ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ КИБЕРНЕТИЧЕСКИХ СИСТЕМ

Кафедра
«Криптология и кибербезопасность»

ОТЧЕТ
О НАУЧНО-ИССЛЕДОВАТЕЛЬКОЙ РАБОТЕ

«Применение современных природных алгоритмов в задаче фильтрации англоязычного спама на основе метода опорных векторов»

Исполнитель:

студент гр.

подпись, дата

Научный руководитель:

подпись, дата

Зам. зав. каф. № 42: Когос К.Г.

подпись, дата

Москва — 2022

РЕФЕРАТ

Пояснительная записка содержит 68 страниц, 11 рисунков, 2 таблицы, 2 приложения и 27 источников.

Ключевые слова: СПАМ, МАШИННОЕ ОБУЧЕНИЕ, МЕТОД ОПОРНЫХ ВЕКТОРОВ, ОПТИМИЗАЦИЯ, ПРИРОДНЫЙ АЛГОРИТМ, ГИПЕРПАРАМЕТР, ACCURACY, PRECISION, RECALL, F1-SCORE.

Целью работы является выявление наиболее эффективного нового природного алгоритма для классификации англоязычного спама на основе метода опорных векторов.

Объект исследования — классификаторы спама на основе машинного обучения.

Предмет исследования — оптимизация гиперпараметров алгоритмов машинного обучения.

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

Результатом данной работы является сравнительная оценка эффективности современных природных алгоритмов при решении задачи распознавания англоязычного спама.

Качественный подбор параметров алгоритма машинного обучения является актуальной задачей, поскольку они во многом определяют точность работы получившегося спам-классификатора.

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)
SI	—	роевой интеллект, коллективное поведение децентрализованных, самоорганизованных систем, естественных или искусственных (Swarm Intelligence)
AO	—	Оптимизатор орла (Aquila optimizer)
HGS	—	Поиск голодных игр (Hunger Games Search)
SSA	—	Алгоритм поиска воробьев (Sparrow Search Algorithm)
MRFO	—	Алгоритм оптимизации кормодобывания скатов манта (Manta Ray Foraging Optimization)
SVM	—	Метод опорных векторов (Support vector machine)
SGD	—	Стохастический градиентный спуск (Stochastic gradient descent)

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 2021 год, 45,56% электронных писем было спамом, причем большая его часть (24,77%), как и в прошлом году, исходила 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 Костенко В. А., 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, обработка 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 существующих подходов к оптимизации алгоритмов машинного обучения, в том числе природных алгоритмов.

В четвертом разделе рассмотрены метрики, используемые для 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.

В девятом разделе приведены результаты экспериментальной оценки эффективности выбранных природных алгоритмов, сделаны соответствующие выводы.

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 за законных отправителей. Хотя этот прием кажется очень эффективным, уклониться от него тоже очень просто. Для повторной попытки спамеры могут использовать зомби.

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. Хеши ранее идентифицированных спам-сообщений хранятся в базе данных на уровне MTA. Все входящие сообщения электронной почты проверяются по этим хешам. 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 хешей.

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:

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

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 Метод опорных векторов (SVM)

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

Гиперплоскость — пространство размерностью на единицу меньше размерности исходного пространства. Основная цель метода 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].

Метод 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 применяется метод опорных векторов, представляющий 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 (алгоритм 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 машинного обучения 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 метод 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 Алгоритмы оптимизации

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 Классификация алгоритмов 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, учиты-

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

3.3 Оптимизация гиперпараметров в машинном обучении

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, метод 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.

3.4 Природные алгоритмы оптимизации

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. При оптимизации гиперпараметров возможно использование более сложных алгоритмов, чем поиск по сетке или случайный поиск. В частности, природных алгоритмов.

Классифицировать природные алгоритмы можно по разным признакам, однако наиболее часто этим признаком служит биологический 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). Про-

цессы 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 Оценка производительности моделей машинного обучения

Для оценки производительности некоторой обученной модели используются различные метрики.

4.1 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 в Таблице 1.

Таблица 1 – Матрица ошибок

—	HAM	SPAM
HAM	TN	FP
SPAM	FN	TP

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.

4.2 Accuracy

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

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

4.3 Recall

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

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

4.4 Precision

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

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

4.5 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 $F1$:

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

4.6 ROC-кривая

ROC-кривая (receiver operating characteristic curve) — также известна как кривая ошибок — график, показывающий производительность классификационной модели при всех пороговых значениях классификации. Часто используется при бинарной классификации.

Кривая ошибок отображает два параметра:

— TPR — Частота истинно-положительных срабатываний (Чувствительность);

— FPR — Частота ложно-положительных срабатываний (Ошибка первого рода).

TPR — то же, что $Recall$(11). Величина $1 - FPR$ называется специфичность модели.

FPR определяется формулой:

$$Precision = \frac{FP}{(FP + TN)} \tag{14}$$

Площадь под кривой ошибок (AUC ROC — area under curve ROC) — пло-

щадь двумерной области под кривой ошибок. Обеспечивает совокупную меру эффективности по всем возможным пороговым значениям классификации.

Чем выше показатель, тем качественнее классификатор. Значение колеблется от 0 до 1: модель, предсказания которой на 100% неверны, имеет $AUC = 0.0$, модель, предсказания которой на 100% верны, имеет $AUC = 1.0$.

Преимущества использования AUC:

— Является масштабно-инвариантным. Измеряет, насколько хорошо ранжируются прогнозы, а не их абсолютные значения.

— Является классификационно-пороговым инвариантом. Измеряет качество предсказаний модели независимо от того, какой порог классификации выбран.

4.7 Перекрестная проверка

Перекрестная проверка применяется для получения более надежных оценок.

В базовом подходе, называемом $k - foldCV$ [14], обучающая выборка разбивается на k меньших наборов.

Для каждого из k наборов выполняется следующее:

— Модель обучается с использованием $k - 1$ частей набора в качестве обучающих данных;

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

Метрика, которую возвращает k-кратная перекрестная проверка, представляет собой среднее значение из полученных значений этой метрики для всех k наборов. Этот подход может быть дорогостоящим в вычислительном отношении, но дает более точную оценку модели.

Вариацией данного метода является стратифицированная перекрестная проверка [15], возвращающая наборы данных с сохранением процентного соотношения образцов для каждого класса.

5 Подготовка к проведению эксперимента

Для подготовки к программной реализации системы оценки производительности алгоритмов оптимизации, необходимо спланировать эксперимент, построить алгоритм проведения эксперимента, выбрать инструменты для его 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 — оценки 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:

— Выбрать наборы данных для обучения;

— Определить набор алгоритмов для подбора параметров модели;

— Предварительно обработать тренировочные данные;

— Реализовать алгоритм глобальной оптимизации функции на основе того или иного природного алгоритма;

— Выбрать оценочную функцию, подлежащую оптимизации природным алгоритмом. Эта функция должна вычислять оценку модели МО для каждой частицы с новыми параметрами. Такой функцией может служить как одна из стандартных метрик (раздел 4), так и нестандартная весовая функция;

— Получить набор параметров путем оптимизации оценочной функции;

— Обучить модель с использованием полученных параметров;

— Оценить производительность полученной модели;

— Повторить эксперимент, получив те же параметры с использованием других подходов к оптимизации, описанных в 3.3;

— Повторить эксперимент для других выбранных способов подбора параметров и других наборов данных;

— Применить полученные модели к тестовым наборам данных, также провести оценку;

— 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;

— Объяснить полученные результаты эксперимента, сделать соответствующие выводы.

