Содержание

Введение 4
1. Техническое задание 5
1.1 Назначение разработки и область применения 5
1.2 Технические требования 5
2. Анализ технического задания 6
2.1 Выбор операционной системы 6
2.2 Выбор языка программирования 9
2.3 Выбор среды разработки 11
2.4 Обзор существующих методов классификации спам-сообщений 13
2.5 Выбор подхода к решению задачи детектирования спам-сообщений 15
3. Разработка структуры системы детектирования спам-сообщений 18
3.1 Разработка общей структуры системы 18
3.2 Разработка алгоритма нахождения опорных векторов 21
3.3 Разработка алгоритма принятия решения 22
4. Разработка программных средств 23
4.1 Разработка интерфейса пользователя системы 23
4.2 Программная реализация модулей системы 26
5. Тестирование системы 30
5.1 Описание набора данных 30
5.2 Описание методики тестирования 31
Заключение 33
Список литературы 34

Введение

В последнее время электронная почта стала одним из наиболее распространенных средств связи. Она является достаточно совершенной в техническом отношении и недорогой альтернативой другим средствам связи.

Вместе с развитием электронной почты увеличивается и количество угроз ее нормальному функционированию. Наиболее серьезной и важной проблемой стал так называемый спам – массово рассылаемые сообщения (как напрямую, так и косвенно), предназначенные лицам, не выражавшим желания и получать, несмотря на предпринятые меры по предотвращению этой рассылки. Борьба со спамом идет уже не один десяток лет, и несмотря на успехи в разработке и исследовании различных подходов к решению данной задачи, согласно исследованию, проведенному экспертами «Лаборатории Касперского» [1], в 2018 году доля спама в мировом почтовом трафике составляет 53,5%, а в отечественном 54,2%. Приведенная статистика указывает на необходимость разработки новых и улучшения существующих алгоритмов фильтрации спама.

1. Техническое задание

1.1 Назначение разработки и область применения

Разрабатываемая программная система предназначена для классификации текстовых сообщений на «спам» и «не спам» сообщения. Для работы системы необходимо заранее подготовить базу классифицированных сообщений, для обучения модели классификации.

Область применения разрабатываемой системы:

- Использование системы вручную для классификации теста электронного письма.

- Интеграция с почтовыми клиентами для классификации почтового трафика.

Разрабатываемая система предназначается для использования на стационарных и портативных компьютерах.

1.2 Технические требования

Рассмотрим требования, предъявляемые разрабатываемой системой к ЭВМ:

- операционные системы семейства MS Windows, начиная с версии XP и выше или дистрибутив операционной системы Linux, аппаратное обеспечение ЭВМ должно поддерживать работу соответствующей ОС;
- устройства ввода: мышь, клавиатура;
- устройства вывода: монитор.

Рассмотрим, какой функциональностью должна обладать проектируемая программная система детектирования спам-сообщений:

1) Пользователь системы имеет возможность выбрать текстовый файл с сообщением из директории, используя графический интерфейс программы.

2) Программа выполняет формирование признаков сообщения и с помощью обученной модели классифицирует его.

3) По завершению определенного периода времени, программа выводит пользователю результат о том, к какому классу относится предоставленное им сообщение.

2. Анализ технического задания

2.1 Выбор операционной системы

Одним из важнейших этапов при разработке ПО является определение целевой операционной системы, под которой будет происходить непосредственно сама разработка.

Для решения этой задачи необходимо проанализировать существующие и доступные на сегодняшний день операционные системы. Результатом этого анализа должна быть операционная система, которая в наибольшей степени удовлетворяет требованиям к разрабатываемой системе. Выбор операционной системы усложняет наличие у каждой из систем уникальных свойств, которые могут оказать как положительное, так и негативное влияние в том числе и на процесс разработки.

В сравнительном анализе будут участвовать самые популярные на сегодняшний день операционные системы, такие как Linux, Windows и macOS:

1) Linux – это обобщенное название семейства UNIX-подобных операционных систем, основанных на Linux Kernel, соответствующего стандартам POSIX. Разработка первой версии ядра началась 1991 году. Отличительной чертой Linux Kernel является то, что оно разрабатывается и распространяется в соответствии с моделью свободного и открытого программного обеспечения. А так как Linux Kernel всего лишь ядро, основа операционной системы, то и не существует никаких официальных дистрибутивов этой операционной системы. Как правило Linux-системы распространяются в виде бесплатных готовых дистрибутивов, специализированных под конкретные пользовательские задачи и укомплектованных определенным набором программ и утилит прикладного характера. Несмотря на сравнительно не давнее появление на рынке операционных систем, популярность Linux-систем становится все больше, этому способствует их доступность и гибкость в настройке, позволяющая наиболее корректно использовать ресурсы машины, для решения пользовательских задач.

2) Windows – это операционная система производства компании Microsoft Corporation, появившаяся на рынке в 80-х годах 20 века. На сегодняшний день семейство ОС Windows является самой популярной операционной системой в мире, за счет того, что ее используют как на домашних компьютерах для решения пользовательских задач, так и на производстве, где решаются более сложные задачи, с более высокими требованиями к надежности. Операционная система семейства Windows стала первой ОС, которая получила графический интерфейс, взамен

консольному интерфейсу. При детальном рассмотрении всех операционных систем Windows их можно разделить на 4 группы:

- MS-DOS (все версии до Windows-95);

- Windows-9x (все версии после Windows-95 до Windows ME);

- Windows NT (все версии старше Windows ME, включая современные такие как Windows 10);

- Windows для смартфонов;

- Windows Embedded (встраиваемые ОС).

3) macOS – это продукт компании Apple появившийся на рынке в 1984 году, в качестве единственной системы для компьютера Macintosh. На рынке операционных систем занимает второе по популярности место. Главной отличительной чертой macOS является ее принадлежность к семейству ОС UNIX.

Теперь перейдем непосредственно к сравнительному анализу представленных выше операционных систем и выявим их преимущества и недостатки. Отбор операционной системы, наиболее подходящей под разработку нашей программной системы будет производиться по следующим критериям: требования к аппаратному обеспечению, стабильность работы, наличие прикладного программного обеспечения, удобство разработки и использования программного обеспечения, защищенность.

1) Требования к аппаратному обеспечению.

По данному критерию безусловно проигрывает продукт компании Apple – macOS. Обусловлено это тем, что данную операционную систему, ввиду лицензионного соглашения, возможно использовать только на оборудовании компании Apple. В тоже время ОС Windows и Linux, можно установить на оборудование, представляемое почти всеми производителями, присутствующими на рынке (MSI, ASUS, Lenovo, HP и т.д.).

2) Стабильность работы.

По данному критерию с лучшей стороны себя показала операционная система Linux. Из-за ее гибкой настройки, различного рода нарушения в работе системы, происходят на ней значительно реже чем на ее конкурентах в лице Windows и macOS.

3) Наличие прикладного программного обеспечения.

По данному критерию лидером становится операционная система Windows. Обусловлено это ее большой популярностью, в следствии которой,

библиотека программного обеспечения у данной системы значительно больше чем у Linux и macOS. Большая часть ПО на текущий момент времени написана для Windows, и не имеет аналогов или портированных версий, предназначенных для установки на другие операционные системы.

4) Защищенность.

По данному критерию операционные системы Linux и macOS опережают Windows. Это обусловлено высокой популярностью и спецификой программной архитектуры системы Windows. Следствием выше перечисленных факторов, стало создание большого количества вредоносного программного обеспечения, которое в народе получило название «вирус», и используются для умышленного нарушения нормального функционирования системы, в целях личной выгоды.

5) Удобство разработки и использования программного обеспечения.

По данному критерию лидером вновь становится операционная система Windows. Это связано с тем, что большая часть ПО, разработанного для ОС Windows, написаны на высокоуровневых языках программирования, таких как C++, C#, Java, Python и других. Часть этих языков программирования имеют возможность компиляции исполняемых модулей для их последующего применения на операционных системах macOS и Linux.

Для разработки программной системы детектирования спам-сообщений была выбрана операционная система Windows. Причиной для этого стало наибольшее количество программного обеспечения для разработки, что является важным критерием для разработки нашей системы. Так же ОС Windows соответствует всем необходимым функциональным и техническим требованиям для разработки данного проекта.

