Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Информационная безопасность

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

Методика обнаружения вредоносного спама на основе методов машинного обучения

Студент ИУ8-112 (Группа)	_____ (Подпись, дата)	В. В. Волнухин (И.О.Фамилия)
Руководитель ВКР	_____ (Подпись, дата)	А. Ю. Быков (И.О.Фамилия)
Консультант по исследовательской части	_____ (Подпись, дата)	А. Ю. Быков (И.О.Фамилия)
Консультант по конструкторской части	_____ (Подпись, дата)	А. Ю. Быков (И.О.Фамилия)
Консультант по технологической части	_____ (Подпись, дата)	А. Ю. Быков (И.О.Фамилия)
Консультант по организационно-экономической части	_____ (Подпись, дата)	Н. Е. Беняев (И.О.Фамилия)
Консультант по организационно-правовому обеспечению ИБ	_____ (Подпись, дата)	Е. А. Тарапанова (И.О.Фамилия)
Нормоконтролер	_____ (Подпись, дата)	Т. Е. Завадская (И.О.Фамилия)

2020 г.

АННОТАЦИЯ

Отчёт 148 с., 5 ч., 20 источников, 35 рис., 11 табл.

МАШИННОЕ ОБУЧЕНИЕ, АНТИ-СПАМ, PYTHON, ТОКЕНИЗАЦИЯ, ВЕКТОРИЗАЦИЯ, СТЕММИНГ ПОРТЕРА, SVM, RANDOM FOREST, KNN, PAA, НАИВНЫЙ БАЙЕС, СТОХАСТИЧЕСКИЙ ГРАДИЕНТНЫЙ СПУСК, TF-IDF, АГРЕГАЦИЯ АЛГОРИТМОВ

Объектом исследования выпускной квалификационной работы являются подходы и алгоритмы обнаружения вредоносного спама.

Цель работы – разработать методику обнаружения спама в реальных условиях (для анти-спам системы) на основе кросс-валидационных исследований.

В исследовательской части исследуются используемые в данной работе алгоритмы МО и их модификации для классификации текстовых сообщений, обосновывается необходимость разработки методики обнаружения, выбираются инструменты разработки ПО, рассматриваются найденные наборы данных для проведения исследований.

В конструкторской части рассматриваются алгоритмы, методики и исследования, сделанные в рамках данной работы. В частности, ведётся создание методики обнаружения.

Технологическая часть посвящена непосредственно вопросам реализации алгоритмов и методик, программному обеспечению проведения исследований.

Четвёртая часть работы – организационно-экономическая, она посвящена определению этапов разработки, построению сетевой модели проекта, анализу затрат на выполнение проекта и прогнозированию прибыли.

Пятая часть представляет анализ нормативно-правовых актов Российской Федерации, утверждающих вопросы, связанные с разработкой ПО в целом и анти-спам систем в частности.

В заключении подводятся итоги проделанной работы.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ

Датасет (от англ. «dataset») – набор данных для работы с моделью.

Признак (англ. «feature») – термин из области машинного обучения, характеризирующий свойство объекта или его поведения, имеющее числовое или категориальное значение.

Семпл (от англ. «sample») – в МО так называется экземпляр/образец данных (случай, событие), для которого нужно найти целевую функцию (сделать прогноз); применительно к задаче обнаружения спама семпл также парой называется документом.

Спам (от англ. «spam») – нежелательные или вредоносные сообщения, не являющиеся полезными в рамках прошлых, текущих и будущих целей и задач рабочего процесса защищаемой организации или желаемыми конечным пользователем, передаваемые в рамках какого-либо социального информационного сервиса и имеющие специфику последнего (SMS-спам, спам электронной почты, спам в системах мгновенных сообщений).

N-грамма — последовательность из n элементов строки. С семантической точки зрения, это может быть последовательность звуков, слогов, слов или букв. На практике чаще встречается N-грамма как ряд слов, устойчивые словосочетания называют коллокацией. Последовательность из двух последовательных элементов (#слов) часто называют биграмма, последовательность из трёх элементов называется триграмма.

Документ – необработанный текст.

Словарь – множество уникальных слов (как правило собираются на основе датасета).

Токены – это признаки, извлекаемые из текста (другое название n-грамм в нашем случае); токенизация – их поиск и извлечение.

Векторизация – расчёт значений признаков токенизированных семплов.

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ЛВП	– Линейное векторное пространство
МО	– Машинное обучение
ЯП	– Язык программирования

ВВЕДЕНИЕ

Обоснование актуальности темы

Современные социальные информационно-коммуникационные сервисы и их пользователи ежедневно подвергаются постоянно усложняющимся спам-атакам, которые приводят не только к возникновению существенных избыточных аппаратных затрат на обработку сообщений, уменьшению КПД социальных рабочих процессов, но и могут спровоцировать НСД к сети организации или устройству пользователя в том случае, если конечный пользователь не распознает пришедшее фишинговое сообщение как спам и правильно не отреагирует.

Таким образом, необходимо разрабатывать и регулярно совершенствовать системы анти-спам для своевременного обнаружения таких сообщений. Современные технологии обнаружения базируются на методах из области машинного обучения.

Цель

Разработать методику обнаружения спама в реальных условиях (для анти-спам системы) на основе кросс-валидационных исследований.

Задачи

1. Провести обзор предметной области.
2. Сформулировать математическую постановку задачи.
3. Отобрать алгоритмы МО, создать их модификации и проработать варианты их комбинирования.

4. Разработать алгоритм поиска лучших комбинаций алгоритмов МО для обнаружения вредоносного спама.
5. Разработать методику предобработки текста.
6. Проанализировать и отобрать методы извлечения признаков из текста (токенизации и векторизации).
7. Разработать исследовательское ПО для валидации алгоритмов, получения сравнительных результатов и определить правила исследования.
8. Провести исследования на нескольких датасетах с семплами разного характера для обнаружения особенностей и свойств комбинаций алгоритмов МО, подходов к агрегированию в связке с методами извлечения признаков, методами предобработки текста и поиска наилучших комбинаций.
9. Разработать методику обнаружения вредоносного спама для sms-сообщений и сообщений электронной почты.
10. Рассмотреть организационно-правовые аспекты проекта.
11. Рассмотреть организационно-экономические аспекты проекта.

1 ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ

В данной части рассмотрены используемые в данной работе алгоритмы МО и их модификации для классификации текстовых сообщений, обосновывается необходимость разработки методики обнаружения, выбираются инструменты разработки ПО, рассматриваются найденные наборы данных для проведения исследований, производится теоретический анализ как самих алгоритмов в контексте их комбинирования, так и их гиперпараметров.

1.1 Обоснование необходимости разработки методики обнаружения вредоносного спама на основе методов МО

Современные социальные информационные сервисы и их пользователи ежедневно подвергаются постоянно усложняющимся спам-атакам, которые приводят не только к возникновению существенных избыточных аппаратных затрат на обработку сообщений, уменьшению КПД социальных рабочих процессов, но и могут спровоцировать НСД к сети организации или устройству пользователя в том случае, если конечный пользователь не распознает пришедшее фишинговое сообщение как спам и правильно не отреагирует.

Таким образом, необходимо разрабатывать и регулярно совершенствовать системы анти-спам для своевременного обнаружения таких сообщений. Современные технологии обнаружения базируются на методах из области машинного обучения (МО).

Методика – это порядок действий, который необходимо выполнить для достижения заданного результата – для обнаружения спама полноценной анти-спам системой.

1.2 Поиск алгоритмов МО для обнаружения вредоносного спама

Поиск алгоритмов МО можно разделить на 2 основных этапа: теоретический отбор и практический отбор, последний фактически пронизывает всю данную работу.

Для теоретического отбора воспользуемся двумя подходами: анализом источников в свободном доступе, где сравнивалась эффективность алгоритмов при работе с текстом и для обнаружения спама и теоретическим анализом свойств алгоритмов.

Воспользуемся официальными рекомендациями создателей библиотеки для научных вычислений и МО Scikit-learn (см. рисунок 1, где показано сравнение алгоритмов при классификации текстовых семплов по показателям точность (метрика accuracy), время обучения, время предсказания; за критерий отбора выбираем только метрику точности) для выявления перспективных методов обнаружения [1]. Далее выберем те алгоритмы, которые теоретически отличаются от остальных, поскольку в перспективе требуется, чтобы каждый алгоритм исправлял недостатки коллег для обеспечения лучшего качества обнаружения и по-своему обобщал выборку. Синхронизируем это с рядом других научных и инженерных источников по МО [2,3]. Получим следующий список алгоритмов для исследования:

1. Наивная Байесовская классификация (Naive Bayes Classifier).
2. Классификатор по ближайшим центроидам (Nearest Centroid Classifier, NCC).
3. Стохастический градиентный спуск (Stochastic Gradient Descent, SGD).
4. Метод опорных векторов (Support Vector Machine, SVM).
5. Пассивно-агрессивный классификатор (Passive Aggressive Algorithm, PAA).
6. Гребневая регрессия (Ridge Classifier, RC).
7. Метод k-ближайших соседей (K- Nearest Neighbor Classifier, KNN).
8. Персептрон – модификация стохастического град. спуска (Perceptron).

9. Алгоритм случайного леса (Random Forest Classifier).
10. Решающие деревья.
11. Логистическая регрессия.

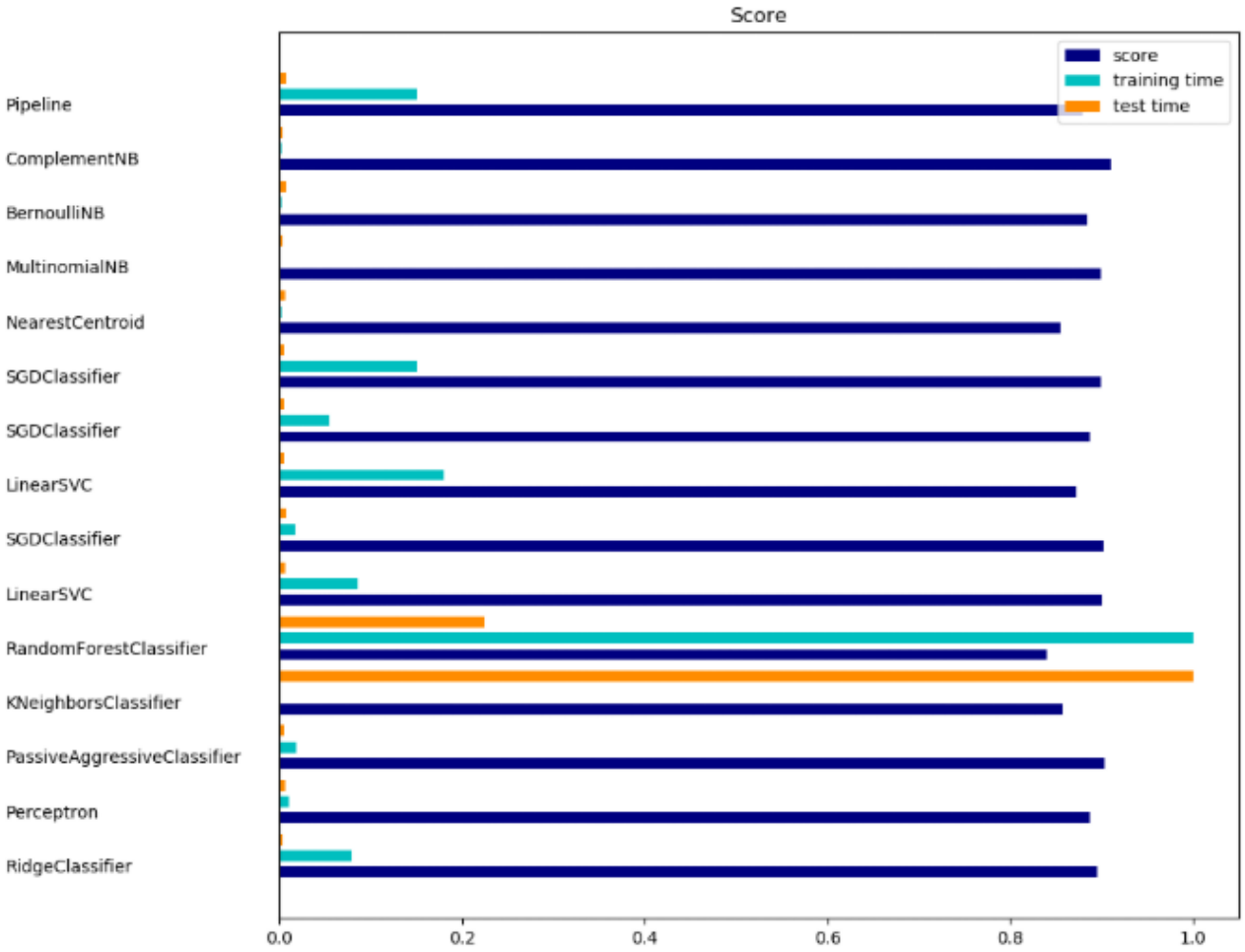

Рисунок 1 – Характеристики алгоритмов при прогнозировании тем новостных текстов

Не существует надёжных теоретических критериев выбора алгоритмов для классификации текста. Лучший способ – практический: выбрать датасет с несколькими тысячами семплов (не меньше), посредством кросс-валидации проверить все возможные комбинации алгоритмов на уровень качества, отсортировать комбинации в порядке убывания значений выбранных метрик качества, отфильтровать результаты.

Таким образом, в общем и целом, для отбора алгоритмов МО и формирования их модификаций были предприняты следующие шаги:

- список алгоритмов для исследований был получен при обзоре научных и инженерных источников, для рассмотрения были

отобраны наиболее популярные и перспективные алгоритмы для решения задачи классификации текста (не обязательно очень эффективные поодиночке),

- теоретические данные (знание принципов работы алгоритмов и их гиперпараметров) использовались для формирования модификаций алгоритмов, обнаружения случаев дубликата принципов работы (алгоритмы должны отличаться, характер отличий задаётся типом комбинации), из списка алгоритмов исключаются те, что дублируют работу более качественных конкурентов, также анализировались модификации, добытые из научных и инженерных источников,
- было отобрано 3 больших набора данных с отличающимися характеристиками семплов (разные года сбора, длины сообщений и словари), написаны предобработчики и экстракторы признаков для них,
- был написан алгоритм, который посредством кросс-валидации (учитывающей дисбаланс классов в датасетах) валидировал все комбинации алгоритмов на заданном датасете и выводил отсортированные и отфильтрованные результаты,
- на основе результатов экспериментов добавлялись новые модификации алгоритмов, не созданные ранее при теоретическим рассмотрении, для выявление более удачных модификаций, в тоже время неудачные алгоритмы исключались из исследования – таким образом, во время экспериментов удалось лучше выделить наиболее способные алгоритмы, достойные для создания на их основе комбинаций, а также было оценено влияние разных гиперпараметров на конечный результат для каждого конкретного алгоритма.

После проведения всех этапов были исключены следующие алгоритмы из первоначального списка:

1. Решающие деревья - их отсутствие компенсировано присутствием модификаций алгоритма RandomForest с небольшим кол-вом деревьев.
2. Логистическая регрессия - заменена стохастическим градиентным спуском с соотв. функцией потерь для упрощения работ.
3. Классификатор по ближайшим центроидам – не было замечено явных выдающихся свойств, алгоритм исключён для упрощения работ.
4. Гребневая регрессия – не было замечено явных выдающихся свойств, алгоритм исключён для упрощения работ.
5. Мультиномиальный алгоритм Наивного Байеса – исключен, так как теоретически и практически проигрывает комплементарному алгоритму.

В итоге был выведен перечень алгоритмов с модификациями, рассмотрим его подробнее.

1.2.1 Перечень алгоритмов МО и их модификаций

Для начала следует пояснить, что данные являются в целом линейно-сепарабельными, что обычно влияет на выбор алгоритмов, однако есть смысл задействовать и алгоритмы с нелинейной функцией, поскольку это может помочь обнаружить недостающие семплы [2].

Цель создания модификаций – повысить разнообразность комбинаций для увеличения охвата поиска лучших комбинаций.

Ниже описаны все используемые алгоритмы (названы именами классов из ПО) с модификациями и описанием только исследуемых гиперпараметров, в том числе изменённых или специально незатронутых. Для модификаций указаны изменения относительно конфигурации по умолчанию.

Для алгоритмов, поддерживающих распараллеливание вычислений, включена максимальная многопоточность (параметр n_jobs=-1).

Приписка «_Default» к названию алгоритма означает установку гиперпараметров по умолчанию.

Случайный лес

Пусть обучающая выборка состоит из N образцов, размерность пространства признаков равна M, и задан параметр m (в задачах классификации обычно $m \approx \sqrt{M}$) как неполное количество признаков для обучения.

Наиболее распространённый способ построения деревьев ансамбля следующий (называется бэггинг (англ. bagging, сокращение от англ. bootstrap aggregation)). Сгенерируем случайную подвыборку семплов с повторениями размером N из обучающей выборки (это и есть процесс бутстрапа). Построим решающее дерево, классифицирующее образцы данной подвыборки, причём в ходе создания очередного узла дерева будем выбирать набор признаков, на основе которых производится разбиение (не из всех M признаков, а лишь из m случайно выбранных, этот фактор настраивается). Выбор наилучшего из этих m признаков может осуществляться различными способами – используются различные критерии удачности разбиения подвыборки: критерий Джини (Gini impurity), например, который применяющийся также в алгоритме построения решающих деревьев CART. В некоторых реализациях алгоритма вместо него используется критерий прироста информации (information gain) [3].

Дерево строится до полного исчерпания подвыборки и не подвергается процедуре прунинга (англ. pruning — отсечение ветвей) (в отличие от решающих деревьев, построенных по таким алгоритмам, как CART или C4.5). Однако прунинг можно включить дополнительно [4].

Классификация объектов проводится путём голосования: каждое дерево ансамбля относит классифицируемый объект к одному из классов, и побеждает класс, за который проголосовало наибольшее число деревьев (принцип большинства).

Таким образом, RF – это ансамблевый алгоритм, который очень разнообразно настраивается, последствием является широкая вариативность результатов качества обнаружения. Рассмотрим подробнее настройки и модификации

- RandomForestAlg_Default (конфигурация по умолчанию):
    - n_estimators=100 (кол-во деревьев), важно понимать, что при решении нашей задачи как правило используется много признаков (тысячи - сотни тысяч), поэтому не стоит пренебрегать кол-вом деревьев в ансамбле,
    - criterion= «gini» (критерий качества разделения выборки в каждом дереве при обучении),
    - max_depth=None (не ограничена искусственно),
    - min_samples_split=2 (минимальный размер подмножества, которое можно разделить при обучении),
    - min_samples_leaf=1 (минимальный размер листового подмножества, которое соответствует определённому классу),
    - max_features= «auto» (макс. кол-во признаков, которые используются для разбиении множества во время обучения каждого дерева) – данный параметр нигде не изменялся, поскольку необходимо проводить дополнительные практические исследования его влияния,
    - max_leaf_nodes=None (максимальное кол-во листьев-подмножеств, помеченных классом после процедуры обучения)- данный параметр нигде не изменялся, поскольку необходимо проводить дополнительные практические исследования его влияния,
    - bootstrap=True (включена ли функция использования бутстрап-выборок, когда каждое дерево обучается на своём случайном подмножестве семплов),
    - class_weight=None (используются для семплов веса, позволяющие повысить универсальность модели),
    - max_samples=None (максимальное кол-во семплов в бутстрап-подмножестве, по умолчанию используются все семплы обучающего датасета).
- RandomForestAlg_Small:

    - n_estimators=10.
- RandomForestAlg_Big:
    - n_estimators=300.
- RandomForestAlg_Medium:
    - n_estimators=50.
- RandomForestAlg_BigBootstrap75:
    - RandomForestAlg_Big,
    - max_samples=0.75 (75% семплов присутствует в каждом бутстрап-подмножестве).
- RandomForestAlg_Bootstrap90:
    - max_samples=0.9.
- RandomForestAlg_MDepth20:
    - max_depth=20.
- RandomForestAlg_MDepth30:
    - max_depth=30.
- RandomForestAlg_Balanced:
    - class_weight=«balanced» (веса обратно пропорциональные популярности их класса, это позволяет сделать модель менее предвзятой к новым данным, не похожим на обучающий датасет).

Наивный Байес

Существует три основных типа моделей на основе наивного байесовского алгоритма:

- Gaussian (нормальное распределение). Модель данного типа используется в случае непрерывных признаков и предполагает, что значения признаков имеют нормальное распределение.
- Multinomial (мультиномиальное распределение). Используется в случае дискретных признаков. Например, в задаче классификации текстов

признаки могут показывать, сколько раз каждое слово встречается в данном тексте.

- Bernoulli (распределение Бернулли). Используется в случае двоичных дискретных признаков (могут принимать только два значения: 0 и 1). Например, в задаче классификации текстов с применением подхода «мешок слов» (bag of words) бинарный признак определяет присутствие (1) или отсутствие (0) данного слова в тексте [2].

Рассмотрим 2 варианта алгоритма: Multinomial Naive Bayes (класс MultinomialNB) и Complement Naive Bayes (Дополнение Наивного Байеса) (класс ComplementNB). Оба алгоритма - вероятностные методы обучения.

MNB

Пусть имеется *n* *m*-грамм(слов) в словаре. Тогда распределения задаются векторами на каждый класс *y* следующим образом: $\theta_y = (\theta_{y1}, \ldots, \theta_{ym})$, где θ_{yi} вероятность $P(x_i \mid y)$ слова *i* появится в случайном образце класса *y*:

$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$ (1) , где $N_{yi} = \sum_{x \in T} x_i$ (2) - кол-во появлений слова *i* в семплах, соответствующих классу *y*, находящимся в тренировочном наборе T, $N_y = \sum_{i=1}^{n} N_{yi}$ (3) - общее кол-во появлений всех слов в семплах класса *y*, $\alpha \geq 0$ – сглаживание, предотвращающее нулевую вероятность (такое бывает, если обучающие семплы не содержат совершенно никаких слов словаря), $\alpha = 1$ – сглаживание Лапласа, $\alpha < 1$ – сглаживание Линдстона. Таким образом, мы построили распределения вероятностей признаков (появления слов) для каждого класса. Это и есть процесс обучения.

Теперь рассмотрим процесс предсказания спама.

Формула теоремы Байеса:

$$P(A \mid B) = \frac{P(AB)}{P(B)} = \frac{P(B \mid A)\,P(A)}{P(B)} \quad (4)$$

Формула для предсказания спама похожа на классический аналог, однако знаменатель раскрыт по правилу полной вероятности:

$$P(y = 1 \mid x) = \frac{\Pi\ P(x_i \mid y = 1) * P(y = 1)}{P(x_1, x_2, .., x_n)} (5)$$

Недавние статистические исследования показали, что на сегодняшний день вероятность любого сообщения быть спамом составляет по меньшей мере 80 % => p(y=1) = 0.8, p(y=0) = 0.2.Однако большинство байесовских программ обнаружения спама делают предположение об отсутствии априорных предпочтений у сообщения быть «spam», а не «ham», и полагает, что у обоих случаев есть равные вероятности 50 %. О фильтрах, которые используют эту гипотезу, говорят как о фильтрах «без предубеждений». Это означает, что у них нет никакого предубеждения относительно входящей электронной почты. Мы также воспользуемся данным предположением и тогда получим упрощенную формулу предсказания:

$$P(y = 1 \mid x) = \prod P(x_i \mid y = 1) * P(y = 1)\ (6)$$

в виде функции:

$$\hat{y} = arg\ \max_{y_c} \prod P(x_i \mid y = y_c) * P(y = y_c)\ (7)$$

Данные формулы демонстрирует «наивность» - мы предполагаем, что все признаки равноправны и независимы, это даёт нам право перемножить вероятности наличия их значений (вероятность наличия слова в семпле).

Для решения не/спам устанавливается пороговое значение h, за окончательный вердикт отвечает следующий предикат: $P(y = 1 \mid x) > h$

Итоговая функция в линейном виде:

$$\hat{y} = arg\ \max_{y} \sum_i x_i w_{yi}\ (8),$$

где $w_{ci} = \log \widehat{\theta_{yi}}$ (9) – веса признаков для каждого класса,

CNB

Данный алгоритм является расширением предыдущего. Его цель – нивелировать зависимость от несбалансированных датасетов. Очевидно, что в

случае с MNB, если кол-во семплов для классов будет не 0.5/0.5, а в другом соотношении, то это приведёт к искажениям прогнозов, поскольку алгоритм не учитывает соотношение классов как таковое – он просто рассчитывает распределения для классов по имеющимся семплам. CNB зачастую показывает себя лучше в реальных задачах.

Суть CNB такова: в отличие от MNB, вычисляется вероятность появления слова-признака в случайном семпле не рассматриваемого класса (*y=1* - спам), а в других классах, кроме рассматриваемого – рассчитываются вероятности для «дополнения» рассматриваемого класса, в то время как в MNB – для рассматриваемого класса [2].

Происходит суммирование по всем семплам *j*, которые не класса c, значений признака *i*, d_{ij} – это значение признака *i* в документе *j* (в нашем случае это частотность слов), далее делится на сумму значений признаков по всем семплам всех классов, кроме рассматриваемого *c*. Т.о. мы получаем «дополненные» распределения для каждого класса. Далее из этих распределений формируются веса признаков для каждого класса.

$$\widehat{\theta_{ci}} = \frac{\alpha_i + \sum_{j:y_j \neq c} d_{ij}}{\alpha + \sum_{j:y_j \neq c} \sum_k d_{kj}} \text{ (10)},$$

$$w_{ci} = \log \widehat{\theta_{ci}} \text{ (11)},$$

$$w_{ci} = \frac{w_{ci}}{\sum_j |w_{cj}|} \text{ (12) (нормируем вес)}$$

где c – класс документа/семпла d, $\alpha = \sum_i \alpha_i$ (13).

Итоговая функция предсказания класса в вероятностном и линейном видах [3]:

$$\hat{c} = arg\min_c \prod \frac{1}{P(x_i \mid y \neq c)} * P(y = c) \text{ (14)}$$

$$\hat{c} = arg\min_c \sum_i x_i w_{ci} \text{ (15)}$$

В работе используется Комплементарный алгоритм с настройками по умолчанию.

Метод опорных векторов

Метод опорных векторов (англ. SVM, support vector machine) — набор алгоритмов обучения с учителем, использующихся для задач классификации, регрессионного анализа и обнаружения выбросов. Особым свойством метода опорных векторов является непрерывное уменьшение эмпирической ошибки классификации и увеличение зазора, поэтому метод также известен как метод классификатора с максимальным зазором. Машина опорных векторов строит гиперплоскость (часто рассматривают 2 вспомогательные, находящиеся на одинаковом расстоянии от основной – центральной гиперплоскости) в пространстве высокой или бесконечной размерности. Хорошее разделение достигается с помощью гиперплоскости, которая имеет наибольшее расстояние до ближайших точек обучающих данных любого класса (так называемый функциональный запас), поскольку в общем случае, чем больше запас, тем ниже ошибка обобщения классификатора (см. рисунок 2) [5].

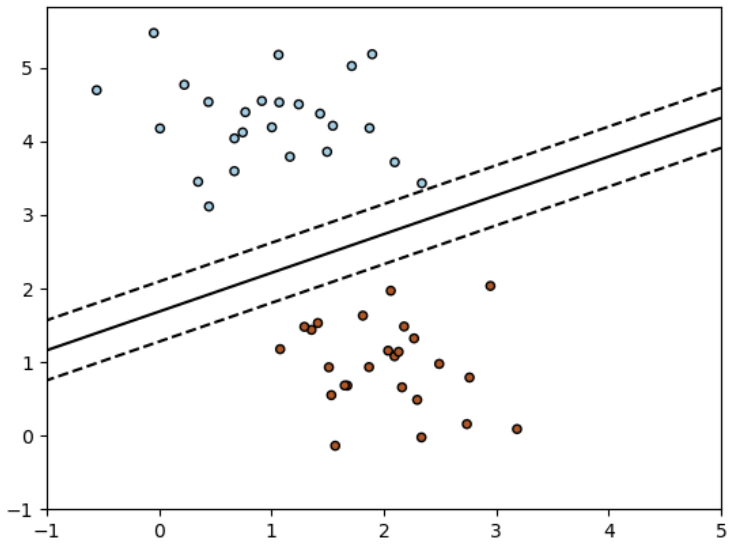

Рисунок 2 – Работа SVM при 2 признаках

Этот метод обычно считается лучшим из алгоритмов классификации текста (хотя, он немного медленнее чем наивный Байес). Он резко теряет в качестве обнаружения при уменьшении обучающей выборки.

SVM отличается от персептрона (и др. аналогов, использующих взвешенную сумму весов со значениями признаков) обеспечением зазоров по бокам

центральной гиперплоскости, в первую очередь (см. рисунок 3). Это обеспечено: 1) функцией, которая минимизируется, 2) формулой корректировки весов (реакция на ошибку), основанной на функции активации (дифференцируемой функции, что позволяет рассмотреть понятие градиента), а не на пороговой, как у персептрона (см. рисунок 4) [5].