5.1.1 Перекрестная проверка

В данном эксперименте планируется применять перекрестную проверку как при поиске подходящих параметров, так и для оценки получившихся классификаторов.

5.1.2 Подбор параметров

Для настройки параметров с помощью природных алгоритмов, необходимо задать число частиц и число популяций, а также метрику для максимизации (например, $Accuracy$). Затем — обучить модели с перекрестной проверкой на различных наборах параметров, предоставляемых оптимизационным алгоритмом. Продолжать процесс до тех пор, пока не будет достигнуто максимальное число популяций.

5.2 Архитектура системы для проведения эксперимента

На Рисунке 1 представлена блок-схема описанной системы.

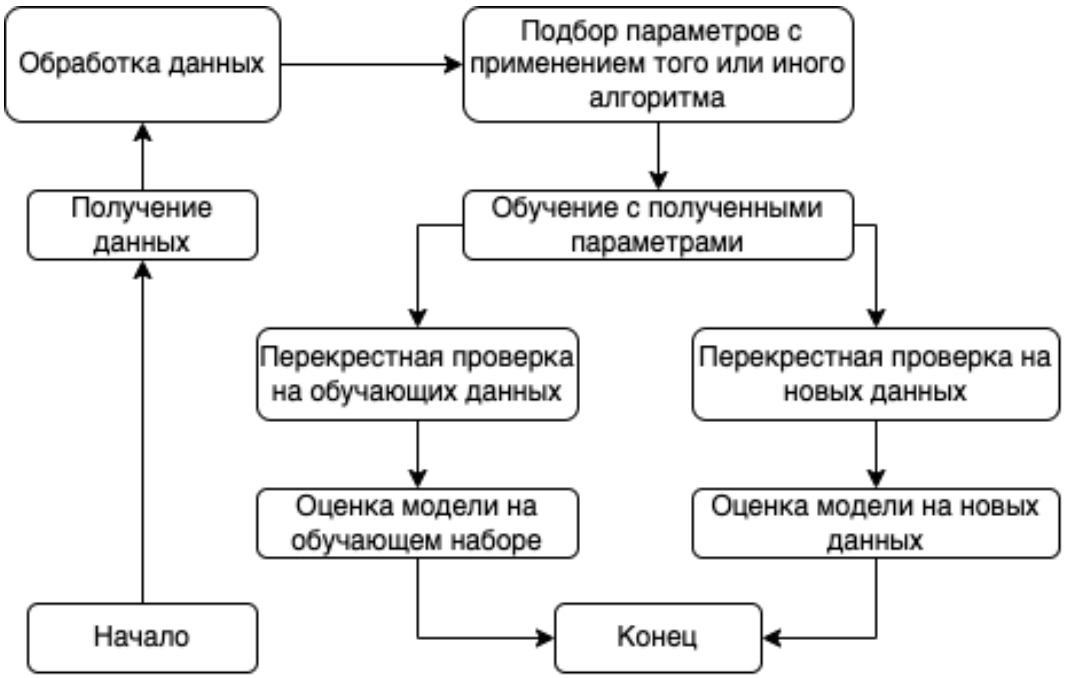

Рисунок 1 – Блок-схема системы для проведения эксперимента

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 [16] 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 сильную мотивацию и причины поведения всех живых существ. Кроме того, почти все животные используют вычислительно-логические 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 [17].

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 [18].

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 [19].

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). Для реализации программной системы выбран язык Python версии 3.9.7, так как данный язык программирования 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.

Основные модули, использованные в работе:

— 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.

Были использованы следующие наборы данных:

— Набор данных Ling-Spam (2893 писем) [20];

— Набор данных Spam-Assasin (9352 писем) [21];

— Набор данных Enron (33715 писем) [22].

Соотношение спама и обычных писем для набора данных Ling-Spam представлено на 2, для Spam-assasin — на 3, для Enron — на 4.

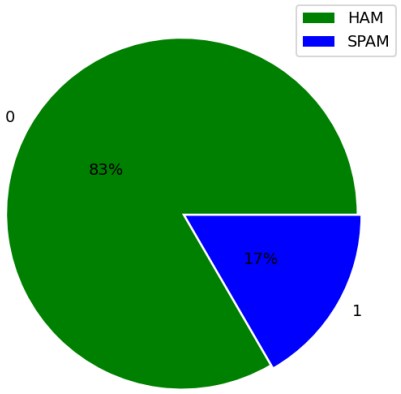

Рисунок 2 – Соотношение спама и обычных писем для набора данных Ling-Spam

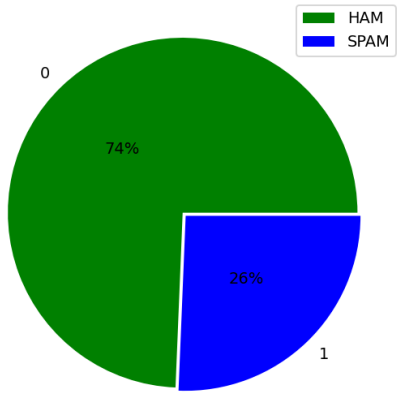

Рисунок 3 – Соотношение спама и обычных писем для набора данных Spam-Assasin

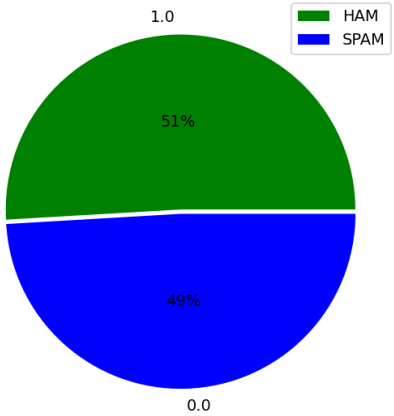

Рисунок 4 – Соотношение спама и обычных писем для набора данных Enron

7.3 Предварительная обработка данных

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.

7.3.1 Приведение к общему виду

С помощью программы, приведенной в Приложении А, 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).

7.3.2 Символьная обработка

Символьная обработка данных включена в программный код, приведенный в Приложении А.

Необходимо очистить строки от бесполезных для распознавания спама 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.

7.3.3 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 матрица подсчета была преобразована 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 данных. Tf означает частоту слова, а tf-idf — это частота слова, умноженная на величину, обратную частоте, с которой в наборе данных встречаются документы, его содержащие [23].

Формула, лежащая в основе статистического показателя TF-IDF:

$$tf - idf(w, d, D) = tf(w, d) * idf(w, D) \tag{15}$$

где:

— d — данный документ из нашего набора данных;

— D — набор документов;

— w — заданное слово в документе.

Частота слова среди всех слов документа:

$$tf(w,d) = log(1 + f(w,d)) \tag{16}$$

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

Обратная частота слова среди всех документов:

$$idf(w,D) = log(\frac{N}{f(w,D)}) \tag{17}$$

где N — количество всех документов в наборе.

