Московский государственный университет
имени М. В. Ломоносова

Факультет вычислительной математики и кибернетики
Кафедра автоматизации систем вычислительных комплексов

Магистерская диссертация

«Автоматическая система классификации текстов для базы знаний предприятия»

Работу выполнил
Попков Максим Иванович

Научный руководитель
Кандидат физико-математических наук
Писковский Виктор Олегович

«____» _____ 2014 г.

МОСКВА – 2014 г.

Оглавление

Аннотация

В работе рассмотрены методы машинного обучения для решения задачи классификации данных. Проведено исследование методов индексации, взвешивания и классификации для корпуса документов базы знаний предприятия. Рассмотрены метрики сравнения классификаторов и получены результаты сравнений в рамках существующей инфраструктуры.

Предложен способ использования существующего поискового индекса для решения задачи классификации документов. Разработана реализация и сценарии работы автоматической системы классификации текстов.

Введение

В связи с научно-техническим прогрессом объем документации на предприятиях имеет тенденцию к постоянному и все ускоряющемуся увеличению. Согласно результатам исследования компании IDC в работе Digital Universe Study, объем накопленных данных в компаниях будет удваиваться каждые 18 месяцев (утверждение на 2011 год). [1]

На рисунке 1 показан объем прироста документов в разрезах по типам данных. [2] Данные на диаграмме свидетельствуют об увеличении объема корпоративных документов: электронной почты и офисных документов. Согласно исследованиям компании Docflow [3], на поиск необходимой информации и документов уходит более 9 часов в неделю. Поиск является одним из проблемных мест в управлении корпоративными данными. Для успешного ведения бизнеса компаниями необходимы автоматические системы поиска и анализа информации.

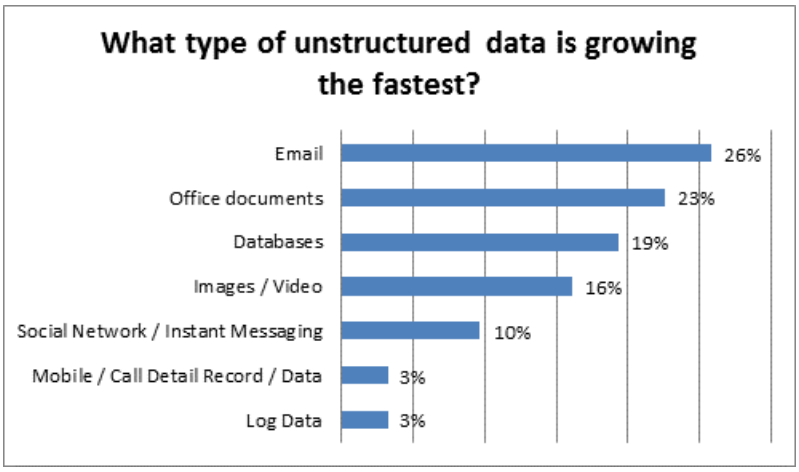


Рисунок 1: Прирост документов в разрезах по типам данных

В рамках данной работы рассматривается создание автоматической системы классификации текстов. Вот некоторые из бизнес задач, решить которые помогают системы автоматической классификации:

- Обнаружение спама – почта, комментарии в блоге компании
- Классификация обращений – определений категорий новых обращений в службы клиентской поддержки
- Классификация документов – определение новых документов в одну из предопределенных категорий
- Обнаружение мошеннических операций при проведении финансовых транзакций

Целью данной работы является создание автоматической системы классификации текстов базы знаний. Рассмотрены инструменты для создания автоматической системы и способы её интеграции в существующую инфраструктуру.

Если данные никак не структурировать, не распределять по категориям, то через некоторое время в предприятии будут накоплены массивы плохо структурированных данных. Отсутствие возможности вовремя и быстро получить необходимую информацию по нужной теме сделает бесполезной большую часть накопленной базы знаний компании.

Применение автоматических систем классификации текстов позволит сократить трудозатраты человека на рубрикацию и поиск необходимой информации предоставляемой

информационными системами, а также позволит сократить участие или вовсе не привлекать специалистов при размещении информации в базе знаний предприятия.

Постановка задачи

1. Описание исходных данных

Под электронной базой знаний предприятия будем понимать совокупность программных средств, обеспечивающих поиск, хранение, преобразование и запись сложно структурированных информационных единиц (знаний). [4]

Знания – это совокупность сведений, образующих описание, соответствующее некоторому уровню осведомленности об описываемом предмете, событии или проблеме. Знание в электронном виде – это документ. [5]

В рамках данной работы, рассматривается база знаний небольшого предприятия. Типичные виды документов базы знаний рассматриваемого предприятия – это тексты по информационным технологиям, научные статьи, описания алгоритмов и презентации с конференций.

Каждый документ имеет свою уникальную категорию. Между категориями нет иерархической зависимости, и они организованы в виде плоской одноуровневой структуры.

Характерная особенность рассматриваемых документов в том, что количество слов в статье может сильно варьироваться от 350 до 7000 слов и более. Количество категорий базы знаний не большое, от 4 до 10, при этом все категории тематически близки из-за узкого профиля предприятия. Количество документов внутри категорий сильно варьируется, от 2 до 60.

2. Содержательная постановка задачи

Для существующей базы знаний предприятия необходимо провести исследование данных, предложить модель для автоматической классификации данных и разработать классификатор удовлетворяющий требованиям предприятия.

Для существующей базы знаний предприятия необходимо

- Предложить методы для индексации и автоматической классификации данных
- Разработать классификатор удовлетворяющий требованиям предприятия

В рассматриваемом предприятия существует несколько баз знаний, поэтому разрабатываемый классификатор данных должен быть внешним по отношению к разрабатываемой системе и не зависеть от её технических особенностей.

Предполагается что количество документов в системе относительно мало и не превышает 10 000 документов.

3. Формальная постановка задачи

Пусть $D = \{d_1, \dots, d_{|D|}\}$ — множество документов, $\mathrm{C} = \{c_1, \dots, c_{|C|}\}$ — множество категорий, $\Phi: D \times C \to \{0, 1\}$ — неизвестная целевая функция, которая по паре $< d_i, c_j >$ говорит, принадлежит ли документ d_i категории c_j (1 или T) или нет (0 или F). [6]

Задача классификации состоит в построении классификатора $\Phi': D \times C \to \{0, 1\}$, максимально близкого к Ф.

Выше была поставлена задача точной классификации, т.е. каждый документ относится только к одной категории. В работе используются классификаторы с ранжированием, при котором множество значений целевой функции – это значения из интервала [0,1]. Документ при ранжировании может относится сразу к нескольким категориям с разной степенью принадлежности.