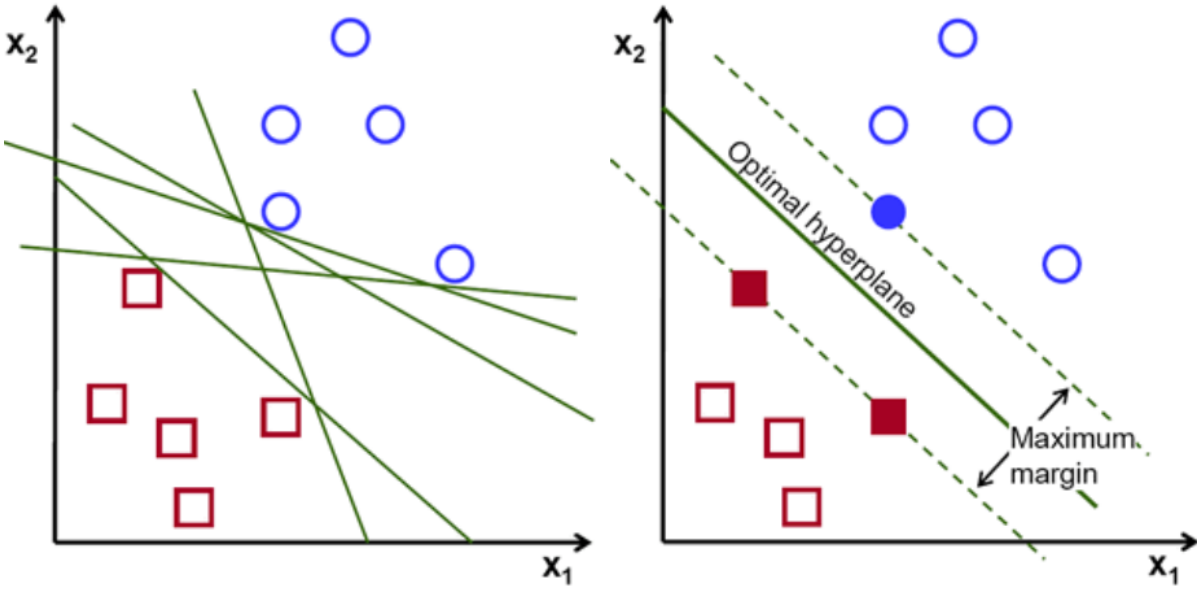

Рисунок 3 – Слева показана работа персептрона, справа - SVM

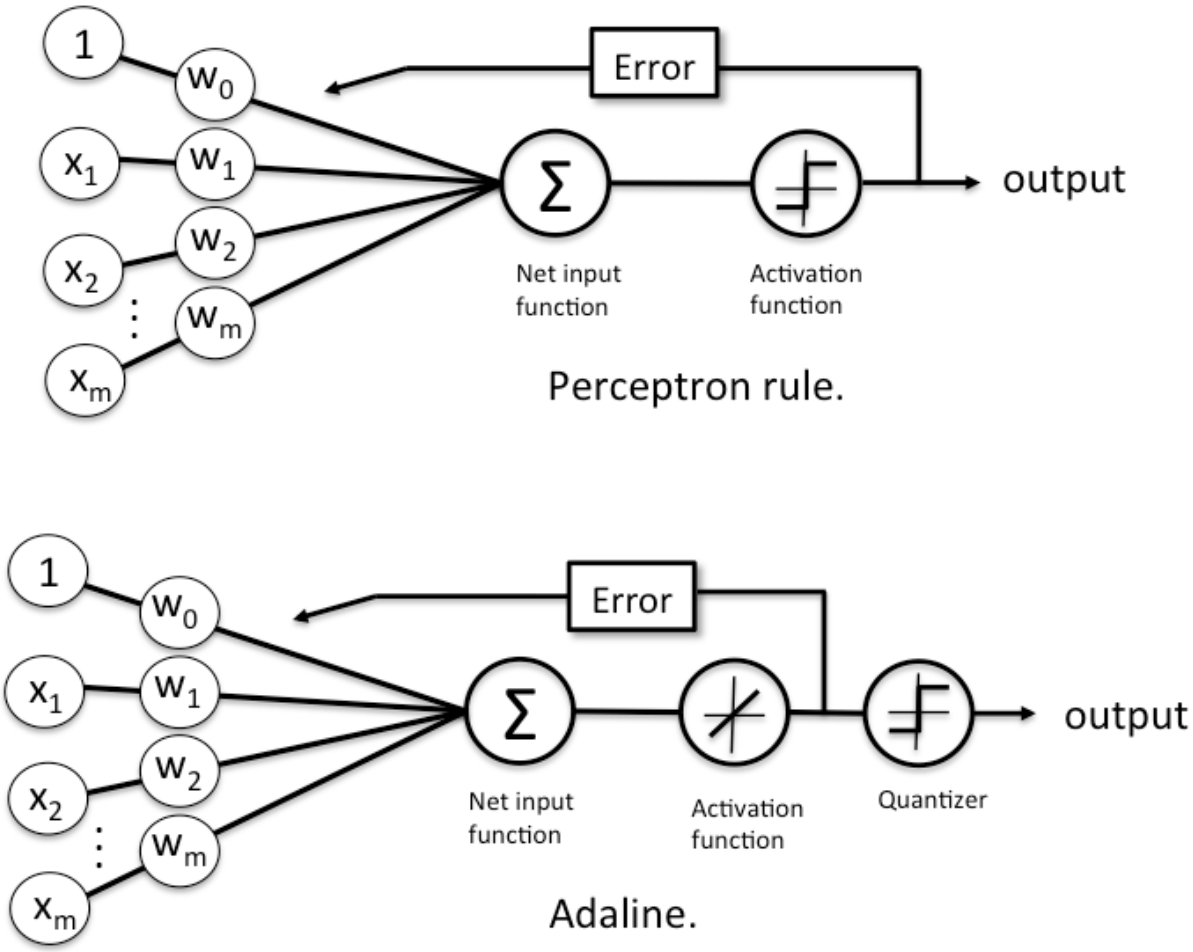

Рисунок 4 – Принципы функционирования персептрона и адаптивного линейного нейрона

Гиперплоскость максимизирует запас с ближайшими тренировочными экземплярами. Эти экземпляры называются опорными векторами, потому что они фиксируют положение и ориентацию гиперплоскости. Линейные SVM предполагают, что данные обучения линейно разделимы. di – дистанции до тренировочных семплов, которые оказались после обучения принадлежащими не своему классу, сумма этих дистанций может быть мерой качества разделения семплов (см. рисунок 5).

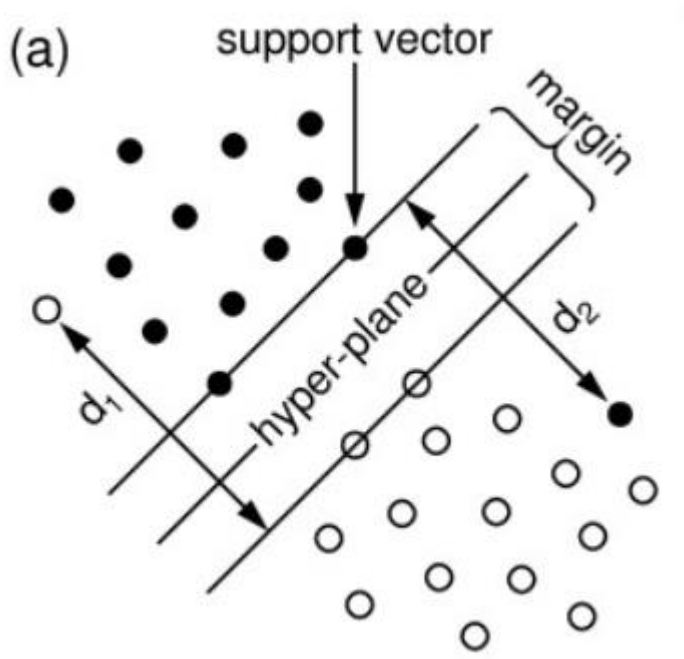

Рисунок 5 – Принцип работы линейного SVM

При наличии нелинейной связи между признаками и откликом качество линейных классификаторов часто может оказаться неудовлетворительным. Машину опорных векторов SVM (Support Vector Machine) можно рассматривать как нелинейное обобщение линейного классификатора, основанное на расширении размерности исходного пространства предикторов с помощью специальных ядерных функций. Это позволяет строить модели с использованием разделяющих поверхностей самой различной формы (см. рисунок 6) [5].

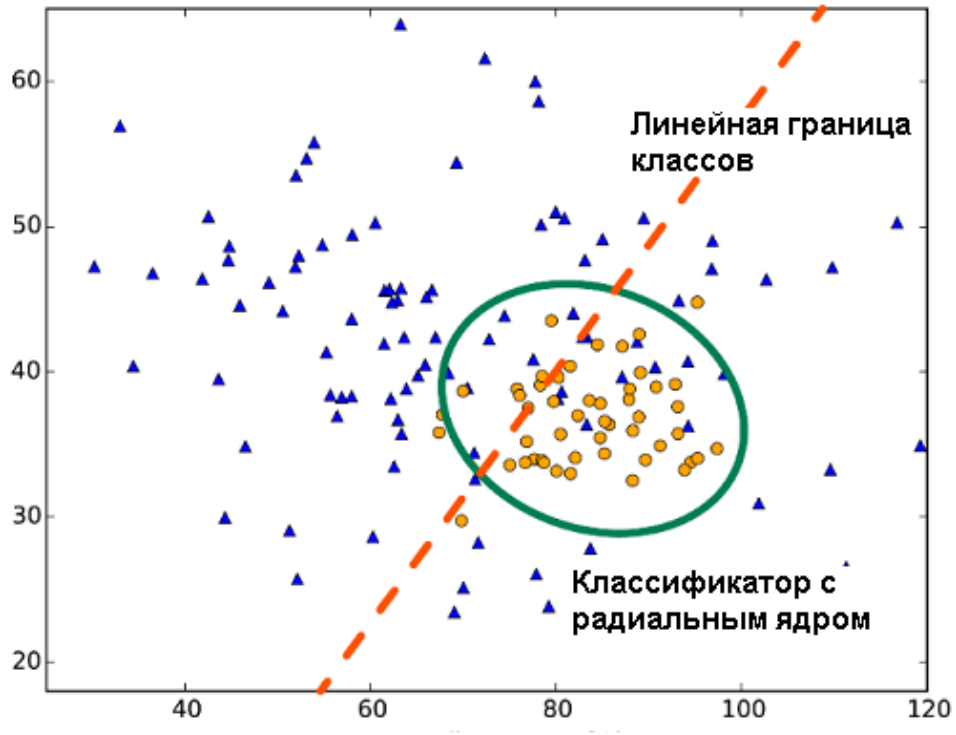

Рисунок 6 – Работа SVM с радиальным ядром

Целевая функция потерь по умолчанию - «square hinge»:

$$L(y, \hat{y}) = \sum_{i=0}^{N} \left(max(0, 1 - y_i \cdot \hat{y}_i)^2 \right) \quad (16)$$

(ŷ - спрогнозированная метка класса -1 или 1)

Можно использовать и предшественницу – «hinge», у которой нет квадрата. Очевидно, без квадрата падает чувствительность функции к ошибкам. Именно такая функция используется персептроном в данной работе.

Функция потерь минимизируется каждым алгоритмом и по ней каждый алгоритм отслеживает степень сходимости/обученности, SVM – не исключение.

- LinearSVCAlg_Default:
  - Линейное ядро с настройками по умолчанию, преимущественно,
  - регуляризация L1 (обновляются лишь веса ненулевых фич, а не всех подряд) может быть использована только лишь в случае, когда кол-во семплов превосходит кол-во фич, а в нашем случае кол-во фич

значительно превосходит кол-во семплов, поэтому используем стандартную L2,

- параметр class_weight мог быть установлен на 'balanced', поскольку в нашем случае датасеты существенно, но не критично дисбалансированы, включение данной функции ведёт к изменению шага изменения весов обратно пропорционально популярности класса семпла, это делается для избегания предвзятости модели к семплам популярного класса (данная проблема изображена ниже на рисунке 7),

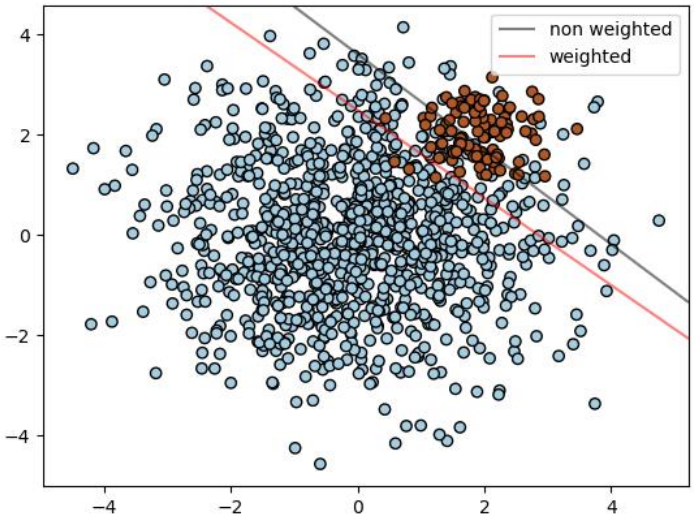

Рисунок 7 – Проблема проведения гиперплоскости при дисбалансе классов

- штраф C по умолчанию (=1.0), поскольку этот параметр приведёт к приспособленности к одному датасету, но ухудшенным результатам для случайных семплов или др. датасетов,
- loss='square_hinge' – квадратичная функция потерь, которая минимизируется при обучении, можно ещё использовать loss='hinge', однако в этом не обнаружено необходимости,
- tol=1e-4 по умолчанию, это допуск для обнаружения незначительно удалённого вектора, который будет принят за опорный, в идеале вектор должен находится строго на границе с правой/левой

гиперплоскостью, изменение этого параметра ведёт к изменению состава участвующих семплов в корректировке весов,

- max_iter=1000 по умолчанию, это кол-во итераций на схождение, т.е. даже если максимальное приближение не будет достигнуто, алгоритм будет остановлен. Данный параметр не важен для предотвращения переобучения, он влияет на точность положения гиперплоскости.

- LinearSVCAlg_Balanced:
  - class_weight = 'balanced' для проверки концепции балансировки и результативности самостоятельной, а также в составе комбинаций (в теории ожидается повышение FP-rate).
- LinearSVCAlg_MoreSupports:
  - tol=0.1 (увеличиваем кол-во опорных векторов, а значит критериев для перемещения гиперплоскости),
- SVCAlg_RBF_Default:
  - Радиальное (RBF) ядро с настройками по умолчанию,
  - Предположительно (по результатам обзора научных и инженерных источников), может быть эффективно как минимум в сочетании с др. алгоритмами несмотря на то, что наша задача имеет линейно-сепарабельные семплы,
  - Есть 2 важнейших параметра, влияющих на степень обобщения выборки: gamma и C, их совместное влияние на характер классификации показано на рисунке 8:

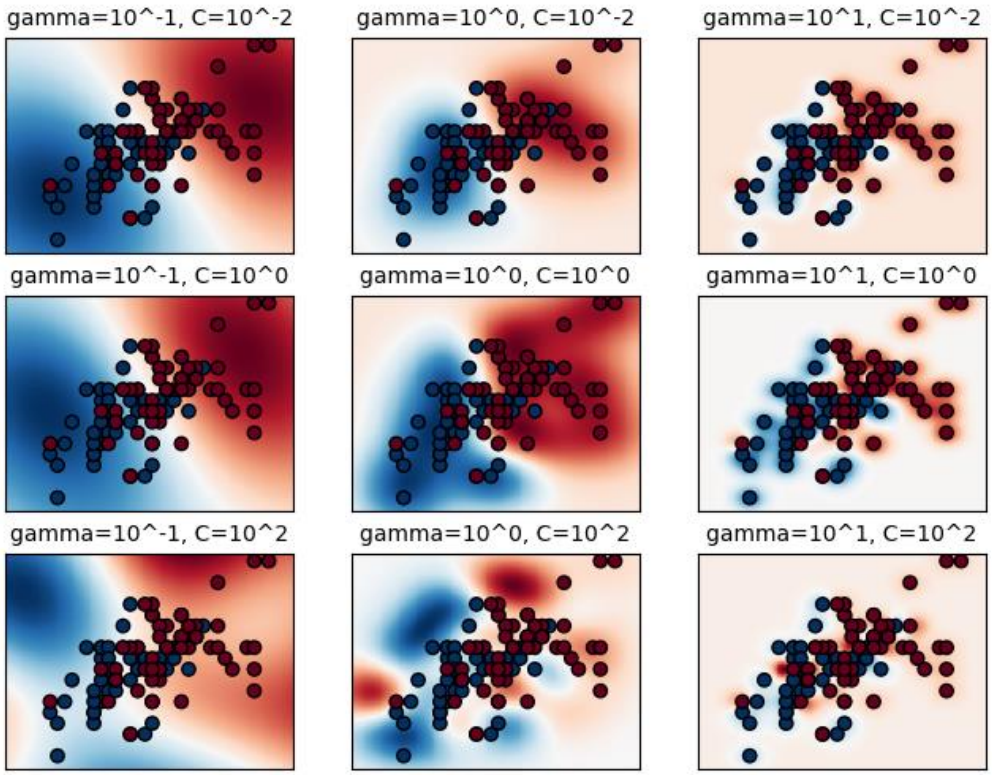

Рисунок 8 – Влияние гиперпараметров gamma и C на обобщение выборки

В конфигурации по умолчанию gamma = ‘scale’ (=1 / (n_features * дисперсия семплов)) и не обнаружено причин для изменения этих параметров. В противовес есть значение ‘auto’ = 1 / n_features, но в таком случае гамма не будет уменьшаться при увеличении дисперсии между семплами, а это полезная функция для нашей задачи. Также можно установить C и gamma в числовом варианте, однако лучше это выполнять в рамках алгоритма подбора гиперпараметров, в данной работе это не рассматривается [6].

Стохастический градиентный спуск

Данный алгоритм представляет собой способ оптимизации целевой линейной функции (например, linear SVM, логистическая регрессия). Оптимизировать (минимизировать) можно разные функции потерь (изменение функции потерь влечёт изменение формулы корректировки весов (из-за разных функций потерь), функции активации и пороговой функции), общим остаётся

принцип градиентного спуска – сам по себе он определяет лишь характер корректировки весов. В данной работе используется именно стохастический алгоритм, а не пакетный, хотя преимущества стохастичности в виде возможности дообучения (онлайн обучения) не используются.

Преимущества стохастического градиентного спуска:

- эффективность,
- простота реализации,

К недостаткам стохастического градиентного спуска относятся:

- SGD требует наличия ряда гиперпараметров, таких как параметр регуляризации и количество итераций (а значит требуется адаптация к набору данных).
- SGD чувствителен к масштабированию объектов – нормирование значений признаков обязательно.

Рассмотрим подробнее принцип минимизации с помощью градиента функции (см. рисунок 9). Градиент $Q(w)$ аппроксимируется градиентом каждого тренировочного семпла по формуле $\mathrm{w} \coloneqq \mathrm{w} - \eta \nabla \mathrm{Q}_i(\mathrm{w})$. Через тренировочный набор может быть осуществлено несколько проходов, прежде чем алгоритм сойдётся. Если такие проходы происходят, данные перемешиваются перед каждым проходом, чтобы избежать зацикливания. На схеме под стоимостью (cost) понимаются функция потерь/ошибки, a derivative of cost – её производная. Получаем так называемый спуск к наименьшей ошибке, используя для корректировки весов градиент [6].

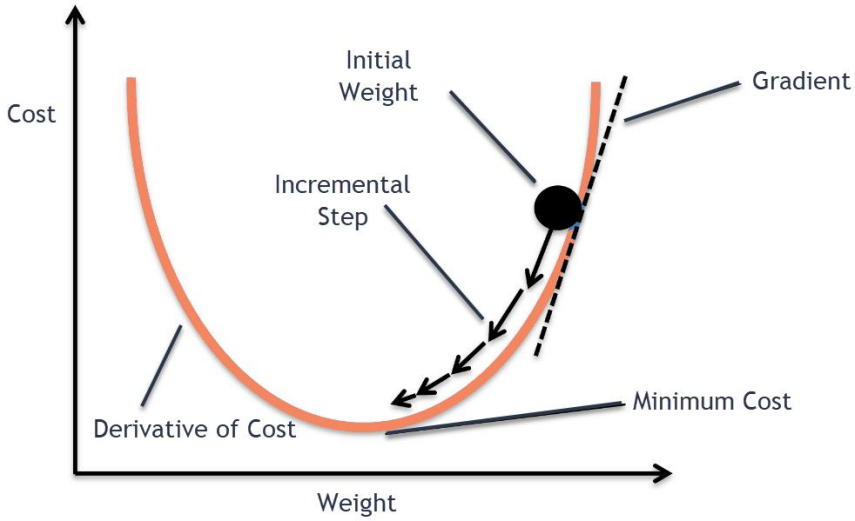

Рисунок 9 – Смысл применения градиента функции в SGD

Рассмотрим модификации данного алгоритма.

- SGDAlg_Default:
  - Функция потерь: loss="hinge", такая же, как в вероятном варианте SVM (однако в данной работе у SVM др. функция), принцип такой же, как в SVM: веса обновляются только в том случае, если семпл/вектор нарушает ограничение отступа, т.е. слишком близко приближается к разделяющей гиперплоскости, тогда мы корректируем на основе этого семпла/опорного вектора положение гиперплоскости.
  - Регуляризация: l2.
  - Использование усреднённого алгоритма Averaged Stochastic Gradient Descent (ASGD): average = False. Усреднение используется, чтобы уменьшить влияние шума. Градиентный спуск может быть близок к оптимальному, но на самом деле не сходиться к нему, а колебаться вокруг оптимального. В этом случае усреднение весов даст решение с большей вероятностью ближе к оптимальному. ASGD работает наилучшим образом при большом кол-ве признаков. Принцип коррекции: алгоритм SGD работает как обычно, корректируя свои

веса, однако после окончания поступления семплов выполняется усреднение весов для всех признаков по формуле [6]:

$$\overline{w} = \frac{1}{N} \sum_{t=1}^{N} w_t \quad (17)$$

- tol («tolerance») - в отличие от SVM, это допуск при сходимости, чем выше значение, тем быстрее алгоритм перестанет сходиться, критерий остановки: loss > best_loss – tol, tol = 1e-3.
- max_iter = 1000.

- SGDAlg_LogLoss:
  - loss='log' (логистическая регрессия)
- SGDAlg_AdaptiveIters:
  - max_iter=⌈(10^6 / n)⌉, где n - кол-во семплов в обучающей выборке. Данная формула была выведена опытным путем исследователями алгоритма и именуется «первым разумным предположением о количестве итераций».
- ASGDAlg_Default:
  - average=True.

Пассивно-агрессивный классификатор

Пассивно-агрессивные алгоритмы (Passive Aggressive Classifier) - это семейство алгоритмов для крупномасштабного обучения (большой диапазон значений признаков). Они похожи на персептрон в том, что они не требуют параметра скорости обучения. Однако, в отличие от персептрона, они включают параметр регуляризации C. Являются потенциальной заменой персептрона в ряде задач.

Классические PA-алгоритмы (есть и др. варианты этих алгоритмов), описанные ниже, являются хорошим возможным выбором при классификации текста, важно попробовать 1 и 2 варианты [3], классический вариант вовсе без не имеет параметра C. На рисунке 10 изложены варианты алгоритма.

INPUT: aggressiveness parameter $C > 0$
INITIALIZE: $\mathbf{w}_1 = (0, \ldots, 0)$
For $t = 1, 2, \ldots$

- receive instance: $\mathbf{x}_t \in \mathbb{R}^n$
- predict: $\hat{y}_t = \text{sign}(\mathbf{w}_t \cdot \mathbf{x}_t)$
- receive correct label: $y_t \in \{-1, +1\}$
- suffer loss: $\ell_t = \max\{0\,,\ 1 - y_t(\mathbf{w}_t \cdot \mathbf{x}_t)\}$
- update:

1. set:

$$\tau_t = \frac{\ell_t}{\|\mathbf{x}_t\|^2} \quad \text{(PA)}$$

$$\tau_t = \min\left\{C\,,\ \frac{\ell_t}{\|\mathbf{x}_t\|^2}\right\} \quad \text{(PA-I)}$$

$$\tau_t = \frac{\ell_t}{\|\mathbf{x}_t\|^2 + \frac{1}{2C}} \quad \text{(PA-II)}$$

2. update: $\mathbf{w}_{t+1} = \mathbf{w}_t + \tau_t y_t \mathbf{x}_t$

Рисунок 10 – Основные пассивно-агрессивные алгоритмы

Данные алгоритмы также вычисляют взвешенную сумму весов и значений признаков, а потом используют пороговую функцию сигнум:

$$\operatorname{sgn} x = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases} \quad (18)$$

От семпла к семплу все веса обновляются. Функция потерь: «hinge». Изменение веса происходит похожим на персептрон образом, однако может применяться регуляризация.

В работе используются 2 варианта алгоритма и одна модификация 2 варианта с весами для балансировки классов.

Метод k-ближайших соседей

KNN запоминают обучающие семплы, а все прогнозируемые семплы рассматривают поодиночке совместно с уже известными обучающими. Семплы рассматриваются как точки в n-мерном пространстве, между которыми можно вычислить расстояния.

Рассмотрим схему алгоритма на рисунке 11. Выбирается метрика расстояния, в данной работе используется метрика Минковского. Часто рисуют радиус вокруг семпла, для которого выполняется прогнозирование, на практике его нет и быть не может, ведь распределение семплов может сильно разниться, Для каждой прогнозируемой вершины необходимо перебрать все обучающие семплы k раз для определения минимальных дистанций, на которую они отстоят от семпла, возможно есть оптимизация - перебираются лишь 1 раз и формируется упорядоченное представление вершин для каждой прогнозируемой. Поэтому при прибавлении хотя бы одного семпла во всей обучающей выборке, работы прибавляется на O(k) или на O(1) и O(m), если нам нужно прогнозировать m элементов в случае оптимизации. Как только известны обучающие семплы-соседи, прогнозируемому семплу сопоставляем метку того класса, которому соответствует большее кол-во семплов-соседей. На рисунке это черная метка [3].

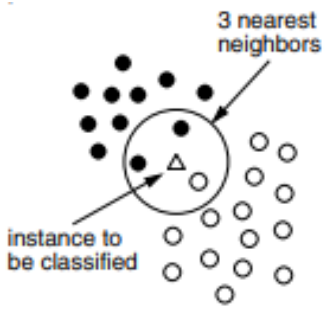

Рисунок 11 – Демонстрация основного принципа работы алгоритма

Преимуществом KNN является мгновенная адаптация под новые обучающие данные. Недостатком являются вычислительные затраты, которые линейно возрастают с ростом обучающих семплов, о чем сказано подробнее выше [6].

Персептрон

В данной работе используется не классическая версия, а модификация стохастического град. спуска с соотв. целевой функцией, пороговой функцией и только с настройками по умолчанию. Алгоритм работает с потенциально линейно сепарабельными данными.

Целевая функция потерь:

- классический алгоритм: 1-0 loss, но эта функция не дифференцируемая, а потом не может использоваться с градиентом,
- адаптация для стохастического градиента: «hinge», рассмотренная в теме о SVM (см. рисунок 12).

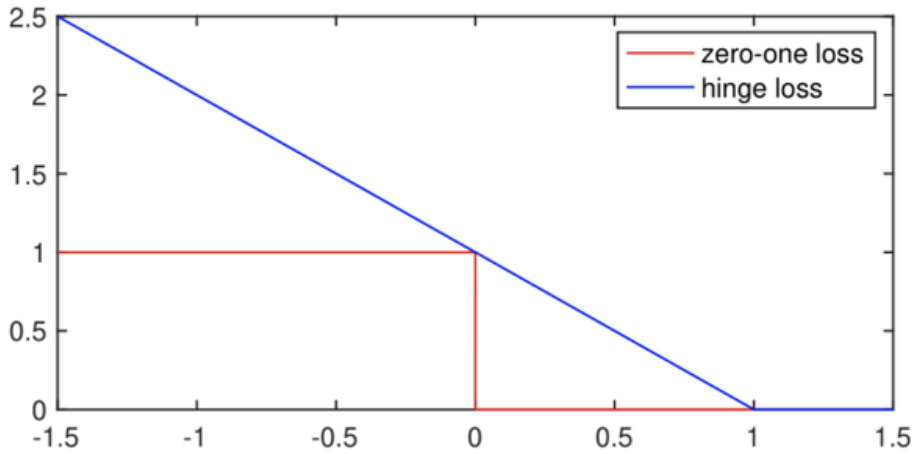

Рисунок 12 – Сравнение функций потерь классического и адаптированного алгоритма

Функция активации, совмещенная с пороговой (строго говоря, у персептрона есть только функция пороговая, а функции активации нет вовсе):

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases} \quad (19)$$

Выход может быть представлен как «1» или «0». Он также может быть представлен как «1» или «-1» в зависимости от того, какая функция активации используется. $w * x + b$ (20) – уравнение гиперплоскости.

Особенностью классического алгоритма является тот факт, что корректировка весов происходит на основе работы ступенчатой функции, а не

вычисленного значения взвешенной суммы. Однако в данном случае используется градиент для корректировки [6].

В данной работе используются только настройки по умолчанию.

1.3 Выбор инструментов разработки ПО с использованием методов МО

Для разработки исследовательского ПО был справедливо выбран ЯП Python, являющийся самым популярным языком для решения задач в области МО. В паре с ним используется наиболее известная и популярная библиотека инструментов и алгоритмов МО scikit-learn. Также используются библиотеки, обеспечивающие набор базовых алгоритмов и специальных структур данных для организации процессов МО и обработки данных: NumPy и Pandas. Для предобработки текста используется библиотека NTLK. И библиотеки, и ЯП базируются на программной платформе научных программных инструментов Anaconda 2020.02, которая позволяет быстро развёртывать и обновлять подключаемые библиотеки и инструментарий разработки. Средой разработки (IDE) выбрана Visual Studio 2019 от компании Microsoft, которая осуществляет качественную поддержку ЯП Python и позволяет быстро работать с программным кодом, осуществляя отладку, переименования и замены, автозаполнение кода, рефакторинг и подсветку спорного или откровенно неправильного синтаксически кода. Для фиксации процесса разработки и контроля версий использовался сервис GitHub компании Microsoft.

1.4 Поиск наборов данных вредоносного спама и безвредных сообщений для обучения и тестирования модели МО

Было отобрано 3 крупных датасета, отличающихся друг от друга по-своему:

- датасет конкурса соревновательной платформы Kaggle от 2017 года, содержащий семплы (5695 семплов) в виде сообщений электронной почты: K2017_Email (краткое наименование),
- датасет SMS-сообщений (5161 семплов) конкурса Kaggle: K2016_SMS; характерная особенность: SMS-ки характеризуются малой длинной, повышенным кол-вом ошибочно написанных слов и специфическим сленгом, что усложняет обработку и обнаружения спама.
- датасет (4991 семплов), собранный предположительно американской энергетической компанией Enron: E_Email; характерная особенность: старые сообщения электронной почты, до 2005 года.

Крайне важно проводить испытания на разных наборах, в том числе смешивая их, это позволяет более объективно оценить эффективность алгоритмов МО.

Примеры сообщений (K2017_Email):

- спам:
  - «security alert - confirm your national credit union information - - >».
  - «want to accept credit cards ? 126432211 aredit cpproved no cecks do it now 126432211».
- не спам:
  - «iv kirstee hewitt fifth floor se 5005 intervew schedule 17 . 30 - 18 . 00 vince kaminski & anjam ahmad 18 . 00 - 18 . 30 ben parsons 18 . 30 - 19 . 00 stephen leppard».
  - «password login kkindal password marketcredit!»

2 КОНСТРУКТОРСКАЯ ЧАСТЬ

В данной части рассматриваются алгоритмы, методики и исследования, сделанные в рамках данной работы для достижения цели. В частности, ведётся создание методики обнаружения.

2.1 Формулировка математической постановки задачи обнаружения вредоносного спама

Исходные данные и операции:

- $D = \{d_1, .., d_z\}$ – множество сообщений/документов (в исходном виде, без обработок).
- $Y = \{y_1, .., y_z\}$ - сопоставленные каждому семплу истинные метки классов, $y_i \in C$.
- $C = \{0,1\}$ – множество классов, где 0-не спам,1-спам.
- $P: D \to D'$ – алгоритм предобработки датасета. D'– предобработанный датасет, в котором каждое сообщение (семпл) представлено списком обработанных слов (подход «bag of words»).
- $fe_i: D' \to X$ – алгоритм извлечения признаков (извлечение n-грамм и расчёт числовой характеристики каждой из них).
- X – датасет после извлечения признаков, где каждое сообщение представлено числовым вектором в таблице признаков, др. словами: X – матрица размером $[z \text{ x } n]$.
- В процессе кросс-валидации датасет многократно разбивается на обучающую и валидационную выборки: $X_{train} \subset X$, $X_{valid} \subset X$, $X_{train} \cap X_{valid} = \emptyset$. Им соответствуют Y_{train}, Y_{valid}, ${D'}_{train}, {D'}_{valid}$ и D_{train}, D_{valid}.
- $D_{splits} = \{(D_{train_i}, D_{valid_i})\}$ – множество разбиений исходного набора сообщений. Аналогично определяются множества ${D'}_{splits}$ и X_{splits}.

- $F = \{f_1, .., f_n\}$ – множество извлекаемых признаков.
- $A = \{a_1, .., a_l\}$ – множество алгоритмов обнаружения, где $a_i: X \to Y^{pred}$ – алгоритм обнаружения, где $Y^{pred} = \{y_1, .., y_z\}, y_i \in C$ [2].