8 Программная реализация эксперимента

Основные этапы работы программы:

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

— Перед обучением или применением всех классификаторов осуществляется преобразование текста в числовую матрицу слов с использованием TfidfVectorizer[24] и набора стоп-слов от модуля nltk;

— Получение параметров для каждого классификатора методом случайного поиска с перекрестной проверкой и с помощью био-алгоритмов с перекрестной проверкой (из 3 обучающих наборов и 6 методов оптимизации получено $3 \times 6 = 18$ классификаторов); Природные алгоритмы применяются с использованием модуля mealpy [25];

— Оценка точности классификаторов с помощью процедуры перекрестной проверки;

— Оценка точности и площади под ROC-кривой классификаторов на двух не использовавшихся при обучении наборах данных;

— Построение графиков ROC-кривых для всех комбинаций обучающего и тестового набора (всего 6 графиков).

8.1 Настройка параметров модели

Параметры настройки модели имеют большое влияние на обнаружение спам-писем и скорость обучения. Для алгоритма SGD подбирались следующие параметры:

— 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 3.3);

— Природных алгоритмов (раздел 6).

Границы для всех трех параметров были установлены 10^{-3} до 10^{3} с шагом в одну степень. Была применена стратифицированная перекрестная проверка с числом разбиений, равным 10.

Недостатки использования случайного поиска и поиска по сетке очевидны — и тот, и другой 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 быстрее, чем поиск по сетке.

8.2 Применение природных алгоритмов оптимизации

Для поиска внутри области, а не по точкам, можно использовать природные алгоритмы.

В контексте данной работы задача оптимизации — максимизировать метрическую функцию (раздел 4). В качестве такой функции была использована метрика $Accuracy$.

Алгоритмы оптимизации были запущены с использованием стратифицированной перекрестной проверки с разбиением на 10 частей. Число частиц было принято равным 10, число популяций — равным 40. Границы поиска для каждого параметра — от 10^{-3} до 10^{3}.

9 Результаты

Классификаторы, обученные на трех различных наборах данных с наборами подобранных разными алгоритмами параметрами, а также с параметрами по умолчанию, были оценены величиной Accuracy с применением перекрестной проверки.

Полученные оценки изображены на диаграмме — Рисунок 5. Полученные в результате выполнения программы измерения *Accuracy* приведены в Таблице 2.

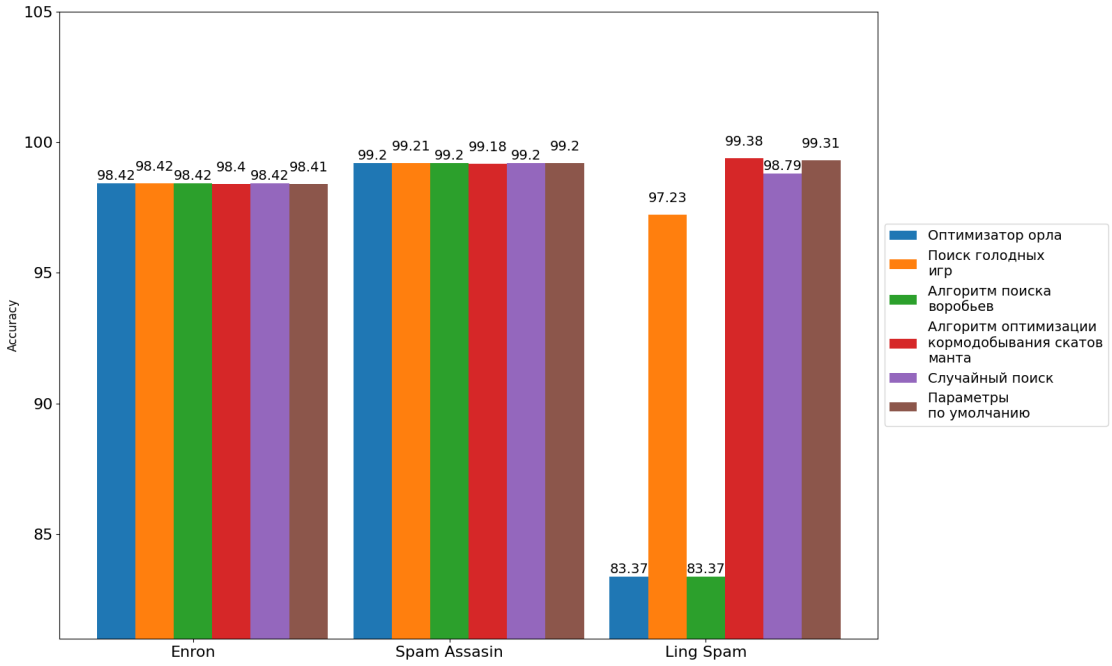

Рисунок 5 – Accuracy для возможных сочетаний наборов данных и алгоритмов оптимизации

Затем обученные на одних данных классификаторы были применены к данным, не применявшихся при обучении. Также была измерена Accuracy. Результаты измерений занесены в Таблицу 2

Также по результатам применения полученных классификаторов к новым данным построены кривые ошибок, с указанием площади под ней. На Рисунке 6 изображена кривая ошибок для классификатора, обученного на наборе Enron и примененного к набору Ling Spam, на Рисунке 7 — для обученного на Enron

Таблица 2 – Результаты эксперимента — метрика Accuracy

Обучающие данные	Данные	MRFO	HGS	AO	SSA	RSCV	DEFAULT
LS	SA	90.217	89.480	75.287	75.287	88.082	88.711
	EN	70.675	76.482	50.927	50.915	76.183	70.343
SA	LS	94.988	95.092	95.022	95.299	94.469	94.469
	EN	72.852	74.901	74.901	73.086	74.587	73.768
EN	LS	90.840	91.116	89.803	89.803	89.803	81.645
	SA	87.606	86.696	86.457	86.457	86.457	77.356

и примененного к Spam Assasin, и так далее.

