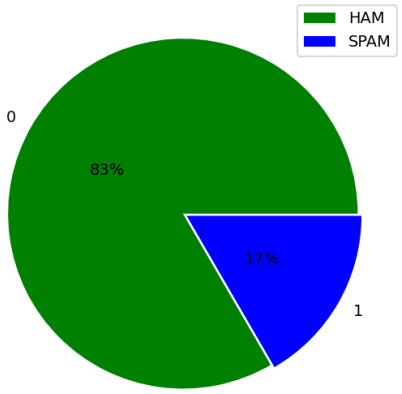

Рисунок 3 – Соотношение спама и обычных писем для набора данных Ling-Spam

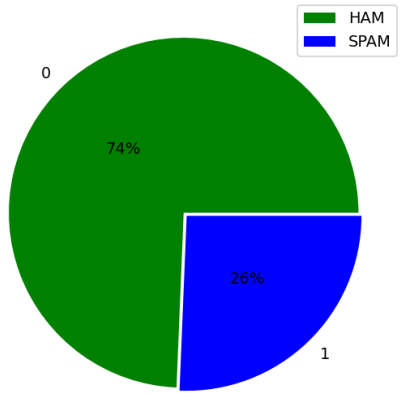

Рисунок 4 – Соотношение спама и обычных писем для набора данных Spam-Assasin

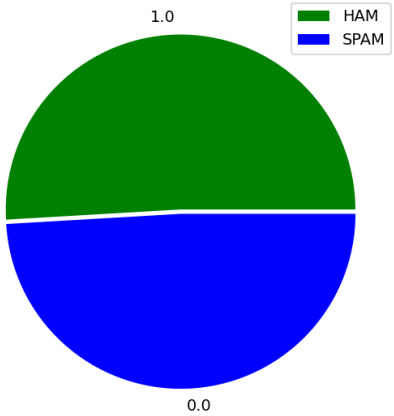

Рисунок 5 – Соотношение спама и обычных писем для набора данных Enron

7.3 Предварительная обработка данных

Текстовые данные требуют специальной подготовки, прежде чем их можно будет использовать для моделирования.

Программную реализацию подготовки данных можно найти в Приложении А.

7.3.1 Символьная обработка

В первую очередь необходимо очистить строки от ненужных символов. Обработка включает в себя:

— Замена символов переноса строки и переноса каретки пробелом;

— Замена повторяющихся пробелов одним;

— Удаление знаков препинания;

— Удаление подстрок, не являющихся словами;

— Удаление цифр;

— Приведение всех букв к строчному виду.

7.3.2 Токенизация

При работе с текстовыми данными нам необходимо было представить их в виде целых чисел или чисел с плавающей точкой. Набор текстовых документов преобразовывался в матрицу слов (токенов), из которых состоял текст. При

этом нужно было также исключить из обработки некоторые стоп-слова – различные артикли, предлоги, вводные слова. Это полезно, поскольку такие слова не очень полезны для определения того, является ли электронное письмо спамом или нет.

Далее матрицу подсчета преобразовывалась в нормализованное представление tf-idf.

TF-IDF (Term Frequency – Inverse Document Frequency) – это оценка, направленная на определение важности слова в документе, а также для учета его связи с другими документами из того же набора даннных. Tf (term frequency) означает частоту слова, а tf-idf – это частота слова, умноженная на величину, обратную частоте, с которой в наборе данных встречаются документы, его содержащие [24].

Формула, лежащая в основе статистического показателя TF-IDF:

$$tf - idf(w, d, D) = tf(w, d) * idf(w, D) \tag{14}$$

где:

— d – данный документ из нашего набора данных;

— D – набор документов;

— w – заданное слово в документе;

Частота слова среди всех слов документа:

$$tf(w, d) = log(1 + f(w, d)) \tag{15}$$

где $f(w, d)$ – частота слова w в документе d.

Обратная частота слова среди всех документов:

$$idf(w, D) = log(\frac{N}{f(w, D)}) \tag{16}$$

где N – количество всех документов в наборе.

8 Программная реализация тестовой системы

Основные этапы работы программы:

— Считывание и преобразования данных с помощью библиотек numpy и pandas;

— Преобразование текста в числовую матрицу слов с использованием TfidfVectorizer[25] и набора стоп-слов от модуля nltk;

— Настройка параметров использованием подхода Scikit-Learn [9] и био-алгоритмов для повышения точности моделей;

— Использование природных алгоритмов в реализации от модуля mealpy [26];

— Запись результатов.