Алгоритмы извлечения признаков работают по разным принципам:

- варианты извлекаемых признаков:
  - униграммы (1 признак – 1 слово, n слов – n признаков),
  - униграммы + биграммы (на основе 2 слов получаем 3 признака: 2 слова по отдельности и их пару),
  - биграммы (только пары слов),
  - униграммы + биграммы + тернограммы (на основе 3 слов получаем 6 признаков: 3 слова по отдельности, 2 пары, 1 тройка слов).
- варианты вычисления значений признаков (для n-граммы g):
  - $counts(g)$ – частность = кол-ву присутствия в семпле конкретного признака,
  - $tf(g) = \frac{counts(g)}{s}$ (21), где s – суммарная частотность признаков,
  - $tf - idf(g) = tf * idf$ (22), где $idf(g) = log\frac{m}{1+df(g)}$ (23), $df(g)$-кол-во семплов, в которых встречается n-грамма g (др. словами, idf – корректор значения признака в контексте популярности в рамках датасета/языка).

Получаем множество различных способов извлечения признаков:

$$FE = fe_{counts} \cup \{fe_{tf}, fe_{tf-idf}\} \text{ x } \{1 - grams, (1,2) - grams, (1,3) - grams, 2 - grams\}$$

Решение задачи представляет собой поиск такой модели, для которой значения метрик качества обнаружения (функция q_d) и производительности (функция q_p) для заданного датасета D оптимальны. При использовании кросс-

валидации в рамках датасета мы получаем множество разбиений D_{splits} на обучающую и валидационную выборки, в таком случае необходимо рассматривать метрики как случайные непрерывные величины (в теории, на практике же можно поспорить о дискретности, поскольку имеем датасеты ограниченного размера и метрики при этом не будут бесконечными дробями никогда), поэтому итоговым значением каждой является её математическое ожидание.

В результате получаем следующую формулировку задачи:

$$\begin{cases} \mathrm{M}\left[q_d\left(D_{splits},\ P, fe_i, a_j\right)\right] \rightarrow max_{fe_i\,\epsilon\,FE, a_j\,\epsilon\,A} \\ \mathrm{M}\left[q_p\left(D_{splits}, P, fe_i, a_j\right)\right] \rightarrow min_{fe_i\,\epsilon\,FE, a_j\,\epsilon\,A} \end{cases} \quad (24)$$

2.2 Разработка и использование алгоритма поиска лучших комбинаций найденных алгоритмов МО для обнаружения вредоносного спама

Данная работа построена целиком и полностью на данном алгоритме, который уникален своей бескомпромиссностью - проверяет все возможные комбинации алгоритмов и обеспечивает ранжирование по метрикам. Данный подход позволяет получить неожиданные результаты и надёжно проверить теоретические предположения насчет сочетаний алгоритмов.

Цель комбинирования (агрегирования) алгоритмов-одиночек (обозначаются в работе аббревиатурой «SA») – улучшить их результаты по-отдельности.

2.2.1 Типы комбинаций алгоритмов

Любые комбинации алгоритмов есть смысл для удобства разделить на тривиальные и усложнённые (например, стекинг с логистическим суммированием на последнем слое).

Тривиальные комбинации

Это комбинации, в которых результаты прогнозирования алгоритмов на одних и тех же данных агрегируются каким-то законом в итоговый результат - прогноз.

Максимальная длина комбинаций – 4 в большинстве случаев. Это обусловлено экономией времени вычислений и недостаточной целесообразностью – вряд ли удастся получить качество обнаружения значительно более высокое и один тест подтверждает это. И хотя всё же можно было проверить это подробнее, многопоточный алгоритм валидации разрабатывался, но не был доведен до финальной реализации, поэтому массового тестирования длинных комбинаций не проводилось.

Виды тривиальных комбинаций:

- Дизъюнктивные (обозначаются знаком +, аббревиатурой «DC»): если хотя бы 1 алгоритм классифицирует семпл как спам, значит комбинация будет считать семпл спамом.
- Конъюнктивные (обозначаются знаком *, аббревиатурой «CC»): только если все алгоритмы классифицируют как спам, комбинация будет считать семпл спамом.
- Мажоритарные (обозначаются знаком ↔, аббревиатурой «MC»): позволяют избежать переобучения классификаторов и лучше обобщить представленные семплы простым способом – обеспечить голосование за итоговый результат, в данном случае будет поставлен тот вердикт, за который выступает большинство классификаторов. Не генерируются на основе ансамблевых алгоритмов МО.

Пусть имеется 2 алгоритма в комбинации, тогда с помощью диаграмм Эйлера можно наглядно продемонстрировать характер работы дизъюнктивной и конъюнктивной комбинаций (рисунок 13).

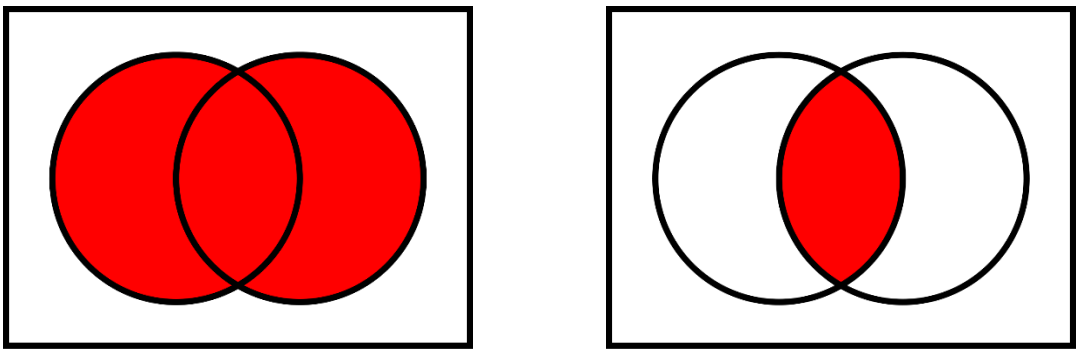

Рисунок 13 – Дизъюнкция и конъюнкция

Можно наблюдать следующие особенности деятельности типов комбинаций (полужирным шрифтом выделены преимущества):

- дизъюнктивные:
  - склонны к понижению precision
  - **склонны к recall**
  - алгоритмы дополняют друг друга путем добавления обнаруженных семплов в общий вклад обнаружения, однако необходима повышенная аккуратность алгоритмов (низкое значение метрики precision)
- конъюнктивные:
  - **склонны к повышению precision**
  - склонны к понижению recall
  - алгоритмы должны обнаруживать как можно больше семплов (демонстрируя пониженную аккуратность), в целом происходит совместное исключение FP-вердиктов и улучшение суммарной метрики качества (f1).

Усложнённые комбинации

В данной работе была реализована генерация и валидация следующих типов таких комбинаций:

- Бэггинговые (к голосованию добавлено варьирование обучающих выборок и признаков для каждого алгоритма ансамбля), обозначаются аббревиатурой «BAGC».

Бэггинговые добавляют к голосованию (как в мажоритарных) принцип бутстрапа (bootstrap): каждый классификатор (их 10, для экономии вычислительных ресурсов, это вызвано невозможностью включить штатное распараллеливание из-за конструктивных недостатков библиотеки scikit-learn) работает со случайным подмножеством семплов (рассматривает задачу под несколько иным углом; максимально 95% случайных семплов достаётся каждому алгоритму в текущей реализации).

Оба этих типа комбинаций строятся на сокращённом списке алгоритмов, нежели тривиальные комбинации:

- исключены ансамблевые алгоритмы (данные комбинации и так строят ансамбли, а необходимость строить ансамбли ансамблей выглядит избыточной),
- исключены алгоритмы со стабильно недостаточно высокими precision или recall (возможно, есть смысл проверить такие комбинации в рамках более подробного исследования, однако в связи с отсутствием технической возможности в рамках scikit-learn распараллелить работу комбинаций и недостаточной перспективности, была проведена такая оптимизация).

2.2.2 Связь исследовательского ПО с полноценным анти-спамом

Для обучения и прогнозирования в исследовательских целях применяют следующие типы датасетов:

- тренировочные,
- валидационные: как правило, тренировочный и валидационный датасеты являются частями одного общего датасета, др. словами: валидация – это облегчённое тестирование, которое позволяет оценить большинство

особенностей качества модели, но не все, во время валидации производится составление словаря и извлечение признаков на базе всего датасета (потенциально мы можем обнаруживать лишь семплы данного датасета), не только его тренировочной части, таким образом любой признак семпла из валидационного набора будет гарантированно использован алгоритмами МО),

- тестовые: модель прогнозирует неизвестные ранее данные, это моделирует работу полноценного (реального) анти-спама, когда случайное сообщение из трафика попадает на анализ и должно быть классифицировано, при этом модель может извлечь не все признаки из сообщения, которые оно потенциально содержит, поскольку словарь создавался универсальным и за основу не брался какой-либо конкретный датасет.

В данной работе речь идёт лишь о первых 2-ух типах, потому что для исследований на данном этапе этого достаточно (**разрабатывается предварительная методика обнаружения, хоть и для анти-спам системы, но всё же на основе кросс-валидационных исследований**), исследования на тестовых датасетах были бы следующим закономерным этапом. Работа с тестовыми датасетами может происходить лишь после исследований на валидационных наборах (фолдах), т.е. данный этап нельзя игнорировать.

Таким образом, есть единственный коренной фактор, мешающий качественному обнаружению спама – чрезмерно разряженный вектор семпла в связи с недостатком признаков, которые описывают его и в случае с полноценной анти-спам системой это явление сильнее выражено, поскольку не только не все возможные признаки (из набора признаков/словаря) можно извлечь из каждого семпла (это и для кросс-валидации характерно), так ещё и в принципе неизвестны все потенциальные признаки, которые потребуется извлекать.

Чтобы в рамках кросс-валидации немного усилить этот фактор (для того, чтобы смелее экстраполировать результаты кросс-валидации на полноценную

систему), было решено не убирать весь шум из наборов данных (датасетов), подробнее об этом написано в разделе о предобработке текста.

2.2.3 Правила поиска лучших комбинаций алгоритмов

Перейдём к описанию правил:

- Как описано выше, используется кросс-валидация с учетом дисбаланса классов (выбирается соответствующий алгоритм фолдинга).
- Комбинация определяется 3 характеристиками:
    - списком алгоритмов-участников (подмножеством исходного множества алгоритмов), он характеризуется ещё и длинной,
    - правилом агрегации результатов алгоритмов,
    - (опциально) правилами организации обучения, во время которых варьируется кол-во используемых признаков, обучающие наборы данных и т.д. от алгоритма к алгоритму комбинации.
- Участниками являются как алгоритмы со стандартными настройками, пригодные для классификации текста, так и их модификации.
- В рамках каждой комбинации необходимо присутствие различных алгоритмов если не по принципу, то хотя бы по конфигурации.
- Лучшей комбинацией называется такая, которая лучшая для всех датасетов с похожими типами семплов. К комбинациям-лидерам существует ещё одно требование отбора: это высокая способность к обобщению, т.е. алгоритмическая комбинация должна обладать всеми алгоритмами с таких свойством. Поскольку данная работа исследовательская, в ней есть модификации более специфические, не так хорошо обобщающие обучающую выборку (тем не менее их на практике теоретически можно использовать адаптивно: динамически менять параметры алгоритмов на основе пре-классификации датасетов, этот процесс можно назвать

процессом настройки моделей). Если лидером стал алгоритм-одиночка, то требование аналогичное.

- Лучшая комбинация при одних условиях не может быть лучшая и в других – может быть получено несколько лучших комбинаций и необходимо определить условия применения каждой такой комбинации.
- Для поисков лучших комбинаций проводится множество кросс-валидационных тестов на разных датасетах с разными параметрами, определенными мат. постановкой. Согласно мат. постановке, осуществляется поиск лучшей комбинации на конкретном зафиксированном датасете. Однако это можно расширить до поиска лучшей комбинации на нескольких датасетах с семплами одного типа (например, семплами электронной почты) не противореча мат. постановке.
- Требования к комбинациям-кандидатам:
    - высокая способность к обобщению (не допускается конфигурирование алгоритмов под конкретные датасеты (определённые словари), например с помощью алгоритмов подбора гиперпараметров, необходимо получить высокое качество для любого датасета с семплами определённых свойств и потенциально случайного семпла с соотв. свойствами),
    - приемлемая скорость обучения и прогноза (это не очень важно, но комбинация не должна быть слишком медленной),
    - $\text{precision} \geq 0.9$, $\text{recall} \geq 0.85$ – это необходимый порог, комбинации, не прошедшие его, не считаются приемлемыми для создания модели,
    - устойчивость к выбросам – это задача сознательно наложена не на механизм предобработки датасета обучения (такая функция есть, но фильтрует семплы по длине умеренно), а на комбинацию алгоритмов – по той причине, что в реальности семпл может содержать состав слов, по частоте не сбалансированный (важно понимать, что на комбинацию не возлагается задача устойчивости к любой

несбалансированности, только к умеренной, нужны дополнительные механизмы контроля этих случаев),

- модификации алгоритмов должны составляться в том числе с использованием изменённых значений ранее использованных гиперпараметров (в др. модификациях алгоритма в составе др. комбинаций) для установления влияния каждого параметра при отрицательной или положительной динамике после перенастройки,

- Важно то, что для тривиальных комбинаций используется один и тот же список алгоритмов. Обычно, важно обосновывать выбранный путь практических исследований теоретически, определять рамки экспериментов, однако в случае с работой алгоритмов МО остаётся много неочевидного, потому что как минимум многое зависит от наборов данных, а не только от характера работы алгоритмов. Можно сделать индивидуальные списки алгоритмов для каждого типа тривиальных комбинаций, обратившись к следующим особенностям комбинаций:
  - алгоритмы для конъюнкции должны иметь высокий TP-rate на используемых датасетах (без этого нет смысла пытаться уменьшить кол-во FP-случаев, потому что анти-спам должен иметь высокую полноту (recall) в любом случае),
  - алгоритмы для дизъюнкции должны иметь низкий FP-rate на используемых датасетах (иначе с увеличением длины комбинации FP-rate будет недопустимо расти).

  Алгоритм поиска лучших комбинаций в любом случае позволяет получить интересные данные о сочетаемости различных алгоритмов и проверяет все возможные сочетания – в этом его основная ценность. В итоге, общий список алгоритмов содержит как отлично сочетающиеся алгоритмы, так и не очень.

2.2.4 Разработка алгоритма поиска лучших комбинаций

Алгоритм задуман как валидатор алгоритмов, который не только непосредственно валидирует алгоритмы, он также фильтрует результаты и выводит их. Способен проверять как поданые на вход алгоритмы МО без генерации комбинаций, этому режиму работы соответствуют нижеперечисленные пункты №: 3, 4,…, так и используя функцию генерации подмножеств алгоритмов, что и является наиболее важной его функцией.

Поисковый алгоритм решает следующие задачи (под-алгоритмы):

1. Сгенерировать комбинации: алгоритм должен решить комбинаторную задачу формирования подмножеств множества алгоритмов, которые были поданы на вход (для каждого типа комбинаций подаётся уникальный список алгоритмов-одиночек). Получим следующий псевдокод:

   для m=1..lmax:

   найти C_a^mсочетаний алгоритмов без повторений (комбинаций длинны m)

   на множестве алгоритмов A, $|A| = a$, lmax – макс. длинна комбинаций
2. Инициализировать на основе подмножеств комбинации алгоритмов заданных типов, используя соответствующие программные классы.
3. Создать разбиения на непересекающиеся подмножества датасета (назовём их фолдами в данном контексте), используя алгоритм k-folding для кросс-валидации (на обучающих фолдах обучаем алгоритм, на валидационных вычисляем метрики качества, при этом каждую итерацию разбиения меняются обучающие фолды и валидационный фолд, подробнее об этом расписано далее).
4. Обучить каждый алгоритм в отдельности и проверить на сформированных фолдах, запоминать результаты прогнозирования, таким образом получим сопоставление каждому алгоритму на конкретном разбиении фолдов результатов прогнозирования, также получим метрики производительности (время прогнозирования, время обучения) и метрики качества обнаружения для алгоритмов-одиночек.

5. Скомбинировать результаты прогнозирования на фолдах алгоритмов-одиночек (см. п.3) в соответствии с типом тривиальной комбинации, если валидируется усложнённая комбинация, то результаты алгоритмов-одиночек не используются, идёт процесс обучения и валидации на фолдах.
6. Вычислить различные оценочные метрики для каждой комбинации алгоритмов, при этом комбинации следует сопоставить среднее значение каждой метрики по всем разбиениям фолдов. Любая функция вычисления метрики качества обнаружения принимает на вход 2 аргумента: результаты прогнозирования (метки класса, сопоставленные каждому семплу) и правильные метки на каждый семпл.
7. Отфильтровать и отсортировать результаты.
8. Экспортировать результаты.

Оценим под-алгоритм, отвечающий за генерацию комбинаций (п.1). Пусть величина a – количество алгоритмов, m – количество мест в комбинации (длина комбинации). Кол-во подмножеств алгоритмов длинны m (сочетания без повторений): $C_a^m = \frac{a!}{(a-m)!m!}$ (25). Кол-во перебираемых алгоритмом вариантов: $T = \sum_{m=1}^{lmax} C_a^m$ (26).

2.2.5 Методика разбиения датасета (k-folding)

Подробнее рассмотрим проблему несбалансированности классов в датасете (дисбаланс в датасете K2017_Email показан на рисунке 14). Популярна ситуация, когда датасет содержит дисбаланс классов и возникает 4 возможных решения [7]:

1) уменьшить кол-во семплов избыточного класса до соотношения 1:1, 2:1 (undersampling), однако в таком случае уменьшается словарь и точность вычисленных признаков снижается, что повлечёт за собой снижение качества модели, так можно поступать в том случае, если семплов с избыточное количество,

2) увеличить кол-во семплов миноритарного класса – для этого существуют различные техники (например, дублирование семплов), не всегда приводящие к улучшению ситуации,

3) добавить семплы миноритарного класса из других датасетов (так, чтобы алгоритм предобработки был совместим, что далеко не всегда легко осуществимо),

4) оставить как есть (выбран данный вариант для всех датасетов, поскольку дисбаланс не существенный – 3-6-кратный, в то время как абсолютно критичным является 10-тикратный), но использовать специальные алгоритмы кросс-валидации и ориентироваться на лишь некоторые метрики, способные эффективно оценивать качество модели и в условиях дисбаланса классов.

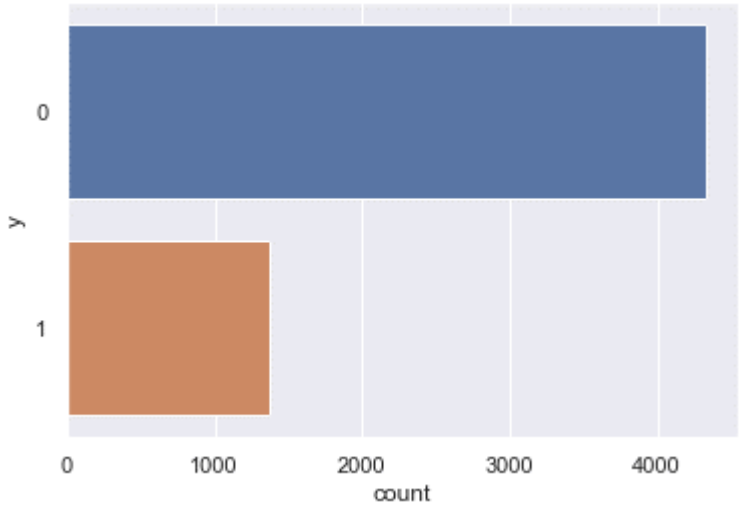

Рисунок 14 – Дисбаланс классов датасета K2017_Email

То, что модель будет предвзята к чистым сообщениям не является проблемой, главное, чтобы предвзятость не была слишком существенной и применялись подходы по смягчению предвзятости на уровне алгоритмов МО (введение для семплов весов, которые обратнопропорциональны популярности класса).

Базовый алгоритм k-folding-а [8] предусматривает разбиение датасета на k фолдов, k-1 из которых обучающие (они, в свою очередь, должны содержать не менее 3 тысяч семплов в обучающих фолдах для обеспечения наилучшего

качества классификации текста, см. рисунок 15), а k-й – валидационный. Можно также k-й валидационный фолд скомпоновать из случайных семплов. Проблема в том, что при таком подходе дисбаланс классов в исходном датасете создаёт некорректные фолды, содержащие разный баланс классов, что серьёзно может испортить качество оценки моделей.

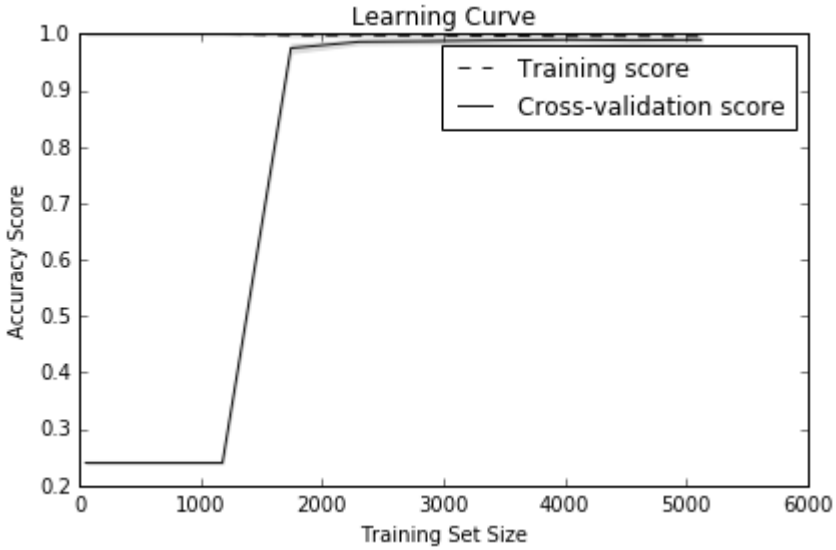

Рисунок 15 – Увеличение качества обнаружения спама (K2017_Email) с увеличением размера обучающей выборки

Требуется учесть долю классов в датасете и сохранить её в каждом фолде. Существует 2 подхода: алгоритмы, реализованные в классах библиотеки scikit-learn, логика этих алгоритмов отражена на рисунках 16 и 17.

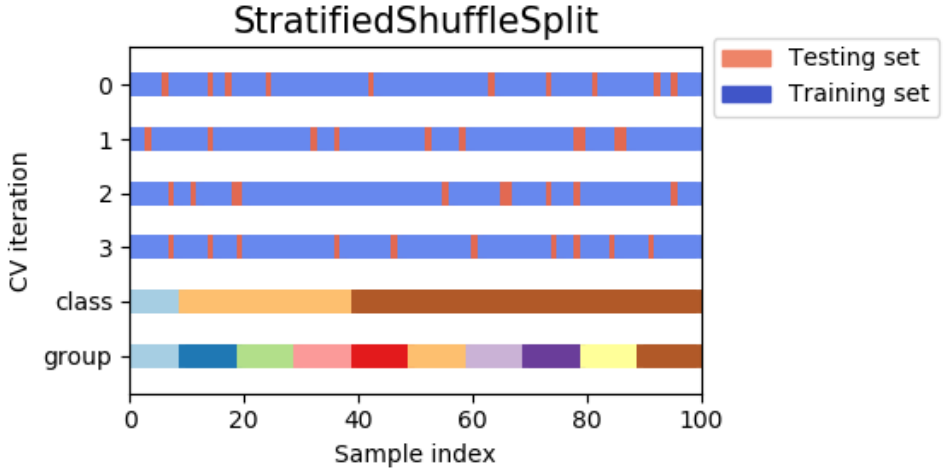

Рисунок 16 – Фолды заполняем случайными семплами каждого класса

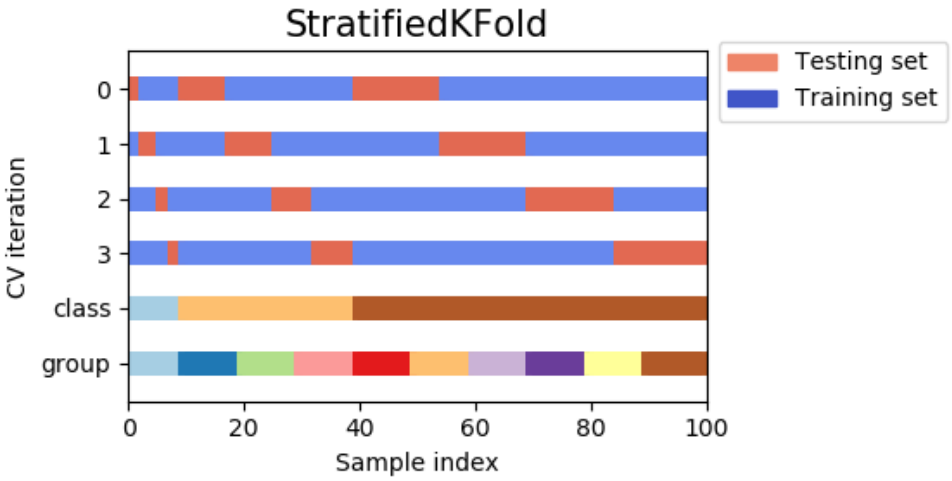

Рисунок 17 – Фолды заполняем последовательно расположенными семплами каждого класса

В данной работе использован 2 подход, поскольку таким образом низка вероятность повторения семплов от разбиения к разбиению.

2.2.6 Метрики качества алгоритмов

Метрики обнаружения

Основной оценочной метрикой выбрана F1. Данная метрика эффективна по той причине, что учитывает false negative (fn) и false positive (fp) – не обнаруженный спам и ложно обнаруженный соответственно, а не только правильные вердикты и позволяет за счет управляющего коэффициента усилить/уменьшить важность recall, при beta=1 recall и precision одинаково важны – полнота и аккуратность обнаружения равноценны для нас, однако есть смысл сделать полноту важнее аккуратности в ряде эксплуатационных ситуаций. F1 устойчива в условиях дисбаланса классов и её значение снижается быстрее, чем значение метрики accuracy (совершенно не устойчивой к дисбалансу) в том случае, если алгоритм какие-то семплы не обнаруживает. Программа

рассчитывает и ряд др. метрик, но они носят исключительно информационный характер, но не исследовательский [2].

Суть используемых метрик:

- accuracy (точность) – сколько правильных вердиктов дала модель:

$$\text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn} \quad (27)$$

- True Positive Rate (recall) – сколько модель обнаружила реального спама относительно всего спама в датасете (полнота обнаружения):

$$\text{Recall} = \frac{tp}{tp + fn} \quad (28)$$

- Positive Predictive Value (precision) – сколько модель обнаружила реального спама относительно всех вердиктов «спам» (аккуратность обнаружения):

$$\text{Precision} = \frac{tp}{tp + fp} \quad (29)$$

- f1 (при $beta = 1$) – результат агрегирования двух выше индикаторов:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}} \quad (30)$$

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \quad (31)$$

Отступление об ошибках 1 и 2 родов

В бинарной классификации, в машинном обучении и области Data Science такие не используются часто или вообще не используются, используются повсеместно более наглядные аналоги: recall (аналог ошибки 2 рода, O2=1-recall), precision (O1 прямым образом не выразить, потому что это не аналог для O1). O1 прямым образом не выразить, потому что зависит её величина от fp и tn, ни precision, ни recall не используют эти метрики. Если сравнить O1 и precision,

становится очевидным, почему O1 часто не подходит: полезнее знать соотношение правильных вердиктов «спам» ко всем таковым (включая неправильные), нежели пытаться соотносить потенциальные вердикты «не спам» (fp) и фактические (tn): $fp/(fp + tn)$, так мы ответим на вопрос «какова доля ошибочно обнаруженных легитимных семплов из всех легитимных», это не самая ценная метрика в контексте текущей задачи, важнее сделать метрику зависимой от кол-ва семплов в классе «спам», ведь проводится оценка его экспертизы и это более чувствительная метрика из-за более малого значения знаменателя [3].

Иногда под ошибками O2 имеют в виду fp-rate (fp/общее кол-во семплов), а под O1 fn-rate. fp-rate и fn-rate не достаточно для оценки качества обнаружения, поскольку в знаменателе у них все возможные семплы и 1 такие метрики не могут быть равны в принципе.

Метрики производительности

- Время обучения (сек, среднее на фолд) – вычисляется фиксацией меток времени перед вызовом метода обучения и после, вычисляется дельта между ними.
- Время прогнозирования (сек, среднее на фолд) – по аналогии.

Важной особенностью полученных значений является то, для их вычисления задействуется функция фиксации текущего времени системы в секундах, а не операций процессора, поэтому полученные данные не самые точные (погрешность, обнаруженная на практике, до 2 мс при соблюдении организационных условий), однако этого достаточно при соблюдении организационных условиях тестирования для дифференцирования алгоритмов обнаружения по временным критериям. Подсчет операций позволил бы на системах с разными процессорами получать одинаковые объективные цифры и тем самым исключить любое внешнее влияние на конечные цифры. Процессор адаптивно задействует свой вычислительный ресурс, поэтому конечное время

работы алгоритма может быть разным, есть и др. внешние факторы. При исследованиях было соблюдено организационное условие – при тестировании алгоритмов-одиночек (на основе которых потом формировались данные о производительности комбинаций) остальные программы в ОС не должны использовать её ресурсы, чтобы почти все ресурсы достались исследовательскому ПО.

2.2.7 Методика предобработки текста

Обнаружение спама базируется на NLP. NLP (Natural Language Processing, обработка естественного языка) – это область вычислительных наук, которая изучает взаимодействие между компьютером и человеком. Техники, разработанные NLP, используются для анализа текста, предоставляя компьютерам возможность понять человеческий язык. NLP используется в автоматическом суммировании и анализе тональности высказываний. Одной из основных составляющих NLP являются инструментарий по предобработке текста и извлечению из него признаков – на этом строится переход от речевых исходных данных к математическим конструкциям, понятным алгоритмам МО.

Методика представляет собой решение двух задач в области предобработки [9]:

- удаление шума и нежелательных элементов документов,
- нормализация текста.

В работе используются алгоритмы МО на базе подхода «Bag of words» («мешок слов»), который подразумевает, что каждый семпл (сообщение) описывается бессвязным набором присутствующих в датасете слов или групп слов (n-граммы). Самое простое значение, которое может принять признак – кол-во присутствий слова-признака в рассматриваемом семпле.

Именно на этапе предобработки исходные сообщения будут представлены в виде «мешков слов».

Данная методика использует в основном базовые популярные подходы для достижения необходимого качества представления данных, которые потом используется для кросс-валидации алгоритмов МО.

Оптимальной методикой по сложности реализации и эффективности является следующая последовательность действий с входными документами (функционал предобработка preprocessor1):

1. Удаление семплов-дубликатов.
2. Удаление слишком длинных и коротких семплов относительно медианы. Такая мера позволяет избежать переобучения, уменьшая шумность входных данных. Однако шумность не была устранена полностью. Причина: в идеале обучать алгоритмы классификации необходимо на очищенных качественный наборах, а валидировать и тестировать на разных, однако обеспечить это в рамках кросс-валидации чревато дополнительными сложностями и смысла в этом весомого нет, поэтому при предобработке было просто уменьшено кол-во шума в контексте длин сообщений, но полностью он не был устранён, важнее проверить способность алгоритмов противостоять шумности валидационных семплов. Если и решать вопрос с шумностью, то рассуждать следует так: нет возможности избежать обнаружения ряда потенциально неудачных семплов, поэтому необходимо под каждую длину разработать свою модель, тем более, что полноценный анти-спам имеет больших риск столкнуться с подобными семплами (в таком случае наборы признаков будут одинаковыми, но обучение и обнаружение ожидаются более качественными).
3. Удаление стоп-слов – с помощью NTLK.
4. Удаление пунктуации и неизвестных символов.
5. Группировка специфических лексем, уникальность которых не имеет значения (во избежание наполнения словаря избыточными n-граммами и повышения ценности такой информации для самих моделей):
    a. номеров телефонов (заменяются на слово «phonenumbr»),

b. любых других номеров («numbr»),
c. адресов эл. почты («emailaddr»),
d. URL («httpaddr»),
e. денежные символы («moneysymb»).