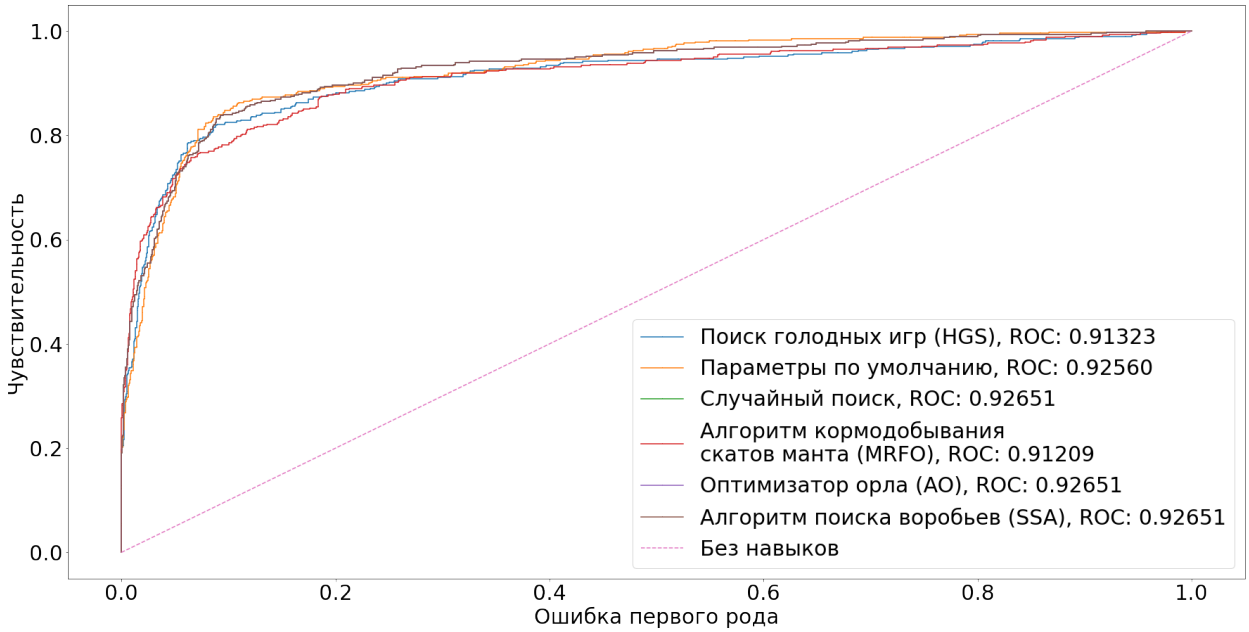

Рисунок 6 – Кривые ошибок для классификаторов, обученных на Enron, оцененных — на Ling Spam

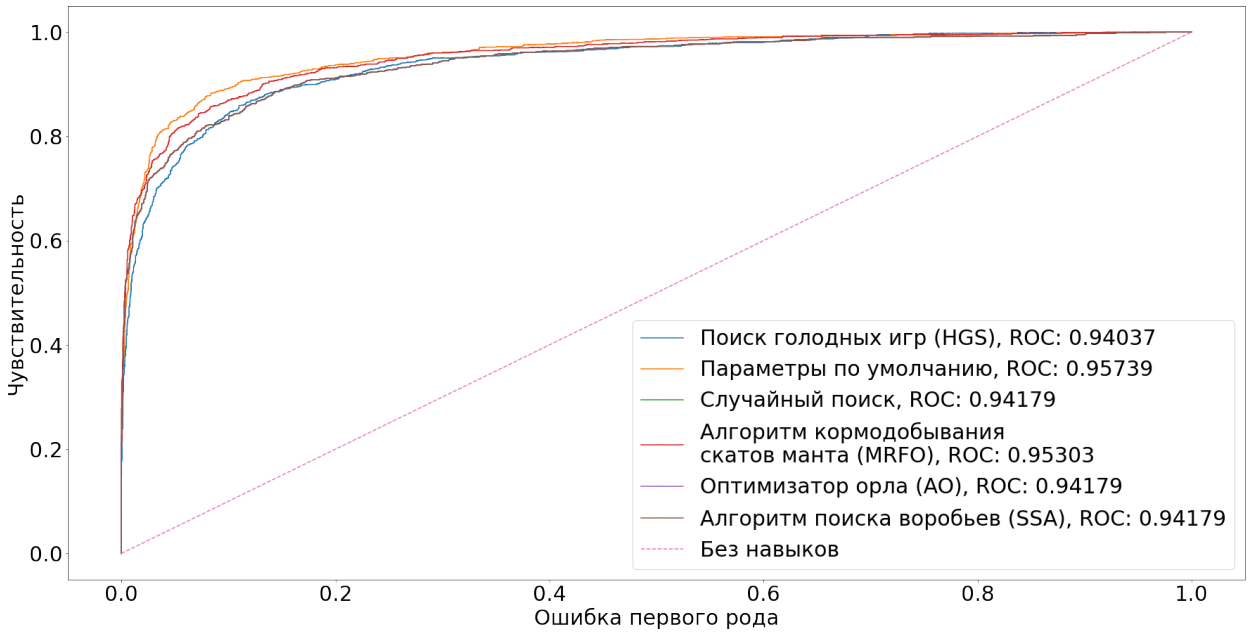

Рисунок 7 – Кривые ошибок для классификаторов, обученных на Enron, оцененных — на Spam Assasin

На следующих рисунках — Рисунке 8, Рисунке 9, Рисунке 10 и Рисунке 11 — изображены кривые, соответствующие наборам данных, указанным в названиях рисунков.

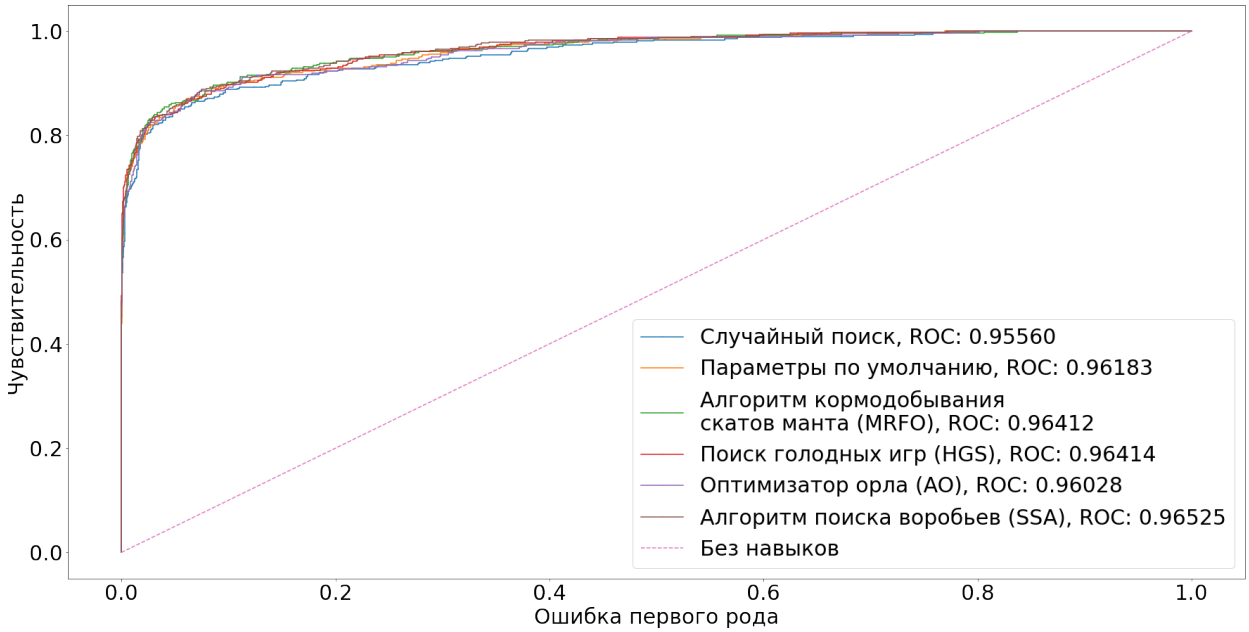

Рисунок 8 – Кривые ошибок для классификаторов, обученных на Spam Assasin, оцененных — на Ling Spam