4. Коллекции документов для классификации

Методы машинного обучения, используемые для классификации, полагаются на наличие коллекции $\Omega = \{d_1, \dots, d_{|\Omega|}\}$, $\Omega \subset D$ заранее классифицированных документов, то есть таких, для которых точно известно значение целевой функции Ф. Для того, чтобы после построения классификатора можно было оценить его эффективность, Ω разбивается на две части, не обязательно равного размера. [6]

1. Обучающая (*training-and-validation*) коллекция. Классификатор строится на основании характеристик этих документов (то есть, вообще говоря, зависит от обучающей коллекции).
2. Тестовая (*test*) коллекция. На ней проверяется качество классификации. Документы из тестовой коллекции никаким образом не должны участвовать в процессе построения классификатора, иначе полученные результаты теста уже не будут соответствовать действительности.

Кроме приведённого выше разбиения, авторы рекомендуют дополнительно выделить из учебной коллекции часть документов в проверочную (*validation*) коллекцию, на основании которой оптимизируются параметры классификатора. [6]

Глава 1 «Обзор методов классификации данных»

1. Основные этапы классификации

Первый этап решения задачи автоматической классификации текстов – это преобразование документов. На этом этапе документы, имеющие вид последовательности символов, преобразуются к виду, пригодному для алгоритмов машинного обучения в соответствии с задачей классификации. Обычно алгоритмы машинного обучения имеют дело с векторами в пространстве, называемом пространством признаков. [7]

Вторым этапом является построение классифицирующей функции Ф` при помощи обучения на примерах. Качество классификации зависит и от того, как документы будут преобразованы в векторное представление, и от алгоритма, который будет применен на втором этапе. При этом важно отметить, что методы преобразования текста в вектор специфичны для задачи классификации текстов и могут зависеть от коллекции документов, типа текста (простой, структурированный) и языка документа. Методы машинного обучения, применяемые на втором этапе, не являются специфичными для задачи классификации текстов и применяются также в других областях, например, для задач распознавания образов. [7]

Итак, классическая задача классификации может быть разбита на два основных этапа:

1. Предобработка/Индексация – отображение текста документа на его логическое представление, например, вектор весов $\overline{d_j}$, который затем подается на вход алгоритму классификации
2. Классификация/Обучение – этап классификации документа или обучения на множестве документов, основанный на логическом представление документа. Важно отметить, что для классификации и обучения может быть использован общий метод предобработки/индексации текстов.

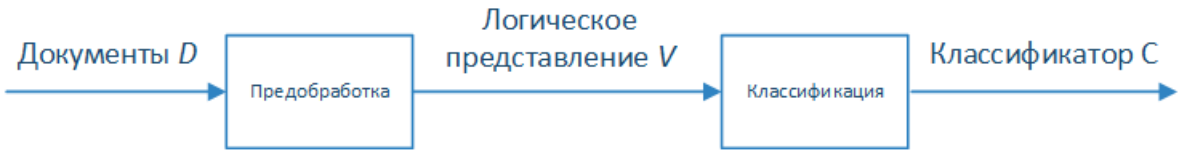


Рисунок 2: Основные этапы классификации

2. Индексация и предобработка документа

Как было сказано выше, этап предобработки отображает текст документа на логическое представление.

Текст представляется в виде мультимножества термов (слов). Множество всех термов $T = \{\tau_0, .., \tau_{|T|}\}$. Каждому терму $\tau_i \in T$ сопоставлен некоторый вес w_{ij}, $0 \leq w_{ij} \leq 1$, числовая характеристика встречаемости этого слова в документе $d_j \in D$. Порядок слов, как правило, не учитывается, учитывается только частота встречаемости слова в тексте и, возможно, другие признаки, такие как: «слово встретилось в заголовке», «слово заключено в теги» и т.д. На основании этих признаков каждому слову в тексте сопоставляется его вес.

Логическое представление документа $\overline{d_j}$ в таком случае может быть получено извлечением всех значимых термов из всех документов D и присваиванием каждому терму документа $\overline{d_j}$ своего веса. Более формально, каждый документ представлен вектором n-мерной размерности $\overline{d_j} = < w_{1j}, .., w_{|T|j} >$. где каждый компонент w_{ij} является весом i-го терма из множества термов T в документе $\overline{d_j}$ Полученное в итоге n-мерное пространство векторов принято называть *пространством признаков* для категории документов D. Каждый документ - это точка в пространстве признаков.

Размерность вектора — это количество термов, которые встречаются в документах множества D. Из-за того, что учитываются все слова, которые когда-либо встретились в документах, вектор документа имеет огромное количество координат, большинство из которых нули.

Предобработка документа, в таком случае, это преобразование последовательности термов документа в n-мерное векторное пространство.

Процесс получения вектора весов $\overline{d_j}$ для документа называют индексацией документа. Индексацию можно представить в виде трех этапов:

1. Извлечение термов *(Term extraction)* – на этом этапе применяются методы для поиска и выбора наиболее значимых термов в корпусе документов.
2. Взвешивание термов *(Term weighting)* – определение значимости терма для выбранного документа
3. Уменьшение размерности векторов (Dimensionality reduction) – процесс сокращения векторного пространства

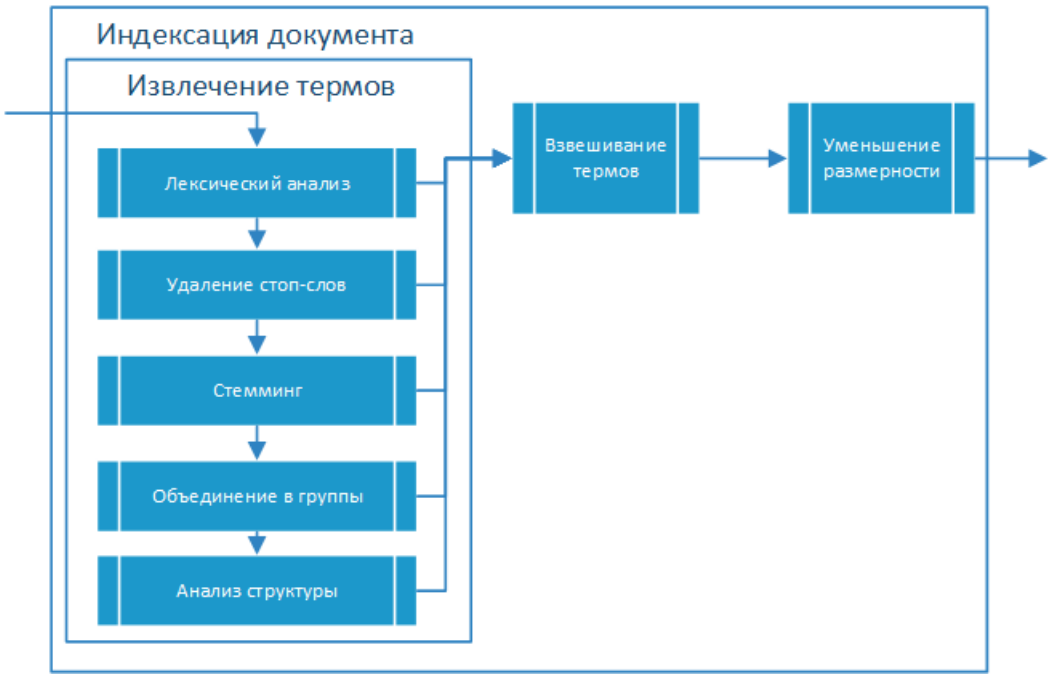


Рисунок 3: Индексация документа

2.1. Извлечение термов

Извлечение термов, или извлечение признаков – это процесс разбиения текста на более простые объекты, которые так же называются термами. Результат данного процесса – это множество термов *T*, которое используется для получения весовых характеристик документа.