6. Стемминг Портера – с помощью NTLK. Это алгоритм нормализации, он не использует базы основ слов, а лишь, применяя последовательно ряд правил, отсекает окончания и суффиксы, основываясь на особенностях языка, в связи с чем работает быстро, но не всегда безошибочно [9].
7. Представление всех документов в виде семплов по схеме «Bag of words».

Примеры предобработки некоторых небольших (для компактности) семплов исходного (raw) датасета в формате «до и после» показаны в таблице 1.

На этих примерах наглядно виден один из подходов злоумышленников – намеренные ошибки в словах (например, семпл №2), что сбивает с толку простой механизм извлечения, применённый в работе, в итоге в словарь попадают слова с опечатками как совершенно новые, хотя должен применяться механизм обнаружения истинных слов, написанных с ошибками.

Интересны также ситуации преобразования вида «08452810075over18's → numbrovernumbr». Может создаться впечатление, что предобарботка проведена небрежно, ведь цифры смешаны с буквами и надо разделить это, но склеенные слова нужно рассматривать как отдельные лишь тогда, когда все слова написаны правильно и склеены для запутывания детектора или из-за ошибки, однако в данном случае лучше рассматривать именно всю склейку целиком, желательно обобщив написанные цифры (предобработчик блок цифр разменяет на «numbr») – предобработчик в данной работе всегда поступает так, что тоже не всегда верно, почему - уже сказано выше.

Таблица 1 – Семплы до и после предоботки

K2017_Email	
"Subject: security alert - confirm your national credit union information - - > "	secur alert confirm nation credit union inform
"Subject: want to accept credit cards ? 126432211 aredit cpproved no cecks do it now 126432211"	want accept credit card numbr aredit cpprove ceck numbr
"Subject: get your babies diapers bill paid for a year . your family could definately use this , now go . mjirartt"	get babi diaper bill paid year famili could defin use go mjirartt
K2016_SMS	
Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...	go jurong point crazi avail bugi n great world la e buffet cine got amor wat
Ok lar... Joking wif u oni...,	ok lar joke wif u oni
Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's	free entri numbr wkli comp win fa cup final tkt numbrst may numbr text fa numbr receiv entri question std txt rate c appli numbrovernumbr

В последнем семпле слово «tkts» преобразовано в «tkt», это связано с не всегда качественной работой лемматизации Стемминга Портера: в данном случае он просто посчитал s за окончание-указание множественного числа предметов и удалил его. Походим образом стемминг поступает со словом «receive», превращая его в лемму «receiv». Это не является критическим недостатком, поскольку важнее обобщить все слова к одной конкретной лемме и для алгоритма, в сущности, неважно как она читается и правильная ли она в принципе.

Рассмотрим баланс классов и длины семплов каждого датасета (для каждого класса по-отдельности, это важно, чтобы оценить равноценны ли

семплы по длинам хотя бы ориентировочно – в каждом классе должны быть семплы разного соизмеримого кол-ва признаков и желательно, чтобы и доля в каждом классе сообщений каждого диапазона длин была похожа; см. рисунок 18, рисунок 19, рисунок 20) после предобработки, поскольку они являются одними из важнейших характеристик датасета.

Распределения многих датасетов по длинам семплов позволяет утверждать однозначно следующее:

- имеется единичный шум – во время предобработки кол-во шума было уменьшено, но он не был устранён полностью намеренно по уже описанным ранее причинам,
- не всегда подчиняется хотя бы ориентировочно нормальному распределению – в идеале медиана и среднее также должны совпадать, но допускается несущественное расхождение (медиана устойчива к выбросам, среднее – нет, при наличии слишком шумных данных, разница между средним и медианой будет больше),
- в целом распределения от класса к классу похожи в рамках датасета (это соответствие важно, чтобы ни у одного класса не было преимуществ/недостатков).

K2017_Email

```
Соотношение классов:
0    0.759789
1    0.240211
Длины семплов:
-- Статистики для класса: spam
---- max: 2581
---- min: 1
---- mean: 128.0
---- median: 64.0
-- Статистики для класса: ham
---- max: 4345
---- min: 1
---- mean: 162.0
---- median: 112.0
```

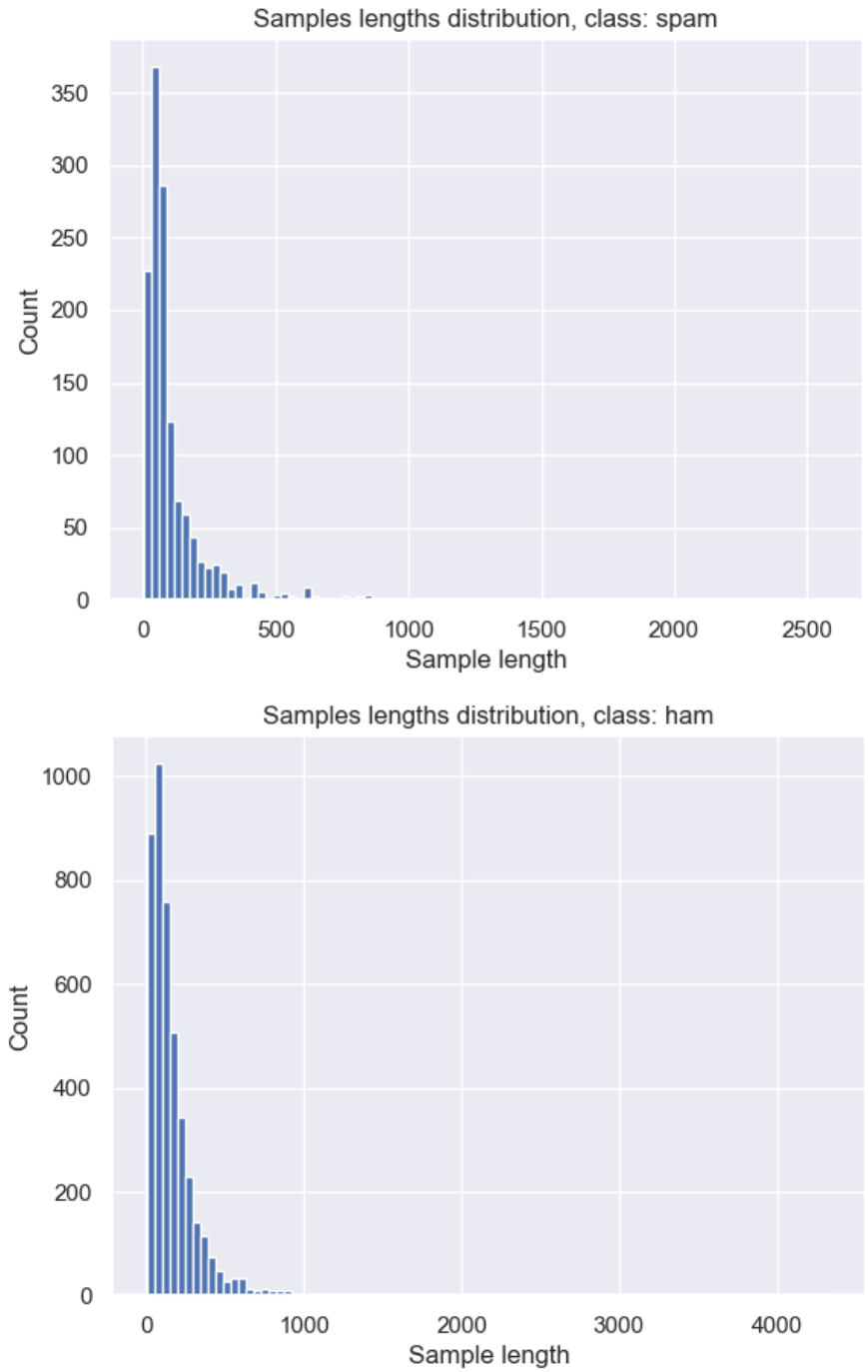

Рисунок 18 – Распределения длин семплов датасета K2017_Email

Примечание: может создаться впечатление, что гистограмма зря разнесена по такому широкому диапазону значений длин, однако это из-за наличия очень немногих семплов с соответствующими длинами, просто масштаб не позволяет отобразить четко на гистограмме популярность таких семплов.

E_Email

```
Соотношение классов:
0    0.707473
1    0.292527
Длины семплов:
-- Статистики для класса: spam
---- max: 2591
---- min: 1
---- mean: 135.0
---- median: 64.5
-- Статистики для класса: ham
---- max: 3474
---- min: 1
---- mean: 107.0
---- median: 59.0
```

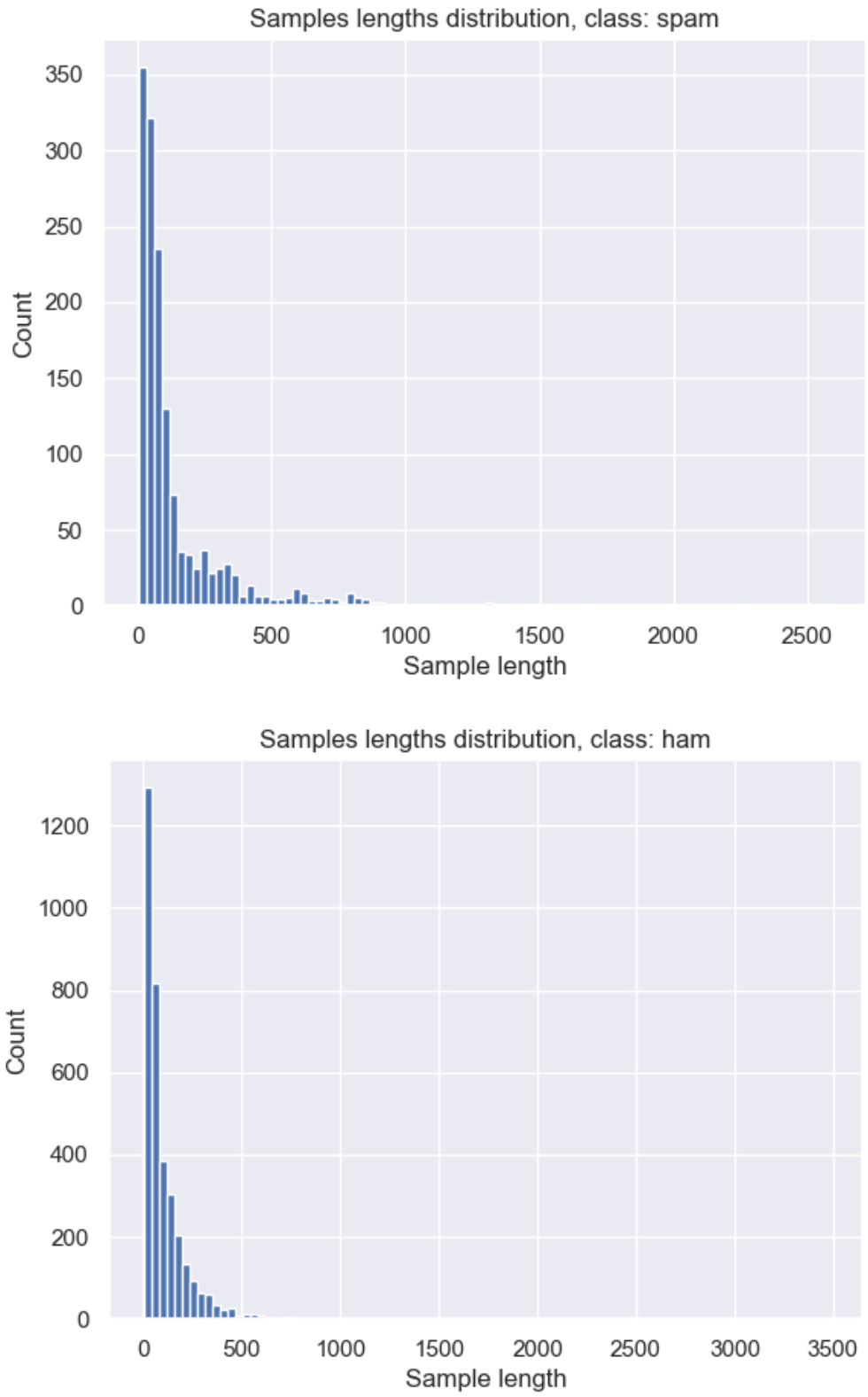

Рисунок 19 – Распределения длин семплов датасета E_Email

K2016_SMS

```
Соотношение классов:
0    0.873474
1    0.126526
Длины семплов:
-- Статистики для класса: spam
---- max: 31
---- min: 2
---- mean: 17.0
---- median: 18.0
-- Статистики для класса: ham
---- max: 78
---- min: 1
---- mean: 8.0
---- median: 6.0
```

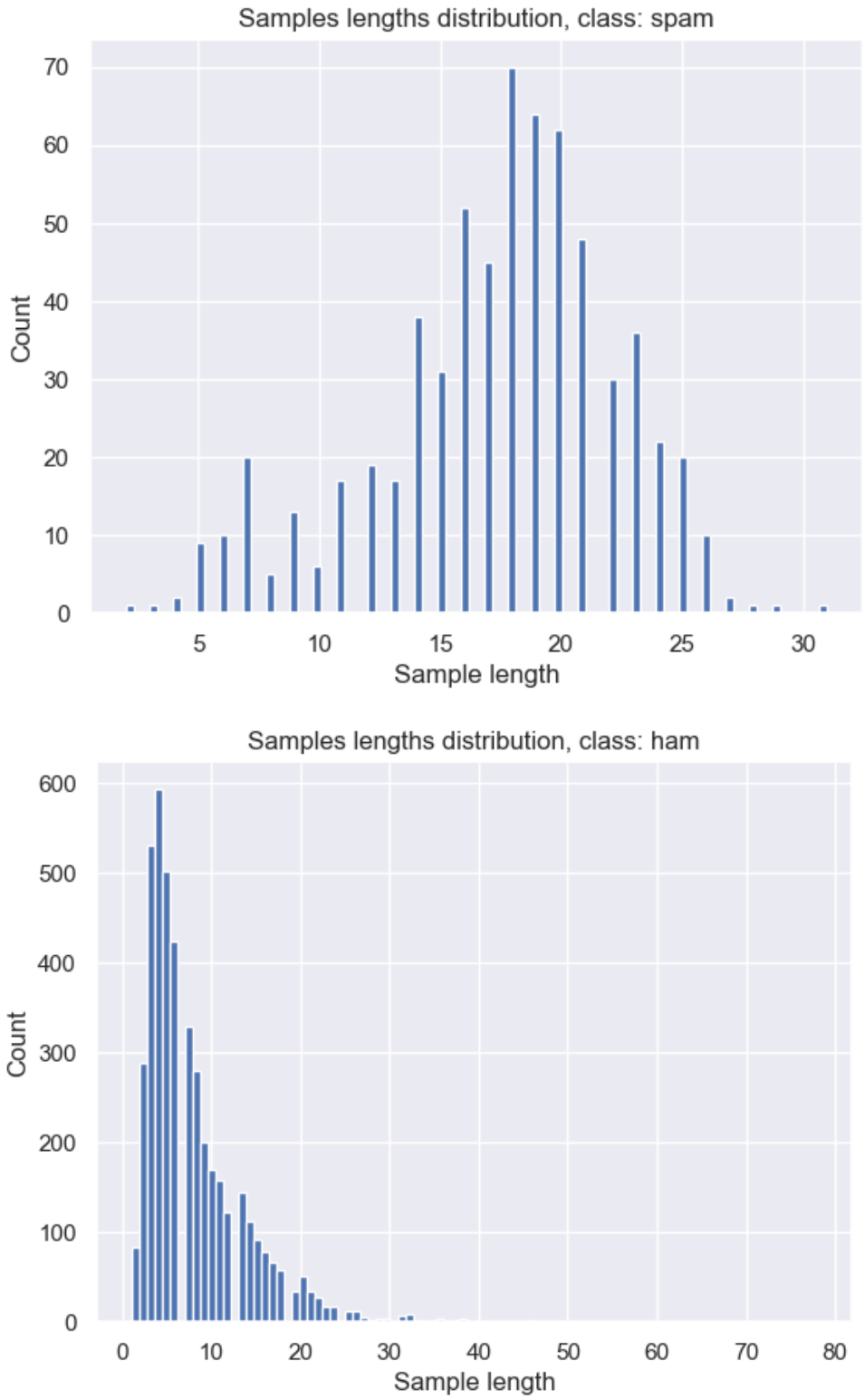

Рисунок 20 – Распределения длин семплов датасета K2016_SMS

2.2.8 Методы извлечения признаков

Алгоритмы извлечения признаков используются для двух задач (выполняются в указанном порядке над предобработанным семплом вида «bag of words»):

1. Токенизация.
2. Векторизация.

В математической постановке уже изложены варианты токенизации (типы извлекаемых n-грамм) и формулы векторизации. Подробнее рассмотрим эти подходы.

В библиотеке scikit-learn подсчет частотностей словоупотреблений выполняет класс CountVectorizer. Если мы вызовем конструктор класса CountVectorizer(ngram_range =(1,2)), то он будет извлекать не только одиночные слова, но и пары:
'Bi-grams are cool!' -> ['bi', 'grams', 'are', 'cool', 'bi grams', 'grams are', 'are cool'] (2n-1 признаков, если n-кол-во слов).

По умолчанию n-граммы представляют собой слова целиком. Однако, можно реализовать извлечение n-грамм из слов без сохранения слова самого по себе. Зачем: чтобы избавиться от лишнего шума, созданного ошибочными действиями создателя документов (семплов), если слово написано ошибочно, оно будет принято за отдельный признак. Для этого используют char_wb-анализатор, который создает n-граммы только из символов внутри слова (границы слова обозначены дополнительным пробелом в соответствующих фичах – они соответствуют пограничным буквам):
CountVectorizer(analyzer='char_wb', ngram_range=(2, 2)): ['words', 'wprds'] → [' w', 'ds', 'or', 'pr', 'rd', 's ', 'wo', 'wp']. Как видно, слова не похожи по написанию, но это одно и тоже слово в сущности, при этом набор признаков почти одинаков, за исключением признака «wp», который можно дополнительно и намеренно исключить [9].

char-анализатор, в качестве альтернативы, создает n-граммы, которые охватывают >1 слова при создании фичи:
CountVectorizer(analyzer='char', ngram_range=(5, 5)): ['jumpy fox'] → ['jumpy', 'mpy f', 'py fo', 'umpy ', 'y fox'].

Вариант, учитывающий границы слов (char_wb), особенно интересен для языков, которые используют пробелы для разделения слов, поскольку он генерирует значительно меньше шумных элементов, чем необработанный char вариант в этом случае. Для таких языков это может повысить как точность прогнозирования, так и скорость сходимости классификаторов, обученных с использованием таких функций, сохраняя при этом устойчивость к ошибкам и деривациям слов (формам слов) [10].

Конфигурация вышеуказанного класса по умолчанию токенизирует строку, извлекая слова не менее чем из 2 букв.

Подсчет словоупотреблений (такой экстрактор в проекте называется «words_counts_1») — это хорошее начало, но есть проблема: в длинных документах среднее количество словоупотреблений будет выше, чем в коротких, даже если они посвящены одной теме.

Чтобы избежать этих потенциальных несоответствий, достаточно разделить количество употреблений каждого слова в документе на общее количество слов в документе (однако если мы извлекаем биграммы и любые другие n-граммы, то эта логика не работает, при кросс-валидации экстрактор признаков сначала выделяет признаки на основе датасета, потом высчитывает частотность каждого и делит их на суммарную частотность всех признаков в документе). Этот новый признак называется tf — «Term Frequency»(частота термина).

Следующее уточнение меры tf — это снижение веса слова, которое появляется во многих документах в корпусе, и отсюда является менее информативным, чем те, которые используются только в небольшой части корпуса. Примером низко ифнормативных слов могут служить служебные слова, артикли, предлоги, союзы и т.п. Это снижение называется tf–idf, что значит

“Term Frequency times Inverse Document Frequency” (обратная частота термина) [11].

Логарифм* (idf-множитель) может быть любой, ибо нам не на важно абсолютное значение, важно отличие значений фич семплов между собой (дифференциация). Таким образом, характеристика idf сама по себе – вес слова (популярность в рамках датасета, причем idf тем больше, чем меньше популярность) и не обязательно имеет значение от 0 до 1 [12].

**log-арифм помогает сделать вес не сильно модифицирующей функцией, а скорее корректирующей.*

Необходимо упомянуть, что прорабатывались и менее популярные подходы:

- PoS (Part of Speech)-теги, сопоставляемые словам семпла, являющиеся частями речи этих слов.
- Word Embeddings и библиотека Word2Vec.

Важно в дополнение заметить, что хотя некоторую информацию о локальном позиционировании можно сохранить, извлекая более длинные n-граммы вместо отдельных слов, концепция «мешка слов» разрушает большую часть внутренней структуры документа и, следовательно, большую часть значения, которое несет эта внутренняя структура [13]. Таким образом, для решения более широкой задачи понимания естественного языка необходимо учитывать местную структуру предложений и абзацев. Есть смысл исследовать применение принципа «Word Embeddings» для представления слов «мешка» в виде взаимосвязанных по смыслу и с точки зрения речевых особенностей векторов в ЛВП, т.е. библиотеку Word2Vec, однако в данной работе от этого решено отказаться.

Покажем кол-во признаков при применении каждого токенизатора в таблице 2 ниже. Заметно, на сколько непропорционально возрастает кол-во биграмм относительно униграмм, что не удивительно.

Таблица 2 – Связь кол-ва признаков с используемыми токенизаторами.

	K2017_Email	**E_Email**	**K2016_SMS**
ngram=1,1	25609	37893	6337
ngram=1,2	318608 (+1145%)	249917	35927 (+467%)
ngram=1,3			66755
ngram=2,2			29590

2.2.9 Фильтрация и сортировка результатов

Цель фильтрации – обеспечить доказательство перспективности выбранных подходов к агрегированию алгоритмов МО путем удаления и выделения из определённых комбинаций.

Процесс фильтрации имеет 2 составляющие:

- удаление бесполезных результатов (ремуверы, от англ. «remover»),
- выделение интересующего подмножества результатов в отдельное место в выводе (хайлайтеры, от англ. «highlighter»), работает исключительно после сортировки данных по убыванию качества и работы всех ремуверов.

Данная стадия выполняется в 3 шага:

1. Работают ремуверы.
2. Сортируются результаты (алгоритм TimSort, который является оптимальным выбором для большинства ситуаций, оценка вычислительной сложности в наилучшем случае $\theta(n)$, в наихудшем $\theta(n * log(n))$, затраты памяти не превышают $\theta(n)$).
3. Работают хайлайтеры.

Используемые ремуверы:

1. Удаляет бесполезные комбинации – те комбинации, которые бесполезны в контексте необходимости комбинировать алгоритмы-одиночки, поскольку хотя бы один алгоритм-одиночка как минимум не уступает комбинации в основных метриках качества обнаружения (recall и precision).
2. Удаляет избыточные комбинации – это такие комбинации, добавление алгоритмов к которым не сопровождается улучшением одной из метрик качества обнаружения (precision или recall), если комбинация уже признана избыточной, это означает, что эта комбинация с добавлением др. алгоритмов являясь уже новой комбинацией также избыточна, а значит можно удалять целое поддерево комбинаций "под корень", а не удалять их перебором (оптимизация). Алгоритм начинает проход с комбинаций длинны 2 и постепенно увеличивает длину до максимума, как только рассматриваемая комбинация маркируется на удаление (ни одна из её метрик не лучше метрик комбинации-родителя, из которой она образовалась), запускается рекурсивных проход по всем возможным комбинациям, которые могут быть основаны на рассматриваемой (текущая комбинация рассматривается как родительская и на её основе происходит генерация всевозможных имен комбинаций) и тоже помечаются на удаление, потом все помеченные данные разом удаляются. Ниже представлен пример избыточной комбинации:

```
---PAA_I_Default <-> PAA_II_Default
['f1=0.9824', 'auc=0.99', 'acc=0.9896', 'prec=0.9738', 'rec=0.9911', 'train_time=0.0391088', 'pred_time=0.0005001']
---SGDAlg_LogLoss <-> PAA_I_Default <-> PAA_II_Default
['f1=0.9824', 'auc=0.99', 'acc=0.9896', 'prec=0.9738', 'rec=0.9911', 'train_time=0.0590127', 'pred_time=0.0012008']
```

3. Удаляет слабые комбинации – удаляются не удовлетворяющие множествам допустимых значений метрик качества обнаружения (precision: [0.9; 1], recall: [0.85; 1]) комбинации.

Используемые хайлайтеры:

1. Выделяет подмножество уникальных алгоритмов. Осуществляет проход от вершины сортированного списка вниз перебором. Уникальный алгоритм – это тот, который содержит в своём составе алгоритмов-участников хотя бы один ещё не встречавшийся выше в отсортированном списке среди алгоритмов-участников приведенных алгоритмов. Таким образом, кол-во алгоритмов в таком подмножестве не может превышать m, где m – кол-во алгоритмов-участников, из которых составлялись комбинации. Каков смысл данного подхода: каждый алгоритм-участник будет показан в той комбинации, которая показала лучшее качество с его участием.

Для алгоритмов - одиночек не используется фильтрация, т.к. их совсем немного и все результаты важны, не только хорошие. В коде обеспечено включение и выключение методик фильтрации в зависимости от типа фильтруемых комбинаций.

2.2.10 Экспорт результатов (логирование)

Используется штатный функционал ЯП Python - модуль logging. Основная идея в том, что гораздо лучше использовать популярное решение, предназначенное для решения задачи, нежели изобретать уникальную никому неизвестную реализацию. Модуль позволяет переключаться между файлами с логами и дописывать в них различную информацию. Сам лог представляет собой текстовой файл с кодировкой UTF-8 и расширением .log, он может быть открыт в любом текстовом редакторе. Примеры логов показаны далее в соотв. разделе.

Виды лог-файлов:

- результаты, сортированные по метрике f1 в первую очередь: по одному лог-файлу на каждый тип комбинаций (вместе с комбинациями каждого

типа в каждый такой лог для наглядности экспортируются результаты алгоритмов-одиночек), отдельный лог для алгоритмов-одиночек, а также лог с совместными результатами для всех алгоритмов и их комбинаций,

- результаты, сортированные по метрике recall: всё тоже самое, за исключением алгоритмов-одиночек, они сортируются только по метрике f1,
- служебные лог-файлы: в данной работе используется только 1 такой файл, в который сохраняются промежуточные результаты расчётов, которые производит программа, это необходимо для контроля правильно её работы.

2.2.11 Функционал тестовых сценариев

В ПО реализована автоматизация проведения тестовых мероприятий. Был написан обработчик, который обеспечивает возможность задать множество тестовых сценариев и их последовательное проведение с сохранением лог-файлов в соответствующих каталогах файловой системы, каждый каталог назван именем тестового сценария. Каждый тестовый сценарий определяется наличием:

- названия тестового сценария, формат названия: *{краткое название датасета с годом его создания}_{тип спама}{предобработчик текста}{экстрактор признаков и его параметры}*
- датасета,
- функции предобработки текста,
- функции извлечения признаков,
- иных параметров для алгоритма поиска лучших комбинаций.

Об этом процессе подробнее написано в технологической части.

2.2.12 Валидационные мероприятия и результаты

Результаты позволяют решить 2 задачи:

1. Отбор комбинаций-лидеров для создания методики.
2. Исследовать др. алгоритмические комбинации для выявления свойств типов комбинаций и их особенностей, что позволит дополнить методику с их участием и определить перспективность дальнейших исследований.

ПО проводит многослойную фильтрацию результатов, о которой можно прочесть в разделе о ПО. Результаты для комбинаций специфичны тем, что удалены особенно бесполезные комбинации, которые не лучше алгоритмов-одиночек, из которых состоят. Это не правильная во всех смысла мера, но она позволяет анализировать именно вопрос превосходства выбранных концепций комбинирования. Т.е. приоритет отдаётся алгоритмам-одиночкам, а комбинации должны доказать своё концептуальное превосходство.

Сортировка производится в следующем порядке:

- f1 (по убыванию),
- recall (по убыванию),
- среднее на фолд время прогнозирования (сек)* (по возрастанию),
- среднее на фолд время обучения (сек)* (по возрастанию);

для значений метрик качества обнаружения округление производится до 4 знаков после запятой, для метрик производительности – до 7.

**Т.е. это время, требующееся на обработку* x *семплов, где* $x = \{\text{кол} - \text{во семплов в датасете}\}/\{\text{кол} - \text{во фолдов}\}$.

В данной работе результаты исследований представлены в упрощенном виде и сгруппированы в группы по интересующим критериям:

- ТОП сверху выводится до тех пор, пока важнейшая метрика (f1) не меняется, т.е. ТОП не может состоять из комбинаций с разными значениями важнейшей метрики, однако остальные метрики должны быть приемлемыми, иначе даже при высокой f1 комбинация не будет выведена.
- ТОП снизу не выводится, т.к. в нем нет смысла в контексте данной работы. ТОП-списки содержат как алгоритмы-одиночки, так и комбинации.
- В противовес ТОП-спискам указан либо список лучших алгоритмов одиночек, либо список лучших комбинаций, чтобы можно было отследить

динамику изменения качества от лучших одиночек к лучшим комбинациям, др. словами есть ли смысл использовать комбинации.

- Также указаны лучшие комбинации из других (не вошедших в ТОП) категорий.

SMS-семплы с Email-семлами не смешивались (отменённый тестовый сценарий), потому что слишком очевидно в теории и не требует проверка того, что качество модели, которая пытается обнаруживать оба типа сообщений, будет слишком низкой при работе по обоим типам спама, по 2 причинам: SMS-ки слишком короткие и содержат много жаргона и ошибок, в основном, в отличие от основной массы email-сообщений. Исследовать решение этих проблем нужно дополнительно: модели обнаружения SMS можно применять и для email-сообщений определённых длин и свойств (теоретически sms-спам могут рассылать по почте), предлагается исследовать преклассификацию разговорности сообщения (использовать модуль МТС, который показан в разделе по архитектуре).

При сравнении метрик качества обнаружения важно учитывать некоторую погрешность - разбиение на фолды имеет случайный характер и значения метрик могут незначительно изменяться от разбиения к разбиению.