2.2 Выбор языка программирования

Следующая по значимости задача, которая встает перед нами - это выбор языка программирования, на котором будет производиться разработка всего приложения. Выбор конкретного языка должен зависеть от специфики разработки и навыков разработчика. Рассмотрим наиболее популярные языки программирования для разработки Windows-приложений:

1) **Си –** относится к компилируемым статически типизированным языкам программирования. Разработанный в 1973 году, как развитие языка Би, изначально предназначался для реализации ОС UNIX, но в дальнейшем был портирован на множество других платформ. Благодаря большой схожести конструкций языка с машинными инструкциями, Си хорошо показал себя в проектах для которых был свойственен язык ассемблера, например, в операционных системах или программном обеспечении для суперкомпьютеров. Синтаксис языка СИ в дальнейшем стал основой для таких языков как C++, Java, C# и Objective-C.

2) **C++** – как и Си является компилируемым статически типизированным языком программирования. C++ имеет поддержку процедурного, объектно-ориентированного и обобщенного программирования. Этот язык имеет в себе свойства как высокоуровневых, так и низкоуровневых языков программирования. Являясь одним из самых популярных языков программирования C++ имеет широчайшую область применения начиная от создания операционных систем и драйверов устройств, заканчивая прикладными программами и играми. Помимо этого, C++ является полностью совместимым с языком Си, это значит, что программа написанная на языке Си может быть собрана с помощью компилятора C++.

3) **Python** – является высокоуровневым языком программирования, с минималистическим синтаксисом, направленным на повышение читаемости кода. Этот язык обладает динамической типизацией и автоматическим управлением памятью, а также поддерживает объектно-ориентированное, императивное, структурное, функциональное и аспектно-ориентированное программирование. В область его применения входят back-end разработка, разработка графических интерфейсов, веб-сайтов, разработка баз данных, научные и числовые вычисления.

4) **C# –** относится к семейству языков с Си-подобным синтаксисом. Является объектно-ориентированным языком программирования со статической типизацией. В основном применяется в области инженерии, профессиональных услуг, дизайна, управления и контроля качества. C# позволяет разработчикам создавать веб-службы XML и приложения Microsoft .NET для ОС Windows.

5) **Java** – изначально данный объектно-ориентированный язык программирования разрабатывался для применения в области бытовой электроники. Основной причиной появления такого языка как Java стало нежелание производителей различного рода электроники писать код отдельно под каждую платформу или архитектуру оборудования с полного нуля. Таким образом, был создан язык программирования Java, для осуществляется трансляция исходного кода в байт код, который затем выполняется на виртуальной java-машине (JVM). Эта виртуальная машина пишется отдельно под каждую архитектуру, и обрабатывает байт-код, передавая его в виде инструкций непосредственно на аппаратное обеспечение и, таким образом, работает как интерпретатор, но значительно быстрее. В настоящее время для всех популярных платформ существуют виртуальные java-машины, позволяющие выполнять на них исходный код на языке Java.

Для разработки системы был выбран язык программирования Python, так как данный язык показывает высокую скорость вычислений, простой и удобный синтаксис, а также предоставляет возможность создания графических интерфейсов.

2.3 Выбор среды разработки

Еще одним не маловажным фактором при разработке является правильный подход к выбору среды разработки. Правильно подобранная IDE позволит разработчику значительно повысить эффективность работы. Рассмотрим наиболее популярные среды разработки для Windows на языке Python:

1) **Spyder** - межплатформенная среда разработки с открытым исходным кодом. Включает в себя основные библиотеки для обработки больших наборов данных, такие как*: NumPy, SciPy, Matplotlib* и *IPython*, кроме того, он может быть расширен с помощью плагинов. *Spyder* предлагает интерактивную справку, которая позволяет искать конкретную информацию о библиотеках. Особенности *Spyder* заключаются в следующих функциях: это текстовый редактор с подсветкой синтаксиса, завершение кода и исследование переменных, которые можно редактировать значениями с помощью графического интерфейса пользователя.

2) **PyCharm** - это *IDE*, созданная компанией JetBrains, ответственной за одну из самых известных Java IDE - IntelliJ *IDEA*. PyCharm интегрирует инструменты и библиотеки, такие как NumPy и Matplotlib, что позволяет работать с обзором массивов и интерактивными графиками. Как и другие IDE, PyCharm имеет интересные функции, такие как редактор кода, подсветку ошибок, мощный отладчик с графическим интерфейсом, помимо интеграции Git, SVN и Mercurial. Кроме того, можно расширить возможности PyCharm, добавив плагины.

3) **Atom** - текстовый редактор с открытым исходным кодом, разработанный Github. У Atom есть интересные функции, которые создают хороший опыт для разработчиков Python. Одним из лучших преимуществ Atom является его сообщество, главным образом благодаря совершенствованиям констант и плагинам, которые они разрабатывают для настройки IDE и улучшения рабочего процесса. В Atom есть возможность визуализировать свои результаты, не открывая дополнительных окон. Кроме того, есть плагин под названием «Markdown Preview Plus», который предоставляет встроенную поддержку для редактирования и визуализации файлов Markdown и который позволяет открыть предварительный просмотр, визуализировать уравнения LaTeX и многие другие функции. К недостаткам Atom можно отнести слабую производительность на старых процессорах.

4) **Jupyter Notebook** - веб-приложение, основанное на архитектуре клиент-сервер и позволяющее создавать и обрабатывать документы. Jupyter Notebook предоставляет удобную интерактивную среду для научных исследований на многих языках программирования, которая работает не только как среда разработки, но и как инструмент презентации или

образования. Благодаря Jupyter можно легко видеть и редактировать свой код, чтобы создавать интересные презентации. Например, можно использовать библиотеки визуализации данных, такие как Matplotlib и Seaborn, и показывать графики в том же документе, где находится код. Помимо всего этого можно экспортировать работу в файлы PDF и HTML, или как .py-файл.

Для разработки программной системы была выбрана среда PyCharm, главным фактором выбора этой среды было наличие у нее PyCharm Community версии, которая распространяется бесплатно и имеет открытый исходный код, также данная IDE обладает удобным отладчиком, имеет возможность работать с несколькими версиями Python и прекрасно интегрируется с Git.

2.4 Обзор существующих методов классификации спам-сообщений

На сегодняшний день разработан ряд технологий построения фильтров – сервисов для отсеивания нежелательной информации. Все технологии можно разделить на настраиваемые вручную и интеллектуальные [2]. Настраиваемые вручную фильтры основываются на списках доступа и настраиваются непосредственно пользователем, который выбирает либо нежелательные адреса, при политике пропуска по «черному списку», либо разрешенные при политике пропуска по «белому списку». Однако ручные способы фильтрации нежелательных сообщений малоэффективны и требуют постоянного обновления списков доступа, создавая дополнительную нагрузку на пользователя.

Фильтры, построенные с использованием технологий искусственного интеллекта, требуют обучения только на начальном этапе, продолжая обучаться в дальнейшем самостоятельно, существенно снижая нагрузку на пользователя.

В данной работе будут рассматриваться только фильтры построенные с использованием технологии искусственного интеллекта (машинного обучения).

На текущий момент существует множество алгоритмов идентификации спама. Все эти алгоритмы можно разделить на группы по используемому подходу, или категории, положенного в основу алгоритма фильтрации спама, математического аппарата:

- Вероятностные (байесовские классификаторы, логистическая регрессия, MRF-классификатор);
- Линейные (персептрон, алгоритм Winnow, метод опорных векторов);
- На основе сходства (методы ближайших соседей);
- Логические (дерево принятия решений, логический вывод на основе набора правил);
- На основе моделей сжатия данных (DMC, PPM).

Самым распространенным на сегодняшний день является фильтр, основанный на наивном байесовском подходе, в котором предполагается, что различные слова сообщения независимы друг от друга [3]. Максимальный результат, достигнутый байесовскими фильтрами на сегодняшний день, составляет порядка 95% отфильтрованного спама. Для повышения эффективности байесовского фильтра необходимо учитывать семантические связи между словами, что требует привлечения методов семантического анализа и существенно повышает нагрузку на систему и увеличивает время работы самого фильтра, при незначительном повышении эффективности фильтрации.