Лексический анализ

Лексический анализ - первый шаг извлечения термов. На этом этапе отсеиваются все символы, которые не являются буквами, например: знаки препинания и html-теги.

Удаление стоп-слов

Стоп-слова – это слова, не несущие какой-либо самостоятельной смысловой нагрузки. К стоп-словам относятся предлоги, союзы и местоимения. [8] В целях уменьшения размерности пространства термов индексатор не учитывает стоп-слов и удаляет их при анализе. Так же стоп-слова сильно влияют на отбор ключевых слов. Если их не удалить, они засоряют множество термов, так как встречаются часто в тексте.

Дополнительно стоит отметить, что бывают ситуации, когда есть смысл отказаться от удаления стоп-слов. В статье «Little Words Can Make a Big Difference for Text Classification» для новостных статей на английском языке было показано, что от тематике к тематике сильно меняются фразовые глаголы и точность классификации с учетом стоп-слов может возрасти. [9] Для русских статей нет упоминаний о проведении схожих исследований.

Лемматизация и стемминг

Лемматизация – это приведение каждого слова в документе к его нормальной форме.[10] В русском языке нормальными формами считаются:

- Для существительных и прилагательных – именительный падеж, единственное число, мужской род.
- Для глаголов, причастий и деепричастий – глагол в неопределенный форме.

При построении множества термов часто пренебрегают формами слова. Это оправдано, так как, сохраняя формы слов, пространство термов и структуры хранения будут быстро расти, что ухудшит производительность, а статистика будет делится между формами одного слова, ухудшая общую картину.

Стемминг – отбрасывание изменяемых частей слов, главным образом, окончаний. Эта технология более простая, не требует хранения словаря слов или большого набора правил. Технология основана на правилах морфологии языка. Недостаток стемминга – это большое число ошибок. Стемминг хорошо подходит для английского языка, но хуже для русского.[10]

Одна из проблем при рассмотрении слов в качестве термов – это их семантическая неоднозначность, которую условно можно подразделить на две группы:

1. Синонимы – слова одной части речи, различные по звучанию и описанию, но имеющие похожее лексическое значение (идти – шагать, смелый – храбрый) [12]
2. Омонимы – разные по значению, но одинаковые по написанию единицы языка (Лук – оружие или растение) [13]

Разрешить эту неопределенность можно, используя контекст слова в предложении. Для этого используются методы морфологического и лингвистического анализа. [10]

Объединение в группы, N-граммы

Объединение в группы – это процесс объединения нескольких последовательных слов в одну группу, которую так же называют N-граммой. В таком случае, каждая N-грамма, рассматривается как самостоятельный терм документа.

N-граммы нашли применение в выявлении плагиата. Если разделить текст на несколько небольших фрагментов, представленных N-граммами, их легко сравнить друг с другом и таким образом получить степень сходства контролируемых документов. [10]

N-граммы успешно используются для классификации текста и языка.

Используя N-граммы, можно эффективно найти кандидатов, чтобы заменить слова с ошибками правописания. [10]

Основной недостаток применения N-грамм, это быстро растущий объем памяти необходимый для их хранения.

2.2. Взвешивание термов с использованием статистических мер

Один из популярных методов представить вес терма — метод TF-IDF. Вес вычисляется специальной весовой функцией. Ниже приведены возможные её варианты:

TF ***(term frequency — частота терма)*** — отношение числа вхождения некоторого терма к общему количеству термов документа. Таким образом, оценивается важность терма τ_i в пределах отдельного документа d_j. [14]

Пусть fr_{ij} — число вхождений терма τ_i в документ d_j. Тогда частота терма определяется как:

$$TF(\tau_i, d_j) = \frac{fr_{ij}}{\sum_i fr_{ij}}, \quad \text{где } 0 \le i \le |T|, 0 \le j \le |D| \qquad (1)$$

IDF ***(inverse document frequency — обратная частота документа)*** — инверсия частоты, с которой некоторое слово встречается в документах коллекции. Учёт IDF уменьшает вес широкоупотребительных слов. Для каждого уникального слова в пределах конкретной коллекции документов существует только одно значение IDF. [14]

$$IDF(\tau_i, D) = \frac{|D|}{|(d_i \supset t_i)|} \quad (2)$$

где, |D| — количество документов в коллекции, $|(d_i \supset \tau_i)|$ — количество документов, в которых встречается τ_i (когда $fr_i \neq 0$), $0 \leq i \leq |T|$.

TF-IDF — статистическая мера, используемая для оценки важности слова в контексте документа, являющегося частью коллекции документов или корпуса. Вес некоторого слова пропорционален количеству употребления этого слова в документе и обратно пропорционален частоте употребления слова в других документах коллекции. [14]

$$w_{ij} = TF(\tau_i, d_j) * IDF(\tau_i, D) \quad (3)$$

TF-SLF

Мера TF-IDF рассматривает важность терма в рамках всего корпуса документов. При такой оценке игнорируется важность терма в рамках отдельно взятой категории. Предположение что оценка терма не должна зависеть от категории и должна быть одинаковой в рамках всего корпуса может работать неэффективно, когда документов в корпусе меньше 1000 и они являются тематически близкими. Это было показано в работе [15] и подобное ухудшение качества классификации наблюдается для корпуса документов рассматриваемой базы знаний. Результаты приведены в пункте 6 данной работы.