Сценарии:

1. *K2017_Email pr1 Tfidf1(ngram=(1,1)):*

- *верхний ТОП:*

```
---LinearSVC_Balanced + RandomForest_Big
['f1=0.988', 'auc=0.9934', 'acc=0.9942', 'prec=0.9842', 'rec=0.992', 'train_time=0.9398128', 'pred_time=0.1044235']
```

- *лучшие алгоритмы-одиночки:*

```
---LinearSVC_Balanced
['f1=0.9873', 'auc=0.9927', 'acc=0.9939', 'prec=0.9842', 'rec=0.9905', 'train_time=0.0346078', 'pred_time=0.0004001']
```

- *лучшие комбинации других типов:*

Если представители каких-то типов не написаны, значит ни одна из протестированных комбинаций данного типа не прошла фильтрацию.

```
---ASGDAlg_Default <-> LinearSVC_Balanced
```

['f1=0.9879', 'auc=0.9919', 'acc=0.9942', 'prec=0.9884', 'rec=0.9876', 'train_time=0.0496112', 'pred_time=0.0008001']

---LinearSVC_Balanced * PAA_II_Balanced

['f1=0.9861', 'auc=0.9901', 'acc=0.9933', 'prec=0.9883', 'rec=0.9839', 'train_time=0.0454103', 'pred_time=0.0004001']

---BAGGING: ['LinearSVC_Balanced']

['f1=0.9854', 'auc=0.9904', 'acc=0.993', 'prec=0.9855', 'rec=0.9854', 'train_time=0.4218962', 'pred_time=0.0130029']

2. *K2017_Email pr1 Tfidf1(ngram=(1,2)):*

- *верхний ТОП:*

---LinearSVC_Balanced + PAA_I_Default + PAA_II_Balanced + RandomForest_Balanced

['f1=0.9887', 'auc=0.9937', 'acc=0.9946', 'prec=0.9856', 'rec=0.992', 'train_time=1.6137651', 'pred_time=0.1056239']

- *лучшие алгоритмы-одиночки:*

---LinearSVC_Balanced

['f1=0.9864', 'auc=0.99', 'acc=0.9935', 'prec=0.9898', 'rec=0.9832', 'train_time=0.1054238', 'pred_time=0.0009002']

- *лучшие комбинации других типов:*

---LinearSVC_Balanced <-> PAA_II_Balanced

['f1=0.9868', 'auc=0.9901', 'acc=0.9937', 'prec=0.9905', 'rec=0.9832', 'train_time=0.1340303', 'pred_time=0.0011002']

---PAA_I_Default * PAA_II_Default

['f1=0.9841', 'auc=0.9868', 'acc=0.9925', 'prec=0.9926', 'rec=0.9759', 'train_time=0.0587132', 'pred_time=0.0010002']

---BAGGING: ['LinearSVC_Balanced']

['f1=0.9822', 'auc=0.9847', 'acc=0.9916', 'prec=0.9933', 'rec=0.9715', 'train_time=1.1486307', 'pred_time=0.0656478']

3. *K2017_Email pr1 Tf1(ngram=(1,2)):*

- *верхний ТОП:*

---ComplementNB_Default + PAA_I_Default + RandomForest_Default + RandomForest_Balanced

['f1=0.9876', 'auc=0.9926', 'acc=0.994', 'prec=0.9855', 'rec=0.9898', 'train_time=2.8860535', 'pred_time=0.212848']

---ComplementNB_Default + PAA_I_Default + RandomForest_Bootstrap90 + RandomForest_Balanced

['f1=0.9876', 'auc=0.9926', 'acc=0.994', 'prec=0.9855', 'rec=0.9898', 'train_time=2.8474446', 'pred_time=0.2128481']

---ComplementNB_Default + PAA_I_Default + RandomForest_BigBootstrap75 + RandomForest_Balanced

['f1=0.9876', 'auc=0.9926', 'acc=0.994', 'prec=0.9855', 'rec=0.9898', 'train_time=4.7943853', 'pred_time=0.3138709']

---ComplementNB_Default + PAA_I_Default + RandomForest_Big + RandomForest_Balanced

['f1=0.9876', 'auc=0.9926', 'acc=0.994', 'prec=0.9855', 'rec=0.9898', 'train_time=5.373216', 'pred_time=0.3253737']

- *лучшие алгоритмы-одиночки:*

---PAA_II_Default

['f1=0.9861', 'auc=0.9911', 'acc=0.9933', 'prec=0.9856', 'rec=0.9868', 'train_time=0.0414094', 'pred_time=0.0005001']

- *лучшие комбинации других типов:*

---LinearSVC_Balanced <-> SVCAlg_RBF_Default <-> PAA_I_Default <-> PAA_II_Balanced

['f1=0.9872', 'auc=0.9922', 'acc=0.9939', 'prec=0.9855', 'rec=0.989', 'train_time=7.3134555', 'pred_time=0.75237']

---LinearSVC_Balanced * PAA_II_Balanced

['f1=0.9858', 'auc=0.9915', 'acc=0.9932', 'prec=0.9834', 'rec=0.9883', 'train_time=0.1367311', 'pred_time=0.0014001']

---BAGGING: ['PAA_II_Balanced']

['f1=0.985', 'auc=0.9908', 'acc=0.9928', 'prec=0.9833', 'rec=0.9868', 'train_time=0.2686608', 'pred_time=0.034808']

4. K2017_Email pr1 Counts1:

(для бэггинг-комбинаций здесь макс. длинна комбинаций = 5)

- *верхний ТОП:*

---ComplementNB_Default * SGDAlg_AdaptiveIters

['f1=0.9797', 'auc=0.9844', 'acc=0.9903', 'prec=0.9867', 'rec=0.973', 'train_time=4.2359592', 'pred_time=0.0985218']

- *лучшие алгоритмы-одиночки:*

---ComplementNB_Default

['f1=0.9768', 'auc=0.9907', 'acc=0.9886', 'prec=0.9594', 'rec=0.9949', 'train_time=1.937139', 'pred_time=0.0481103']

- *лучшие комбинации других типов:*

---SGDClf_Default <-> SGDAlg_LogLoss <-> LinearSVC_Balanced <-> PAA_I_Default

['f1=0.9757', 'auc=0.9855', 'acc=0.9882', 'prec=0.9712', 'rec=0.9803', 'train_time=9.8178218', 'pred_time=0.1977449']

---SGDAlg_LogLoss + RandomForest_Big + RandomForest_Medium + RandomForest_BigBootstrap75

['f1=0.9787', 'auc=0.9892', 'acc=0.9896', 'prec=0.9694', 'rec=0.9883', 'train_time=22.8354678', 'pred_time=0.4657057']

---SGDAlg_LogLoss + RandomForest_Big + RandomForest_MDepth30 + RandomForest_BigBootstrap75

['f1=0.9787', 'auc=0.9887', 'acc=0.9896', 'prec=0.9708', 'rec=0.9868', 'train_time=23.6117438', 'pred_time=0.4668056']

---BAGGING: ['SGDAlg_LogLoss']

['f1=0.9737', 'auc=0.9861', 'acc=0.9872', 'prec=0.964', 'rec=0.9839', 'train_time=71.3991587', 'pred_time=0.6801544']

5. *E_Email pr1 Tfidf1(ngram=(1,1)):*

- *верхний ТОП:*

---LinearSVC_Balanced * PAA_I_Default

['f1=0.9809', 'auc=0.9877', 'acc=0.9888', 'prec=0.9769', 'rec=0.9849', 'train_time=0.0442101', 'pred_time=0.0001']

---LinearSVC_Default * PAA_I_Default

['f1=0.9809', 'auc=0.9875', 'acc=0.9888', 'prec=0.9776', 'rec=0.9842', 'train_time=0.034908', 'pred_time=0.0003']

---LinearSVCAlg_MoreSupports * PAA_I_Default

['f1=0.9809', 'auc=0.9875', 'acc=0.9888', 'prec=0.9776', 'rec=0.9842', 'train_time=0.0209049', 'pred_time=0.0003']

---SGDClf_Default * PAA_I_Default

['f1=0.9809', 'auc=0.987', 'acc=0.9888', 'prec=0.9789', 'rec=0.9829', 'train_time=0.0180041', 'pred_time=0.0']

- *лучшие алгоритмы-одиночки:*

---PAA_I_Default

['f1=0.9803', 'auc=0.9876', 'acc=0.9884', 'prec=0.975', 'rec=0.9856', 'train_time=0.009002', 'pred_time=0.0']

- *лучшие комбинации других типов:*

---ComplementNB_Default + PAA_I_Default

['f1=0.9793', 'auc=0.9877', 'acc=0.9878', 'prec=0.9711', 'rec=0.9877', 'train_time=0.0115026', 'pred_time=0.0004001']

---BAGGING: ['PAA_I_Default']

['f1=0.9805', 'auc=0.9871', 'acc=0.9886', 'prec=0.9776', 'rec=0.9836', 'train_time=0.0609138', 'pred_time=0.0067015']

---SGDAlg_LogLoss <-> PAA_I_Default <-> PAA_II_Default

['f1=0.9806', 'auc=0.9881', 'acc=0.9886', 'prec=0.9743', 'rec=0.987', 'train_time=0.0288066', 'pred_time=0.0001']

---SGDAlg_LogLoss <-> PAA_I_Default <-> PAA_II_Balanced

['f1=0.9806', 'auc=0.9881', 'acc=0.9886', 'prec=0.9743', 'rec=0.987', 'train_time=0.0290067', 'pred_time=0.0001']

---BAGGING: ['PAA_I_Default']

['f1=0.9805', 'auc=0.9871', 'acc=0.9886', 'prec=0.9776', 'rec=0.9836', 'train_time=0.0609138', 'pred_time=0.0067015']

6. *E_Email pr1 Tfidf1(ngram=(1,2)):*

- *верхний ТОП:*

```
---PAA_I_Default <-> PAA_II_Default
['f1=0.9824',  'auc=0.99',  'acc=0.9896',  'prec=0.9738',  'rec=0.9911',  'train_time=0.0391088',
'pred_time=0.0005001']
---LinearSVCAlg_MoreSupports * PAA_II_Default
['f1=0.9824',  'auc=0.9898',  'acc=0.9896',  'prec=0.9745',  'rec=0.9904',  'train_time=0.0511117',
'pred_time=0.0006001']
---LinearSVCAlg_MoreSupports * PAA_I_Default
['f1=0.9824',  'auc=0.9898',  'acc=0.9896',  'prec=0.9745',  'rec=0.9904',  'train_time=0.0478108',
'pred_time=0.0007001']
---LinearSVC_Default * PAA_II_Default
['f1=0.9824',  'auc=0.9898',  'acc=0.9896',  'prec=0.9745',  'rec=0.9904',  'train_time=0.0857196',
'pred_time=0.0009']
---LinearSVC_Default * PAA_I_Default
['f1=0.9824',  'auc=0.9898',  'acc=0.9896',  'prec=0.9745',  'rec=0.9904',  'train_time=0.0824187',
'pred_time=0.0010001']
```

- *лучшие алгоритмы-одиночки:*

```
---PAA_I_Default
['f1=0.982',  'auc=0.9899',  'acc=0.9894',  'prec=0.9732',  'rec=0.9911',  'train_time=0.0179039',
'pred_time=0.0003001']
```

- *лучшие комбинации других типов:*

```
---PAA_I_Default + PAA_II_Default
['f1=0.982',  'auc=0.9901',  'acc=0.9894',  'prec=0.9725',  'rec=0.9918',  'train_time=0.0391088',
'pred_time=0.0005001']
```

7. *K2016_SMS pr1 Tfidf1(ngram=(1,1)):*

- *верхний ТОП:*

```
---SGDClf_Default + kNN_Default + RandomForest_Medium + RandomForest_Balanced
['f1=0.9393',  'auc=0.9554',  'acc=0.9851',  'prec=0.9648',  'rec=0.9157',  'train_time=0.5617274',
'pred_time=0.3555801']
```

- *лучшие алгоритмы-одиночки:*

```
---SGDClf_Default
['f1=0.9278',  'auc=0.9415',  'acc=0.9826',  'prec=0.9738',  'rec=0.8866',  'train_time=0.0038008',
'pred_time=0.0001']
```

- *лучшие комбинации других типов:*

```
---SGDAlg_LogLoss <-> ASGDAlg_Default <-> PAA_I_Default <-> PAA_II_Default
```

['f1=0.9295', 'auc=0.943', 'acc=0.9829', 'prec=0.9739', 'rec=0.8897', 'train_time=0.018004', 'pred_time=0.0003']

---SGDClf_Default <-> LinearSVCAlg_MoreSupports <-> PAA_I_Default <-> PAA_II_Balanced

['f1=0.9295', 'auc=0.9431', 'acc=0.9829', 'prec=0.9741', 'rec=0.8897', 'train_time=0.0162036', 'pred_time=0.0005002']

---SGDClf_Default <-> LinearSVC_Balanced <-> SVCAlg_RBF_Default <-> PAA_II_Balanced

['f1=0.9295', 'auc=0.943', 'acc=0.9829', 'prec=0.9741', 'rec=0.8896', 'train_time=0.5204435', 'pred_time=0.0513152']

---ComplementNB_Default * LinearSVC_Balanced

['f1=0.9271', 'auc=0.9421', 'acc=0.9824', 'prec=0.9705', 'rec=0.8881', 'train_time=0.0127029', 'pred_time=0.0002']

---BAGGING: ['PAA_II_Balanced']

['f1=0.9241', 'auc=0.9378', 'acc=0.9818', 'prec=0.975', 'rec=0.8789', 'train_time=0.067115', 'pred_time=0.0085019']

8. K2016_SMS pr1 Tfidf1(ngram=(1,2)):

- *верхний ТОП:*

---PAA_I_Default + PAA_II_Balanced + RandomForest_Default + RandomForest_Medium

['f1=0.9308', 'auc=0.9464', 'acc=0.9831', 'prec=0.9673', 'rec=0.8973', 'train_time=1.2256774', 'pred_time=0.2065468']

---PAA_I_Default + PAA_II_Balanced + RandomForest_Medium + RandomForest_Balanced

['f1=0.9308', 'auc=0.9464', 'acc=0.9831', 'prec=0.9673', 'rec=0.8973', 'train_time=1.1881689', 'pred_time=0.2067467']

- *лучшие алгоритмы-одиночки:*

---PAA_II_Balanced

['f1=0.9226', 'auc=0.937', 'acc=0.9814', 'prec=0.973', 'rec=0.8775', 'train_time=0.0053012', 'pred_time=0.0001']

- *лучшие комбинации других типов:*

---ComplementNB_Default * PAA_II_Balanced

['f1=0.9253', 'auc=0.9361', 'acc=0.9822', 'prec=0.983', 'rec=0.8744', 'train_time=0.0067015', 'pred_time=0.0006002']

---SGDClf_Default <-> PAA_II_Balanced

['f1=0.9227', 'auc=0.9376', 'acc=0.9814', 'prec=0.9716', 'rec=0.8789', 'train_time=0.0112026', 'pred_time=0.0002001']

---BAGGING: ['PAA_II_Balanced']

['f1=0.9262', 'auc=0.9375', 'acc=0.9824', 'prec=0.9816', 'rec=0.8774', 'train_time=0.0353079', 'pred_time=0.0054013']

9. K2016_SMS pr1 Tfidf1(ngram=(1,3)):

- *верхний ТОП:*

```
---ComplementNB_Default * Perceptron_Default
['f1=0.9261', 'auc=0.9407', 'acc=0.9822', 'prec=0.9718', 'rec=0.8852', 'train_time=0.0083017',
'pred_time=0.0013004']
```

- *лучшие алгоритмы-одиночки:*

```
---PAA_II_Balanced
['f1=0.9159', 'auc=0.929', 'acc=0.98', 'prec=0.9795', 'rec=0.8606', 'train_time=0.0060013',
'pred_time=0.0002001']
```

- *лучшие комбинации других типов:*

```
---SGDClf_Default <-> PAA_II_Balanced
['f1=0.9174', 'auc=0.9324', 'acc=0.9802', 'prec=0.9729', 'rec=0.8683', 'train_time=0.0124026',
'pred_time=0.0003001']


---PAA_II_Balanced + RandomForest_Medium + RandomForest_Small + RandomForest_Balanced
['f1=0.924', 'auc=0.9372', 'acc=0.9818', 'prec=0.9763', 'rec=0.8774', 'train_time=2.015456',
'pred_time=0.3101703']
```

10.K2016_SMS pr1 Tfidf1(ngram=(2,2)):

- *верхний ТОП:*

```
---SGDAlg_LogLoss + RandomForest_Small + RandomForest_Balanced + Perceptron_Default
['f1=0.8936', 'auc=0.9262', 'acc=0.974', 'prec=0.9288', 'rec=0.8621', 'train_time=1.7334925',
'pred_time=0.2062464']
---SGDAlg_LogLoss + RandomForest_Big + RandomForest_Small + Perceptron_Default
['f1=0.8936', 'auc=0.9262', 'acc=0.974', 'prec=0.9288', 'rec=0.8621', 'train_time=4.3583864',
'pred_time=0.2063466']
---SVCAlg_RBF_Default + RandomForest_Small + RandomForest_Balanced + Perceptron_Default
['f1=0.8936', 'auc=0.9262', 'acc=0.974', 'prec=0.9288', 'rec=0.8621', 'train_time=2.8806522',
'pred_time=0.3296743']
---SVCAlg_RBF_Default + RandomForest_Big + RandomForest_Small + Perceptron_Default
['f1=0.8936', 'auc=0.9262', 'acc=0.974', 'prec=0.9288', 'rec=0.8621', 'train_time=5.505546',
'pred_time=0.3297745']
```

- *лучшие алгоритмы-одиночки:*

```
---Perceptron_Default
['f1=0.8875', 'auc=0.9189', 'acc=0.9729', 'prec=0.9338', 'rec=0.8467', 'train_time=0.0034007',
'pred_time=0.0001']
```

- *лучшие комбинации других типов:*

-

11.K2016_SMS pr1 Tfidf1(ngram=(1,2)) max_combination_length=7:

Данный сценарий обосновывает необходимость тестирования более длинных тривиальных комбинаций – тестировались только они

- *верхний ТОП:*

```
---PAA_II_Balanced + kNN_Default + RandomForest_Medium + RandomForest_MDepth30
['f1=0.93', 'auc=0.9457', 'acc=0.9829', 'prec=0.9676', 'rec=0.8958', 'train_time=1.0018275', 'pred_time=0.3877875']
---PAA_II_Balanced + kNN_Default + RandomForest_Medium + RandomForest_Bootstrap90
['f1=0.93', 'auc=0.9457', 'acc=0.9829', 'prec=0.9676', 'rec=0.8958', 'train_time=1.4320246', 'pred_time=0.3877877']
---PAA_II_Balanced + kNN_Default + RandomForest_Default + RandomForest_Medium
['f1=0.93', 'auc=0.9457', 'acc=0.9829', 'prec=0.9676', 'rec=0.8958', 'train_time=1.4975394', 'pred_time=0.3881877']
```

- *верхний ТОП (по метрике recall):*

```
---ComplementNB_Default
['f1=0.8175', 'auc=0.9482', 'acc=0.9461', 'prec=0.7176', 'rec=0.951', 'train_time=0.0021005', 'pred_time=0.0004001']
---SGDAlg_AdaptiveIters + SGDAlg_LogLoss + PAA_II_Balanced + kNN_Default + RandomForest_Medium + Perceptron_Default
['f1=0.9182', 'auc=0.9527', 'acc=0.9793', 'prec=0.9199', 'rec=0.9172', 'train_time=0.5381221', 'pred_time=0.2799631']
```

- *лучшие алгоритмы-одиночки:*

```
---PAA_II_Balanced
['f1=0.9258', 'auc=0.9387', 'acc=0.9822', 'prec=0.9765', 'rec=0.8805', 'train_time=0.0062013', 'pred_time=0.0003001']
```

- *лучшие комбинации других типов:*

```
---ASGDAlg_Default <-> PAA_II_Balanced
['f1=0.927', 'auc=0.9414', 'acc=0.9824', 'prec=0.9719', 'rec=0.8866', 'train_time=0.0135031', 'pred_time=0.0003001']


---ComplementNB_Default * PAA_II_Balanced
['f1=0.927', 'auc=0.9376', 'acc=0.9826', 'prec=0.983', 'rec=0.8774', 'train_time=0.0083019', 'pred_time=0.0007002']
```

Исследовательские мероприятия и задействованные для этого тестовые сценарии

- Выявления наиболее эффективного подхода подсчета признаков (counts, tf, tf-idf): 2-4. Важно то, что при тестировании каждого подхода каждый раз ищется наиболее эффективный алгоритм, а не сравниваются результаты зафиксированного алгоритма. **Выводы:** Очевидно превосходство вычисления tf-idf значений над использованием ненормированных частотностей встречаемости в сообщениях, но превосходство над tf также имеется. Можно предположить, что необходимо провести тесты на извлечениях различных признаков, а не только униграмм и биграмм и только тогда пытаться делать однозначные выводы, но это видится избыточной проверкой, поскольку разница между tf и tf-idf большой и не предполагается. На рисунке 21 показаны результаты эксперимента.

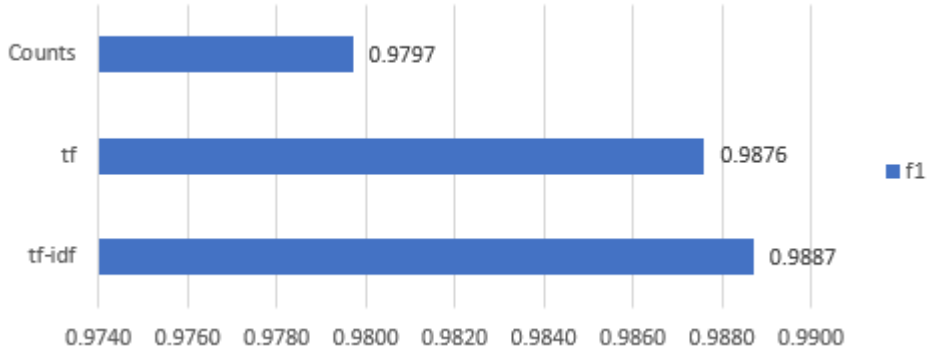

Рисунок 21 – Сравнение алгоритмов векторизации

- Выявление наиболее эффективного подхода извлечения n-грамм. Результаты показаны на рисунке 22, построены на основе сценариев 1,2,5,6,7-10. В данном тесте важно учесть тот факт, что критерием является не результат, который демонстрирует какой-то фиксированный алгоритм при разных экстракторах признаков, вопрос стоит немного иначе – как отличаются результаты лучших алгоритмов, найденных при разных экстракторах признаков. Более того, необходимо провести эти тесты на разных датасетах, чтобы точнее понять преимущества того или иного подхода, поскольку имеется проблема производительности: чем больше

признаков извлекается, тем больше требуется вычислительных ресурсов. Не все устройства, которым нужна фильтрация сообщений на уровне клиента, имеют много ресурсов, а для некоторых важна ещё и автономность работы. К тому же, от датасета к датасета семплы имеют разные длины и разный характер применения слов. **Выводы:** Очевидно стабильное, хоть и численно мизерное, превосходство извлечения униграмм и биграмм на крупных датасетах с сообщениями электронной почты, это логичный результат, хорошо и то, что извлечение униграмм и биграмм не требует сильного увеличения вычислительных затрат. Однако, удивительные результаты получились для датасета с SMS-семлами – с добавлением связующих признаков (биграмм,…) результат уверенно снижается. Если разница между (1,2) и (1,3) – граммами не позволяет делать однозначных выводов, то относительное возвышение униграмм над остальными и явно низкий результат для биграмм говорят о многом. Необходимо обратить внимание на особенности датасета K2016_SMS, которые и являются причиной таких результатов, как минимум повышенная уникальность пар и троек слов, это характерно для крайне не нормированного и парой не грамотного разговорного текста (при итак небольших размерах семплов, что провоцирует излишнюю разряженность векторов), который ещё и часто пестрит крайне отличающимися формами одних и тех слов (например, словами пишутся смайлики вместо обычных слов, выражающих эмоции или целые характеризующие предложения заменяются словами-одиночками из интернет-сленга: «crazy mistake» -> «oh lol», «very funny and stupid person» -> «lalka» и т.д.) - даже одно и тоже слово может быть написано по разному, например в разговорном стиле или с ошибками, поскольку SMS-сообщения набираются с мобильного устройства, что увеличивает вероятность ошибиться: «you» -> «u», «o my god» -> «omg», «you want» -> «u w»). Усложняет ситуацию тот факт, что Интернет постоянно плодит новые языковые конструкции и слова, которые вырабатываются людьми или в принципе коверкается смысл уже

имеющихся в языке слов, всё это происходит связи с популярностью соц. сервисов Интернета и нежеланием людей уделять много внимания литературе и языку, зато крайне много уделяется медийным развлекательным сервисам и социальным сервисам, это очевидная и стабильная тенденция. SMS-сообщения и мгновенные сообщения из мессенджеров являются более изменчивыми в этом контексте, что сильно усложняет регулярное обнаружение спама. Для эффективного обнаружения SMS-спама явно не стоит всерьёз надеяться на помощь популярных средств извлечения признаков, на это указывает в целом относительно более низкий результат, а **лучшим выбором из имеющихся экстракторов является извлечения только лишь униграмм.** Требуется глубокая предобработка SMS-семплов, которая будет включать приведение слов к единому виду, либо использование алгоритмов вероятностного связывания, когда с определённой вероятностью будет установлена схожесть одного слова с другим. Также наверняка необходимо использовать метаданные сообщения помимо его словарного содержания.

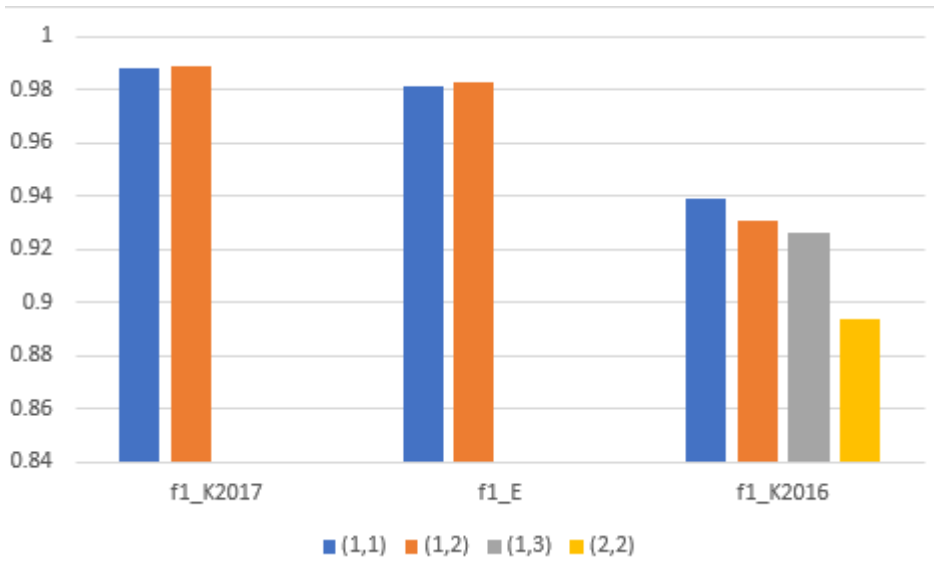

Рисунок 22 – Результаты разных токенизаторов на разных датасетах

- Проверка существенной важности исследовать тривиальные комбинации длинной больше 4: 11. **Выводы:** лидерами теста стали комбинации длинной 4, это скорее подтверждает тезис о том, что больше 4 и не следует тестировать. Однако это если рассуждать о наилучшем соотношении аккуратности (precision) и полноты (recall) обнаружения. Лидером по полноте стал алгоритм Наивного Байеса, однако у него слишком низкий показатель precision, зато на втором месте расположилась комбинация из 6 алгоритмов. Сравним метрики качества обнаружения этой комбинации с комбинацией-лидером по f1 в данном тесте на рисунке 23:

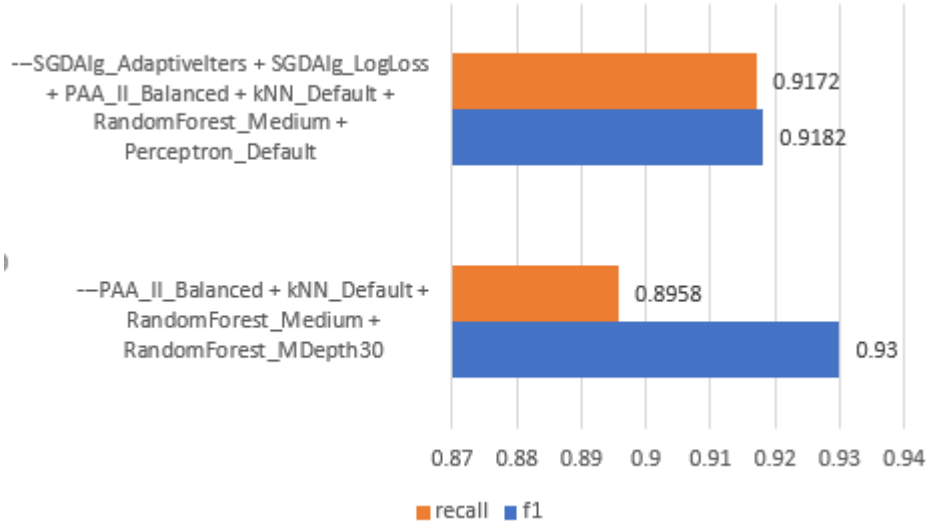

Рисунок 23 – Дизъюнктивная комбинация длинной 6 лучшая по recall при высоком значении precision

Другие наблюдения и выводы

- Дизъюнктивные комбинации лучше остальных в большинстве сценариев.
- Конъюнктивные комбинации не часто успешны, но успешность разительно возрастает при увеличении количества фич. Важно то, что в некоторых сценариях они превзошли даже дизъюнктивные. Этот фактор важен, поскольку полноценный анти-спам всегда располагает

ограниченным кол-вом признаков как минимум из-за несовершенства набора признаков по охвату потенциальных семплов.