8.1 Настройка параметров модели

Параметры настройки модели имеют большое влияние на обнаружение спам-писем и скорость обучения. Станадртные алгоритмы настройки предлагают 3 параметра для алгоритма SGD:

— Альфа (alpha) – чем выше значение, тем сильнее регуляризация. Также может использоваться для вычисления скорости обучения;

— Эпсилон (epsilon) – значение определяет скорость обучения алгоритма;

— Тол (tol) – критерий остановки.

Настройка модели проводилась с использованием:

— Параметров по умолчанию;

— Случайного поиска (подраздел 3.2);

— Природных алгоритмов (раздел 6).

Границы для всех трех параметров были установлены 10^{-3} до 10^{3} с шагом в одну степень. Набор был разделен на тренировочный и тестовый в отношении 70:30. Для усреднения результатов по каждому набору данных с каждым подходом к настройке, обучение проводилось 100 раз.

Недостатки использования случайного поиска и поиска по сетке очевидны – и тот, и дргуой используют и возвращают только те параметры, которые им переданы в качестве словаря, с той лишь разницей, что случайный поиск не предполагает полного перебора, а потому в нашем случае гораздо быстрее, чем поиск по сетке. Но наиболее удачная комбинация параметров может и не принадлежать переданным вариациям. Для поиска внутри области, а не по точкам, используем природные алгоритмы.

8.2 Применение природных алгоритмов оптимизации

В контексте данной работы задача оптимизации – максимизировать функцию точности (подраздел 4.1.2). Алгоритмы оптимизации были запущены с числом частиц (epoch) равным 10 и числом популяций (pop_size) равным 50. Границы поиска для каждого параметра – от 10^{-3} до 10^{3}.

9 Полученные результаты

Алгоритм	Данные	Тест	Accuracy	Precision	Recall	F1-score
AO	ling_spam	30%	99.31%	99.36%	96.05%	97.80%
HGS	ling_spam	30%	99.19%	99.35%	95.76%	97.64%
SSA	ling_spam	30%	93.95%	100%	74.00%	77.85%
MRFO	ling_spam	30%	99.19%	99.32%	95.51%	97.45%
RSCV	ling_spam	30%	99.10%	99.35%	95.77%	97.55%

Алгоритм	Данные	Тест	Accuracy	Precision	Recall	F1-score
AO	enron	30%	98.84%	98.07%	99.68%	98.87%
HGS	enron	30%	98.86%	98.12%	99.68%	98.89%
SSA	enron	30%	73.37%	70.85%	74.36%	66.12%
MRFO	enron	30%	98.86%	98.11%	99.68%	98.89%
RSCV	enron	30%	97.36%	96.02%	99.61%	97.65%

Алгоритм	Данные	Тест	Accuracy	Precision	Recall	F1-score
AO	spam_assasin	30%	97.95%	99.11%	92.94%	95.90%
HGS	spam_assasin	30%	97.99%	99.10%	93.06%	95.97%
SSA	spam_assasin	30%	86.21%	98.63%	61.61%	64.40%
MRFO	spam_assasin	30%	97.97%	99.07%	93.04%	95.97%
RSCV	spam_assasin	30%	97.86%	98.99%	92.57%	95.73%

Алгоритм	Данные	Тест	Accuracy	Precision	Recall	F1-score
AO	enron	25%	98.87%	98.13%	99.67%	98.89%
HGS	enron	25%	98.86%	98.12%	99.67%	98.89%
SSA	enron	25%	77.83%	80.55%	82.68%	74.69%
MRFO	enron	25%	98.84%	98.08%	99.68%	98.87%
RSCV	enron	25%	98.06%	96.77%	99.65%	98.16%

ЗАКЛЮЧЕНИЕ

В рамках данной работы осуществлен обзор различных подходов к фильтрации спама. В частности, задача фильтрация спама была рассмотрена как задача классификации в контексте машинного обучения. Описаны некоторые алгоритмы машинного обучения, используемые для решения данной задачи. Кроме того, приведена классификация природных алгоритмов. Также приведены различные метрики, используемые для оценки производительности моделей машинного обучения. Перечислены основные подходы к оптимизации гиперпараметров машинного обучения.