Для преодоления данного ограничения рассмотрим метрику TF-SLF [15], основанную на следующих предположениях:

1. Терм является важным в рамках категории, если он встречается в большинстве документов данной категории
2. Оценка терма понижается, если он является важным для нескольких категорий

Введем следующие обозначения:

3. NDF_{tc} – нормализованная частота встречаемости терма *t* в категории *c*
4. df_{tc} – число документов категории *c* в которых встречается хотя бы раз терм *t*
5. N_c – количество документов в категории *c*
6. C – множество категорий в корпусе документов
7. SLF_t – логарифмированная сумма частот терма *t*

$$NDF_{tc} = \frac{df_{tc}}{N_c} \quad (4)$$

Оценка NDF_{tc} локальна для категории. Для получения глобальной оценки R_t в рамках всего корпуса все NDF_{tc} суммируются:

$$R_t = \sum_{c \in C} NDF_{tc} \quad (5)$$

Логарифмированная сумма частот вычисляется как:

$$SLF_t = \log \frac{|C|}{R_t} \quad (6)$$

SLF позволяет устранить дисбаланс между категориями с малым числом документов и категориями с большим числом документов. [15]

Оценка TF-SLF для терма *t* вычисляется как:

$$TFSLF_t = TF_t * SLF_t \quad (7)$$

2.3. Взвешивание термов с использованием графа термов

Построение графа термов

Граф термов строится на основе последовательности термов документа. Для построения графа выбирается окно r, $0 \leq r \leq |T|$, число последовательно идущих термов. Все термы внутри окна объединяются ребрами.

Граф состоит из набора вершин и связей между ними - ребер. Каждая вершина представляет уникальный терм документа. Ребра связывают термы, встретившиеся внутри одного предложения. Таким образом вершины и ребра графа сохраняют зависимости между словами внутри предложений документа.

Пусть L= $(l_1, \dots, l_{|L|})$ — последовательность слов документа *D*, $l_i \in T$. Построим граф $G = \{V, E\}$ документа *D*. Множество *V* – множество вершин графа *G*, где каждая вершина v_i представлена термом t_i. Множество *E* – множество ребер графа *G*, где каждое ребро e_{ij} соединяет вершины v_i и v_j в только том случае, когда термы τ_i и τ_j:

$\exists\, l_i, l_j \in L$ и $i + 1 \leq j \leq i + r$,

то есть встречаются хотя бы один раз в тексте внутри окна r.

Для построения графа алгоритм последовательно проходит по термам текста. Он начинает с первого терма и добавляет вершины на граф, связывая их ребрами внутри каждого окна. Таким образом, вершины и ребра графа сохраняют связи между словами внутри документа.

Ниже приведен пример построения графа с окном длины 1 и 2 для последовательности термов:

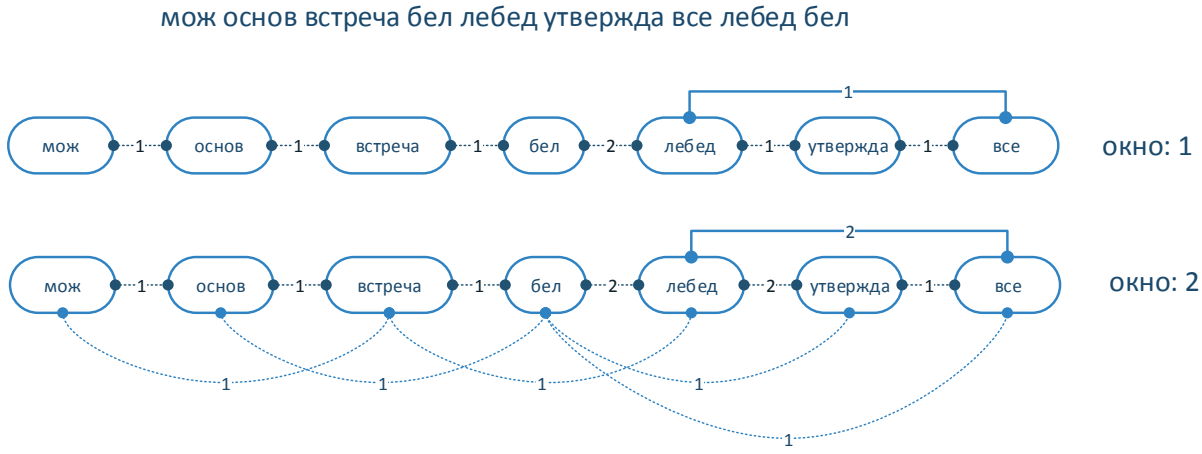


Рисунок 4: Графы термов с разными размерами окна

Алгоритм 1: Построение графа термов

Входные данные

i ← 0 // индекс начала окна
r ← размер окна
V ← {} // множество вершин графа G
E ← {} // множество ребер графа G
G ← {} // граф G

Результат работы

Граф термов

Сценарий

1: **Для каждого** терма $l_i .., l_{i+r} \in$ L:
   a. **Если** $l_k \notin V$, то добавить вершину l_k на граф G
   b. Для каждой пары термов $e = (l_j, l_k)$, $l_j, l_k \in$ L, если $e \notin E$, $i + 1 \leq j \leq i + r$, добавить e на граф G
2: i=i+1
3: **Если** i+r < |L|, перейти на шаг 1
4: **Конец**

Рассмотрим алгоритмы взвешивания термов с использованием графа термов. Один из алгоритмов взвешивания вершин графа – Random-Walk. Основная идея алгоритма Random-Walk – голосования, или рекомендации, которыми обмениваются связанные вершины. Чем больше было отдано голосов за вершину, тем выше её вес. Помимо этого вес вершины в голосовании может определяться важностью самого голосования.

Для алгоритма Random-Walk существует много реализаций, например, PageRank - алгоритм ссылочного ранжирования [17], ниже будут рассмотрены его модификации.

Алгоритм PageRank для ориентированного графа

Рассмотрим механизм работы алгоритма PageRank для ориентированного графа.

Пусть *G = (V,E)* - направленный граф с множеством вершин *V* и множеством ребер *E*. Пусть $V_a \in V$ и $V_b \in V$ - две вершины a и b, принадлежащие графу *G*. Назовем $In(V_a)$ множеством входящих вершин, которые ссылаются на вершину V_a, и множество $Out(V_a)$ – множество исходящих вершин, на которые ссылается V_a, Тогда вес вершины V_a, обозначим его $S(V_a)$, определяется итерационно рекуррентной функцией от всех входящих вершин:

$$S^1 = 0.25$$
$$S^{n+1}(V_a) = (1-d) + d * \sum_{V_b \in In(V_a)} \frac{S^n(V_b)}{|Out(V_b)|} \qquad (8)$$

Где d – коэффициент затухания, экспериментально подобранное значение которого может быть принято от 0 до 1. На основе множества исследований с применением разных коэффициентов затухания в литературе рекомендуют d выбирать равным 0.85. [17]. Введение коэффициента затухания гарантирует, что процесс вычисления веса сходится. Вес вершин S^1 на первой итерации рекомендуют выбирать равным 0.25 на основе экспериментальных исследований сходимости процесса. [17, 46]

Вес каждой вершины пересчитывается последовательно на основе значений веса у входящих вершин. Алгоритм останавливается, когда вес всех вершин достигнет своего, заранее определенного порога сходимости.