- Бэггинговые комбинации не показали выдающихся результатов, возможно из-за недостаточной длинны. Т.е. длину следует увеличивать радикально (больше 10, например, кол-во решающих деревьев в бэггинге RandomForest = 100 для новых версий scikit-learn, 10 для более старых), чтобы окончательно оценить ценность таких комбинаций, однако вряд ли приходится ждать сильного увеличения метрики качества. Но у них уже наблюдается стабильное положительное свойство: они увеличивают аккуратность обнаружения (стойкость к переобучению, устойчивость к выбросам), что не удивительно, учитывая характер их работы, не так сильно понижая полноту (recall) – это ведёт к возрастанию метрики f1. Иногда они могут быть полезными.
- Была найдена как минимум одна комбинация, которая показала высокую метрику при кросс-валидации на датасете с SMS-спамом (f1=0.9393) и отлично себя показала (с небольшим отставанием от лидеров обнаружения email-спама) при кросс-валидации на обоих датасетах с email: f1 = 0.956 и 0.9791, а лидер имеет средне значение метрики f1 = 0.9832. Это говорит о возможности использовать одну алгоритмическую комбинацию в моделях разных тематик.
- При перекрестном сравнении результатов комбинаций (если имеется >1 датасета с семплами одного типа при равных параметрах исследования) станет очевидным следующее - **многие комбинации не могут быть универсальными, т.е. при отличных метриках на одном датасете можно видеть сильно сниженные на другом.**
- Необходимо дать ответ на ещё один важный вопрос – перспективны ли исследованные комбинации в принципе – способны ли они каким-либо образом пройти кросс-валидацию лучше алгоритмов-одиночек и на сколько. Остается много вопросов о некоторых комбинациях-лидерах, которые обгоняют одиночек – сфера применения, универсальность и т.д.,

они здесь опускаются. Рассмотрим сравнительные графики по метрике f1 лучших комбинаций каждого типа и лучших алгоритмов-одиночек за все тестовые сценарии на рисунке 25. К графикам приложена таблица со значениями (рисунок 24), в которой обозначены цветом результаты алгоритмов-лидеров (они стали основой рассмотренных далее моделей методики обнаружения) в наиболее удачных сценариях по SMS-сообщениям и email. По значениям видно, что **применение комбинаций увеличило качество обнаружения (f1): SMS на 0.0115 (DC), email на 0.0004 (DC) в худшем случае, на 0.0023 (DC) в лучшем случае.** Т.е. кардинально качество поднять агрегацией результатов не удалось, тем не менее наблюдается стабильное преимущество комбинаций за все тестовые сценарии.

test/f1	DC	CC	MC	BAGC	SA	max f1	max-SA	
tfidf 1,1	0.988	0.9861	0.9879	0.9854	0.9873	0.988000	0.000700	
tfidf 1,2	0.9887	0.9841	0.9868	0.9822	0.9864	0.988700	**0.002300**	email-модель
Tf 1,2	0.9876	0.9858	0.9872	0.985	0.9861	0.987600	0.001500	
Counts	0.9787	0.9797	0.9757	0.9737	0.9768	0.979700	0.002900	
tfidf 1,1	0.9793	0.9809	0.9806	0.9805	0.9803	0.980900	0.000600	
tfidf 1,2	0.982	0.9824	0.9824		0.982	0.982400	**0.000400**	email-модель
tfidf 1,1	0.9393	0.9271	0.9295	0.9241	0.9278	0.939300	**0.011500**	SMS-модель
tfidf 1,2	0.9308	0.9253	0.9227	0.9262	0.9226	0.930800	0.008200	
tfidf 1,3	0.924	0.9261	0.9174		0.9159	0.926100	0.010200	
tfidf 2,2	0.8936				0.8875	0.893600	0.006100	
							0.000400	min
							0.011500	max

Рисунок 24 – Таблица Excel для сравнения результатов лучших алгоритмов каждого типа за все тестовые сценарии (для построения графиков ниже)

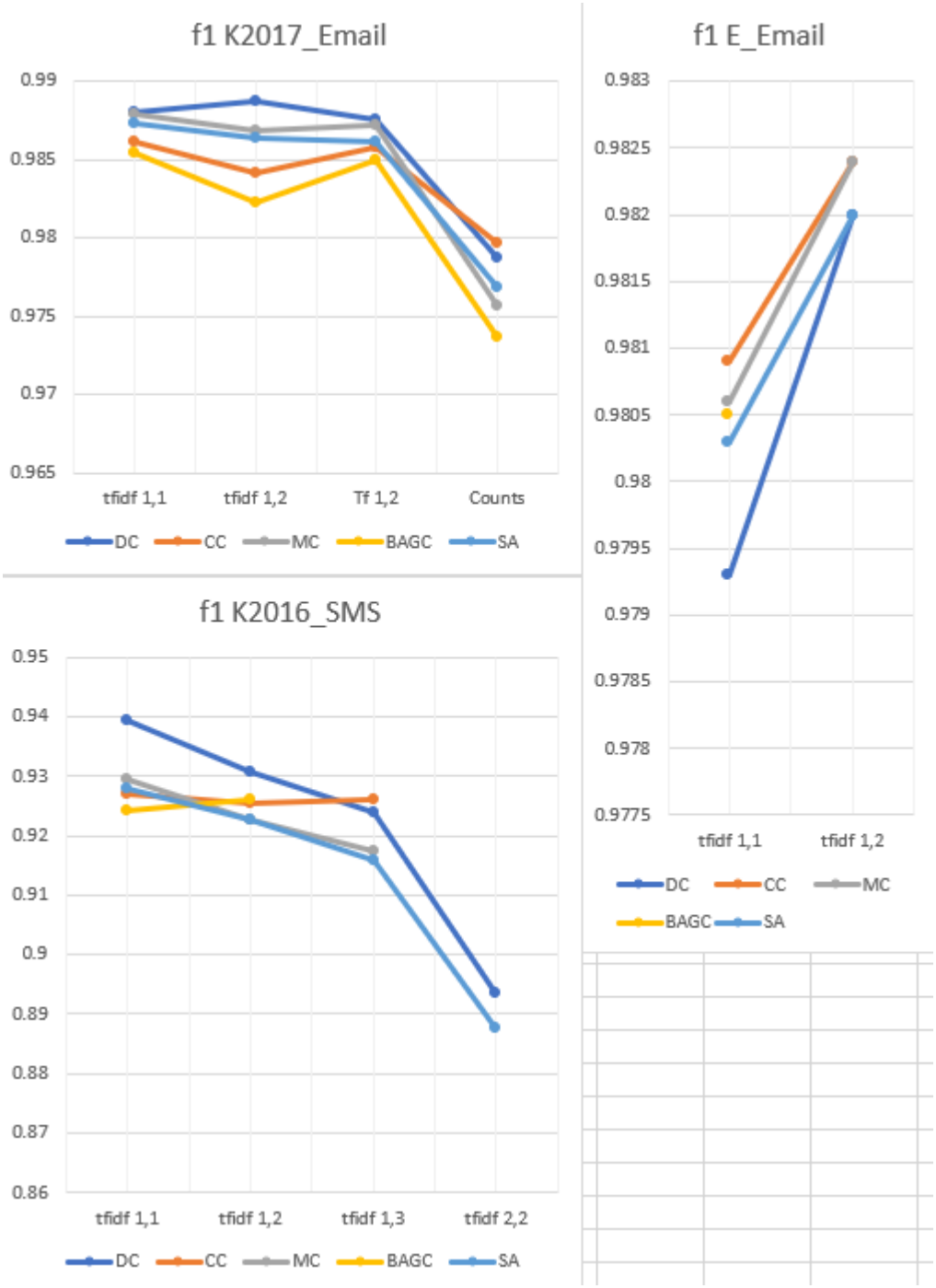

Рисунок 25 – Сравнительные графики лучших алгоритмов каждого типа за все тестовые сценарии

Лучшие комбинации

Алгоритмы отбора лидеров тестов на основе логов результатов (всех типов комбинаций) выполняются для следующих условий применения:

- Email-семплы: имеем результаты двух датасетов с семплами одной группы (параметры исследований должны быть одинаковыми, фильтрация избыточных и бесполезных комбинаций для всех типов комбинаций отключена), выполняем алгоритм вычислительной сложности $O(n^2)$ в худшем случае:
    1. производим перекрестный поиск и сравнение, когда выбирается комбинация сначала в одном списке результатов и ищется в другом, при нахождении алгоритма вычисляем среднее значение метрик f1, запоминаем его как текущий максимум (оптимизация №1 для экономии памяти), также запоминаем для рассматриваемого алгоритма противоположного списка (на основе которого был обновлён максимум f1) его значение f1 как текущий максимум противоположного списка (оптимизация №2 для снижения кол-ва вычислений средних величин), чтобы потом не допускать к рассмотрению те алгоритмы др. списка, которые хуже по f1 уже рассмотренных, продолжаем перекрестный поиск по аналогии и вычисление средних метрик, если очередное среднее значение f1 меньше текущего общего максимума по f1, то не обновляем текущую лучшую комбинацию (оптимизации стали возможны именно благодаря тому, что оба списка отсортированы по убыванию значения метрики f1),
    2. потом в другом списке тоже самое, однако свой первый максимум по f1 там не фиксируется, ибо он уже есть, фиксируется только максимальный f1 противоположного списка,
    3. у нас остаётся комбинация с лучшим средним значением по f1 – она и является лучшей на основе результатов двух датасетов.
- SMS-семплы: фиксируем комбинации, находящиеся выше всех у списке и имеющих одинаковые между собой значения всех метрик, однако

применяется дополнительный фактор: выбирается алгоритм, который наиболее теоретически наиболее обобщён в связи с тем, что для работы не был добыт второй датасет с sms-семплами.

2.3 Разработка методики обнаружения вредоносного спама

2.3.1 Архитектура анти-спам системы

В данной работе не делается упор на архитектуру – исследуются именно алгоритмы и их комбинации и на данном этапе не требуется создавать методику обнаружения спама в реальных условиях эксплуатации. Поэтому архитектура предварительная и не все её части проанализированы в работе (см. рисунок 26) и просто призвана дать минимальное понимание системы, в которой может использоваться методика. Уровень агрегации результатов моделей не рассматривается, как и механизм МТС.

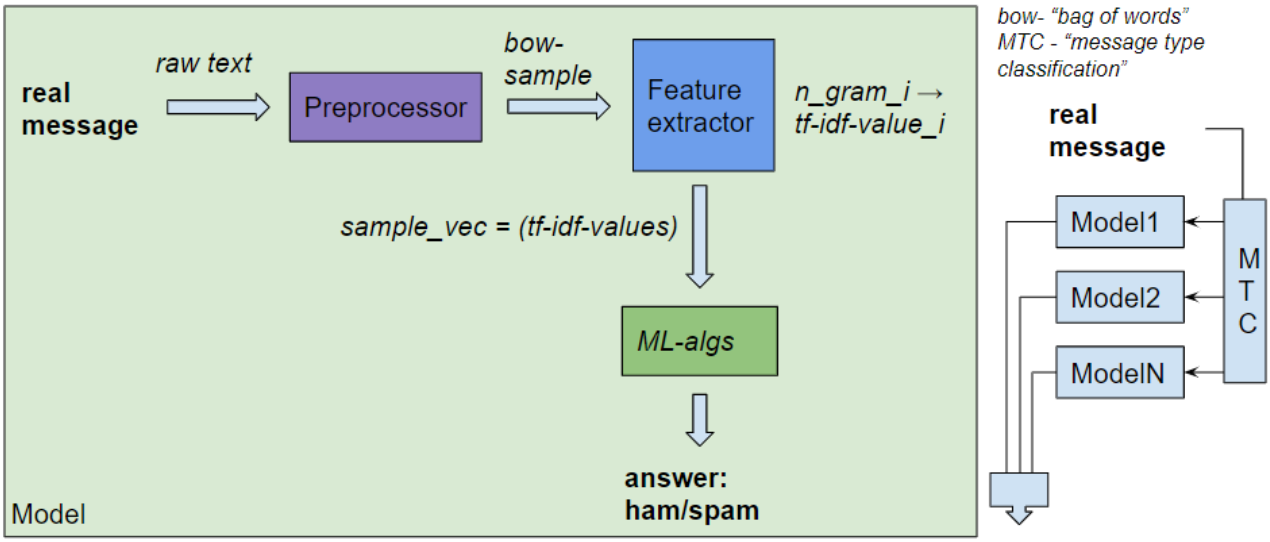

Рисунок 26 – Архитектура потенциального анти-спам детектора

2.3.2 Модели обнаружения спама

Этапы функционирования модели:

1. Работает предобработчик семпла.
2. Работает экстрактор признаков семпла (создаётся набор признаков*).
3. Посредством алгоритма (ов) классифицируется образец.

**Вопросы создания универсального набора признаков (словаря в случае с униграммами) не входят в данную работу в связи с тем, что все исследования построены не на прототипе анти-спам-системы, а на кросс-валидации, когда словарь строится по всему датасету; мы готовим методику на основе кросс-валидационных тестов, а не прототипа реальной анти-спам системы, обходя вопрос компоновки словаря и сопутствующих проблем из-за неспособности извлечь всё, что содержит случайный семпл, ведь словарь ограничен и особенностей работы с алгоритмами МО из-за этого.*

Для обнаружения Email-спама

Найденный алгоритм показывается следующие результаты в каждом из датасетов:

```
---SGDAlg_LogLoss <-> PAA_I_Default <-> PAA_II_Default
['f1=0.9824',    'auc=0.99',    'acc=0.9896',    'prec=0.9738',    'rec=0.9911',
'train_time=0.0590127', 'pred_time=0.0012008']
---SGDAlg_LogLoss <-> PAA_I_Default <-> PAA_II_Default
['f1=0.9841',    'auc=0.9868',    'acc=0.9925',    'prec=0.9926',    'rec=0.9759',
'train_time=0.0858193', 'pred_time=0.0014003']
```

f1_mean = 0.9832

pred_time_mean = 0.00130055 ($\overline{v_{pred}}$ = 416 семплов/мс)

(Скорость округлена с недостатком)

- Алгоритм: SGDAlg_LogLoss ↔ PAA_I_Default ↔ PAA_II_Default
- Экстрактор признаков: Tfidf1(ngram=(1,2)).
- Предобработчик: preprocessor1.
- Условия эксплуатации:
    - Устройства: смартфоны, компьютеры, серверы (малое время прогнозирования).

- Режимы работы: потоковый фоновый* (малое время прогнозирования) на всех перечисленных устройствах.

*Одно сообщение за другим «на лету» без заметных задержек.

Для обнаружения SMS-спама

Отбор алгоритмов-кандидатов:

- Комбинация-лидер: ни один алгоритм комбинации не видится слабым в контексте обобщающей способности, т.е. сильно зависимым от набор данных - на примере 2 датасетов с email-семплами это подтвердилось, более того – на обоих этих датасетах комбинация набрала высокие значения метрик f1 0.956 и 0.9791 и метрика precision при этом осталась примерно той же, что и в нашем текущем случае. Дополнительным фактором является высокая обобщающая способность исходя из принципы их работы - как минимум 3 алгоритмов из 4. Вывод: эта комбинация достойна считаться лучшей с одним условием – конечно же, с учетом особенностей и вариативности SMS-спама текущего исследования крайне не достаточно, но для начала это неплохой выбор.

```
---SGDClf_Default + kNN_Default + RandomForest_Medium + RandomForest_Balanced
['f1=0.9393',   'auc=0.9554',   'acc=0.9851',   'prec=0.9648',   'rec=0.9157',
'train_time=0.5617274', 'pred_time=0.3555801' (v_pred = 1,238 семпла/мс)]
```

- Алгоритм: SGDClf_Default + kNN_Default + RandomForest_Medium + RandomForest_Balanced
- Экстрактор признаков: Tfidf1(ngram=(1,1)).
- Предобработчик: preprocessor1.
- Условия эксплуатации:
  - Устройства: смартфоны, компьютеры, серверы (малое время прогнозирования).
  - Режимы работы: потоковый фоновый* (малое время прогнозирования) на всех перечисленных устройствах.

2.3.3 Методика обнаружения спама

Формирует 2 модели: для email, для sms.

Обучение

I. Сформировать набор признаков (словарь) для каждой модели.
II. Подготовить датасеты с предобработанными SMS и email-сообщениями:
    1. Удалить документы: дубликаты, слишком длинные и короткие.
    2. Далее предобработать каждый документ по алгоритму ниже*.
III. Обучить алгоритмы моделей на соответствующих датасетах:
    - Email-модель: SGDAlg_LogLoss, PAA_I_Default, PAA_II_Default.
    - SMS-модель: SGDClf_Default, kNN_Default , RandomForest_Medium, RandomForest_Balanced.

Данный вопрос (обучения) сильно упрощён для кросс-валидации, поскольку словарь строится на основе датасетов, а не заранее путём кропотливых исследований и экспериментов. Каждый семпл будет разобран на n-граммы (на основе словаря), им будут сопоставления значения, далее каждый семпл будет передан алгоритму с его истинной меткой класса на обучение.

Классификация сообщения в трафике

I. Произвести предобработку документа*:
    1. Удалить стоп-слова, пунктуацию и неизвестные символы.
    2. Сгруппировать специфические лексемы: номера телефонов, любые др. номера, email, URL, денежные символы.
    3. Произвести Стемминг Портера.
    4. Представить документ в виде «bag of words» - семпла.
II. Произвести извлечение признаков:

- Email-модель: извлечь униграммы и биграммы, вычислить tf-idf-значения для них.
- SMS-модель: извлечь униграммы, вычислить tf-idf-значения для них.

III. Классифицировать подготовленный семпл:

- Email-модель:
  1. Запомнить прогнозы алгоритмов SGDAlg_LogLoss, PAA_I_Default, PAA_II_Default.
  2. Итоговый вердикт вывести мажоритарным голосованием.
- SMS-модель:
  1. Запомнить прогнозы алгоритмов SGDClf_Default, kNN_Default , RandomForest_Medium, RandomForest_Balanced.
  2. Итоговый вердикт посчитать дизъюнктивной агрегацией.

Эксплуатация моделей

- Устройства: смартфоны, компьютеры, серверы
- Потоковый фоновый режим работы на всех устройствах

2.4 Заключение

В результате исследования была построена предварительная методика обнаружения спама, которая может быть применена в реальной анти-спам системе. Однако, данных исследований недостаточно и необходимо создать прототип реального анти-спама для проведения более приближённых к реальности тестов.

Были сконструированы следующие этапы работы анти-спам системы:

- предобработка,
- извлечение признаков,
- обучение и прогнозирование: были выбраны лучшие алгоритмы.

3 ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ

В данной части представлено описание реализации исследовательского ПО. Данная часть посвящена непосредственно вопросам реализации алгоритмов и методик, программному обеспечению проведения исследований.

3.1 Обоснование выбора инструментов программирования для разработки ПО

Часть информации об этом изложена в разделе 1.3. Главными критериями для выбора описанного набора инструментов являются их популярность и информационная поддержка как производителей, так и всего сообщества разработчиков-пользователей. Visual Studio 2019 IDE была выбрана как уже знакомая среда разработки, однако у неё есть более сильный конкурент, который более популярен, это PyCharm IDE, однако преимуществ для выполнения данного проекта обнаружены не были. Минимальные требования к IDE были приведены в разделе 1.3.

Стоит упомянуть о вычислительной системе, на базе которой происходила разработка. Это компьютер на базе Windows 10 с процессором Ryzen 3700X и 32 GB ОЗУ, что позволяет быстро проводить даже ресурсоёмкие вычисления на процессоре. Поэтому удалённых вычислительных ресурсов (сервер) не понадобилось. Рассматривался вопрос использования GPU для распараллеливания вычислений, однако надёжных однозначных технических решений для этого найдено не было, видимо из-за избыточности этой затеи при тех алгоритмах МО, которые используются в данной работе.

3.2 Разработка ПО

3.2.1 Архитектура и её реализация

В ЯП Python программный код находится в модулях, модули содержат классы и функции.

Стоит заметить, что главное достоинство программы в контексте архитектуры – это независимость следующего функционала друг от друга (инкапсуляция):

- предобработка текста,
- извлечение признаков,
- алгоритм поиска лучших комбинаций,
- логирование,
- общие статические функции по решению некоторых задач,
- фильтрация результатов.

Таким образом, существует возможность не только быстро добавлять новые типы комбинаций, которые смогут задействовать старый обобщённый функционал, но и изменять независимо друг от друга предобработчики текста, экстракторы признаков, датасеты и параметры исследований.

Рассмотрим подробнее модули проекта: их назначение, классы и функции.

main_research

Точка входа в ПО. Этот модуль управляет исследовательским кодом проекта.

Не содержит классов, только функции и код, которые:

- выводят характеристики датасетов,
- обеспечивают поддержку автоматизированных тестовых сценариев,

определяют список генерируемых типов алгоритмических комбинаций, списки алгоритмов-одиночек, на основе которых они будут генерироваться.

algs

Содержит классы с алгоритмами МО, фактически являются классами-обёртками над классами библиотеки scikit-learn, которые и реализуют МО.

Главная задача класса - фиксация всех алгоритмов и модификаций и избавление вызывающего кода от подробностей, связанных с параметрами алгоритмов и модификаций (инкапсуляция гиперпараметров).

Применено наследование классов:
MLAlgorithm – родительский класс, который имплементирует 3 базовых метода: learn(X_train, y_train), predict(X_test), learn_predict(X_train, X_test, y_train). Все его классы-наследники реализуют алгоритмы и их модификации, но каждый из них переопределяет чаще всего лишь поле, определяющее классификатор (алгоритм МО, определённый классом из scikit-learn), а все остальное остаётся прежним – вызывающий код будет работать с тремя вышеописанными родительскими имплементациями. Переопределить вышеописанные 3 метода пришлось лишь в нескольких случаях – понадобились индивидуальные гиперпараметры для некоторых алгоритмов МО.

datasets_preprocessing

Определяет классы для предобработки датасетов.

Класс-родитель DatasetPreprocessors является абстрактным, поскольку его экземпляры создавать запрещено – он определяет общий функционал для предобработки, а именно некоторые общие стадии предобработки (например, стоп-слова удаляются в любых дасетах, как и происходит удаление пунктуации и неизвестных символов) – общие предобработчики, а также обеспечивает

сохранение результатов предобработки и загрузку ранее сохранённых результатов предобработки, если таковые имеются, что существенно ускоряет работу программы. Все классы-потомки называются именами соотв. датасетов, например KagleSMS2016DatasetPreprocessors. В каждом таком классе может быть множество предобработчиков, однако в данной работе более 1 предобработчика ни у одного датасета нет. Каждый такой предобработчик вызывает интересующий общий предобработчик, получает нужные данные из датасета (каждый датасет устроен по-разному, чтобы получить одни и те же данные, нужен разный код), определяет кодировку.

Сам процесс предобработки представляет собой поиск в структурах данных pandas, используя базовые функции языка, функции библиотеки NumPy и функции библиотеки Pandas и изменение, удаление информации. Активно применяются лямбда-функции, регулярные выражения.

После предобработки вызывающему коду возвращается Pandas Array с семплами в виде строк с набором слов («bag of words»).

feature_extraction

Определяет экстракторы признаков.

Имеется один абстрактный класс FeatureExtractors со статическими функциями и всего 3 метода, каждый из которых отвечает за свою уникальную векторизацию (вычисление признаков), а токенизация (какие признаки извлекаются) задаётся в каждом методе параметрами.

generic

Модуль общих статических методов и данных на разные общие случаи.

Содержит набор абстрактных классов, в каждом из которых есть статические функции или/и данные: ServiceData, DatasetInstruments,

CollectionsInstruments, MathInstruments, ServiceInstruments. Такое обобщение позволяет зафиксировать и централизованно управлять популярными решенными в коде подзадачами, чтобы не плодить кодовую тавтологию, когда одна и таже подзадача решается разными путями. Например, в классе MathInstruments расположен метод, отвечающий за генерацию подмножеств и без обобщения разработчик вполне может позабыть об уже имеющейся реализации и сделать новую, вторую, однако в этом нет никакого смысла.

logs

Определяет функционал по непосредственному размещению, контролю и записи в лог-файлы.

Реализован стандартный шаблон Singleton не стандартным способом. Шаблон Singleton позволяет использовать плюсы обычного класса в плане наличия контекста не допуская создания более 1 экземпляра класса. Имеем класс A, делаем его конструктор абстрактным, чтобы нельзя было напрямую создать экземпляры, однако делаем в его же составе приватное (недоступное вне класса) статическое поле класса A и контролирует посредством его единственность экземпляра класса A через публичную статическую функцию, которая отвечает за возвращение единственного экземпляра класса A и вызывает приватный конструктор класса A только в случае, если статическое поле не инициализировано. Однако в Python невозможно задать статическое поле типа класса, в котором это поле содержится. Был создан внешний класс LogsFileProvider, который имеет в своём составе внутренний класс __LogsFileProvider. Таким образом, мы сохраняем в статическом поле класса LogsFileProvider единственный экземпляр класса __LogsFileProvider и работаем с его функционалом.

В сущности, задействованная в данном модуле библиотека logging предполагает взаимодействие с лог-файлами через байтовый файловый стриминг – получается дескриптор файла от ОС и файл занимается процессом

программы. Важно данный файл освободить в конце работы с ним, чтобы не провоцировать ошибки.

ml_algs_validation

Данный модуль является самым оригинальным и целевым для данной работы. Он занимается генерацией комбинаций на основе алгоритмов-одиночек, кросс-валидацией алгоритмов МО, фильтрацией результатов и их экспортом (логированием). Это самый большой по объёму и самый сложный модуль.

Одним из родительских классов модуля является AlgsCombinationsValidator. Он определяет общие данные для валидации и функционал, а также задаёт абстрактные методы, нуждающиеся в определении дочерними классами:

- параметры округления метрик при логировании,
- разбиения на фолды,
- максимальная длинна комбинации,
- список названий метрик обнаружения и производительности,
- управление дочерним валидаторами, фильтрацией и логированием,
- задаёт общие для всех валидаторов методы фильтрации результатов,
- сортирует результаты,
- форматирует данные для экспорта,
- управляет выбором имеющихся лог-файлов,
- рассчитывает метрики обнаружения,
- округляет значения метрик,
- определяет внутренний класс AlgsCombination, который является родителем для дочерних классов, определяющих комбинации алгоритмов.

Его потомками являются 2 дочерних класса: ComplexCombinationsValidator и TrivialCombinationsValidator. Принципиально

отличаются тем, что валидатор тривиальных комбинаций адаптирован для запоминания результатов алгоритмов-одиночек и комбинирования этих результатов, используя простые логические операции, в то время как валидатор усложнённых комбинаций использует меньше оригинальных решений, ведь использует чаще классы scikit-learn, которые определяют некоторые усложнённые комбинации, однако ничего не мешает написать и более оригинальный код, если в этом будет смысл. Также, он не использует результаты алгоритмов-одиночек, поскольку в этом нет для него никакого смысла, т.е. он гораздо большее кол-во раз нагружает процессор тем, что для поиска вердиктов комбинаций каждый раз обучает и валидирует алгоритмы МО на всех фолдах. В этом нет большой проблемы в теории, поскольку используемые для комбинаций агрегирующие алгоритмы МО классы scikit-learn имеют встроенную функцию многопоточности, однако на текущий момент она не работает – это проблема на стороне библиотеки. Т.е. бэггинг с оригинальным состав алгоритмов МО я не могу распараллелить, а Random Forest, реализуемый др. классом данной библиотеки, который является бэггингом с деревьями решений, без проблем распоточивается.

Подробнее рассмотрим класс AlgsCombination. Он определяет общие свойства (поля) и методы, описывающие любую комбинацию, будь то усложнённую или тривиальную:

- метрики качества (тип dict – словарь, где названию метрики сопоставлено значение),
- имена алгоритмов-участников (тип list – список, может быть не одномерным, для стекинга, например, можно инициализировать для этого поля список список, где каждый подсписок будет отражать слой стекинга),
- объекты для работы с алгоритмами-участниками (list, гибкость аналогичная предыдущему свойству),
- размер комбинаций – кол-во алгоритмов, в зависимости от сложности комбинации размер может быть и кортежем, когда

показан размер на каждом уровне, например на каждом слое стекинга,

- абстрактный метод создания имени комбинации.

Это пример, когда код обобщён даже для комбинаций, которые в перспективе могут быть реализованы, но которых ещё нет в программе. Это важно сколько не из-за перспектив, даже если они есть, а из-за общей культуры разработки, когда архитектура должна предусматривать больше, чем непосредственно задумывается на реализацию, ведь архитектура отражает не запросы разработчика или заказчика, а продумывается как отражение с рядом упрощений существующих вещей со всеми их свойствами, поведением и иерархией.

Дочерними классами являются ComplexCombination, который определяет дополнительно поле классификатора – инициализируется объектом алгоритма МО, содержит дополнительно общие для всех методы fit и predict, а так же TrivialCombination, который имеет подкласс-перечисление (тип enum) Type со значениями SINGLE (алгоритмы-одиночки для упрощения кода рассматриваются как тривиальные комбинации длинной 1), DISJUNCTIVE, CONJUNCTIVE, MAJORITY (это разновидности тривиальных комбинаций) и подкласс TrivialCombiTask, который определяет тривиальную комбинацию, поданную на многопроцессную обработку (последняя так и не была доведена до финальной стадии). Дочерние классы ComplexCombination определяют разновидности усложнённых комбинаций, т.е.: BaggingCombination, при дальнейшем развитии это ещё могут быть BoostingCombination, StackingCombination.

В каждом классе-валидаторе заданы в виде полей структуры данных, хранящие комбинации тех типов, которыми валидатор занимается. Важно то, что структура данных – не списки, а словари (dict), важной особенностью которого является константный доступ по ключу, т.е. быстрый поиск нужной комбинации со всеми её свойствами (в этих словаря названию комбинации соответствует объект комбинации со всей информации о ней). При необходимости, из словаря можно создать список с итератором, при этом такой список будет работать

именно с данными словаря, а не делать в памяти копию данных словаря, что очень важно. В виде словаря комбинации представляются куда чаще и это экономит ресурсы компьютера.

Вторым родительским классом модуля является CombinationsFiltration. Фильтрация – процесс, не связанный с валидаторами, поэтому в их составе код фильтрации находиться не должен, зато он связан с разновидностями комбинаций. Имеем следующие абстрактные классы:

- CombinationsFiltration: определяет общие функции фильтрации для комбинаций, если необходимо разработать уникальные реализации или другой фильтрующий функционал, используются дочерние классы:
    - TrivialCombisFiltration,
    - BaggingCombisFiltration.

Третьим внешним классом является TrivialCombisMultiprocessingValidator (он не просто так не является потомком AlgsValidators, поскольку в его задачи входит исключительно многопроцессная валидация, и он должен быть максимально независимым от окружающего кода), было написано 2 его варианта, однако подробно они не будет описаны, т.к. ни один из них не были доведены до финального статуса.

3.2.2 Реализация обработчика автоматизированных тестовых сценариев

На рисунке 27 показан фрагмент кода, показывающий как заданы некоторые сценарии.

```
test_scenarios = {
    'K2017_Email pr1 Tfidf1(ngram=(1,1))': #DONE
    ( kagle2017_preproc1, (FeatureExtractors.extractor_tfidf_1, {'ngram_range':(1,1)}), {} ), #( (
    'K2017_Email pr1 Tfidf1(ngram=(1,2))': #для доказательства, что лучше ngram=(1,2), чем (1,1)
    ( kagle2017_preproc1, (FeatureExtractors.extractor_tfidf_1, {'ngram_range':(1,2)}), {} ), #DONE
    'K2017_Email pr1 Tf1(ngram=(1,2))': #для доказательства, что tfidf1 лучше при тех же n-граммах
    ( kagle2017_preproc1, (FeatureExtractors.extractor_tf_1, {'ngram_range':(1,2)}), {} ), #DONE
    'K2017_Email pr1 Counts1': #для доказательства, что tf1 и tfidf1 лучше #DONE
    ( kagle2017_preproc1, (FeatureExtractors.extractor_words_counts_1, {}), {} ),
```

Рисунок 27 – Пример задания в коде тестовых сценариев

Как можно видеть, сценарий представляет собой словарь, в котором ключ – это название сценария, а значение – кортеж, который состоит из ссылки на функцию предобработки, ссылки на функцию экстракции признаков вместе с набором параметров экстрактора, упакованных в словарь и словаря с дополнительными параметрами для валидатора. Далее в цикле данные элементы словаря будут обрабатываться, аргументы из словарей будут распакованы оператором ** - фрагмент соответствующего кода показан на рисунке 28.

```
for test_name, ( (corpus,y), (extractor_func, extractor_params), research_params ) in test_scenarios.items():
    print('////////////////////////////////////' + test_name)
    print('////////////////////////// preprocessing done')
    X = extractor_func(corpus, **extractor_params) #corpus -> X
    print('////////////////////////// feature extraction done')
```

Рисунок 28 – Фрагмент кода обработчика тестовых сценариев

В конце каждой итерации, когда закончена обработка тестового сценария, необходимо поместить результаты в отдельный каталог, каталог называется именем сценария. Остаётся удалить файлы в каталоге с текущими логами. Проблема в том, что удалить файлы, находящиеся под контролем программы невозможно, дескрипторами этих логов занимается модуль logging. Прежде чем такие файлы удалять, необходимо снять блокировку на стороне модуля логирования – это делается путем удаления хэндлера (обработчика) каждого лога на стороне модуля, далее следует удаление самого лог-файла, потом снова добавляется в модуль новый хэндлер для лог-файла с таким же названием.