В ходе работы получены следующие результаты:

— Спланирован эксперимент по выявлению достоинств и недостатков различных подходов к оптимизации классификаторов спама на основе машинного обучения;

— Спроектирована система для проведения эксперимента по выявлению достоинств и недостатков различных подходов к оптимизации классификаторов спама на основе машинного обучения;

— Для реализации программной системы выбран язык Python версии 3.9 или выше, так как он предоставляет широкий выбор библиотек для машинного обучения, в частности, библиотеку scikit-learn. Таким образом, его использование призвано облегчить программную реализацию алгоритмов машинного обучения и сосредоточиться непосредственно на эксперименте.

В дальнейших исследованиях планируется практическая реализация представленной системы, проведение эксперимента в соответствии с разработанным планом с последующим выявлением достоинств и недостатков различных подходов к оптимизации гиперпараметров классификаторов спама.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Detecting Spam Email With Machine Learning Optimized With Bio-Inspired Metaheuristic Algorithms / Gibson Simran, Issac Biju, Zhang Li, and Jacob Seibu Mary // IEEE Access. — 2020. — Vol. 8. — P. 187914–187932.

2. Т. Куликова Т. Щербакова Т. Сидорина. Спам и фишинг в 2020 году. — 2021. — URL: https://securelist.ru/spam-and-phishing-in-2020/100408/ (Дата обращения 12.05.2021).

3. A comprehensive study of spam detection in e-mails using bio-inspired optimization techniques / Batra Jai, Jain Rupali, Tikkiwal Vinay A., and Chakraborty Amrita // International Journal of Information Management Data Insights. — 2021. — Apr. — P. 100006.

4. Bhowmick Alexy, Hazarika Shyamanta M. E-Mail Spam Filtering: A Review of Techniques and Trends // Lecture Notes in Electrical Engineering. — Springer Singapore, 2017. — Đ□ĐºÑ□. — P. 583–590.

5. Garcia Flavio D., Hoepman Jaap-Henk, Nieuwenhuizen Jeroen. Spam Filter Analysis // Security and Protection in Information Processing Systems. — Springer US, 2004. — P. 395–410.

6. A Bayesian Approach to Filtering Junk E-Mail / Heckerman David, Horvitz Eric, Sahami Mehran, and Dumais Susan // AAAI Workshop on Learning for Text Categorization. — 1998. — July.

7. Multinomial Naive Bayes. — URL: https://scikit-learn.org/stable/modules/naive_bayes.html?highlight=mnb (Дата обращения 15.05.2021).

8. Воронцов К. Лекции По Методу Опорных Векторов. — 2007.

9. sklearn.linear_model.SGDClassifier. — URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html (Дата обращения 15.05.2021).

10. sklearn.svm.SVC. — URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html (Дата обращения 21.05.2021).

11. Yang Xin-She. Firefly Algorithms for Multimodal Optimization // Stochastic Algorithms: Foundations and Applications. — Springer Berlin Heidelberg, 2009. — P. 169–178.

12. Boyd Stephen, Vandenberghe Lieven. Convex Optimization. — Cambridge University Press, 2004. — Mar.

13. Metrics and scoring: quantifying the quality of predictions. — URL: https://scikit-learn.org/stable/modules/model_evaluation.html (Дата обращения 21.05.2021).

14. Перевод: Géron Aurélien. Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow. Концепции, инструменты и техники для создания интеллектуальных систем. — Вильямс, 2018. — Vol. 688.

15. Tuning the hyper-parameters of an estimator. — URL: https://scikit-learn.org/stable/modules/grid_search.html (Дата обращения 21.05.2021).

16. Cross-validation: evaluating estimator performance. — URL: https://www.fbi.gov/news/stories/forging-papers-to-sell-fake-art (Дата обращения 21.05.2021).

17. Aquila Optimizer: A novel meta-heuristic optimization algorithm / Abualigah Laith, Yousri Dalia, Elaziz Mohamed Abd, Ewees Ahmed A., Al-qaness Mo-

hammed A.A., and Gandomi Amir H. // Computers & Industrial Engineering. — 2021. — July. — Vol. 157. — P. 107250. — Access mode: https://doi.org/10.1016/j.cie.2021.107250.

18. Hunger games search: Visions, conception, implementation, deep analysis, perspectives, and towards performance shifts / Yang Yutao, Chen Huiling, Heidari Ali Asghar, and Gandomi Amir H // Expert Systems with Applications. — 2021. — Sep. — Vol. 177. — P. 114864. — Access mode: https://doi.org/10.1016/j.eswa.2021.114864.

19. Xue Jiankai, Shen Bo. A novel swarm intelligence optimization approach: sparrow search algorithm // Systems Science & Control Engineering. — 2020. — Jan. — Vol. 8, no. 1. — P. 22–34. — Access mode: https://doi.org/10.1080/21642583.2019.1708830.