Алгоритм PageRank для неориентированного графа без учета веса ребер

Для взвешивания вершин в графе термов применяется похожий подход. Пусть теперь *G = (V,E)* - неориентированный граф. Пусть $V_a \in V$ и $V_b \in V$ - две вершины a и b, принадлежащие графу *G*. Назовем $Occ(V_a)$ множество смежных вершин вершины V_a, тогда в простейшем случае вес вершины вычисляется по формуле:

$$S_{pr}^1 = 0.25$$
$$S_{pr}^{n+1}(V_a) = (1-d) + d * \sum_{V_b \in Occ(V_a)} \frac{S_{pr}^n(V_b)}{|Occ(V_b)|} \qquad (9)$$

Алгоритм PageRank для неориентированного графа со взвешенными ребрами

Рассмотрим способ взвешивания вершин с учетом веса ребер. Пусть *G = (V,E)* - неориентированный взвешенный граф.

Пусть $TFIDF_{V_k} = \mathrm{TFIDF(k, d)}$, то есть весу TFIDF *k*-го терма в рассматриваемом документе d.

Присвоим каждому ребру $E_{ij} = (V_i, V_j)$ вес M_{ij}, рассчитанный по следующей формуле:

$$\mathrm{M}_{ij} = TFIDF_{V_i} * TFIDF_{V_j} \quad (10)$$

Нормализованный вес ребра E_{ij} в таком случае вычисляется по формуле:

$$W_{ij} = M_{ij}/M_{max} \quad (11)$$

Где M_{max} - наибольший вес ребра в графе G. Таким образом, формула расчета веса вершины выглядит так:

$$S^1_{rwidf} = 0.25$$
$$S^{n+1}_{rwidf}(V_a) = \frac{(1-d)}{|N|} + C * \sum_{V_b \in Occ(V_a)} \frac{W_{ab} * S^n_{rwidf}(V_b)}{|Occ(V_b)|} \quad (12)$$

Где $N = |V|$, количество вершин в графе G, d – коэффициент затухания, C – коэффициент масштабирования [19], его принимают равным 0.95. Для вершин графа без входящих ребер вес устанавливается равным:

$$S_{min} = (1-d)/N \quad (13)$$

Данная модель имеет несколько недостатков:

1. Погрешность определения веса вершины по данной модели увеличивается для вершин, где входящие ребра имеют наибольший вес.
2. Не учитывается плотность связи между двумя термами в тексте, т.е. число вхождений термов вместе

Алгоритм PageRank для неориентированного графа с учетом частоты вхождения пар термов

Предлагается другой способ вычислить вес ребра – присвоить ему частоту вхождения пары термов вместе. Например, если выражение *«Бесплатный софт»* встретилось 4 раза, тогда вес ребра соединяющего вершины *«бесплатный»* и *«софт»* будет равен 4. Формула расчета веса ребра будет выглядеть так:

$$W_{ij} = |E_{ij}| \quad (14)$$

Где $|E_{ij}|$ - число вхождений пары термов в тексте вместе.

Пусть fr_{V_i} – частота вхождения терма τ_i, в рассматриваемом документе d_j.

Вычисление веса вершины можно представить следующей формулой:

$$S^1_{rwoc} = 0.25$$
$$S^{n+1}_{rwoc}(V_a) = \frac{(1-d)}{|N|} + d * \sum_{V_b \in Occ(V_a)} \frac{W_{ab} * S^n_{rwoc}(V_b)}{|Occ_{TF}(V_b)|} \quad (15)$$

Здесь $|Occ_{TF}(V_b)| = \sum_{V_a \in Out(V_b)} fr_{V_a}$ - количество выходящих вершин с учетом частоты вхождения термов вершин.

Данный метод вычисления веса вершин является модификацией метода, предложенного авторами работы: «Random-Walk Term Weighting for Improved Text Classification». [19]

Метод, описанный в статье, представлен ниже:

$$S_{hs}^{n+1}(V_a) = \frac{(1-d)}{|N|} + d * \sum_{V_b \in Occ(V_a)} \frac{W_{ab} * S_{hs}^{n}(V_b)}{|Occ(V_b)|} \qquad (16)$$

Данный метод не всегда сходится, поэтому существуют тексты, на которых итерационный процесс не остановится. Метод не сойдется в случае, если у вершины будет достаточное число ребер, вес которых больше единицы. Т.к. при нормализации в дроби учитывается только число смежных вершин $Occ(V_b)$, а число термов, вошедших в эти вершины, не учитывается, то дробь под суммой может стать больше единицы и будет только увеличиваться на каждой новой итерации.

Существенное отличие предложенной модификации заключается в изменении знаменателя дроби, стоящей под знаком суммы. При нормализации теперь учитывается наряду с количеством смежных вершин, число вхождений термов, описываемых этими вершинами. Т.е. $|Out_{TF}(V_b)| = \sum_{V_a \in Out(V_b)} fr_{V_a}$

Итак, общее описание процесса взвешивания термов алгоритмами семейства PageRank:

1. Для каждого терма вычислить значения *TF, IDF и TFIDF*;
2. Выбрать окно *r*, число термов которые будем считать связанными в графе *G*
3. Последовательно читаем *r* термов
    a. Создаем для термов, попавших в окно *r*, вершины: $V_k..V_{k+r}$, $0 \le k \le |T| - r$, и добавляем их на граф *G*
    b. Связываем вершины из окна *w* ребрами.
4. Последовательно для всех вершин пересчитываем S_{weight} до достижения всеми вершинами порога сходимости
5. $w_{ij} = S(V_i)$

Ниже приведен пример взвешивания текста алгоритмом Random-Walk с окном *r*, равным 2, для текста:

« Землетрясение магнитудой 6,6 произошло в четверг, 31 октября, в районе города Кокимбо в центральной части Чили, передает агентство Reuters.

Очаг землетрясения находился в океане на глубине 10,7 километра и на удалении 54 километров к юго-западу от города, уточняется на сайте Геологической службы США (USGS). Толчки были зафиксированы в 20:03 по местному времени (3:03 1 ноября по Москве).

О жертвах или о вызванных подземными толчками разрушениях пока ничего не сообщается.

Напомним, Чили находится в одном из самых сейсмоактивных районов Земли, поэтому землетрясения различной силы там не редкость. Одно из наиболее серьезных произошло в 2010 году. Тогда землетрясение магнитудой 8,8 и вызванное им цунами привели к гибели в Чили более 550

человек и разрушению 220 тысяч зданий. Кроме того, в Чили произошло самое мощное в новейшей истории землетрясение: в 1960 году там зафиксировали магнитуду 9,5. В результате того стихийного бедствия погибли более пяти тысяч человек. ».