Другим подходом, получающим в последнее время все большее распространение, является использование нейросетей (NN – Neural Networks). Преимущество нейросетевого подхода перед наивным байесовским состоит в том, что не делается никаких предварительных предположений о характере нежелательных сообщений, а семантические связи учитываются автоматически. Наибольшее количество разработок связано с построением фильтра на основе многослойного персептрона [4]. Однако такой подход встречается с рядом трудностей, связанных с выбором пороговых значений, которые задаются в некотором интервале. Эффективность фильтра существенно зависит от выбора порогового значения. При этом пороговое значение требует постоянной подстройки под изменяющийся характер нежелательных сообщений.

2.5 Выбор подхода к решению задачи детектирования спам-сообщений

Все известные на сегодняшний день алгоритмы детектирования спам-сообщений в общем смысле основаны на следующих этапах

- Предварительная обработка сообщения – сообщение обрабатываются и приобретают вид, с которым классификатору будет возможно работать.

- Построение признакового описания объекта классификации – формирование набора признаков, позволяющего точно и однозначно описать объект классификации.

- Принятие решения – определение сообщения к классу спам или не спам.

В данной работе для решения задачи детектирования спам-сообщений было решено использовать метод опорных векторов (SVM – Support Vector Machines) [5]. Этот метод принадлежит к семейству линейных классификаторов, особым свойством метода опорных векторов является непрерывное уменьшение эмпирической ошибки классификации и увеличение зазора, поэтому метод также известен как метод классификатор с максимальным зазором. Схема, описанных выше этапов представлена ниже.

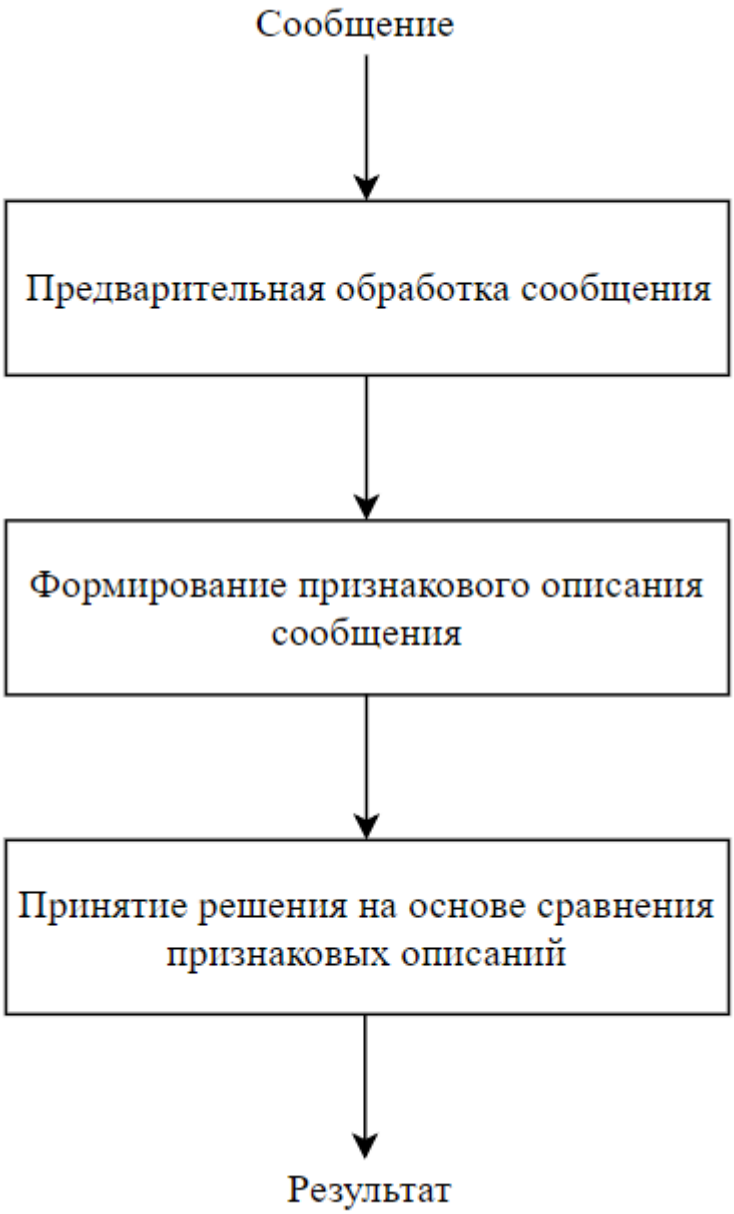

Рисунок 1. Схема этапов решения задачи

В методе опорных векторов каждый объект классификации представляется как вектор (точка) в *p*-мерном пространстве. У каждого объекта есть принадлежность только к одному из двух классов. Такой случай относится к задачам линейной разделимости. Основная задача сводится к нахождению разделяющей гиперплоскости размерности (*p*-1). Эта задача усложняется тем, что искомых гиперплоскостей может быть много, так что необходимо найти гиперплоскость расстояние от которой до ближайших точек разных классов было бы максимальным. Гиперплоскость, которая соответствует требованиям выше называется оптимальной разделяющей

гиперплоскостью, а классификатор соответственно оптимально разделяющим классификатором.

Стоит также отметить некоторые преимущества SVM над NN. Для начала стоит сказать, что производительность SVM существенно выше по сравнению с NN. Для трехслойного персептрона прогнозирование решения требует последовательного умножения входного вектора, на две двумерные матрицы. Для SVM классификация включает в себя определение, с какой стороны от гиперплоскости находится данная точка, другими словами косинусное произведение. Если будем сравнивать точность классификации этих двух моделей, учитывая, что они обе правильно сконфигурированы и обучены, то SVM превосходит NN. Это связано в первую очередь с тем, что эти модели имеют фундаментально разные модели обучения. В NN весовые коэффициенты (параметры подгонки, скорректированные во время обучения) настраиваются таким образом, что суммарная квадратная ошибка между выходом сети и фактическим значением минимизируется. В то время как обучение SVM наоборот означает явное определение границ решения непосредственно из данных для обучения. Это в свою очередь требует дополнительного шага в задаче оптимизации: минимизация совокупного расстояния между гиперплоскостью и опорными векторами.

Но на практике SVM гораздо труднее настроить в отличии от NN. Причина этого заключается в большом количестве параметров, необходимых для настройки.

3. Разработка структуры системы детектирования спам-сообщений

3.1 Разработка общей структуры системы

В соответствии с методом опорных векторов, структура системы должна иметь следующие этапы:

- Разработка механизма предварительной обработки данных

- Формирование признаковой системы

Результатом предварительной обработки будет являться сообщение, очищенное от слов, не несущих никакой смысловой нагрузки, очищенное от всех пунктуационных знаков, а также в котором все знаки табуляции и тире заменены на символ пробела.

Результатом формирования признаковой системы является матрица вхождений популярных слов в сообщения (рисунок 3.1). По горизонтали располагается список самых популярных слов среди всех сообщений из базы классифицированных сообщений. По вертикали располагаются сообщения из базы классифицированных сообщений. В ячейках располагается количество вхождений слова из списка популярных слов в сообщение из базы данных классифицированных сообщений.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	ect	subject	hou	enron	com	deal	please	gas	meter	hpl	daren	thanks	need	corp	volume
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	17	3	10	1	0	0	2	1	0	0	3	1	0	1	0
4	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
5	18	3	10	0	0	2	0	0	1	0	2	1	0	0	1
6	14	2	8	0	0	0	1	1	3	0	1	1	0	0	3
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	1	1	5	4	1	0	1	1	0	0
9	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
10	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0
11	33	1	16	0	0	0	1	1	0	0	0	1	0	0	0
12	0	0	0	0	0	2	0	8	7	4	0	0	5	0	2
13	23	1	12	4	0	1	3	0	0	0	1	5	0	2	2
14	0	0	0	0	0	1	2	0	0	0	0	2	0	0	2
15	3	1	2	0	0	0	0	1	0	2	0	0	0	0	0
16	3	1	2	0	0	0	0	1	0	2	0	0	0	0	0
17	0	0	0	1	0	0	3	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
20	0	0	0	0	0	4	1	0	2	0	0	0	3	0	0
21	9	1	5	0	0	4	3	0	3	0	2	1	3	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	10	1	5	0	0	1	1	0	0	0	0	0	0	0	0
24	3	2	2	13	0	0	1	3	0	4	0	1	0	1	0
25	0	0	0	0	0	1	0	0	2	0	1	1	1	0	0
26	0	0	0	0	0	3	1	0	1	0	1	1	0	0	0
27	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0
28	0	0	0	0	0	1	2	2	2	0	1	1	0	0	1
29	0	1	0	3	1	0	0	0	0	0	0	0	0	2	0
30	0	1	0	3	1	0	0	10	1	0	0	0	0	2	0

Рисунок 3.1 Матрица вхождений популярных слов в сообщения

Для получения данной матрицы, с базой классифицированных сообщений необходимо проделать несколько операций:

1) Предварительная обработка всех сообщений.