20. Zhao Weiguo, Zhang Zhenxing, Wang Liying. Manta ray foraging optimization: An effective bio-inspired optimizer for engineering applications // Engineering Applications of Artificial Intelligence. — 2020. — Jan. — Vol. 87. — P. 103300. — Access mode: https://doi.org/10.1016/j.engappai.2019.103300.

21. Kaggle. — URL: https://aclweb.org/aclwiki/Spam_filtering_datasets (Дата обращения 07.12.2021).

22. Kaggle. — URL: https://spamassassin.apache.org/old/publiccorpus (Дата обращения 07.12.2021).

23. Enron Dataset. — URL: http://nlp.cs.aueb.gr/software_and_datasets/Enron-Spam/index.html (Дата обращения 07.12.2021).

24. Manning Christopher D., Raghavan Prabhakar, Schutze Hinrich. Scoring, term weighting, and the vector space model // Introduction to Information

Retrieval. — Cambridge University Press. — P. 100–123. — Access mode: https://doi.org/10.1017/cbo9780511809071.007.

25. sklearn.svm.SVC. — URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html (Дата обращения 07.12.2021).

26. Thieu Nguyen Van. A collection of the state-of-the-art MEta-heuristics ALgorithms in PYthon: Mealpy. — 2020. — Access mode: https://doi.org/10.5281/zenodo.3711948.

ПРИЛОЖЕНИЕ А

```
#!/usr/bin/python3

import re
import pandas as pd
import email
from bs4 import BeautifulSoup
from sklearn.datasets import load_files


def clear_string(txt):
    non_words_re = re.compile(r'[\W_]+')
    non_numbers_re = re.compile(r'\d+')
    multi_white_spaces = re.compile(r'\s+')

    s = txt.strip().lower()
    s = re.sub(non_words_re, ' ', s)
    s = re.sub(non_numbers_re, '', s)
    s = re.sub(multi_white_spaces, ' ', s)

    return s


def process_email(mail):
    link_re = re.compile(

r'(https|http)?:\/\/(\w|\.|\/|\?|\=|\&|\%)*\b', flags=re.MULTILINE)
```

```
        email_re = re.compile(
            r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b',
                flags=re.MULTILINE)

        b = email.message_from_string(mail)
        body = ""

        if b.is_multipart():
            for part in b.walk():
                ctype = part.get_content_type()
                cdispo = str(part.get('Content-Disposition'))

                if ctype == 'text/plain' and 'attachment' not in cdispo:
                    body = part.get_payload(decode=True)
                    break
        else:
            body = b.get_payload(decode=True)

        soup = BeautifulSoup(body, "html.parser")
        text = soup.get_text().lower()
        text = re.sub(link_re, '', text)
        text = re.sub(email_re, '', text)

        return clear_string(text)


def main():
```

```
emails = load_files(f'./ling_spam',
                    encoding='utf-8', decode_error='replace')
ling_spam = pd.DataFrame(
    {'message': emails.data, 'label': emails.target})

ling_spam['message'] = ling_spam['message'].apply(process_email)
ling_spam.to_csv(f'./ling_spam/messages.csv')

emails = load_files(f'./spam_assasin',
                    encoding='utf-8', decode_error='replace')
spam_assasin = pd.DataFrame(
    {'message': emails.data, 'label': emails.target})


spam_assasin['message'] = spam_assasin['message'].apply(process_email)
spam_assasin.to_csv(f'./spam_assasin/messages.csv')

enron = pd.DataFrame({'message': [], 'label': []})

for i in range(1, 7):
    emails = load_files(
f'./enron/enron{i}', encoding='utf-8', decode_error='replace')

df = pd.DataFrame({'message': emails.data, 'label': emails.target})
    enron = enron.append(df)

enron['message'] = enron['message'].apply( clear_string )
```

```
    enron.to_csv(f'./enron/messages.csv')


if __name__ == '__main__':
    main()
```