Терм	tf	rw-oc
Чил	4	0,426746
землетрясен	5	0,414722
произошл	3	0,334076
магнитуд	3	0,290638
Город	2	0,287796
Район	2	0,265217
зафиксирова	2	0,250155
разрушен	2	0,249372
Одн	2	0,249062
Сам	2	0,239496

Терм	tf	rw-oc
тысяч	2	0,238559
километр	2	0,231826
толчк	2	0,230875
вызва	2	0,227847
центральн	1	0,214739
агентств	1	0,214739
сша	1	0,214739
стихийн	1	0,214739
reuters	1	0,214739
привел	1	0,214739

В таблице приведены результаты взвешивания слов по частоте документа (*TF*) и используя алгоритм Random-Walk (*rw-oc*). Все веса отсортированы по столбцу *rw-oc*. Анализируя полученные веса, можно заметить, что их результаты не сильно отличаются. Такие же наблюдения справедливы для текстов больших размерностей.

Алгоритм Random-walk, используя знания о связности слов, выделил и поднял некоторые ключевые слова, не учтенные методом *TF*, например, слово «Чили». Для данной статьи, это слово есть смысл считать важным ключевым словом, так как статья про землетрясение в «Чили», близость его к другим ключевом словам на графе подняла важность терма.

Метод оценки веса Random-Walk более детально предлагает перераспределять веса. Значения весов представлены вещественным числами и зависят не только от частоты вхождения слова в документе, но и от места слова в тексте. Таким образом там, где метод tf предлагает группу слов с одинаковым значением веса, метод Random-Walk позволяет упорядочить слова внутри данной группы. Учитывая, что группа слов с одинаковым весом tf может быть очень большой, упорядочивание внутри группы с перераспределением важности весов между словами должно сделать результат процесса извлечения термов более качественным.

2.4. Уменьшение размерности векторов

Для сокращения размерности векторов можно не учитывать редкие слова, которые увеличивают размер пространства, но, как правило, не несут полезной для классификатора информации. Также можно не рассматривать слишком часто встречающиеся слова, такие как: предлоги, артикли и т.п. Для каждого терма можно определить его коэффициент значимости, т.е. насколько этот терм полезен для классификации. Эту характеристику можно определить, основываясь на корреляции между встречаемостью слова в документе и принадлежностью этого документа к одной или нескольким категорий.

Кроме удаления лишних термов, можно группировать несколько термов в один. Например, можно группировать вместе синонимы. Ещё один подход — «совместная встречаемость» (cooccurrence): объединять слова, часто встречающиеся в одном окружении. Например, в словосочетаниях «руководитель компании», «директор компании» слова «руководитель» и «директор» встречаются перед словом «компания». Поэтому их можно объединить в один искусственный терм. В общем случае для слов определяется некая метрика близости, и группы близких слов склеиваются в один терм. Вес такого терма в каждом конкретном документе рассчитывается из весов представителей группы, которые встречаются в этом документе.

3. Обучение классификатора

3.1. Линейные классификаторы

Пусть каждой категории c_i соответствует вектор $\bar{c_i} = \{c_{i0}, .., c_{in}\}$, где n — размерность пространства документов. В качестве правила классификатора используется следующая формула:

$$CSV_i(\bar{d}) = \bar{d} \times \bar{c_i} = \sum_j d_j c_{ij} \quad (17)$$

Обычно проводится нормализация так, что итоговая формула для $CSV_i(d)$ — это косинус угла между вектором категории и вектором документа.

$$CSV_i = \frac{\bar{d} \times \bar{c_i}}{|c_i||d|} \quad (18)$$

3.2. Метод Байеса

Метод Байеса основан на анализе совместных распределений признаков документа и категорий. [7] Документу d с вектором признаков $\bar{d} = (d_1, .., d_n)$ сопоставляется наиболее вероятная апостериори категория c_i с вектором признаков $\bar{c_i} = (x_1, .., x_n)$ по формуле:

$$c^* = arg\max_{c \in C} P(c|x_1 = d_1, x_2 = d_2, .., x_n = d_n) \quad (19)$$

В задаче классификации текстов метод Байеса применяется отдельно для каждой категории и принимается решение, принадлежит документ категории или нет.

Апостериорная вероятность принадлежности документа рубрике вычисляется по формуле Байеса, связывающей априорную вероятность с апостериорной:

$$P(c|x_1 = d_1, x_2 = d_2, \dots, x_n = d_n) = \frac{P(x_1 = d_1, x_2 = d_2, \dots, x_n = d_n|c) * P(c)}{P(x_1 = d_1, x_2 = d_2, \dots, x_n = d_n)} \quad (20)$$

И тогда получаем:

$$c^* = Arg \max_{c \in C} \frac{P(x_1 = d_1, x_2 = d_2, \dots, x_n = d_n|c) * P(c)}{P(x_1 = d_1, x_2 = d_2, \dots, x_n = d_n)} \quad (21)$$

Так как знаменатель не зависит от категории, его можно сократить:

$$c^* = Arg \max_{c \in C} P(x_1 = d_1, x_2 = d_2, \dots, x_n = d_n|c) * P(c) \quad (22)$$

Условные вероятности $P(x_1 = d_1, x_2 = d_2, \dots, x_n = d_n|c)$ можно вычислить в предположении условной независимости переменных $x_1, x_2, \dots, x_n$. В этом случае, формула для определения наиболее вероятной категории будет выглядеть следующим образом:

$$c^* = Arg \max_{c \in C} P(c) * \prod_{i=1,..,n} P(x_i = d_i|c) \quad (23)$$

Пусть $Rub(d)$ - это функция, которая по документу d_i возвращает значение категории, в которой состоит документ. *Ex*- множество документов из обучающей выборки.

Для коллекции обучающих документов вероятности $P(x_i=d_i|c)$ вычисляются по формуле

$$P(x_i = d_i|c) = \frac{\left|\{D \in Ex \,|\, c \in Rub(D)\ \&\ D_i = d_i\}\right| + 1}{|\{D \in Ex | c \in Rub(D)|\}|} \quad (24)$$

Добавление единицы в числителе нужно для того, чтобы документы, содержащие не встречающиеся больше нигде признаки (например, уникальные слова), имели отличную от нуля вероятность для всех рубрик. [7]

Метод Байеса обладает высокой скоростью работы и простотой математической модели. Этот метод часто используется в качестве базового метода при сравнении различных методов машинного обучения.