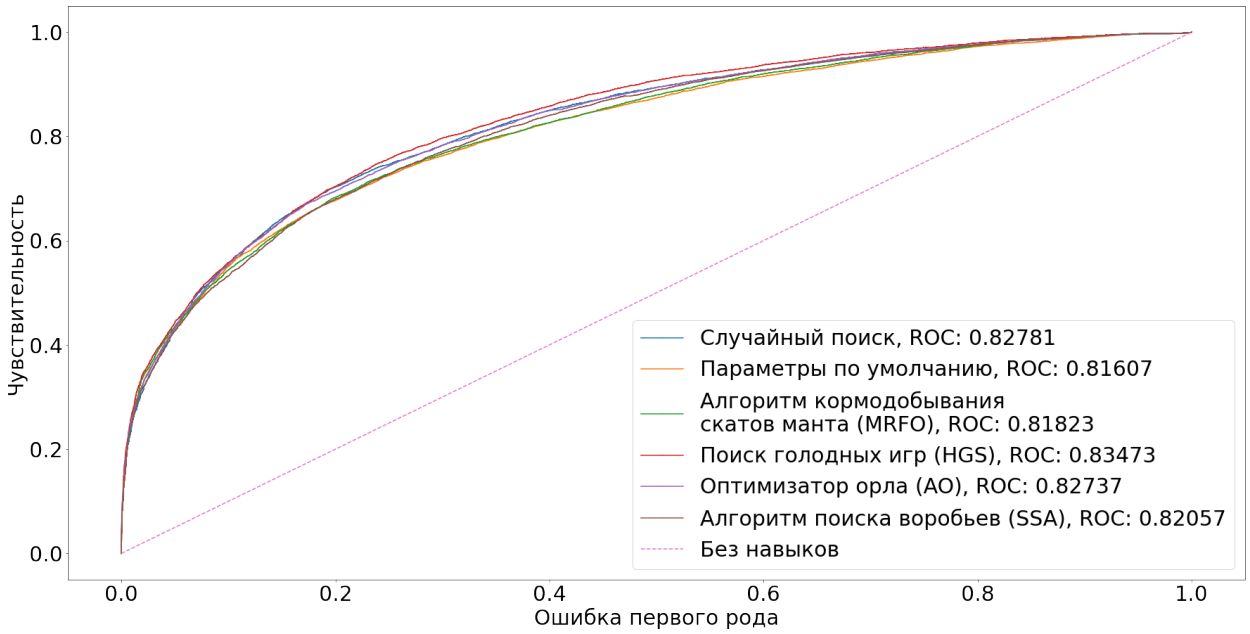

Рисунок 9 – Кривые ошибок для классификаторов, обученных на Spam Assasin, оцененных — на Enron

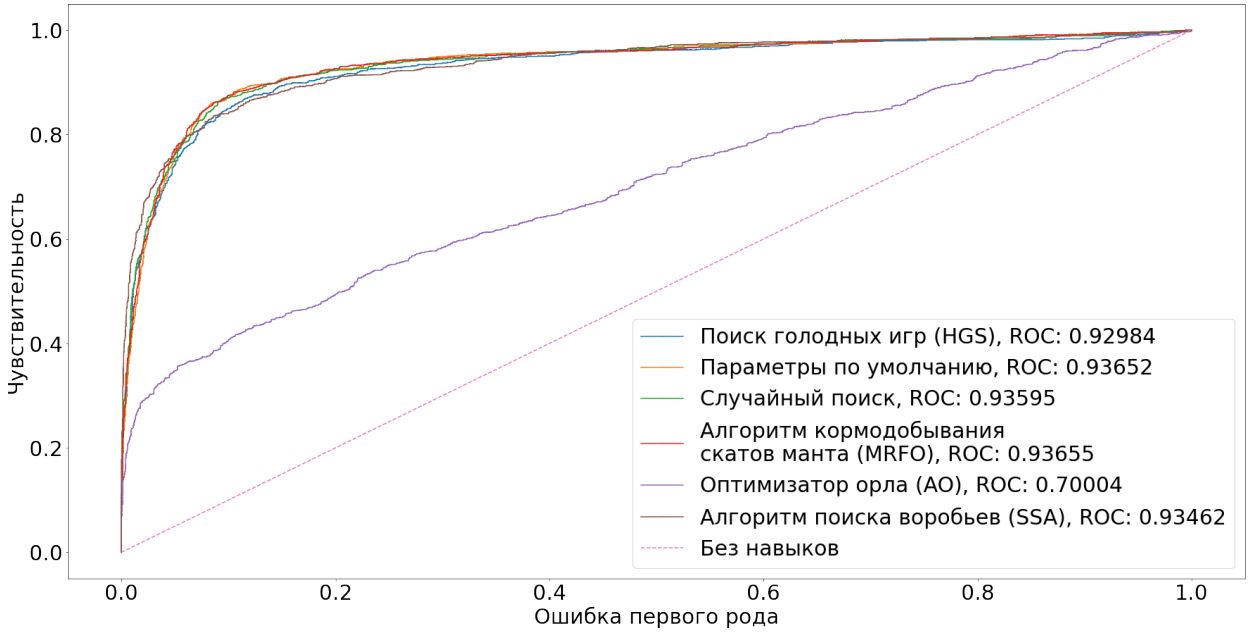

Рисунок 10 – Кривые ошибок для классификаторов, обученных на Ling Spam, оцененных — на Spam Assasin

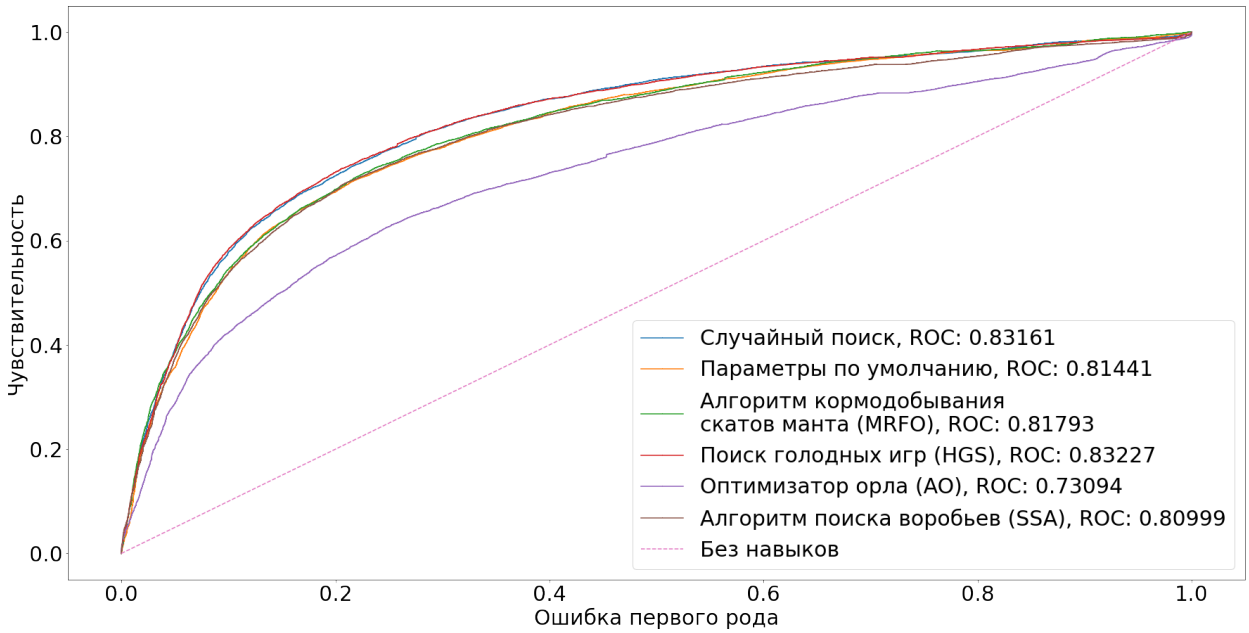

Рисунок 11 – Кривые ошибок для классификаторов, обученных на Ling Spam, оцененных — на Enron