3.3 Оценка возможностей разработанного ПО при обработке вредоносного спама и безвредных сообщений

3.3.1 Положительные особенности созданного ПО

- Эффективные вспомогательные алгоритмы (не касающиеся непосредственно области МО) и структуры данных:

    - алгоритмы фильтрации,
    - работа через словари с комбинациями алгоритмов,
    - сортировка результатов,
    - организация расчётов тривиальных комбинаций с использованием запомненных результатов,
    - запоминание результатов предобработки датасета.
- Алгоритмы МО распараллелены средствами библиотеки scikit-learn.
- ООП: наследование, инкапсуляция, полиморфизм - модульность и универсальность за счет обобщения многих функций.
- Полноценное использование арсенала программной инженерии - были использованы в том числе относительно не популярные решения: шаблон «Singleton», шаблон многопоточного программирования «producer-consumers», внутренние классы, упаковка/распаковка аргументов функций, передача в виде аргументов ссылок на функции и классы, регулярные выражения.
- ПО качественно протестировано: реализована фиксация сложных вычислений, которые производит программа. Для этого в соответствующих местах кода используется функционал логирования вычислений, после чего логи вычислений проверяются разработчиком.

3.3.2 Недостатки созданного ПО

Не реализована многопроцессная обработка тривиальных комбинаций

Т.е. процесс связывания (комбинирования, агрегации) результатов работы алгоритмов-участников комбинаций и расчёт метрик качества можно было распараллелить. Была достигнута высокая степень готовности, для многопроцессного варианта алгоритма валидации и старого однопоточного была реализована единая кодовая база, однако до конца многопроцессный алгоритм

не был отлажен. Использовался модуль processing (а не threading, поскольку интерпретатор Python не поддерживает многопоточность в связи с концепцией Global Interpreter Lock (GIL) [14], см. рисунок 29 – в этом и заключается дополнительная техническая сложность (многопоточная реализация успешно функционировала, в отличие от многопроцессной, но многопоточность в лучшем случае не ускоряет вычисления, в худшем – существенно замедляет), расчёты ведутся подпроцессами – самостоятельными экземплярами интерпретатора ЯП) и целых 2 варианта реализации, один из них включал в себя реализацию шаблона многопоточности «producer-consumers», а другой минимизирует потенциальные проблемы при многопроцессности тем, что исключает работу с каким-либо общим ресурсом данных в многопоточном режиме (задачи каждому подпроцессу формировались заранее и друг от друга эти блоки задач были независимыми структурами данных, не требующие ни атомарности, ни использования монитора для контроля однопоточности доступа к общему ресурсу), однако это не помогло решить проблемы. Это не критично, однако не позволяет провести в короткое время более масштабные эксперименты.

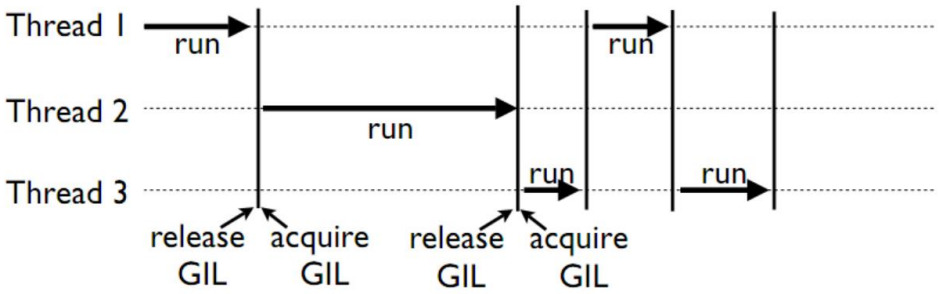

Рисунок 29 – Интерпретатор Python подчиняется механизму GIL

Все параметры валидации должны быть в одном месте

Например, в конфигурационном файле или в управляющем коде. Это необходимо для облегчения управления.

Подсчет метрик производительности

Необходимо использовать функционал ЯП для фиксации операций на процессоре, т.е. процессорное время, а не использовать функцию определения времени. Это повышает точность и объективность исследований. Организационными мерами не удастся сохранить всегда более-менее одинаковые условия – свободу ресурсов процессора и влияние др. факторов ОС и стороннего ПО (антивирусного, например) не исключены. Поэтому даже для дифференциации требовательности к ресурсам алгоритмов МО недостаточно реализованного функционала.

Неполноценная фильтрация ремувером избыточных комбинаций

См. рисунок 30. Механизм не учитывает коммутативность названий, когда перестановка алгоритмов в названии должно продолжать указывать на одну и ту же комбинацию. Поэтому часть избыточных комбинаций не обнаруживается. С алгоритмами, занявших первое место, работают алгоритмы МО, которые в итоге как минимум не приводят к улучшению метрик, а парой приводят к ухудшению. Эта проблема смягчена с помощью ручной фильтрации результатов.

```
---LinearSVCAlg_MoreSupports * PAA_I_Default
['f1=0.9809',   'auc=0.9875',   'acc=0.9888',   'prec=0.9776',   'rec=0.9842',   'train_time=0.0209049',
'pred_time=0.0003']
---LinearSVC_Balanced * LinearSVCAlg_MoreSupports * PAA_I_Default
['f1=0.9809',   'auc=0.9875',   'acc=0.9888',   'prec=0.9776',   'rec=0.9842',   'train_time=0.0561129',
'pred_time=0.0004']
---LinearSVC_Default * LinearSVCAlg_MoreSupports * PAA_I_Default
['f1=0.9809',   'auc=0.9875',   'acc=0.9888',   'prec=0.9776',   'rec=0.9842',   'train_time=0.0468108',
'pred_time=0.0006']
---LinearSVC_Default * LinearSVC_Balanced * LinearSVCAlg_MoreSupports * PAA_I_Default
['f1=0.9809',   'auc=0.9875',   'acc=0.9888',   'prec=0.9776',   'rec=0.9842',   'train_time=0.0820188',
'pred_time=0.0007']
```

Рисунок 30 – Пример не удалённых избыточных комбинаций в финальном логе

3.4 Заключение

В результаты работы над реализацией исследовательского проекта было получено функциональное и в целом производительное ПО (в необходимой степени).

От качества ПО зависит ценность результатов исследования и обширность экспериментов. В некоторой степени, разработанное ПО ограничивает исследователя, однако необходимый перечень испытаний был выполнен.

Большим преимуществом является хорошо обобщённая модульная архитектура с применением ООП – это позволяет использовать это ПО другим исследователям и модернизировать его часто без особых затрат.

4 ОРГАНИЗАЦИОННО-ЭКОНОМИЧЕСКАЯ ЧАСТЬ

Разработка программного обеспечения предполагает необходимость координации значительного количества весьма разноплановых работ, в которых принимают участие специалисты различного профиля и квалификации. Необходимость обеспечения эффективности разработки требует формирования единого плана, предусматривающего окончание всего комплекса работ и отдельных его составляющих в заданные сроки и при ограниченных издержках.

Анализ предстоящей разработки целесообразно проводить, представляя работу в виде экономико-функциональных блоков, что позволяет спланировать деятельность оптимальным образом и обоснованно спрогнозировать конкретные сроки выполнения отдельных этапов работы. Построение диаграммы Гантта позволяет наглядно представить последовательные и параллельные участки, продолжительность и очерёдность работ.

Перед этим подробнее стоит осветить условия лицензии: без срока, выдаётся компании под все её аппаратные потребности (1 на всё), любое количество устройств (это позволяет избежать контроля за соблюдением лицензионного соглашения, затрат на борьбу с кражей интеллектуальной собственности). Возможность самостоятельного дообучения моделей теоретически позволяет использовать его продолжительное время после выпуска на рынок. Система должна иметь на старте продвинутые возможности обнаружения, а далее будет дообучаться пользователем под его потребности. Данная система отлично подойдёт для целевой группы, описанной ниже.

Целевая группа продукта: производители социальных сервисов, которым требуется не обладающая передовыми возможностями (обычно так и происходит, поскольку целевые атаки – удел средних и крупных организаций), но автономная, независимая и недорогая при использовании на большом кол-ве устройств система. Таким образом, она будет помощником модератора, который будет собирать образцы необнаруженного спама и дообучать систему, чтоб на следующий день работы сервиса прежний спам уже не смог разместится. При

это часть спама система будет искать сама, потому что спам куда менее изменчив, как и языковые структуры и принципы, нежели вредоносное ПО.

Таким образом, разработчикам предстоит решить сложные технические задачи, которые сильно удешевят проект.

4.1 Определение основных этапов процесса разработки ПО для разработки анти-спама и расчёт трудоёмкости

Общие затраты труда на разработку и внедрение проекта определим следующим образом:

$$Q_P = \sum_i T_i \tag{32}$$

где T_i – затраты труда на выполнение i-го этапа проекта.

Используя метод экспертных оценок, вычислим ожидаемую продолжительность работ T каждого этапа по формуле:

$$T = \frac{3 \cdot T_{min} + 2 \cdot T_{max}}{5} \tag{33}$$

где T_{min} и T_{max} – максимальная и минимальная продолжительность работы. Они назначаются в соответствии с экспертными оценками, а ожидаемая продолжительность работы рассчитывается как математическое ожидание для β – распределения.

Полный перечень работ с разделением их по этапам приведён в таблице 3.

Таблица 3 – Разделение работ по этапам.

Этап	Работа	T_{min} ч/ч	T_{max} ч/ч	T ч/ч	T ч/дн
1. Разработка технических требований	1. Получение задания, анализ полученных требований к разрабатываемому ПО	8	8	8	1
	2. Разработка, согласование и утверждение ТЗ	16	40	28	4
	3. Анализ предметной области	32	48	40	5
	4. Анализ алгоритмов МО и сопутствующих технологий	32	48	40	5
2. Дизайн ПО	5. Разработка архитектуры ПО	16	32	24	3
	6. Разработка алгоритмов предобработки текста	24	48	36	5
	7. Разработка алгоритмов извлечения признаков	40	56	48	6
	8. Разработка алгоритмов МО и моделей	48	72	60	8
3. Разработка ПО	9. Программная реализация модулей обнаружения	48	64	56	7
	10. Программная реализация конечного ПО	56	72	64	8
4. Тестирование ПО	11. Тестирование модулей обнаружения	36	48	42	6
	12. Тестирование конечного ПО	32	56	44	6
5. Разработка документации	13. Разработка программной и эксплуатационной документации	48	64	56	7
				546	71

$$Q_P = Q_{0Ж} = 71 \text{ чел/дней} = 546 \text{ чел/час.}$$

4.2 Определение численности и квалификации исполнителей

Для оценки возможности выполнения проекта нужно рассчитать среднее количество исполнителей, которое при реализации проекта определяется соотношением:

$$N = \frac{Q_P}{F} \quad (34)$$

где Q_P – затраты труда на выполнение проекта (разработка и внедрение ПО), F – фонд рабочего времени; определяется по следующей формуле:

$$F = T \cdot F_M = T \cdot \frac{t_P \cdot (D_К - D_В - D_П)}{12} \quad (35)$$

где F_M – фонд времени в текущем месяце, который рассчитывается из учёта общего числа дней в году, числа выходных и праздничных дней, t_P – продолжительность рабочего дня, $D_К$ – общее число дней в году, $D_В$ – число выходных дней в году, $D_П$ – число праздничных дней в году.

Таким образом, фонд времени в текущем месяце составляет:

$$F_M = \frac{8 \cdot (365 - 93 - 25)}{12} = 165 \ \frac{\text{часов}}{\text{мес.}} \quad (36)$$

Время выполнения проекта (T) – 3 месяца, следовательно величина фонда рабочего времени составляет 495 часов (F).

Среднее количество исполнителей равно:

$$N = \frac{546}{495} = 1{,}1 \text{ человек}$$

Округляя до большего, получим число исполнителей проекта $N = 2$.

Поиск специалистов, обладающих знаниями и опытом работы в перечисленных выше областях, займет продолжительное время, что может отрицательно сказаться на сроках завершения проекта. Кроме того, заработная плата данного исполнителя будет высока. Это приведет к повышению общей стоимости проекта, а, следовательно, и к повышению стоимости конечного продукта. Предварительный анализ перечня предстоящих работ показал, что для сокращения сроков реализации данного проекта некоторые его этапы могут выполняться параллельно.

Таким образом, для реализации проекта целесообразно привлечь двух специалистов:

- Machine Learning (ML)-инженер;
- Инженер-программист.

4.3 Построение сетевой модели и календарного графика выполнения работ

Для определения временных затрат и трудоёмкости разработки ПО, используем метод сетевого планирования. Метод сетевого планирования позволяет установить единой схемой связь между всеми работами в виде наглядного и удобного для восприятия изображения (сетевого графика), представляющего собой информационно-динамическую модель, позволяющую определить продолжительность и трудоёмкость, как отдельных этапов, так и всего комплекса работ в целом.

Составление сетевой модели включает в себя оценку степени детализации комплекса работ и определения логической связи между отдельными работами. С этой целью составляется перечень всех основных событий и работ. Основные события и работы проекта представлены в таблице 4. Рассчитанные оставшиеся параметры элементов сети (сроки наступления событий, резервы времени событий, полный и свободный резервы времени работ) приведены в таблице 5.

Таблица 4 – Основные события и работы проекта

N_i	Наименование события	Код работы	Работа	Т ч/ч	Т ч/дн
0	Разработка ПО начата	0-1	Получение задания, анализ полученных требований к разрабатываемому ПО	8	1
1	Анализ полученных требований к разрабатываемому ПО проведён	1-2	Разработка, согласование и утверждение ТЗ	28	4
2	ТЗ разработано, согласовано и утверждено	2-3	Анализ предметной области	40	5
3	Анализ предметной области проведён	3-4	Анализ алгоритмов МО и сопутствующих технологий	40	5
4	Анализ алгоритмов МО и сопутствующих технологий проведен	4-5	Разработка архитектуры ПО	24	3
5	Разработка архитектуры ПО проведена	5-6	Разработка алгоритмов предобработки текста	36	5
6	Разработка алгоритмов предобработки текста проведена	6-7	Разработка алгоритмов извлечения признаков	48	6
		6-8	Разработка алгоритмов МО и моделей	60	8
7	Разработка алгоритмов извлечения признаков завершена	7-10	Программная реализация модулей обнаружения	56	7
8	Разработка алгоритмов МО и моделей завершена	8-9	Программная реализация конечного ПО	64	8
9	Программная реализация конечного ПО	9-10	Фиктивная работа	0	0
10	Программная реализация конечного ПО завершена	10-11	Тестирование модулей обнаружения	42	6

Продолжение таблицы 4

N_i	Наименование события	Код работы	Работа	Т ч/ч	Т ч/дн
11	Тестирование модулей обнаружения завершено	11-12	Тестирование конечного ПО	44	6
12	Тестирование конечного ПО завершено	12-13	Разработка программной и эксплуатационной документации	56	7
13	Разработка ПО завершена				

Таблица 5 – Временные затраты на каждый этап работы

N_i	Код работы i - j	t_{ij} чел/день	T_i^P чел/день	$T_i^П$ чел/день	R_i чел/день	$R_{ij}^П$ чел/день	R_{ij}^C чел/день
0	0-1	1	0	0	0	0	0
1	1-2	4	1	1	0	0	0
2	2-3	5	5	5	0	0	0
3	3-4	5	10	10	0	0	0
4	4-5	3	15	15	0	0	0
5	5-6	5	18	18	0	0	0
6	6-7	6	23	23	0	3	0
	6-8	8				0	0
7	7-10	7	29	32	3	3	3
8	8-9	8	31	31	0	0	0
9	9-10	0	39	39	0	0	0
10	10-11	6	39	39	0	0	0
11	11-12	6	45	45	0	0	0
12	12-13	7	51	51	0	0	0
13	-	-	58	58	0	-	-

Ранний срок совершения события T_j^P определяет минимальное время, необходимое для выполнения всех работ, предшествующих данному событию и равен продолжительности наибольшего из путей, ведущих от исходного события к рассматриваемому:

$$T_j^P = max(T_i^P + t_{ij}) \quad (37)$$

Поздний срок совершения события T_i^P – это максимально допустимое время наступления данного события, при котором сохраняется возможность соблюдения ранних сроков наступления последующих событий. Поздние сроки равны разности между поздним сроком совершения *j*-го события и продолжительностью *i-j* работы:

$$T_i^{\text{П}} = min(T_j^{\text{П}} - t_{ij}) \quad (38)$$

Критический путь – это максимальный путь от исходного события до завершения проекта. Его определение позволяет обратить внимание на перечень событий, совокупность которых имеет нулевой резерв времени.

Все события в сети, не принадлежащие критическому пути, имеют **резерв времени** R_i показывающий на какой предельный срок можно задержать наступление этого события, не увеличивая сроки окончания работ:

$$R_i = T_i^{\text{П}} - T_i^{\text{Р}} \quad (39)$$

Полный резерв времени работы $R_{ij}^{\text{П}}$ и **свободный резерв времени** R_{ij}^C работы можно определить, используя следующие соотношения:

$$R_{ij}^{\text{П}} = T_j^{\text{П}} - T_i^{\text{Р}} - t_{ij} \quad (40)$$

$$R_{ij}^C = T_j^P - T_i^{\text{Р}} - t_{ij} \quad (41)$$

Полный резерв работы показывает максимальное время, на которое можно увеличить длительность работы или отсрочить её начало, чтобы не нарушился срок завершения проекта в целом. Свободный резерв работы показывает максимальное время, на которое можно увеличить продолжительность работы или отсрочить её начало, не меняя ранних сроков начала последующих работ.

Сетевой график приведён ниже на рисунке 31.

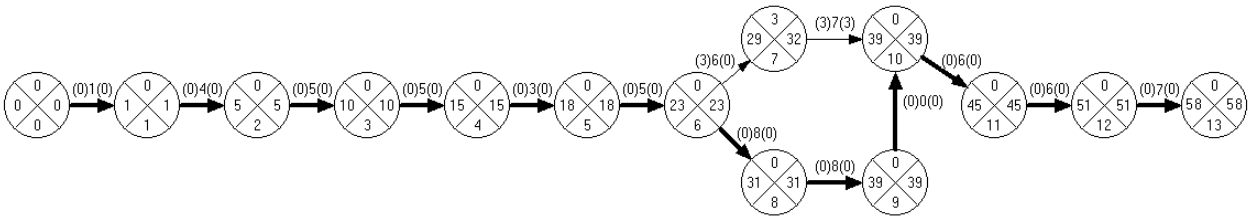

Рисунок 31 – Сетевой график выполнения работ

Как видно из сетевого графика, критический маршрут проходит через вершины 0-1-2-3-4-5-7-8-9-10-11-12-13 и имеет длину $T_{кр}$ = 58 рабочих дней.

Для отображения последовательности проводимых работ приведём диаграмму Гантта (рисунок 32). На рисунке 33 показаны исполнители проекта.

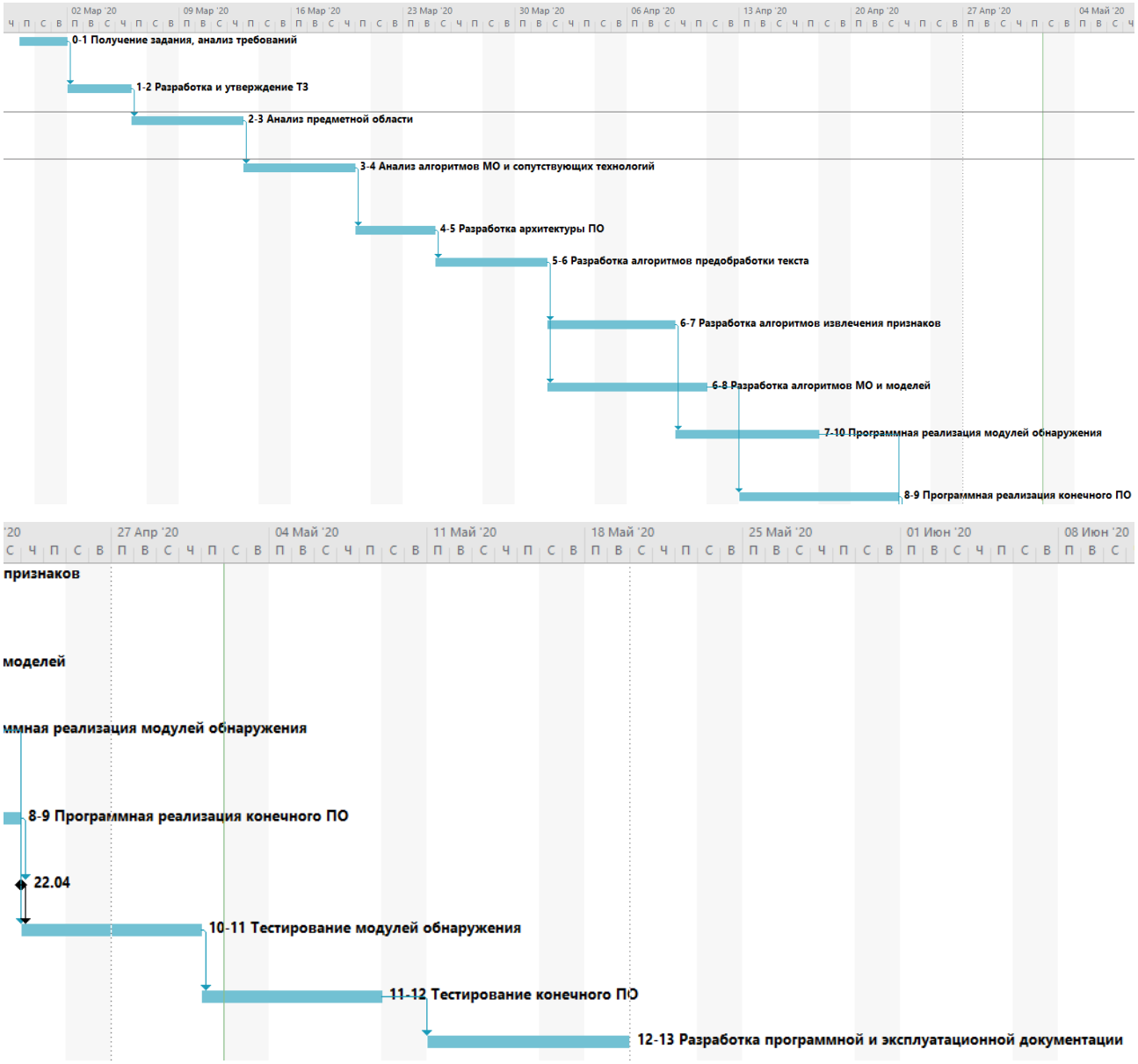

Рисунок 32 – Диаграмма Ганта (Microsoft Project 2016)

0-1 Получение задания, анализ требований	1 день	Пт 28.02.20	Вс 01.03.20	Инженер-программист
1-2 Разработка и утверждение ТЗ	4 дней	Пн 02.03.20	Чт 05.03.20	Инженер-программист
2-3 Анализ предметной области	5 дней	Пт 06.03.20	Чт 12.03.20	ML-инженер
3-4 Анализ алгоритмов МО и сопутствующих технологий	5 дней	Пт 13.03.20	Чт 19.03.20	ML-инженер
4-5 Разработка архитектуры ПО	3 дней	Пт 20.03.20	Вт 24.03.20	Инженер-программист
5-6 Разработка алгоритмов предобработки текста	5 дней	Ср 25.03.20	Вт 31.03.20	ML-инженер
6-7 Разработка алгоритмов извлечения признаков	6 дней	Ср 01.04.20	Ср 08.04.20	Инженер-программист
6-8 Разработка алгоритмов МО и моделей	8 дней	Ср 01.04.20	Пт 10.04.20	ML-инженер
7-10 Программная реализация модулей обнаружения	7 дней	Чт 09.04.20	Пт 17.04.20	ML-инженер
8-9 Программная реализация конечного ПО	8 дней	Пн 13.04.20	Ср 22.04.20	Инженер-программист
9-10 Фиктивная работа	0 дней	Ср 22.04.20	Ср 22.04.20	ML-инженер
10-11 Тестирование модулей обнаружения	6 дней	Чт 23.04.20	Чт 30.04.20	Инженер-программист
11-12 Тестирование конечного ПО	6 дней	Пт 01.05.20	Пт 08.05.20	Инженер-программист
12-13 Разработка программной и эксплуатационной документации	7 дней	Пн 11.05.20	Вт 19.05.20	Инженер-программист

Рисунок 33 – Исполнители проекта (Microsoft Project)

4.4 Анализ структуры затрат, исследование рынка, планирование цены и определение прибыли от реализации продукта

Структура затрат

Затраты на выполнение проекта могут быть представлены в виде сметы затрат, включающей в себя следующие статьи:

- заработная плата исполнителям;
- отчисления на социальные службы;
- материальные затраты;
- прочие затраты.

Затраты на выплату исполнителям заработной платы линейно связаны с трудоёмкостью и определяется следующим соотношением:

$$C_{ЗАРП} = C_{З.ОСН} + C_{З.ДОП} + C_{З.ОТЧ} \quad (42)$$

где $C_{З.ОСН}$ – основная заработная плата,

$C_{З.ДОП}$ – дополнительная заработная плата,

$C_{З.ОТЧ}$ – отчисление с заработной платы.

Расчёт основной заработной платы (оплаты труда непосредственных исполнителей):

$$C_{З.ОСН} = T_{ЗАН} * O_{ДН} \quad (43)$$

где $T_{ЗАН}$ – число дней, отработанных исполнителем проекта,

$O_{ДН}$ – дневной оклад исполнителя.

При 8-и часовом рабочем дне он рассчитывается по соотношению:

$$O_{ДН} = \frac{O_{МЕС} \cdot 8}{F_М} \quad (44)$$

где $O_{МЕС}$ – месячный оклад,

$F_М$ – месячный фонд рабочего времени.

С учётом налога на доходы физических лиц размер оклада увеличивается, что отражено в формуле:

$$О_{МЕС} = О \cdot (1 + \frac{Н_{НДФЛ}}{100}) \quad (45)$$

где О – «чистый» оклад,

$Н_{НДФЛ}$ – налог на доходы физических лиц в размере 13%.

Сведём результаты расчёта в таблицу 6 с перечнем исполнителей и их месячных и дневных окладов, а также времени участия в проекте и рассчитанной основной заработной платой каждого исполнителя:

Таблица 6 – Заработная плата исполнителей

№	Должность	«Чистый» оклад, руб.	Дневной оклад, руб.	Трудозатраты, чел/дни	Затраты на зарплату, руб.
1	Инженер-программист	100 000	4849	47	227 903
2	ML-инженер	140 000	6787	24	162 888

Из таблицы получим общие затраты проекта на заработную плату исполнителей: $С_{З.ОСН} = 390\ 791$ руб.

Расходы на дополнительную заработную плату учитывают все выплаты непосредственным исполнителям за время, не проработанное на производстве, но предусмотренное законодательством РФ. Величина этих выплат составляет 20% от размера основной заработной платы:

$$С_{З.ДОП} = 0{,}2 \cdot С_{З.ОСН} \quad (45)$$

Расходы на дополнительную заработную плату составят:

$$С_{З.ДОП} = 0{,}2 \cdot 390\ 791 = 78\ 158{,}2 \text{ (руб.)}$$

Отчисления на соц. нужды

Согласно нормативным документам, суммарные отчисления в пенсионный фонд, фонд социального страхования и фонды обязательного медицинского страхования составляют 30% от размеров заработной платы.

$$С_{З.ОТЧ} = 0{,}3 \cdot С_{З.ОСН} + С_{З.ДОП} = 195\,395{,}5 \text{ (руб.)} \quad (46)$$

Общие расходы на заработную плату составляют:

$$С_{ЗАРП} = С_{З.ОСН} + С_{З.ДОП} + С_{З.ОТЧ} = 664\,344{,}7 \text{ (руб.)} \quad (47)$$

Материальные затраты

Материальные затраты представлены в таблице 7.

Таблица 7 – Материальные затраты

№	Наименование	Ед. изм.	Кол-во	Цена за ед., руб. (стоимость аренды, руб/мес)	Длительность аренды, мес	Сумма, руб.
1	Компьютер на Ryzen 3600	Шт.	2	70000	Покупается	140000
2	МФУ HP LaserJet Pro MFP M125nw	Шт.	1	7 271	Покупается	7 271
3	Беспроводной маршрутизатор Zyxel Keenetic Lite	Шт.	1	2390	Покупается	2390
4	Бумага Ballet Premier белая А4	Пачка, 500 л.	1	300	Покупается	300
Итого $С_{ОБ}$						149961

Прочие затраты

Материальные затраты представлены в таблице 8.

Таблица 8 – Прочите затраты

№	Наименование	Ед. изм.	Количество	Цена, руб.	Сумма, руб.
1	Услуги связи (интернет) 100мбит/с	месяц	3	500	1 500
Итого:					1 500

Затраты на организацию рабочих мест

Расчёт затрат, связанных с организацией рабочих мест для исполнителей проекта, проводится на основе требований СНИПа (санитарные нормы и правила) и стоимости аренды помещения требуемого уровня сервиса

В соответствии с санитарными нормами, расстояние между рабочими столами с видеомониторами должно быть не менее 2-х метров, а между боковыми поверхностями видеомониторов – не менее 1,2 метра. Площадь на одно рабочее место с терминалом или ПК должна составлять не менее 6 кв. м., а объём – не менее 20 куб. м. Расположение рабочих мест в подвальных помещениях не допускается. Помещения должны быть оборудованы системами отопления, кондиционирования воздуха или эффективной приточно-вытяжной вентиляцией. Таким образом, для размещения двух сотрудников и принтера необходимо помещение площадью 6+6+3=15 кв. м.

Затраты на аренду помещения можно вычислить исходя из следующего соотношения:

$$\mathrm{C_{ОРГ}} = \frac{\mathrm{C_{КВМ}}}{12} \cdot S \cdot \mathrm{T_{АР}} \quad (48)$$

где $C_{КВМ}$ – стоимость аренды одного кв. метра площади за год, S – арендуемая площадь рабочего помещения. $T_{АР}$ – срок аренды (мес).

Таким образом, аренда помещения площадью 15 кв. м. на срок в 3 месяца в районе станции метро «Площадь Ильича» составляет:

$$C_{ОРГ} = \frac{25\,000}{12} \cdot 17 \cdot 3 = 106\,250 \text{ (руб.)}$$

Суммарные затраты на реализацию проекта

Круговая диаграмма, отображающая структуру затрат проекта (см. таблицу 9), приведена на рисунке 34.