3.3. Метод опорных векторов (SVM)

Метод опорных векторов (англ. SVM, support vector machine) — набор схожих алгоритмов вида «обучение с учителем», использующихся для задач классификации и регрессионного анализа. [21]

Основная идея метода опорных векторов — перевод исходных векторов в пространство более высокой размерности и поиск разделяющей гиперплоскости с максимальным зазором в этом пространстве. Две параллельных гиперплоскости строятся по обеим сторонам гиперплоскости, разделяющей наши классы. Разделяющей

гиперплоскостью будет гиперплоскость, максимизирующая расстояние до двух параллельных гиперплоскостей. Алгоритм работает в предположении, что чем больше разница или расстояние между этими параллельными гиперплоскостями, тем меньше будет средняя ошибка классификатора. [20]

Рассмотрим задачу классификации объектов на два непересекающихся класса.

Пусть:

- X – пространство объектов, $X \in R^n$,
- C – классы, $C \in \{-1,1\}$

Дана обучающая выборка: $(x_1,c_1), .. , (x_m,c_m)$

Требуется построить функцию $F: X \to C$, сопоставляющую класс c произвольному объекту x. Для этого рассмотрим учебную коллекцию, в которой для каждого элемента уже задан класс, к которому он принадлежит. Наша задача – это добиться, чтобы метод опорных векторов классифицировал их таким же образом. Для этого метод строит разделяющую гиперплоскость, которая имеет вид:

$$w * x - b = 0 \quad (25)$$

Вектор w – перпендикуляр к разделяющей гиперплоскости. Параметр b равен по модулю расстоянию от гиперплоскости до начала координат. Если параметр b = 0, гиперплоскость пройдет через начало координат, что ограничит решение.

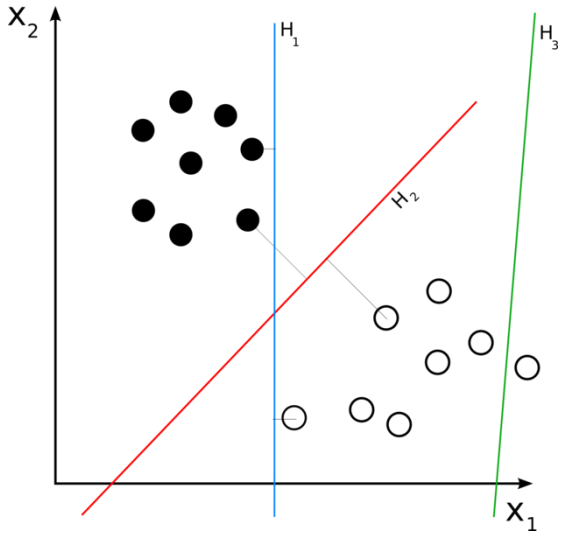


Рисунок 5: Выбор разделяющей гиперплоскости

Оптимальная разделяющая гиперплоскость для метода опорных векторов строится на точках из двух классов. Ближайшие к параллельным гиперплоскостям точки называются опорными векторами.

Так как требуется построить оптимальное разделение, нас будут интересовать и гиперплоскости, которые параллельны оптимальной и расположены максимально близко к опорным векторам двух классов. Не ограничивая общности рассуждений, такие параллельные гиперплоскости можно описать следующими уравнениями:

$$w * x - b = 1$$

$$w * x - b = -1 \quad (26)$$

Если обучающая выборка линейно разделима, то мы можем выбрать гиперплоскости таким образом, чтобы между ними не лежала ни одна точка обучающей выборки и затем максимизировать расстояние между гиперплоскостями. Ширина полосы между ними равна $\frac{2}{||w||}$. Таким образом, задача - минимизировать $||w||$.

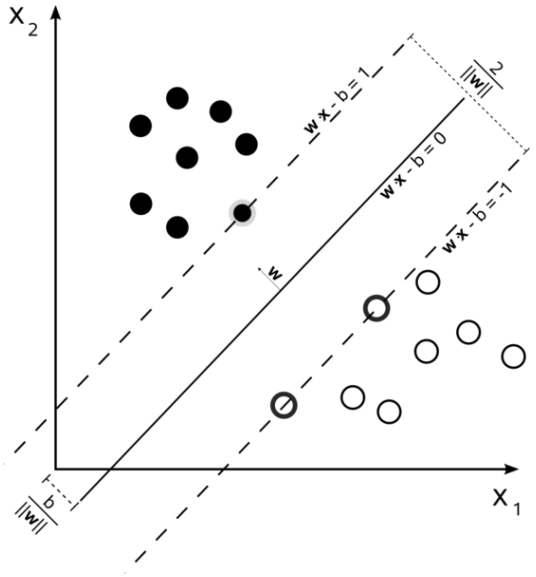


Рисунок 6: Ширина полосы разделения между выборками

Что бы исключить все точки из полосы, мы должны убедиться для всех i, что

$$\begin{bmatrix} w * x - b \geq 1, & c_i = 1 \\ w * x - b \leq 1, & c_i = -1 \end{bmatrix} \quad (27)$$

Или, что тоже самое:

$$c_i(w * x_i - b) \geq 1, \qquad 1 \leq i \leq n$$

Построение оптимальной разделяющей гиперплоскости сводится к задаче минимизации квадратичной формы, которая по теореме Куна-Таккера эквивалента двойственной задаче поиска седловой точки функции Лагранжа. [21]

Ниже представлена эквивалентная задача, содержащая только двойственные переменные $\lambda = (\ \lambda_1, .., \lambda_n)$

$$L_D(\lambda) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j c_i c_j K(x_i, x_j) \rightarrow max\,, \qquad \text{где } i = 1, .., n \qquad (28)$$

$$0 \leq \ \lambda_i \leq C$$

$$\sum_{i=1}^{n} \lambda_i c_i = 0$$

$K(x_i, x_j)$ – функция ядра SVM, которая в простейшем случае равна евклидову скалярному произведению векторов x_i и x_j. Для данной задачи предложены эффективные методы решения. [7]

Константу *C* выбирают по критерию скользящего контроля. Это - трудоемкий способ, так как задачу приходится решать заново при каждом значении *C*.

Подробнее процесс сведения задачи можно посмотреть в [21].

Метод SVM работает с абстрактной векторной моделью предметной области. Это позволяет применять SVM для решения различных задач машинного обучения. SVM используется для задач классификации текстов, распознавания образов и речи.