2) Составление словаря, в который входят все наиболее популярные слова и общее количество их вхождений.

3) На основе словаря из пункта 2 и базы классифицированных сообщений получение матрицы.

Итоговая программа будет содержать в себе следующие блоки:

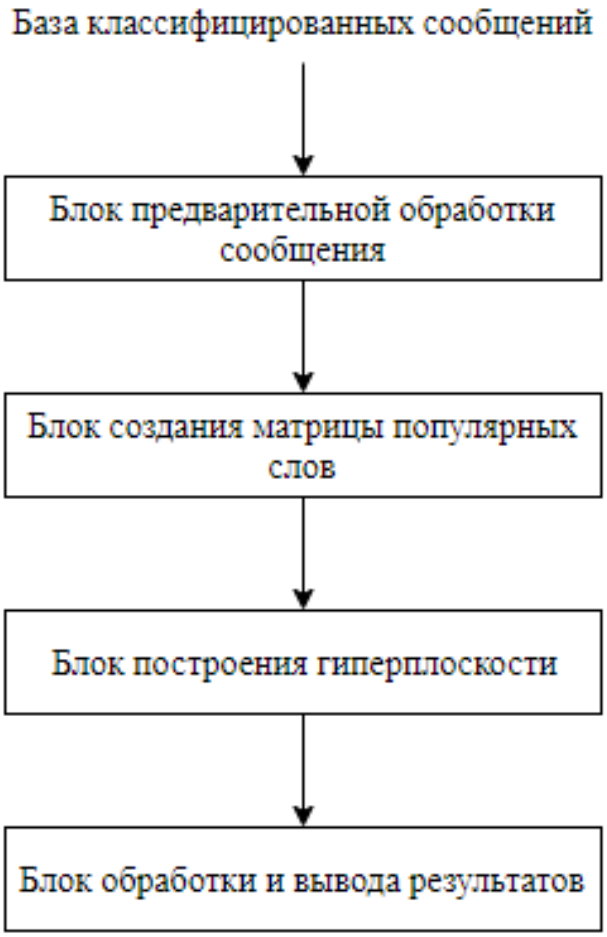

Рисунок 3.2 Структура системы

1) Блок предварительной обработки сообщения включает в себя очистку сообщения от слов не несущих смысловой нагрузки, знаков пунктуации, а также замену всех символов табуляции и тире на символ пробела.

2) Блок создания матрицы популярных слов включает в себя создание словаря популярных слов, и его последующее преобразование.

3) Блок построения гиперплоскости предназначен для обучения модели классификации спам-сообщений. Для этого с помощью обучающей выборки из матрицы популярных слов происходит получение так называемых опорных векторов, на основе которых будет определена разделяющая гиперплоскость. Далее при классификации входящего сообщения происходит определение его положения относительно гиперплоскости.

4) Блок обработки и вывода результатов предназначен для присвоения сообщению класса основываясь на его положении относительно гиперплоскости, и вывода результата классификации.

3.2 Разработка алгоритма нахождения опорных векторов

Последовательность действий по нахождению опорных векторов изображена на рисунке 3.3:

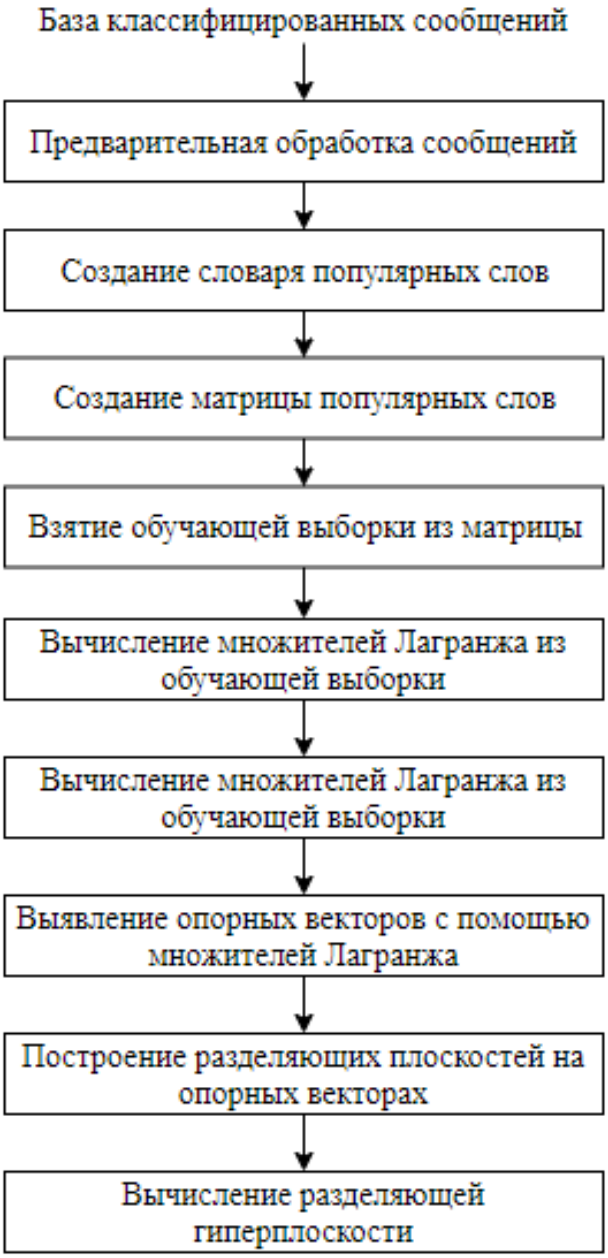

Рисунок 3.3 Алгоритм нахождения опорных векторов

3.3 Разработка алгоритма принятия решения

Для классификации сообщения необходимо определить положение объекта классификации относительно разделяющей гиперплоскости. Алгоритм принятия решения о классе сообщения представлен на рисунке 3.4:

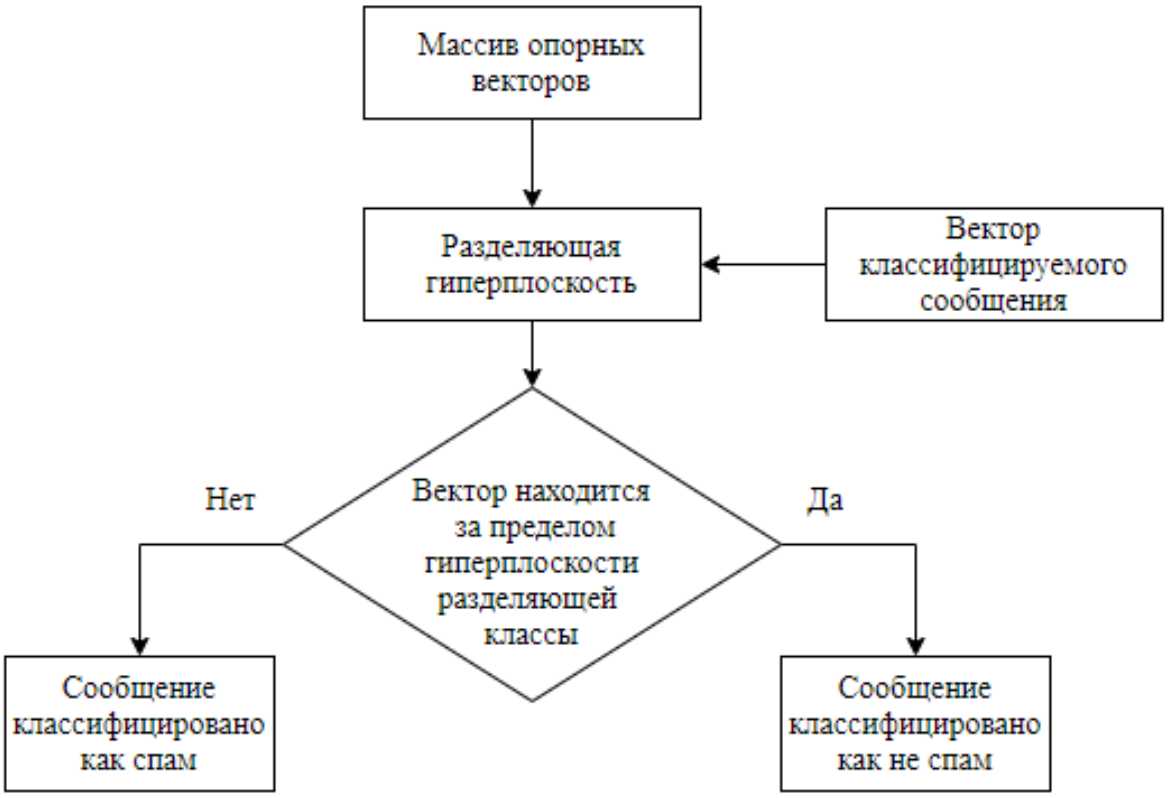

Рисунок 3.4 Алгоритм принятия решения

Классифицировать сообщения на спам и не спам возможно следуя алгоритму выше:

1) Необходим массив опорных векторов.

2) С помощью массива опорных векторов строятся плоскости. После чего вычисляется и строится разделяющая гиперплоскость.

3) Для каждого классифицируемого сообщения вычисляется вектор на плоскости.

4) В зависимости от того с какой стороны от разделяющей гиперплоскости находится вектор сообщения, классифицируем сообщение как спам или не спам.

4. Разработка программных средств

4.1 Разработка интерфейса пользователя системы

Графический пользовательский интерфейс для данной системы был реализован с помощью библиотеки Tkinter. Модуль Tkinter входит в состав стандартной библиотеки языка Python, которая имеет свободное распространение, а значит является простым, а главное легко интегрируемым решением для разработки пользовательского интерфейса.

В модуле Tkinter поддерживается 15 основных виджетов:

1) Frame – виджет контейнера. Он используется для группировки других виджетов, во время создания приложения или макета диалога, а также может иметь фон и границу.

2) Entry – используется для ввода текстовой информации в заданное поле.

3) Button – кнопка, при нажатии которой выполняется заданная ей команда.

4) Label – используется для отображения текстовой информации или изображения.

5) Checkbutton – используется для создания переменной, содержащей два значения. Нажатие на эту кнопку будет переключать значения переменной.

6) Canvas – используется для создания структурированной графики. Под этим подразумевается создание графических редакторов, рисование графиков или реализация пользовательских виджетов.

7) Listbox – используется для создания списков с элементами, которые в последствии можно будет выбрать.

8) Menu – используется для создания всплывающих и раскрывающихся панелей меню.

9) Menubutton – используется для создания выпадающих меню.

10) Radiobutton – используется для создания переменной в которую можно записать одно значение из списка. При нажатии кнопки записывает в переменную выбранное из списка значение и очищает все остальные radiobutton.

11) Scale – используется для создания числовой переменной значение в которую записывается с помощью «ползунка».

12) Scrollbar – используется для создания полосы прокрутки.

13) Toplevel – используется для вызова Frame в отдельном окне верхнего уровня.

14) Text – используется для создания форматированных текстовых дисплеев. Этот виджет имеет возможность работать с текстом с различными атрибутами и стилями и поддерживает встроенные изображения и окна.

15) Message – используется для отображения текстовой информации. Этот виджет схож с виджетом Label, но помимо отображения текста, также может автоматически переносить его в соотношение сторон или заданную ширину.

При разработке пользовательского интерфейса нашей системы были использованы следующие методы: Frame, Button, Label. На рисунке 4.1 представлена программная реализация пользовательского интерфейса.

```
def gui():
    root=Tk()
    root.title("Детектирование")
    root.geometry("500x300")
    root.configure(background="light blue")
    Label(root, text="\nДобро пожаловать в программу\n"
                     "детектирования спам-сообщений! ",
          bg="light blue",
          font=Font(family="Times New Roman", size=14)).pack()
    Label(root, text="\nДля начала работы загрузите файл с сообщением.",
          bg="light blue",
          font=Font(family="Times New Roman", size=14)).pack()
    Label(root, text="\nПо окончанию классификации\n"
                     "результат появится в соответствующем окне.",
          bg="light blue",
          font=Font(family="Times New Roman", size=14)).pack()
    Button(root, text="Загрузить сообщение",
           command = openFolder).place(x=100,
                                       y=200,
                                       width=300,
                                       height=50)
    root.mainloop()
```

Рисунок 4.1 Программная реализация интерфейса

Рассмотрим программные составляющие пользовательского интерфейса:

1) root – объект базового класса Tkinter приложения.

2) root.geometry – параметр объекта, который позволяет задать размер окна приложения.

3) root.configure – параметр позволяющий задать цветовое оформление окна приложения.

4) Label – метод базового класса Tkinter позволяющий отобразить текстовое сообщение для пользователя.

5) Button – метод базового класса Tkinter позволяющий создать в окне приложения кнопку, по нажатию которой у пользователя появится возможность загрузить сообщение

С помощью метода mainloop() происходит запуск цикла обработки событий.

В нашем случае событием является нажатие кнопки «Загрузить сообщение».

При нажатии кнопки происходит вызов метода openfolder(), см. рисунок 4.2.

```
def openfolder():
    root.filename = filedialog.askopenfilename(initialdir="C:\\",
                                               title="Выберите файл",
                                               filetypes=(("txt files", "*.txt"),
                                                          ("all files","*.*")))
    global path
    path = root.filename
    print(path)
```

Рисунок 4.2 Обработчик нажатия кнопки «Загрузить сообщение»

Также для взаимодействия с пользователем в программе существуют диалоговые окна, которые могут сообщить пользователю о следующих событиях:

1) Загрузка сообщения

2) Начало обработки сообщения

3) Окончание обработки

4) Результат обработки

Эти окна реализуются с помощью метода showinfo модуля tkinter.messagebox см.рисунок 4.3.

```
messagebox.showinfo("Сообщение",
                    "Сообщение успешно загружено. Обработка началась.")
```

Рисунок 4.3 Пример реализации диалогового окна

4.2 Программная реализация модулей системы

4.2.1 Модуль создания матрицы популярных слов.

Для того чтобы создать матрицу сначала необходимо подготовить сообщения, а затем создать словарь популярных слов.