ПРИЛОЖЕНИЕ Б

```
#!/usr/bin/python3

import os.path
import csv
import numpy as np
import pandas as pd
import time

from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.metrics import accuracy_score, precision_score
from sklearn.metrics import recall_score, f1_score
from sklearn.linear_model import SGDClassifier
from sklearn.feature_extraction.text import TfidfVectorizer

from mealpy.swarm_based import AO, HGS, SSA, MRFO
from nltk.corpus import stopwords

# bio optimization algorithms
ALGS = ['AO', 'HGS', 'SSA', 'MRFO', 'RSCV']

# log fields
FIELDS = ['alg', 'data', 'test-size', 'time',
          'accuracy', 'precision', 'recall', 'f1-score']

# iterations number
```

```
ITERS = 100

# dataset name
DATASET = 'enron'

# size of test part of dataset
SIZE = 0.25


def resolve_clf(alg):
    if alg == 'AO':
        return AO.OriginalAO
    elif alg == 'HGS':
        return HGS.OriginalHGS
    elif alg == 'SSA':
        return SSA.BaseSSA
    elif alg == 'MRFO':
        return MRFO.BaseMRFO


def test_bio_alg(clf, obj_function):
    problem = {
        'obj_func': obj_function,
        'lb': [0.0001, 0.0001, 0.0001],
        'ub': [1000, 1000, 1000],
        'minmax': 'max',
        'verbose': True,
    }
```

```
    model = clf(problem, epoch=10, pop_size=50)
    model.solve()
    t = sum(model.history.list_epoch_time)

    return [model.g_best, t]


def main():
    for alg in ALGS:

        alpha = 0.0001
        epsilon = 0.0001
        tol = 0.0001

        with open(r'log_new.csv', 'a') as f:
            writer = csv.writer(f)

            if not os.path.isfile('log_new.csv') or \
                    os.path.getsize('log_new.csv') == 0:
                writer.writerow(FIELDS)

            # load dataset
            df = pd.read_csv(f'./input/{DATASET}/messages.csv')
            df = df.fillna('')

            X = np.array(df['message'])
            y = np.array(df['label'])
```

```
sum_acc = 0
sum_prec = 0
sum_recall = 0
sum_f1 = 0
sum_t = 0

for iter in range(0, ITERS):
    print(" Iter ", iter )
    t = 0
    y_pred = []

    # split dataset
    X_train, X_test, y_train, y_test = train_test_split (
        X, y, test_size =SIZE)

    # tokenization

cv = TfidfVectorizer (stop_words=stopwords.words('english'))
    X_train = cv. fit_transform (X_train)
    X_test = cv.transform(X_test)

    # random search
    if alg == 'RSCV':
        tuned_parameters = {

'epsilon': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],
```

```
'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],

'tol': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
                    }

                    t = time.time()
                    clf = SGDClassifier()
                    model = RandomizedSearchCV(clf, tuned_parameters)
                    model.fit(X_train, y_train)
                    y_pred = model.predict(X_test)
                    t = time.time() - t
                    print(model.best_params_)

                # bio optimization
                elif alg in ALGS:
                    def obj_function(solution):
                        alpha, epsilon, tol = solution
                        clf = SGDClassifier(
                            alpha=alpha, epsilon=epsilon, tol=tol)
                        clf.fit(X_train, y_train)
                        y_pred = clf.predict(X_test)

                        # print best params
                        # print(

#       f'Alpha={alpha} Epsilon={epsilon} tol={tol} \

#           Acc score: {accuracy_score(y_test, y_pred)}')
```

```
                return accuracy_score(y_test, y_pred)

            c = resolve_clf(alg)
            best_params_, t = test_bio_alg(c, obj_function)
            print(best_params_)
            alpha, epsilon, tol = best_params_[0]

            clf = SGDClassifier(
                alpha=alpha, epsilon=epsilon, tol=tol)
            clf.fit(X_train, y_train)
            y_pred = clf.predict(X_test)

            # print best params

# print(f'Best with: ', f'{best_params_}'.rjust(29, ' '))

        # default parameters
        else:
            clf = SGDClassifier()
            t = time.time()
            clf.fit(X_train, y_train)
            y_pred = clf.predict(X_test)
            t = time.time() - t

        acc = accuracy_score(y_test, y_pred)
        prec = precision_score(y_test, y_pred)
        recall = recall_score(y_test, y_pred)
```

```
                f1 = f1_score(y_test, y_pred)

                sum_acc += acc
                sum_prec += prec
                sum_recall += recall
                sum_f1 += f1
                sum_t += t

            # calc average metrics
            avg_acc = sum_acc / ITERS
            avg_prec = sum_prec / ITERS
            avg_recall = sum_recall / ITERS
            avg_f1 = sum_f1 / ITERS
            avg_t = sum_t / ITERS

            # write result row
            writer.writerow([alg, DATASET, SIZE, avg_t, avg_acc,
                             avg_prec, avg_recall, avg_f1])

        f.close()


if __name__ == '__main__':
    main()
```