Анализируя результаты, можно сделать следующие выводы:

— Алгоритм MRFO по метрике $Accuracy$ на новых данных превзошел параметры по умолчанию в 5 из 6 случаев, случайный поиск — в 4 из 6, при чем в 2 из этих 4 случаев стал лучшим по сравнению с другими природными алгоритмами. Однако по площади под кривой — только в 2 из 6 случаев алгоритм превзошел случайный поиск и параметры по умолчанию. Лучшим стал 1 раз из этих 2;

— Алгоритм HGS по метрике $Accuracy$ на новых данных превзошел как параметры по умолчанию, так и случайный поиск во всех 6 случаях, являлся лучшим в 3 из 6 случаях. По площади под кривой — трижды превзошел случайный поиск и параметры по умолчанию, дважды дал лучшие показатели;

— Алгоритмы SSA и AO, хоть и показали на новых данных в большинстве случаев более высокую $Accuracy$ по сравнению со случайным поиском и параметрами по умолчанию, по показателю площади под кривой ошибок ROC оказались значительно хуже других алгоритмов;

— Влияние настройки параметров на эффективность алгоритмов более заметно на меньших объемах данных. По результатам в Таблице 2 видно,

что при использовании для обучения большого набора данных Enron, алгоритмы оптимизации в большинстве случаев дают такое же улучшение, как и случайный поиск;

— ”Поиск голодных игр” (HGS) позволил улучшить производительность по обеим метрикам чаще других выбранных алгоритмов.

ЗАКЛЮЧЕНИЕ

Фильтрация спама была рассмотрена в контексте машинного обучения как задача классификации. Описаны некоторые алгоритмы машинного обучения, используемые для решения данной задачи, а именно: наивный байесовский и мультиномиальный байесовский классификаторы, метод опорных векторов и метод опорных векторов со стохастическим градиентным спуском.

Приведены метрики, используемые для оценки производительности моделей машинного обучения: Accuracy, Recall, Precision, F1-мера, кривая ошибок и площадь под кривой ошибок.

В работе перечислены основные подходы к решению задачи оптимизации параметров обучающих алгоритмов, такие как поиск по сетке, случайный поиск, градиентный спуск, байесовская оптимизации и эвристический подход, к которому относится применение природных алгоритмов. В рамках данной задачи спланирован и проведен эксперимент по сравнению эффективности нескольких природных алгоритмов: оптимизатор орла (AO), поиск голодных игр (HGS), алгоритм поиска воробьев (SSA) и алгоритм оптимизации кормодобывания скатов манта (MRFO).

В ходе работы получены следующие результаты:

— Спроектирован программный комплекс для проведения эксперимента по сравнению эффективности выбранных природных алгоритмов в задаче оптимизации классификатора спама на основе метода опорных векторов;

— Выбраны метрики для оценки эффективности природных алгоритмов в вышеуказанной задаче — $Accuracy$ и площадь под кривой ошибок;

— Создан программный комплекс для проведения эксперимента;

— Проведен эксперимент, на основе выбранных метрик сделаны выводы об эффективности выбранных алгоритмов:

а Влияние настройки параметров на эффективность алгоритмов более заметно на меньших объемах данных. При использовании для обучения боль-

шого набора данных, алгоритмы оптимизации в большинстве случаев дают такое же улучшение, как и случайный поиск;

б Алгоритмы SSA и AO, хоть и показали на новых данных в большинстве случаев более высокую $Accuracy$ по сравнению со случайным поиском и параметрами по умолчанию, по показателю площади под кривой ошибок ROC оказались значительно хуже других алгоритмов;

в ”Поиск голодных игр” (HGS) позволил улучшить производительность по обеим метрикам в большем числе случаев, чем у других выбранных алгоритмов;

г ”Алгоритм кормодобывания скатов” (MRFO) также показал хорошие результаты по обеим метрикам, но несколько хуже, чем HGS.

Таким образом, использование алгоритма ”поиск голодных игр” или алгоритма кормодобывания скатов способно повысить точность определения нежелательных рассылок при автоматической фильтрации почтового трафика на величину от 0,1% до 2% по сравнению со случайным поиском и до 9% по сравнению с параметрами по умолчанию.

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. Спам и фишинг в 2021 году [Электронный ресурс] // Securelist.ru: Спам и фишинг в 2021 году | Securelist. URL: https://securelist.ru/spam-and-phishing-in-2021/104407/ (дата обращения 17.04.2022).

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. — 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 [Электронный ресурс] // Scikit-learn.org: machine learning in Python — scikit-learn 1.0.1 documentation. 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 [Электронный ресурс] // Scikit-learn.org: machine learning in Python — scikit-learn 1.0.1 documentation. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html (дата обращения 15.05.2021).

10. Sklearn.svm.SVC [Электронный ресурс] // Scikit-learn.org: machine learning in Python — scikit-learn 1.0.1 documentation. 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 [Электронный ресурс] // Scikit-learn.org: machine learning in Python — scikit-learn 1.0.1 documentation. URL: https://scikit-learn.org/stable/modules/model_evaluation.html (дата обращения 21.05.2021).

14. Cross Validation [Электронный ресурс] // Scikit-learn.org: machine learning in Python — scikit-learn 1.0.1 documentation. URL: https://scikit-learn.org/stable/modules/cross_validation.html (дата обращения 02.04.2022).

15. Stratified KFold Validation [Электронный ресурс] // Scikit-learn.org: machine learning in Python — scikit-learn 1.0.1 documentation. URL: https://scikit-learn.org/stable/modules/generated/sklearn.

model_selection.StratifiedKFold.html (дата обращения 02.04.2022).

16. Aquila Optimizer: A novel meta-heuristic optimization algorithm / Abualigah Laith, Yousri Dalia, Elaziz Mohamed Abd, Ewees Ahmed A., Al-qaness Mohammed A.A., and Gandomi Amir H. // Computers & Industrial Engineering. — 2021. — July. — Vol. 157. — P. 107250.

17. 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.

18. 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.

19. 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.

20. Spam filtering datasets [Электронный ресурс] // Aclweb.org: Spam: filtering datasets - ACL Wiki. URL: https://aclweb.org/aclwiki/Spam_filtering_datasets (дата обращения 07.12.2021).