Для предварительной обработки сообщения был создан метод text _cleanup () см. рисунок 4.4

```
def text_cleanup(text):
    for c in text:
        if c not in string.punctuation:
            text_without_punctuation = c
    text_without_punctuation = ''.join(text_without_punctuation)
    for word in text_without_punctuation.split():
        if word.lower() not in stopwords.words('english'):
            text_without_stopwords = word
    text_without_stopwords = ' '.join(text_without_stopwords)
    cleaned_text = [word.lower() for word in text_without_stopwords.split()]
    return cleaned_text
```

Рисунок 4.4 Реализация метода text_cleanup

Далее необходимо создать словарь всех популярных слов на основе очищенных сообщений, для этого был создан метод popular_dict см. рисунок 4.5.

```
def popular_dict(directory):
    for file in os.listdir(directory):
        file = file.decode("utf-8")
        file_name = str(os.getcwd()) + '/emails/'
        file_name = file_name + file
        file_reading = open(file_name,"r",encoding='utf-8', errors='ignore')
        words = text_cleanup(file_reading.read())
        for word in words:
            if (word.isdigit()==False and len(word)>2):
                word = lmtzr.lemmatize(word)
                if word in count:
                    count[word] += 1
                else:
                    count[word] = 1
```

Рисунок 4.5 Реализация метода popular_dict

После того как получен словарь популярных сообщений, можно создать матрицу популярных сообщений, для этого был реализован метод popular_matrix см. рисунок 4.6.

```
def popular_matrix(directory):
    for file in os.listdir(directory):
        file = file.decode("utf-8")
        file_name = str(os.getcwd()) + '/emails/'
        for i in file:
            if(i!='b' and i!="'"):
                file_name = file_name + i
        file_reading = open(file_name,"r",encoding='utf-8', errors='ignore')
        words_list_array = np.zeros(words.size)
        for word in file_reading.read().split():
            word = lmtzr.lemmatize(word.lower())
            if(word in stopwords.words('english') or word in string.punctuation
                    or len(word)<=2 or word.isdigit()==True):
                continue
            for i in range(words.size):
                if(words[i]==word):
                    words_list_array[i] = words_list_array[i]+1
                    break
        f = open("frequency.csv","a")
        for i in range(words.size):
            f.write(str(int(words_list_array[i])) + ',')
        if(len(file_name)==77):
            f.write("-1")
        elif (len(file_name)==80):
            f.write("1")
        f.write('\n')
        f.close()
```

Рисунок 4.6 Реализация метода popular_matrix

4.2.2 Модуль расчета множителей Лагранжа.

Для расчета множителей Лагранжа был реализован метод compute_multipliers на вход этому методу подается выборка сообщений для обучения классификатора и классы к которому каждое их этих сообщений относится. см. рисунок 4.7.

```
def compute_multipliers(self, X, y):
    n_samples, n_features = X.shape
    K = self.kernel_matrix(X,n_samples)
    P = cvxopt.matrix(np.outer(y, y) * K)
    q = cvxopt.matrix(-1 * np.ones(n_samples))
    G_std = cvxopt.matrix(np.diag(np.ones(n_samples) * -1))
    h_std = cvxopt.matrix(np.zeros(n_samples))
    G_slack = cvxopt.matrix(np.diag(np.ones(n_samples)))
    h_slack = cvxopt.matrix(np.ones(n_samples) * self.c)
    G = cvxopt.matrix(np.vstack((G_std, G_slack)))
    h = cvxopt.matrix(np.vstack((h_std, h_slack)))
    A = cvxopt.matrix(y, (1, n_samples), 'd')
    b = cvxopt.matrix(0.0)
    solution = cvxopt.solvers.qp(P, q, G, h, A, b)
    return np.ravel(solution['x'])
```

Рисунок 4.7 Реализация модуля compute_multipliers

4.2.3 Модуль создания обученной модели

Для классификации сообщений необходимо создать обученную модель на основе которой будет происходить классификация. Для этого был реализован метод construct_predictor. На вход этому методу подаются набор сообщений для обучения, классы к которым эти сообщения относятся, а также множители Лагранжа, см. рисунок 4.8

```
def construct_predictor(self, X, y, lagrange_multipliers):
    support_vector_indices = lagrange_multipliers > MIN_SUPPORT_VECTOR_MULTIPLIER
    support_multipliers = lagrange_multipliers[support_vector_indices]
    support_vectors = X[support_vector_indices]
    support_vector_labels = y[support_vector_indices]
    bias = np.mean(
        [y_k - SVMPredictor(
            kernel=self.kernel,
            bias=0.0,
            weights=support_multipliers,
            support_vectors=support_vectors,
            support_vector_labels=support_vector_labels).predict(x_k)
         for (y_k, x_k) in zip(support_vector_labels, support_vectors)])
    return SVMPredictor(
        kernel=self.kernel,
        bias=bias,
        weights=support_multipliers,
        support_vectors=support_vectors,
        support_vector_labels=support_vector_labels)
```

Рисунок 4.8 Реализация метода construct_predictor

4.2.4 Модуль классификации входящих сообщений.

После того как была получена обученная модель на ее основе происходит классификация всех входящих сообщений. Для классификации был создан метод predict. Этот метод возвращает 1 если сообщение классифицировалось как не спам и -1 если классифицировалось как спам см. рисунок 4.9.

```
def predict(self, x):
    result = self._bias
    for z_i, x_i, y_i in zip(self._weights,
                             self._support_vectors,
                             self._support_vector_labels):
        dot = self._kernel(x_i, x)
        result += z_i * y_i * dot
    return np.sign(result).item()
```

Рисунок 4.9 Реализация метода predict

4.2.5 Модуль принятия решения

После того как были разработаны все ключевые методы, мы можем передать обученной модели исследуемое сообщение, и на основе результата классификации выдать пользователю соответствующий messagebox об окончании классификации и к какому классу относится исследуемое сообщение, для этого был разработан метод final_result см. рисунок 4.10.

```
def final_result(X_train, Y_train, test_message):
    trainer = SVMTrainer(Kernel.linear(), 0.1)
    predictor = trainer.train(X_train, Y_train)
    result = predictor.predict(test_message)
    if result == 1:
        messagebox.showinfo("Сообщение",
        "Обработка завершена. Сообщение классифицировано как не спам")
    elif result == -1:
        messagebox.showinfo("Сообщение",
                            "Обработка завершена. Сообщение классифицировано как спам")
```

Рисунок 4.10 Реализация метода final_result

5. Тестирование системы

5.1 Описание набора данных

Для тестирования системы детектирования спам-сообщений, необходима большая база сообщений, в которой должны содержаться как спам, так и не спам сообщения. Сообщения в этой базе должны быть изначально классифицированы, это значит, что еще до запуска системы мы будем знать к какому классу, относится каждое сообщение из базы.

В ходе тестирования необходимо будет определить, что система работает в соответствии с ожиданиями, а также необходимо определить точность классификации, если она будет высокой значит разработанная система работает корректно.

В качестве набора данных использовалась база классифицированных сообщений общим объемом 5172 сообщения, среди которых:

3672 – не спам сообщения

1500 – спам сообщений.

Для обучающей выборки из базы было взято 3620 случайных сообщений.

5.2 Описание методики тестирования

5.2.1 Работоспособность программы.

Для начала необходимо проверить работоспособность программы. Для этого сначала проверим пользовательский интерфейс, а затем загрузим тестовое сообщение и убедимся, что классификатор выполняет свои задачи. Пользовательский интерфейс представлен на рисунке 5.1.

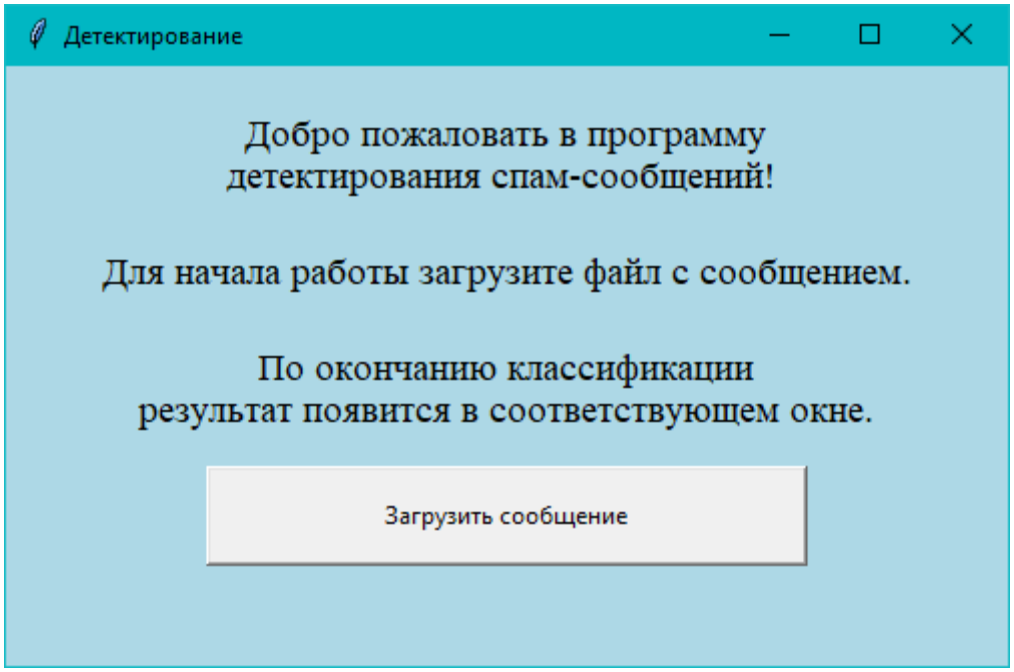

Рисунок 5.1. Пользовательский интерфейс

После загрузки сообщения на экране появляется окно с соответствующим сообщением см. рисунок 5.2.