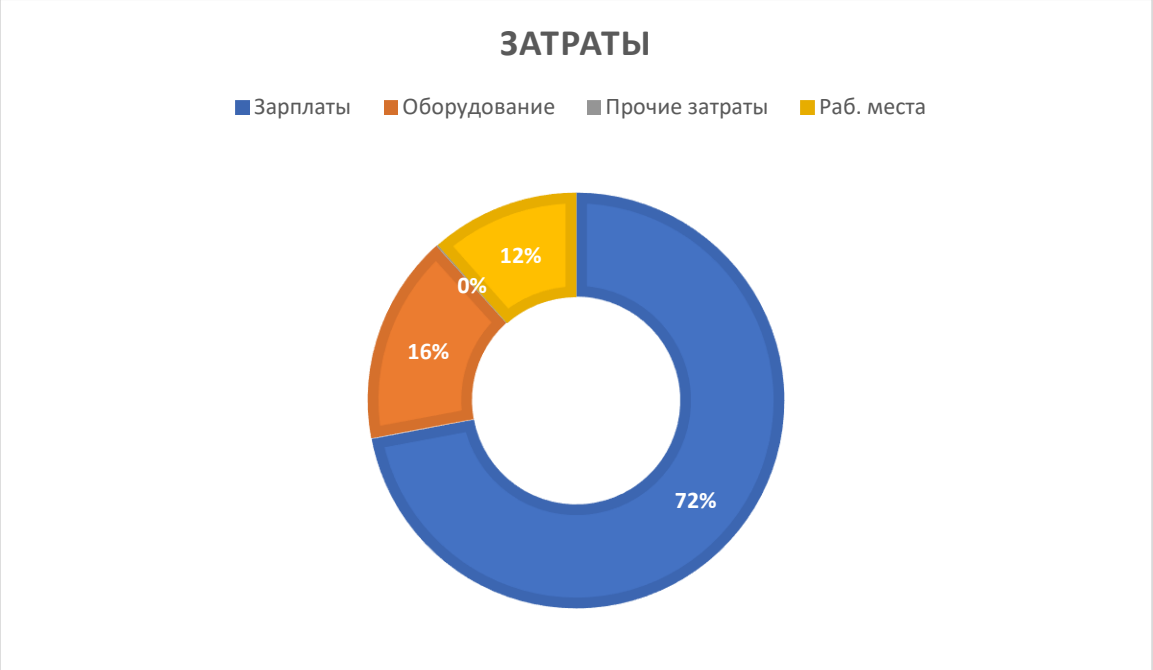

Рисунок 34 – Структура затрат проекта

Таблица 9 – Суммарные затраты на проект

№	Наименование статьи расходов	Затраты, руб.
1	Заработная плата исполнителям $С_{ЗАРП}$	664 344,7
2	Закупка и аренда оборудования $С_{ОБ}$	149 961
3	Прочие затраты	1500
4	Организация рабочих мест $С_{ОРГ}$	106 250
Суммарные затраты К		**922 056**

Сервисное обслуживание

Сервисное обслуживание разработанного программного обеспечения будет выполнять один сотрудник. Сюда входит исправление багов и дообучение моделей в случае их некорректной работы. Для такой работы очень важно нанять человека, имеющего опыт работы с проектом, т.е. имеющегося сотрудника. Для выполнения данной работы привлечём инженера-программиста, который участвовал в разработке ПО. Затраты на сервисное обслуживание приведены в таблице 10. Срок – 2 месяца.

Таблица 10 – Заработная плата исполнителей

№	Должность	«Чистый» оклад, руб.	Дневной оклад, руб.	Трудозатраты, чел/дни	Затраты на зарплату, руб.
1	Инженер-программист	100 000	4849	10	48 490

Расходы за дополнительную заработную плату учитывают все выплаты непосредственным исполнителям на время, не проработанное на производстве,

но предусмотренное законодательством. Величина этих выплат составляет 20% от размера основной заработной платы:

$$С_{З.ДОП} = 0{,}2 \cdot С_{З.ОСН} \quad (49)$$

Расходы на дополнительную заработную плату составят:

$$С_{З.ДОП} = 0{,}2 \cdot 48\,490 = 9\,698 \text{ (руб.)}$$

Согласно нормативным документам, суммарные отчисления в пенсионный фонд, фонд социального страхования и фонды обязательного медицинского страхования составляют 30% от размеров заработной платы.

$$С_{З.ОТЧ} = 0{,}3 \cdot С_{З.ОСН} + С_{З.ДОП} = 24\,245 \text{ (руб.)} \quad (50)$$

Общие расходы на заработную плату составляют:

$$С_{ЗАРП} = С_{З.ОСН} + С_{З.ДОП} + С_{З.ОТЧ} = 82433 \text{ (руб.)} \quad (51)$$

Т.е. затраты на сервисное обслуживание: 82433 рублей.

Исследование рынка

Анти-спам системы довольно популярны в качестве (составе) СЗИ. Однако совершенно другое дело недорогие решения по фильтрации для социальных сервисов: форумов, чатов в игровых сервисах, социальных сетей и т.д. Таким образом, есть ниша недорогих решений, которые не содержат в составе избыточные компоненты и позволяют исключительно фильтровать сообщения, не используя передовые технологии, однако и на сервер ничего отправляться не будет. В данном решении может сочетаться небольшая стоимость, простота развёртывания, приемлемая, но никак не передовая эффективность и конфиденциальность фильтруемых данных.

Целевая аудитория - небольшие команды разработчиков, не обладающие свободными финансовыми ресурсами, которые не хотят к тому же вместе с купленным решением получить др. риски и посторонний функционал – исключительно простая фильтрация, исключительно client-based и за небольшую стоимость.

Монетизироваться проект мог бы за счёт покупки обновляемых моделей обнаружения, стоимость обновлений можно постоянно изменять. Однако, была выбрана другая схема (для упрощения расчетов) – разовая оплата за лицензию. Однако технически клиент получает целый набор конфигураций моделей и встроенный модуль тестирования, где он может проверить эффективность системы на типичных для его трафика сообщениях и выбрать для себя оптимальную конфигурацию моделей. Так же, он получит возможность дообучать модели самостоятельно на основе своих сообщений. Это означает, что долгая поддержка проекту не требуется. Это позволяет сильно сократить расходы на него.

Планирование цены и прогнозирование прибыли

Частичная стоимость разработки, приходящаяся на каждую лицензию ПО, определяется исходя из данных о планируемом объеме установок:

$$\Delta K = \frac{K}{N_P^O} \cdot (1 + H_{\text{ст}}) \tag{52}$$

где K – стоимость проекта, N_P^O – планируемое число лицензий ПО, $H_{\text{ст}}$ – ставка банковского процента по долгосрочным кредитам (более одного года).

Приняв ставку процента по долгосрочным кредитам 13,2% (ПАО «Сбербанк») и используя полученные ранее значения, вычислим:

$$\Delta K = \frac{922\,056}{60} \cdot (1 + 0{,}132) = 17\,396{,}2 \text{ (руб).} \tag{24}$$

Установим значение $K_{\text{ПР}} = 21\,000$ рублей (с учетом прибыли).

Тем самым, сумма от продаж за год составит 60 · 21 000 = 1 260 000 рублей, что обеспечивает срок окупаемости проекта менее 1 года. Определим процент прибыли от одной реализации ПО по формуле:

$$D_{\text{ПРИБ}} = \left(\frac{K_{\text{ПР}}}{\Delta K + K_{\text{ВН}}} - 1\right) \cdot 100\%, \tag{53}$$

где $K_{\text{ВН}} = 0$ – затраты на внедрение.

Для данного проекта:

$$D_{\text{ПРИБ}} = \left(\frac{21000}{17\,396{,}2} - 1\right) \cdot 100\% = 20.7\%.$$

Сумма расчетной прибыли от продажи каждой лицензии ПО с учётом налога на добавочную стоимость $H_{\text{НДС}} = 20\%$:

$$\text{С}_{\text{ПРИБ}} = (\Delta K + K_{\text{ВН}}) \cdot D_{\text{ПРИБ}} \cdot \left(1 - H_{\text{НДС}}\right). \quad (54)$$

Для данного проекта:

$$\text{С}_{\text{ПРИБ}} = (17\,396{,}2\,) \cdot 0{,}207 \cdot (1 - 0{,}2) = 2880{,}8 \text{ (руб)}.$$

Для оплаты расходов на разработку ПО возьмем кредит в банке ПАО «Сбербанк» в размере 1 200 000 рублей на срок 24 месяца. Ежемесячный платеж по данному кредиту составляет 56 600 рублей. Сумма погашения кредита (с учетом комиссии за обслуживание кредита) составляет 1 358 400 рублей.

За первые три месяца разработки продажи равны нулю, т.к. продукт еще в разработке. При этом осуществляются выплаты заработной платы и производятся другие ранее рассчитанные расходы на разработку в размере K рублей.

Будем считать, что через три месяца после начала разработки за каждый последующий год продается 60 лицензий программы.

Фрагмент таблицы общего баланса приведен в таблице 11. Структура дохода показана на рисунке 35.

Таблица 11 – Таблица общего баланса

Месяц	Сальдо начальное по кредиту, руб.	Выплата по кредиту, руб.	Сальдо конечное по кредиту, руб.	Сумма на балансе, руб.	Прибыль от продаж, руб.	Затраты на разработку и обслуживание, руб.
Март 2020	-1358000	56 600	-1301400	1200000	0	407325.9
Апрель 2020	-1301400	56 600	-1244800	736074.1	0	257364.9
Май 2020	-1244800	56 600	-1188200	422109.2	0	257364.9
Июнь 2020	-1188200	56 600	-1131600	213144.3	105000	82433
Июль 2020	-1131600	56 600	-1075000	179111.3	105000	82433
Август 2020	-1075000	56 600	-1018400	145078.3	105000	0
Сентябрь 2020	-1018400	56 600	-961800	193478.3	105000	0
Октябрь 2020	-961800	56 600	-905200	241878.3	105000	0
Ноябрь 2020	-905200	56 600	-848600	290278.3	105000	0
Декабрь 2020	-848600	56 600	-792000	338678.3	105000	0
Январь 2021	-792000	56 600	-735400	387078.3	105000	0
Февраль 2021	-735400	56 600	-678800	435478.3	105000	0

Продолжение таблицы 11

Месяц	Сальдо начальное по кредиту, руб.	Выплата по кредиту, руб.	Сальдо конечное по кредиту, руб.	Сумма на балансе, руб.	Прибыль от продаж, руб.	Затраты на разработку и обслуживание, руб.
Март 2021	-678800	56 600	-622200	483878.3	105000	0
Апрель 2021	-622200	56 600	-565600	532278.3	105000	0
Май 2021	-565600	56 600	-509000	580678.3	105000	0
Июнь 2021	-509000	56 600	-452400	629078.3	105000	0
Июль 2021	-452400	56 600	-395800	677478.3	105000	0
Август 2021	-395800	56 600	-339200	725878.3	105000	0
Сентябрь 2021	-339200	56 600	-282600	774278.3	105000	0
Октябрь 2021	-282600	56 600	-226000	822678.3	105000	0
Ноябрь 2021	-226000	56 600	-169400	871078.3	105000	0
Декабрь 2021	-169400	56 600	-112800	919478.3	105000	0
Январь 2022	-112800	56 600	-56200	967878.3	105000	0
Февраль 2022	-56200	56 600	400	1016278.3	105000	0
Март 2022	400		400	1121278.3	105000	0

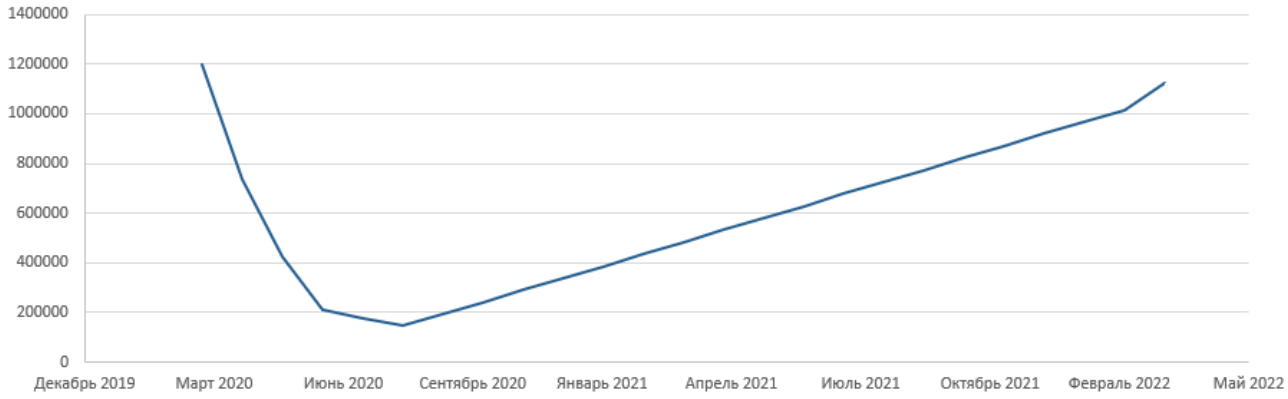

Рисунок 35 – График динамики баланса

4.5 Заключение

Результаты проведённых организационно-экономических расчётов позволили оценить структуру работ, необходимое количество исполнителей, структуру затрат проекта, срок окупаемости проекта.

1. Общие затраты труда для выполнения программного проекта составили 71 чел/дней или 546 чел/часов. Затраты на разработку ПО составляют 922 056 рублей.
2. Исходя из временных требований к реализации проекта, была определена численность исполнителей: 2 человека. По результатам построения сетевого графика и диаграммы Гантта, можно сделать вывод, что введение дополнительных разработчиков не принесёт дополнительного эффекта, поскольку основные этапы работы должны выполняться последовательно.
3. Из структуры затрат проекта видно, что основной статьёй расходов является заработная плата исполнителей (72%).
4. Стоимость продукта оценивается в 21000 рублей, при объеме спроса 60 копий в год. Планирование цены позволило спрогнозировать срок окупаемости проекта, который составляет 24 месяца.

На основании проделанной работы можно сделать вывод о целесообразности проведения работ и внедрения и производство данной разработки.

5 ОРГАНИЗАЦИОННО-ПРАВОВАЯ ЧАСТЬ

В данном разделе производится анализ действующей нормативных правовых актов РФ и нормативных документов, регулирующих правовое обоснование выполнения выпускной квалификационной работы.

Данная выпускная квалификационная работа затрагивает сферу правового регулирования персональных данных, разработки средств защиты информации (СЗИ).

Потенциально разрабатываемая и выводимая далее на рынок анти-спам система (а именно её технологическая часть рассматривается выше) должна использовать личные сообщения пользователей для обнаружения в них спама, больше никакая информация не нужна (используются модели на базе «bag of words»).

Неизбежно возникает вопрос сохранности конфиденциальности данных. Персональные данные не нужны для обнаружения спама, однако они могут содержаться в сообщениях и в словарях детектора однозначно они будут храниться, причем модификация словарей исключена несмотря на то, что они хранятся локально. Это сделано для того, чтобы можно было обновлять одни и те же модели у всех клиентов с одними и теми же словарями.

Необходимо учитывать, что потенциальный анти-спам располагает только client-based детектором (обучается он также локально), т.е. никакие сообщения не будут отправляться на удалённый сервер без ведома пользователя, он сам может пометить сообщение как спам и тогда будет создано локальное надмножество словаря для обнаружения спама, который характерен именно для пользователя, поскольку пользователи спамность трактуют по-своему (в данном случае важно не заменять/вырезать персональные данные, коль они попали в сообщение, потому что это улучшает чувствительность к спаму, который интересует конкретного пользователя). Всё происходит автоматизировано без участия какой-либо человеческой экспертизы.

В рамках данной работы не представляется возможным рассмотреть всю нормативно-правовую базу, поэтому будет рассмотрена только наиболее важная для данной работы её часть:

- Конституция РФ;
- 149-ФЗ от 27.07.2006 г. (ред. от 23.04.2018) «Об информации, информационных технологиях и о защите информации»;
- 152-ФЗ от 27.07.2006 (ред. от 29.07.2017) «О персональных данных»;
- Федеральный закон №99 «О лицензировании отдельных видов деятельности» от 04.05.2011 (ред. от 31.12.2017);
- Постановление Правительства РФ №608 от 26.06.1995 (ред. от 21.04.2010) «О сертификации средств защиты информации»;
- Положение о сертификации средств защиты информации по требованиям безопасности информации (утв. Приказом Гостехкомиссии РФ №199 от 27.10.1995;
- Доктрина информационной безопасности Российской Федерации.

5.1 Конституция РФ

Конституция — это единый, обладающий особыми юридическими свойствами нормативно-правовой акт, посредством которого народ учреждает основные принципы устройства общества и государства, закрепляет правовой статус человека и гражданина.

Согласно Конституции Российской Федерации [15] за гражданином закреплены следующие права и свободы:

- право на неприкосновенность частной жизни, личную и семейную тайну, защиту своей чести и доброго имени - ч. 1 ст. 23;
- право на тайну переписки, телефонных переговоров, почтовых, телеграфных и иных сообщений - ч. 2 ст. 23;
- свобода мысли и слова - ч. 1 ст. 29;

- свобода массовой информации - ч. 5 ст. 29;
- право на свободу выражения своих мнений и убеждений (никто не может быть принужден к выражению своих мнений и убеждений или отказу от них) - ч. 3 ст. 29;
- право свободно искать, получать, передавать, производить и распространять информацию любым законным способом - ч. 4 ст. 29;
- право граждан обращаться лично, а также направлять индивидуальные и коллективные обращения в государственные органы и органы местного самоуправления - ст. 33;
- свобода всех видов творчества - ч. 1 ст. 44;
- право на доступ к культурным ценностям - ч. 2 ст. 44.

Таким образом, граждане вправе обеспечивать защиту информации с использованием средств защиты информации, если это не противоречит нормативным актам Российской Федерации.

5.2 149-ФЗ «Об информации, информационных технологиях и о защите информации»

Данный ФЗ [16] определяет следующие 2 важных понятия:

- **информация** - сведения (сообщения, данные) независимо от формы их представления;
- **конфиденциальность информации** - обязательное для выполнения лицом, получившим доступ к определенной информации, требование не передавать такую информацию третьим лицам без согласия ее обладателя.

5.3 152-ФЗ «О персональных данных»

Данный ФЗ [17] регулирует отношения, связанные с обработкой персональных данных, осуществляемой органами государственной власти, юридическими лицами и физическими лицами.

В данном законе закреплены следующие определения:

- **персональные данные** - любая информация, относящаяся к прямо или косвенно определенному или определяемому физическому лицу (субъекту персональных данных);
- **оператор** - государственный орган, муниципальный орган, юридическое или физическое лицо, самостоятельно или совместно с другими лицами организующие и (или) осуществляющие обработку персональных данных, а также определяющие цели обработки персональных данных, состав персональных данных, подлежащих обработке, действия (операции), совершаемые с персональными данными;
- **обработка персональных данных** - любое действие (операция) или совокупность действий (операций), совершаемых с использованием средств автоматизации или без использования таких средств с персональными данными, включая сбор, запись, систематизацию, накопление, хранение, уточнение (обновление, изменение), извлечение, использование, передачу (распространение, предоставление, доступ), обезличивание, блокирование, удаление, уничтожение персональных данных [ссылка].

Необходимо разъяснить ряд определяющих нюансов о работе с персональными данными:

- Под персональными данными, как следует из статьи 3, подразумевается любая информация прямо или косвенно относящаяся к субъекту персональных данных (то есть физическому лицу). Таким образом, персональные данные — это персонализированная информация, которая

позволяет однозначно определить, о каком именно лице идет речь. *Используемая рассматриваемым анти-спамом информация может случайно содержать персональные данные, однако при автоматизированной обработке это не имеет значения.*

- Биометрические данные, предполагают использование оператором специальных технических средств для анализа биометрических данных субъекта и их сопоставления с эталонными образцами.
- Для применения специального режима обработки биометрических данных необходимо, чтобы их обработка осуществлялась именно оператором. Как отмечает Роскомнадзор, «отнесение сведений персонального характера к биометрическим персональным данным и их последующая обработка должны рассматриваться в рамках проводимых оператором мероприятий, направленных на установление личности конкретного лица». Если же биометрические данные хранятся и обрабатываются исключительно на устройстве пользователя, а оператору сообщаются лишь сведения о результатах идентификации, то говорить об обработке оператором биометрических данных нельзя». *Разрабатываемая система обрабатывает их без участия оператора.*
- Статья 5 часть 2 ФЗ №152 утверждает, что «обработка персональных данных должна ограничиваться достижением конкретных, заранее определенных и законных целей».
- Согласно статье 5 части 7 ФЗ №152 не разрешено хранить персональные данные на срок больший, чем этого требуют цели обработки персональных данных, кроме случаев, когда иной срок уставлен другими федеральными законами. При этом, в случае достижения целей обработки персональных данных, их следует удалить. *Хранятся персональные данные лишь на стороне самого пользователя в виде дополнительного словаря для обнаружения спама, специфичного для конкретного пользователя, они никуда не отправляются и их вид исключает какое-либо злонамерное их использование при утечке. Срок хранения определяется пользователем.*

- Статья 19 регламентирует меры по обеспечению безопасности персональных данных. Так, обеспечение безопасности персональных данных достигается, в частности с применением прошедших в установленном порядке процедуру оценки соответствия средств защиты информации. То есть здесь идет речь о необходимости сертификации СЗИ, обеспечивающих защиту персональных данных.

5.4 99-ФЗ «О лицензировании отдельных видов деятельности»

Статья 12 данного ФЗ [18] регламентирует виды деятельности, которые подлежат обязательному лицензированию:

- разработка, производство, распространение шифровальных (криптографических) средств, информационных систем и телекоммуникационных систем, защищенных с использованием шифровальных (криптографических) средств, выполнение работ, оказание услуг в области шифрования информации, техническое обслуживание шифровальных (криптографических) средств, информационных систем и телекоммуникационных систем, защищенных с использованием шифровальных (криптографических) средств (за исключением случая, если техническое обслуживание шифровальных (криптографических) средств, информационных систем и телекоммуникационных систем, защищенных с использованием шифровальных (криптографических) средств, осуществляется для обеспечения собственных нужд юридического лица или индивидуального предпринимателя);
- разработка и производство средств защиты конфиденциальной информации (под этот пункт могла бы попадать анти-спам система, но не попадает).

5.5 Постановление Правительства РФ №608 «О сертификации средств защиты информации»

Постановление Правительства РФ утвердило положение, устанавливающее порядок сертификации средств защиты информации в Российской Федерации. Согласно положению, криптографические средства для защиты информации, отнесенной к государственной тайне, подлежат обязательной сертификации и должны быть выполнены на основе криптографических алгоритмов, рекомендованных ФСБ РФ.

На анти-спам системы данное постановление не распространяется.

5.6 Положение о сертификации средств защиты информации по требованиям безопасности информации

Это положение распространяется на технические, программные и другие средства защиты информации, предназначенные для защиты информации, содержащей сведения, составляющие государственную тайну, от утечки, несанкционированных и непреднамеренных воздействий, несанкционированного доступа и от технической разведки, а также средства контроля эффективности защиты информации.

В положении в явном виде не прописана необходимость обязательной сертификации ПО анти-спам систем.

5.7 Доктрина информационной безопасности Российской Федерации

Доктрина информационной безопасности – это основной концептуальный документ, определяющий политику Российской Федерации в информационной сфере, содержание национальных интересов и потребность государства в обеспечении безопасности.

Доктрина утверждена Указом Президента Российской Федерации от 5 декабря 2016 г. №646 «Об утверждении Доктрины информационной безопасности Российской Федерации».

Принятие новой доктрины является важным этапом для законодательства Российской Федерации, ее утверждение указывает на изменившиеся приоритеты в безопасности, создает новые направления развития и основания для принятия новых законодательных актов.

Доктрина информационной безопасности не является нормативным правовым актом. Это документ стратегического назначения, где четко обозначены границы обеспечения ИБ, указаны направления взаимодействия всех структур. Она определяет стратегию РФ в информационной сфере на ближайшие годы и служит основой для совершенствования правового, методического, научно-технического и организационного обеспечения ИБ РФ. Положения доктрины должны быть реализованы в соответствующих законах и подзаконных нормативно-правовых актах [19].

В Доктрине информационной безопасности Российской Федерации от 5 декабря 2016 обозначены наиболее важные национальные интересы России в информационной сфере, среди которых обеспечение и защита конституционных прав и свобод человека, обеспечение устойчивого и бесперебойного функционирования информационной инфраструктуры и развитие информационных технологий [20].

Доктрина обозначает основные информационные угрозы в положениях Стратегии национальной безопасности РФ, касающихся:

- возрастающего противоборства в глобальном информационном пространстве;
- угроз нарушения безопасности и устойчивости функционирования критической информационной инфраструктуры РФ;
- деятельности, связанной с использованием информационных и коммуникационных технологий в экстремистской деятельности;

- а также импортозамещения и снижения критической зависимости от зарубежных технологий и промышленной продукции.

В условиях непростой геополитической обстановки наличие зависимости отечественной промышленности от зарубежных информационных технологий и средств обеспечения информационной безопасности становится недопустимым. За последнее время широкой общественности стало известно множество фактов, подтверждающих наличие уязвимостей в популярном зарубежном программном и программно-аппаратном обеспечении, в том числе и в средствах защиты информации. Данные уязвимости, известные зарубежным техническим разведкам или иным враждебно настроенным лицам могут нести реальную угрозу государственной и общественной безопасности страны.

В этих условиях любые СЗИ отечественных производителей имеют преимущество перед зарубежными конкурентами.

В Доктрине подчеркивается, что создание благоприятных условий для осуществления деятельности на территории Российской Федерации российских ИТ-компаний должно являться одним из основных направлений развития экономической сферы обеспечения информационной безопасности страны. Эти меры должны позволить увеличить доли продукции отрасли информационных технологий в валовом внутреннем продукте, а также в структуре экспорта страны.

5.8 Заключение

Персональные данные также являются конфиденциальными данными, а, следовательно, регуляторами ФСТЭК и ФСБ предъявляются требования к обеспечению их защиты.

ФСТЭК России описывает требования к лицензиатам, процедуры получения лицензий на осуществление разработки, производства, распространения и обслуживания криптографических средств; лицензий на деятельность по технической защите конфиденциальной информации; лицензий

на деятельность по разработке и(или) производству средств защиты конфиденциальной информации.

В Доктрине информационной безопасности Российской Федерации акцентируется необходимость в разработке современных передовых средств и технологий защиты информации, что особенно важно ввиду последних геополитических событий и формирования многополярного мира.

При разработке анти-спам системы в обязательном порядке не требуется получать соответствующие лицензии. Информация о сертификации таких систем не была найдена.

С точки зрения технической разработки, анти-спам системы не регулируются законодательством.

Проанализированные в данном разделе НПА повлияли на внесение корректировок в техническую реализацию потенциальной системы для того, чтобы минимизировать работу с лишней конфиденциальной информацией пользователей.

На основании вышеизложенного можно сделать вывод, о том, что данная выпускная квалификационная работа полностью соответствует требованиям действующей законодательной базы Российской Федерации.

ЗАКЛЮЧЕНИЕ

В ходе работы был проведена обширная экспериментальная и инженерная работа по исследованию способов агрегации алгоритмов и разработке методики.

Обзор предметной области и математическая постановка задачи позволили определить границы исследования, его инструментарий, цель и задачи.

Отбор алгоритмов МО, создание модификаций на их основе и проработка вариантов их комбинирования позволили определить множество способов детектирования спама.

Разработка алгоритма поиска лучших комбинаций алгоритмов, методики предобработки текста, анализ и отбор методов извлечения признаков из него позволили определить основной функционал для валидации алгоритмов.

Разработка эффективного и функционального исследовательского ПО для валидации алгоритмов позволила создать удобную платформу для исследований.

Проведение исследований на нескольких датасетах с семплами разного характера позволило обнаружить особенности комбинаций алгоритмов МО, подходов к агрегированию и обнаружить лучшие комбинации для обнаружения вредоносного спама для email и sms-сообщений – разработать методику.

По результатам исследований перспективными признаны мажоритарные и дизъюнктивные комбинации, все свойства которых обязательно раскроются при использовании в полноценной анти-спам системе и должны быть исследованы с помощью использования тестовых наборов, а не валидационных в рамках кросс-валидации. Бэггинг и конъюнктивные комбинации не являются исключениями из-за небольшого отставания от других типов и их свойства также желательно исследовать таким образом, но во вторую очередь.

Полученные результаты посредством кросс-валидации можно оценивать как в целом правдоподобные для полноценной анти-спам системы, поскольку шумность данных не была полностью устранена намеренно, особенно учитывая, что как минимум по каждому диапазону длин необходима своя

специализированная модель обнаружения в полноценной системе (в данной работе не было этого разделения).

Во время проведенных кросс-валидационных исследований не было обнаружено весомого преимущества при использовании комбинаций вместо алгоритмов-одиночек, однако преимущество было во всех сценариях – стабильным, а само значение метрики f1 высоким и достойным в сравнении с результатами по классификации текста из научных и инженерных источников (с учетом сознательной весьма мягкой борьбы с шумом).

Был рассмотрен организационно-экономический сценарий разработки исследованных технологий в виде анти-спам системы, её продажи и рассмотрены организационно-правовые аспекты такого проекта.

Исходный код проекта и ход работ над ним доступен в репозитории на GitHub: https://github.com/Hockwell/AntiSpamRaven.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Scikit-learn-lib: classifier comparison. // SCIKIT-LEARN.ORG: сайт. — URL: https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html (дата обращения: 16.04.2020).

2 MacKay D. On-line book: Information Theory, Inference, and Learning Algorithms. — Morgan Kaufmann, 2005.

3 Ian H. Witten. Data Mining: Practical Machine Learning Tools and Techniques (Second Edition). — Morgan Kaufmann, 2005.

4 Алгоритмы создания дерева принятия решений. // ECONF.RAE.RU: сайт. — URL: http://econf.rae.ru/pdf/2014/03/3245.pdf (дата обращения: 16.04.2020).

5 Cristianini N. Shawe-Taylor J. An Introduction to Support Vector Machines. — Cambridge University Press, 2001.

6 Shalev-Shwartz Shai, Ben-David Shai. Understanding Machine Learning: From Theory to Algorithms. — Cambridge University Press, 2014. — 409 p.

7 Imbalanced-learn: user guide. // IMBALANCED-LEARN.IO: сайт. — URL: https://imbalanced-learn.readthedocs.io/en/stable/user_guide.html (дата обращения: 16.04.2020).

8 Cross-validation: evaluating estimator performance. // SCIKIT-LEARN.ORG: сайт. — URL: https://scikit-learn.org/stable/modules/cross_validation.html (дата обращения: 16.04.2020).

9 Working With Text Data. // SCIKIT-LEARN.ORG: сайт. — URL: https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html (дата обращения: 16.04.2020).

10 Э. Лопер, Э. Кляйн. Обработка естественного языка с Python. — O'Reilly Media Inc, 2019.

11 Manning C.D., Schütze H. Foundations of statistical natural language processing. — MIT press, 1999. — 237 p.

12 Щипицина Л.Ю. Информационные технологии в лингвистике: Учебное пособие. — М.: Флинта: Наука, 2013. — 128 с.

13 Чатуев М.Б., Чеповский А.М. Частотные методы в компьютерной лингвистике: Учебное пособие. — М.: МГУП, 2011. — 88с.

14 Micha Gorelick. High Performance Python: Practical Performant Programming for Humans. — O'Reilly Media, 2014. — 370 p.

15 Конституция Российской Федерации // KREMLIN.RU: сайт президента Российской Федерации. — URL: http://constitution.kremlin.ru/ (дата обращения: 16.04.2020).

16 Федеральный закон №149 от 27.07.2006 г. (ред. от 23.04.2018) «Об информации, информационных технологиях и о защите информации» // CONSULTANT.RU: база данных правовой информации. — URL: http://www.consultant.ru/document/cons_doc_LAW_61798/ (дата обращения: 16.04.2020).

17 Федеральный закон №152 от 27.07.2006 (ред. от 29.07.2017) «О персональных данных» // CONSULTANT.RU: база данных правовой информации. — URL: http://www.consultant.ru/document/cons_doc_LAW_61801/ (дата обращения: 16.04.2020).

18 Федеральный закон №99 «О лицензировании отдельных видов деятельности» от 04.05.2011 (ред. от 01.06.2018) // FSTEC.RU : сайт регулятора. — URL: https://fstec.ru/normotvorcheskaya/litsenzirovanie/74-zakony/222-federalnyj-zakon-rossijskoj-federatsii-ot-4-maya-2011-g-n-99-fz/ (дата обращения: 16.04.2020).

19 Анализ положений Доктрины информационной безопасности РФ // SECURITYLAB.RU: сайт. — URL: https://www.securitylab.ru/analytics/485289.php/ (дата обращения: 16.04.2020).

20 Указ Президента Российской Федерации от 05.12.2016 г. № 646 «Об утверждении Доктрины информационной безопасности Российской Федерации». // KREMLIN.RU: сайт президента Российской Федерации. —

URL: http://www.kremlin.ru/acts/bank/41460/page/1/ (дата обращения: 16.04.2020)