21. Index of /old/publiccorpus [Электронный ресурс] // Spamassassin.apache.org: Index of /old/publiccorpus. URL: https://spamassassin.apache.org/old/publiccorpus (дата обращения 07.12.2021).

22. The Enron-Spam datasets [Электронный ресурс] // Nlp.cs.aueb.gr: The Enron-Spam datasets. URL: http://nlp.cs.aueb.gr/software_and_datasets/Enron-Spam/index.html (дата обращения 07.12.2021).

23. 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.

24. Sklearn.svm.SVC [Электронный ресурс] // Scikit-learn.org: machine learning in Python — scikit-learn 1.0.1 documentation. URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html (дата обращения 07.12.2021).

25. 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.

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

27. Tuning the hyper-parameters of an estimator [Электронный ресурс] // Scikit-learn.org: machine learning in Python — scikit-learn 1.0.1 documentation. URL: https://scikit-learn.org/stable/modules/grid_search.html (дата обращения 21.05.2021).

ПРИЛОЖЕНИЕ А

Код программы для предварительной обработки данных

```
#!/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():
```

```
unbalanced_datasets = ['ling_spam', 'spam_assasin']

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

    shuffled_df = ling_spam.sample(frac=1, random_state=4)
    fraud_df = shuffled_df.loc[shuffled_df['label'] == 1]

non_fraud_df = shuffled_df.loc[shuffled_df['label'] == 0].sample(
        n=len(fraud_df), random_state=42)

    pd.concat([fraud_df, non_fraud_df]).sample(

frac=1, random_state=4).to_csv(f'./{dataset}/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

Код программы для проведения эксперимента

```
#!/usr/bin/python3

from sklearn.model_selection import StratifiedKFold
from mealpy.swarm_based import AO, HGS, SSA, MRFO
from matplotlib import pyplot
from nltk.corpus import stopwords
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.metrics import accuracy_score, f1_score
from sklearn.model_selection import cross_val_score,
    StratifiedKFold, RandomizedSearchCV
import numpy as np
import pandas as pd
from numpy import mean

DEFAULT_PARAMS = [0.0001, 0.1, 1e-3]

EN = pd.read_csv(f'./input/enron/messages.csv').fillna('')
X_EN = np.array(EN['message'])
y_EN = np.array(EN['label'])

LS = pd.read_csv(f'./input/ling_spam/messages.csv').fillna('')
X_LS = np.array(LS['message'])
```

```
y_LS = np.array(LS['label'])

SA = pd.read_csv(f'./input/spam_assasin/messages.csv').fillna('')
X_SA = np.array(SA['message'])
y_SA = np.array(SA['label'])


def resolve_dataset(name):
    if (name == 'EN'):
        return [X_EN.copy(), y_EN.copy()]
    elif (name == 'LS'):
        return [X_LS.copy(), y_LS.copy()]
    elif (name == 'SA'):
        return [X_SA.copy(), y_SA.copy()]
    else:
        return


def resolve_alg(alg):
    if alg == 'AO':
        return AO.OriginalAO
    elif alg == 'HGS':
        return HGS.OriginalHGS
    elif alg == 'SSA':
        return SSA.OriginalSSA
    elif alg == 'MRFO':
        return MRFO.BaseMRFO
```

```
def alg_to_label(alg):
    if alg == 'AO':
        return Оптимизатор' орла (AO)'
    elif alg == 'HGS':
        return Поиск' голодных игр (HGS)'
    elif alg == 'SSA':
        return Алгоритм' поиска воробьев (SSA)'
    elif alg == 'MRFO':
        return Алгоритм' кормодобывания \скатовn манта (MRFO)'
    elif alg == 'RSCV':
        return Случайный' поиск'
    elif alg == 'DEFAULT':
        return Параметры' по умолчанию'


def get_best(alg, X, y):
    if (alg == 'RSCV'):
        distributions = {
            'clf__epsilon': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],
            'clf__alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],
            'clf__tol': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],
        }
        skf = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
        clf = Pipeline([
            ('tfidf_vectorizer', TfidfVectorizer(
                stop_words=stopwords.words('english'))),
            ('clf', SGDClassifier(random_state=0, class_weight='balanced',
```

```
                n_jobs=-1))])

        clf_random = RandomizedSearchCV(

clf, distributions, scoring='accuracy', cv=skf, random_state=0)
        clf_random.fit(X, y)
        best = clf_random.best_params_


return [best['clf__alpha'], best['clf__epsilon'], best['clf__tol']]

 alg = resolve_alg(alg)
 cv = TfidfVectorizer(stop_words=stopwords.words('english'))
 skf = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)

 alpha, epsilon, tol = [], [], []

 for train_index, test_index in skf.split(X, y):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        X_train = cv.fit_transform(X_train)
        X_test = cv.transform(X_test)

        def obj_function(solution):
            alpha, epsilon, tol = solution
            clf = SGDClassifier(random_state=0, class_weight='balanced',
                alpha=alpha, epsilon=epsilon, tol=tol, n_jobs=-1)
```

```
            clf . fit (X_train , y_train )
            y_pred = clf . predict (X_test)
            return accuracy_score( y_test , y_pred)

        problem = {
            ' fit_func ': obj_function ,
            'lb ': [0.0001, 0.0001, 0.0001],
            'ub': [1000, 1000, 1000],
            'minmax': 'max',
            'verbose ': True,
        }

        model = alg(problem, epoch=10, pop_size=40)
        model.solve()
        a, e, t = model.g_best[0]
        alpha.append(a)
        epsilon .append(e)
        tol .append(t)

    return [mean(alpha), mean(epsilon), mean(tol)]


def test ( clfs , train , test ):
    print (f' Test models trained with {train} on {test}')

    for alg in clfs :
        X, y = resolve_dataset ( test )
        clf = clfs [alg]
```

```
        y_score = clf.decision_function(X)
        y_pred = clf.predict(X)

        print(alg)
        print('Accuracy %.5f: ' % accuracy_score(y, y_pred))
        print('ROC: %.5f' % roc_auc_score(y, y_score))
        print('F1: %.5f\n' % f1_score(y, y_pred))

        y_fpr, y_tpr, _ = roc_curve(y, y_score)
        pyplot.plot(y_fpr, y_tpr, marker=',', label=alg_to_label(
            alg) + ', ROC: %.5f' % roc_auc_score(y, y_score))

    ns_probs = [0 for _ in range(len(y))]
    ns_fpr, ns_tpr, _ = roc_curve(y, ns_probs)
    pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label Без=' навыков')
    pyplot.xlabel Ошибка(' первого рода')
    pyplot.ylabel Чувствительность ('')
    pyplot.legend()
    pyplot.show()


pyplot.rcParams['figure.figsize'] = [30, 15]
pyplot.rcParams.update({'font.size': 30})


best_LS_RSCV = get_best("RSCV", X_LS, y_LS)
best_LS_MRFO = get_best("MRFO", X_LS, y_LS)
```