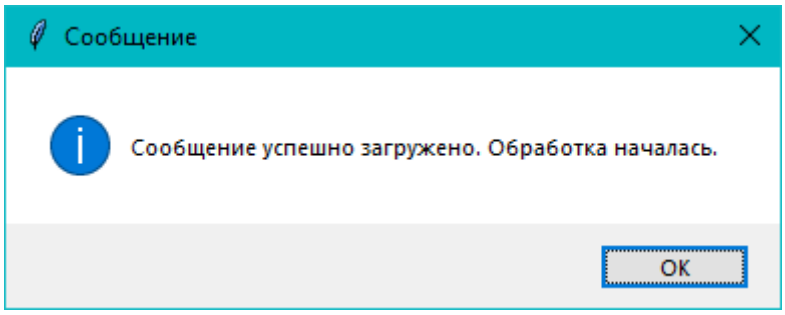

Рисунок 5.2 Окно с сообщением о начале обработки данных

Спустя какое-то время на экран выведется сообщение с результатами классификации см. рисунок 5.3.

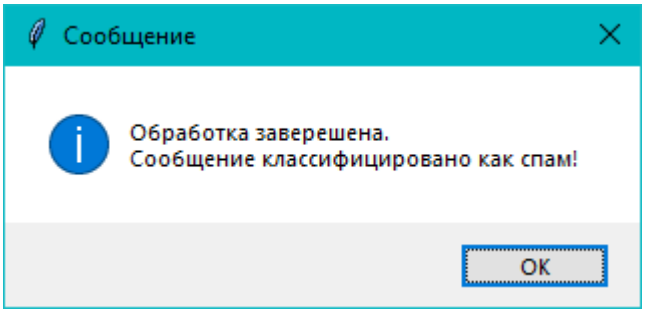

Рисунок 5.3 Результат работы классификатора

5.2.2 Точность классификации.

Для тестирования системы на точность классификации были взяты оставшиеся в базе 1552 сообщения.

Результаты тестирования представлены в виде ошибок первого и второго рода см. рисунок 5.4.

		Верная гипотеза	
		«Не спам»	«Спам»
Результат применения критерия	«Не спам»	1049 верно принятых	29 неверно принятых (Ошибка второго рода)
	«Спам»	69 неверно отвергнутых (Ошибка первого рода)	405 верно отвергнутых

В ходе тестирования системы, сбоев в работе пользовательского интерфейса обнаружено не было, а итоговая точность классификации составила порядка 95%. Исходя из полученных результатов, можно сделать вывод, что разработанная система работает корректно.

Заключение

В результате выполнения выпускной квалификационной работы была спроектирована и программно реализована система детектирования спам-сообщений. Общий алгоритм работы программы был основан на методе опорных векторов.

Разработанная система предназначена для классификации почтового трафика. Тестирование программы подтвердило ее работоспособность и возможность использования для решения данной задачи.

Список литературы

1. М.Вергелис, Н.Демидова, Т.Щербакова. Годовой отчет по спаму и фишингу за 2018 год // АО «Лаборатория Касперского». - 2018

2. Anders Wiehes. Научная работа Comparing Anti-Spam Methods // Department of Computer Science and Media Technology Gjøvik University College. – 2005

3. Rish, Irina. «An empirical study of the naive Bayes classifier» // IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence – 2001

4. Брюхомицкий, Ю. А. Нейросетевые модели для систем информационной безопасности // Таганрог: Изд-во ТРТУ - 2005.

5. В.Вьюгин. Математические основы теории машинного обучения и прогнозирования // МЦМНО - 2013

6. Nello Cristianini, John Shawe-Taylor. An Introduction to Support Vector Machines and Other Kernel-based Learning Methods // Cambridge University Press – 2000

7. Томас Х. Кормен и др. Глава 29. Линейное программирование // Алгоритмы: построение и анализ = Introduction to Algorithms. — 2-е изд. — М.: «Вильямс», 2006. — С. 1296. — ISBN 5-8459-0857-4.

8. Акулич И. Л. Глава 1. Задачи линейного программирования, Глава 2. Специальные задачи линейного программирования // Математическое программирование в примерах и задачах. — М.: Высшая школа, 1986. — 319 с. — ISBN 5-06-002663-9

9. Кузнецов А. В., Сакович В. А., Холод Н. И. Высшая математика. Математическое программирование. — Минск.: Вышейшая школа, 1994. — 286 с

10. Гасс С. Линейное программирование. — М.: Физико-математическая литература, 1961. — 300 с

Приложение А

svm_implementation.py

```
import itertools
import numpy as np
import pandas as pd
from time import time
import cvxopt.solvers
import numpy.linalg as la
import matplotlib.cm as cm
import matplotlib.pyplot as plt
from prettytable import PrettyTable
from tkinter import messagebox


MIN_SUPPORT_VECTOR_MULTIPLIER = 1e-5

class Kernel(object):

    @staticmethod
    def linear():
        return lambda x,y:np.dot(x.T,y)

class SVMTrainer(object):

    def __init__(self, kernel, c):

        self.kernel = kernel
        self.c = c

    def train(self, X, y):
        lagrange_multipliers = self.compute_multipliers(X, y)
        return self.construct_predictor(X, y, lagrange_multipliers)

    def kernel_matrix(self, X, n_samples):
        K = np.zeros((n_samples, n_samples))
        for i, x_i in enumerate(X):
            for j, x_j in enumerate(X):
                K[i, j] = self.kernel(x_i, x_j)
        return K

    def construct_predictor(self, X, y, lagrange_multipliers):
        support_vector_indices = lagrange_multipliers >
MIN_SUPPORT_VECTOR_MULTIPLIER
        support_multipliers = lagrange_multipliers[support_vector_indices]
        support_vectors = X[support_vector_indices]
        support_vector_labels = y[support_vector_indices]
        bias = np.mean(
            [y_k - SVMPredictor(
                kernel=self.kernel,
                bias=0.0,
                weights=support_multipliers,
                support_vectors=support_vectors,
                support_vector_labels=support_vector_labels).predict(x_k)
             for (y_k, x_k) in zip(support_vector_labels, support_vectors)])
        return SVMPredictor(
            kernel=self.kernel,
            bias=bias,
            weights=support_multipliers,
```

```
            support_vectors=support_vectors,
            support_vector_labels=support_vector_labels)

    def compute_multipliers(self, X, y):
        n_samples, n_features = X.shape
        K = self.kernel_matrix(X,n_samples)
        P = cvxopt.matrix(np.outer(y, y) * K)
        q = cvxopt.matrix(-1 * np.ones(n_samples))
        G_std = cvxopt.matrix(np.diag(np.ones(n_samples) * -1))
        h_std = cvxopt.matrix(np.zeros(n_samples))
        G_slack = cvxopt.matrix(np.diag(np.ones(n_samples)))
        h_slack = cvxopt.matrix(np.ones(n_samples) * self.c)
        G = cvxopt.matrix(np.vstack((G_std, G_slack)))
        h = cvxopt.matrix(np.vstack((h_std, h_slack)))
        A = cvxopt.matrix(y, (1, n_samples) , 'd')
        b = cvxopt.matrix(0.0)
        solution = cvxopt.solvers.qp(P, q, G, h, A, b)
        return np.ravel(solution['x'])

class SVMPredictor(object):

    def __init__(self,
                 kernel,
                 bias,
                 weights,
                 support_vectors,
                 support_vector_labels):
        self._kernel = kernel
        self._bias = bias
        self._weights = weights
        self._support_vectors = support_vectors
        self._support_vector_labels = support_vector_labels
        assert len(support_vectors) == len(support_vector_labels)
        assert len(weights) == len(support_vector_labels)

    def predict(self, x):
        result = self._bias
        for z_i, x_i, y_i in zip(self._weights,
                                 self._support_vectors,
                                 self._support_vector_labels):
            dot = self._kernel(x_i, x)
            result += z_i * y_i * dot
        return np.sign(result).item()

def calculate(true_positive,false_positive,false_negative,true_negative):
    result = {}
    result['precision'] = true_positive / (true_positive + false_positive)
    result['recall'] = true_positive / (true_positive + false_negative)
    return result

def
confusion_matrix(true_positive,false_positive,false_negative,true_negative):
    matrix = PrettyTable([' ', 'Ham' , 'Spam'])
    matrix.add_row(['Ham', true_positive , false_positive])
    matrix.add_row(['Spam', false_negative , true_negative])
    return matrix ,
calculate(true_positive,false_positive,false_negative,true_negative)

def implementSVM(X_train,Y_train,X_test,Y_test):
    ham_spam = 0
    spam_spam = 0
    ham_ham = 0
```

```
    spam_ham = 0
    trainer = SVMTrainer(Kernel.linear(),0.1)
    predictor = trainer.train(X_train,Y_train)
    for i in range(X_test.shape[0]):
        ans = predictor.predict(X_test[i])
        if(ans==-1 and Y_test[i]==-1):
            spam_spam+=1
        elif(ans==1 and Y_test[i]==-1):
            spam_ham+=1
        elif(ans==1 and Y_test[i]==1):
            ham_ham+=1
        elif(ans==-1 and Y_test[i]==1):
            ham_spam+=1
    return confusion_matrix(ham_ham,ham_spam,spam_ham,spam_spam)


def final_result(X_train, Y_train, test_message):
    trainer = SVMTrainer(Kernel.linear(), 0.1)
    predictor = trainer.train(X_train, Y_train)
    result = predictor.predict(test_message)
    if result == 1:
        messagebox.showinfo("Сообщение",
        "Обработка завершена. Сообщение классифицировано как не спам")
    elif result == -1:
        messagebox.showinfo("Сообщение",
                            "Обработка завершена. Сообщение классифицировано
как спам")


def write_to_file(matrix,result,start_time):
    f = open("results.txt","a")
    f.write("Linear model")
    f.write("\n")
    f.write(matrix.get_string())
    f.write("\n")
    f.write("Precision : " + str(round(result['precision'],2)))
    f.write("\n")
    f.write("Recall : " + str(round(result['recall'],2)))
    f.write("\n")
    f.write("Time spent for model : " + str(round(time()-start_time,2)))
    f.write("\n")
    f.write("\n")
    f.write("\n")
    f.close()

global_start_time = time()

cvxopt.solvers.options['show_progress'] = False

df1 = pd.read_csv('wordslist.csv')
df2 = pd.read_csv('frequency.csv',header=0)

input_output = df2.as_matrix(columns=None)
X = input_output[:,:-1]
Y = input_output[:,-1:]

total = X.shape[0]
train = int(X.shape[0] * 70 / 100)

X_train = X[:train,:]
Y_train = Y[:train,:]
X_test = X[train:,:]
```

```
Y_test = Y[train:,:]

f = open("results.txt","w+")
f.close()
k=0

start_time = time()
matrix , result = implementSVM(X_train,Y_train,X_test,Y_test)
write_to_file(matrix,result,start_time)
k+=1
print("Done : " + str(k))

f = open("results.txt","a")
f.write("Time spent for entire code : " + str(round(time()-
global_start_time,2)))
f.close()
```

unique_words_from_emails.py

```
import os
import nltk
import time
import string
import operator
import numpy as np
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer

def text_cleanup(text):
    text_without_punctuation = [c for c in text if c not in
string.punctuation]
    text_without_punctuation = ''.join(text_without_punctuation)
    text_without_stopwords = [word for word in
text_without_punctuation.split() if word.lower() not in
stopwords.words('english')]
    text_without_stopwords = ' '.join(text_without_stopwords)
    cleaned_text = [word.lower() for word in text_without_stopwords.split()]
    return cleaned_text

start_time = time.time()

lmtzr = WordNetLemmatizer()
k=0
count = {}

directory_in_str = "emails/"
directory = os.fsencode(directory_in_str)

for file in os.listdir(directory):
    file = file.decode("utf-8")
    file_name = str(os.getcwd()) + '/emails/'
    file_name = file_name + file
    file_reading = open(file_name,"r",encoding='utf-8', errors='ignore')
    words = text_cleanup(file_reading.read())
    for word in words:
        if (word.isdigit()==False and len(word)>2):
            word = lmtzr.lemmatize(word)
            if word in count:
                count[word] += 1
```

```
            else:
                count[word] = 1
    k+=1
    if(k%100==0):
        print("Done " + str(k))

sorted_count = sorted(count.items(),key=operator.itemgetter(1),reverse=True)
sorted_count = dict(sorted_count)

f= open("wordslist.csv","w+")
f.write('word,count')
f.write('\n')
for word , times in sorted_count.items():
    if times < 100:
        break
    f.write(str(word) + ',' + str(times))
    f.write('\n')
f.close()

print('Time (in seconds) to pre process the emails ' + str(round(time.time()
- start_time,2)))
```

find_occurance_of_words_in_emails.py

```
import os
import string
import numpy as np
import pandas as pd
from time import time
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer

start_time = time()

df = pd.read_csv('wordslist.csv',header=0)
words = df['word']

lmtzr = WordNetLemmatizer()

directory_in_str = "emails/"
directory = os.fsencode(directory_in_str)

f = open("frequency.csv","w+")
for i in words:
    f.write(str(i) + ',')
f.write('output')
f.write('\n')
f.close()

k=0
for file in os.listdir(directory):
    file = file.decode("utf-8")
    file_name = str(os.getcwd()) + '/emails/'
    for i in file:
        if(i!='b' and i!="'"):
            file_name = file_name + i
    k+=1
    file_reading = open(file_name,"r",encoding='utf-8', errors='ignore')
    words_list_array = np.zeros(words.size)
```

```
    for word in file_reading.read().split():
        word = lmtzr.lemmatize(word.lower())
        if(word in stopwords.words('english') or word in string.punctuation
or len(word)<=2 or word.isdigit()==True):
            continue
        for i in range(words.size):
            if(words[i]==word):
                words_list_array[i] = words_list_array[i]+1
                break
    f = open("frequency.csv","a")
    for i in range(words.size):
        f.write(str(int(words_list_array[i])) + ',')
    if(len(file_name)==77):
        f.write("-1")
    elif (len(file_name)==80):
        f.write("1")
    f.write('\n')
    f.close()
    if(k%100==0):
        print("Done " + str(k))

print("Time (in seconds) to segregate entire dataset to form input vector " +
str(round(time() - start_time,2)))
```

gui.py

```
from tkinter import *
from tkinter.font import Font
from tkinter import filedialog
from tkinter import messagebox

root = Tk()
path = None


def openfolder():
    root.filename = filedialog.askopenfilename(initialdir="C:\\",
                                               title="Выберите файл",
                                               filetypes=(("txt files",
"*.txt"),
                                                          ("all
files","*.*")))
    global path
    path = root.filename
    mesagebox_test()
    print(path)


def gui():
    global root
    root.title("Детектирование")
    root.geometry("500x300")
    root.configure(background="light blue")
    Label(root, text="\nДобро пожаловать в программу\n"
                     "детектирования спам-сообщений! ",
          bg="light blue",
          font=Font(family="Times New Roman", size=14)).pack()
    Label(root, text="\nДля начала работы загрузите файл с сообщением.",
          bg="light blue",
          font=Font(family="Times New Roman", size=14)).pack()
```

```
Label(root, text="\nПо окончанию классификации\n"
                 "результат появится в соответствующем окне.",
      bg="light blue",
      font=Font(family="Times New Roman", size=14)).pack()
Button(root, text="Загрузить сообщение",
       command = openfolder).place(x=100,
                                   y=200,
                                   width=300,
                                   height=50)
root.mainloop()
```