```
best_LS_HGS = get_best(”HGS”, X_LS, y_LS)
best_LS_AO = get_best(”AO”, X_LS, y_LS)
best_LS_SSA = get_best(”SSA”, X_LS, y_LS)

best_SA_RSCV = get_best(”RSCV”, X_SA, y_SA)
best_SA_MRFO = get_best(”MRFO”, X_SA, y_SA)
best_SA_HGS = get_best(”HGS”, X_SA, y_SA)
best_SA_AO = get_best(”AO”, X_SA, y_SA)
best_SA_SSA = get_best(”SSA”, X_SA, y_SA)

best_EN_RSCV = get_best(”RSCV”, X_EN, y_EN)
best_EN_MRFO = get_best(”MRFO”, X_EN, y_EN)
best_EN_HGS = get_best(”HGS”, X_EN, y_EN)
best_EN_AO = get_best(”AO”, X_EN, y_EN)
best_EN_SSA = get_best(”SSA”, X_EN, y_EN)



LS_EN = {1.: 10,  0.:  1}
LS_SA = {1.:  1,  0.:  1.5}

SA_EN = {1.: 10,  0.:  1}
SA_LS = {1.:  1,  0.:  8}

EN_LS = {1.:  1,  0.:  100}
EN_SA = {1.:  1,  0.:  100}



def  create_clf (params,  class_weight=None):
```

```
    alpha, epsilon, tol = params
    return Pipeline([
        ('tfidf_vectorizer', TfidfVectorizer(
            stop_words=stopwords.words('english'))),
        ('clf', SGDClassifier(random_state=0,
            alpha=alpha, epsilon=epsilon, tol=tol, class_weight=class_weight,
            n_jobs=-1))
    ])


X, y = X_SA, y_SA

SA_clfs = {
    "RSCV": create_clf(best_SA_RSCV),
    "DEFAULT": create_clf(DEFAULT_PARAMS),
    "MRFO": create_clf(best_SA_MRFO),
    "HGS": create_clf(best_SA_HGS),
    "AO": create_clf(best_SA_AO),
    "SSA": create_clf(best_SA_SSA)
}

for alg in SA_clfs:
    print(alg, mean(cross_val_score(
        SA_clfs[alg], X, y, cv=10, scoring='accuracy')))

SA_EN_clfs = {
    "RSCV": create_clf(best_SA_RSCV, SA_EN),
    "DEFAULT": create_clf(DEFAULT_PARAMS, SA_EN),
```

```
    ”MRFO”: create_clf(best_SA_MRFO, SA_EN),
    ”HGS”: create_clf (best_SA_HGS, SA_EN),
    ”AO”: create_clf (best_SA_AO, SA_EN),
    ”SSA”: create_clf (best_SA_SSA, SA_EN)
}

for alg in SA_EN_clfs:
    SA_EN_clfs[alg]. fit (X_SA, y_SA)

test (SA_EN_clfs, ”SA”, ”EN”)

SA_LS_clfs = {
    ”RSCV”: create_clf(best_SA_RSCV, SA_LS),
    ”DEFAULT”: create_clf(DEFAULT_PARAMS, SA_LS),
    ”MRFO”: create_clf(best_SA_MRFO, SA_LS),
    ”HGS”: create_clf (best_SA_HGS, SA_LS),
    ”AO”: create_clf (best_SA_AO, SA_LS),
    ”SSA”: create_clf (best_SA_SSA, SA_LS)
}

for alg in SA_LS_clfs:
    SA_LS_clfs[alg]. fit (X_SA, y_SA)

test (SA_LS_clfs, ”SA”, ”LS”)

X, y = X_LS, y_LS

LS_clfs = {
```

```
        "RSCV": create_clf(best_LS_RSCV),
        "DEFAULT": create_clf(DEFAULT_PARAMS),
        "MRFO": create_clf(best_LS_MRFO),
        "HGS": create_clf(best_LS_HGS),
        "AO": create_clf(best_LS_AO),
        "SSA": create_clf(best_LS_SSA)
}

for alg in LS_clfs:
    print(alg, mean(cross_val_score(
        LS_clfs[alg], X, y, cv=10, scoring='accuracy')))

LS_EN_clfs = {
        "RSCV": create_clf(best_LS_RSCV, LS_EN),
        "DEFAULT": create_clf(DEFAULT_PARAMS, LS_EN),
        "MRFO": create_clf(best_LS_MRFO, LS_EN),
        "HGS": create_clf(best_LS_HGS, LS_EN),
        "AO": create_clf(best_LS_AO, LS_EN),
        "SSA": create_clf(best_LS_SSA, LS_EN)
}

for alg in LS_EN_clfs:
    LS_EN_clfs[alg].fit(X_LS, y_LS)

test(LS_EN_clfs, "LS", "EN")

LS_SA_clfs = {
        "HGS": create_clf(best_LS_HGS, LS_SA),
```

```
    ”DEFAULT”: create_clf(DEFAULT_PARAMS, LS_SA),
    ”RSCV”: create_clf(best_LS_RSCV, LS_SA),
    ”MRFO”: create_clf(best_LS_MRFO, LS_SA),
    ”AO”:  create_clf (best_LS_AO, LS_SA),
    ”SSA”:  create_clf (best_LS_SSA, LS_SA)
}

for  alg  in  LS_SA_clfs:
    LS_SA_clfs[alg]. fit (X_LS, y_LS)

test (LS_SA_clfs, ”LS”,  ”SA”)

EN_LS_clfs = {
    ”HGS”:  create_clf (best_EN_HGS, EN_LS),
    ”DEFAULT”: create_clf(DEFAULT_PARAMS, EN_LS),
    ”RSCV”: create_clf(best_EN_RSCV, EN_LS),
    ”MRFO”: create_clf(best_EN_MRFO, EN_LS),
    ”AO”:  create_clf (best_EN_AO, EN_LS),
    ”SSA”:  create_clf (best_EN_SSA, EN_LS)
}

for  alg  in  EN_LS_clfs:
    EN_LS_clfs[alg]. fit (X_EN, y_EN)

test (EN_LS_clfs, ”EN”, ”LS”)


EN_SA_clfs = {
```

```
        ”HGS”: create_clf(best_EN_HGS, EN_SA),
        ”DEFAULT”: create_clf(DEFAULT_PARAMS, EN_SA),
        ”RSCV”: create_clf(best_EN_RSCV, EN_SA),
        ”MRFO”: create_clf(best_EN_MRFO, EN_SA),
        ”AO”: create_clf(best_EN_AO, EN_SA),
        ”SSA”: create_clf(best_EN_SSA, EN_SA)
}

for alg in EN_SA_clfs:
    EN_SA_clfs[alg].fit(X_EN, y_EN)

test(EN_SA_clfs, ”EN”, ”SA”)

X, y = X_EN, y_EN

EN_clfs = {
        ”RSCV”: create_clf(best_EN_RSCV),
        ”DEFAULT”: create_clf(DEFAULT_PARAMS),
        ”MRFO”: create_clf(best_EN_MRFO),
        ”HGS”: create_clf(best_EN_HGS),
        ”AO”: create_clf(best_EN_AO),
        ”SSA”: create_clf(best_EN_SSA)
}

for alg in EN_clfs:
    print(alg, mean(cross_val_score(
        EN_clfs[alg], X, y, cv=10, scoring=’accuracy’)))
```