Преимущества метода:

1. Метод сводится к решению задачи квадратичного программирования в выпуклой области, которая всегда имеет единственное решение [21]
2. Метод находит разделяющую полосу максимальной ширины, что позволяет в дальнейшем осуществить более уверенную классификацию

Недостатки метода:

1. Метод чувствителен к шумам и стандартизации данных
2. Не существует общего подхода к автоматическому выбору ядра в случае линейной неразделимости классов

3.4. Метод Роше

Классификатор Роше (Rocchio) — один из самых простых методов классификации.[7]

Введем следующие обозначения:

1. POS_i – множество документов, принадлежащих категории c_i

$$POS_i = \{d_j \in Tr | \Phi(d_j, c_i) = 1\}$$

2. NEG_i – множество документов, не принадлежащих категории c_i

$$NEG_i = \{d_j \in Tr | \Phi(d_j, c_i) = 0\}$$

3. $\bar{c_i}$- профиль или взвешенный центроид для категории c_i, $\bar{c_i} = < w_{1i}, .., w_{|T|i} >$

Для каждой категории вычисляется взвешенный центроид по формуле:

$$\bar{c_i} = \alpha \frac{1}{|POS_i|} \sum_{d \in POS_i} \bar{d} - y \frac{1}{|NEG_i|} \sum_{d \in NEG_i} \bar{d} \qquad (29)$$

Здесь α и y — контрольные параметры, которые характеризуют значимость положительных и отрицательных примеров. Например, если $\alpha = 1$ и $y = 0$, $\bar{c_i}$ будет центром масс всех положительных примеров категории c_i. Обычно $\alpha = 1$ и $y < 1$.

После вычисления взвешенных центроидов для каждой категории, классификатор Роше определяет принадлежность документа рубрике при помощи вычисления расстояния между вектором обрабатываемого документа и центроидом каждой рубрики. Полученное расстояние сравнивается с заданным порогом. В качестве функции расстояния часто используется косинус между векторами. Тогда классифицирующая функция определяется как величина, обратная расстоянию документа $d \in D$ до центроида категории c_i

$$CSV(d, c_i) = \frac{1}{d \times c_i} \quad (30)$$

Преимущества метода:

1. Метод просто реализуем и существует много готовых реализаций
2. Центроид можно быстро пересчитать при добавлении новых примеров. Эта особенность полезна, например, в задаче адаптивной фильтрации, когда пользователь постепенно указывает системе, какие документы выбраны правильно, а какие - нет.
3. Метод дает хорошие результаты, когда документы из одной категории близки друг другу (в смысле расстояния)

Недостатки метода:

1. Если расстояние между документами внутри категории велико, тогда классификатор любому классифицируемому документу будет присваивать малое значение, и не один документ не попадет в эту категорию. Расстояние между документами внутри категории может быть большим, например, когда документы внутри категории на двух разных языках.

3.5. Метод k-ближайших соседей

Метод k-ближайших соседей (k-nearest neighbours, k-NN), в отличие от других, не требует фазы обучения. [7] Для того, чтобы найти рубрики, релевантные документу d, этот документ сравнивается со всеми документами из обучающей выборки. Для каждого документа $d_e \in Ex$ из обучающей выборки находится расстояние - косинус угла между векторами признаков:

$$\rho(d, d_e) = cos(d, d_e) \quad (31)$$

Далее из обучающей выборки выбираются k документов, ближайших к d (k - параметр). Пусть $Ex_k(d)$ – k ближайших к документу d обучающих документов. Для каждой рубрики вычисляется релевантность по формуле

$$CSV(d, c_i) = \sum_{d_e \in Ex_k(d)} \cos(d, d_e) \quad (32)$$

Рубрики с релевантностью выше некоторого заданного порога считаются соответствующими документу. Число ближайших соседей k обычно выбирается в интервале от 1 до 100. Рекомендуется брать k ~ 20-50.

Данный метод показывает довольно высокую эффективность, но требует довольно больших вычислительных затрат на этапе рубрикации.

3.6. Деревья решений

Деревья решений – это способ представления правил в иерархической, последовательной структуре, где каждому объекту соответствует единственный узел, дающий решение. Под правилом понимается логическая конструкция, представленная в виде "если ... то ...". [22]

Дерево принятия решений – ориентированное дерево.

Структура дерева:

- Не листовая вершина – один из признаков
- Ребро – значение признака
- Листовая вершина – значений целевой функции (класс)

Алгоритм 2: Общий алгоритм построения дерева принятия решений

Входные данные
P ← множество признаков
T ← тестовые данные
Результат работы
Дерево решений
Сценарий
пока не рассмотрены все признаки **повторять**
  p ← следующий признак из P
  для каждого значения v выбранного признака p:
    из тестовых данных T оставить только те, у которых p = v
    рекурсивно построить дерево на выбранных данных, рассматривая P\p признаков
  остановится, **если** верно одно из:
      все тестовые объекты принадлежат одному классу
      закончились признаки
      свой критерий

Существует множество алгоритмов построения дерева принятия решений: ID3, C4.5, CART, CHAID, MARS. Они уже реализованы и используются в современных библиотеках для машинного обучения.

Подробнее про деревья решений можно прочитать в лекциях [23].

Достоинства метода [24]

- Прост в понимании и интерпретации. Люди способны интерпретировать результаты модели дерева принятия решений после краткого объяснения

- Способен работать как с категориальными, так и с интервальными переменными
- Использует модель «белого ящика». Если определенная ситуация наблюдается в модели, то её можно объяснить при помощи булевой логики
- Позволяет оценить модель при помощи статистических тестов
- Метод хорошо работает даже в том случае, если были нарушены первоначальные предположения, включенные в модель
- Позволяет работать с большим объемом информации без специальных подготовительных процедур. Данный метод не требует специального оборудования для работы с большими базами данных.

Недостатки метода

- Проблема получения оптимального дерева решений является NP-полной
- Те, кто изучает метод дерева принятия решений, могут создавать слишком сложные конструкции, которые не достаточно полно представляют данные. Данная проблема называется проблемой «чрезмерной подгонки»
- Существуют концепты, которые сложно понять из модели, так как модель описывает их сложным путем.
- Для данных, которые включают категориальные переменные с большим набором уровней, больший информационный вес присваивается тем атрибутам, которые имеют большее количество уровней

3.7. Случайный лес (Random Forest)

Случайный лес (Random Forest) – алгоритм машинного обучения, использует ансамбль (или комитет) решающих деревьев. Рассмотрим алгоритм обучения классификатора. [25]

Обозначения:

- N – размер обучающей выборки
- M – размерность пространства признаков
- m – параметр, число случайно выбираемых признаков для дерева

Все деревья строятся независимо друг от друга по следующей процедуре:

1: Сгенерируем случайную подвыборку с повторением размером N из обучающей выборки.
2: Построим решающее дерево, классифицирующее примеры данной подвыборки. Причём в ходе создания очередного узла дерева будем выбирать признак, на основе которого производится разбиение, не из всех M признаков, а лишь из m случайно выбранных.
3: Дерево строится до полного исчерпания подвыборки и не подвергается процедуре отсечение ветвей

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

Достоинства:

- Высокое качество получаемых моделей, сравнимое с SVM и бустингом, и лучшее, чем у нейронных сетей
- Способность эффективно обрабатывать данные с большим числом признаков и классов
- Нечувствительность к масштабированию (и вообще к любым монотонным преобразованиям) значений признаков
- Одинаково хорошо обрабатываются как непрерывные, так и дискретные признаки
- Существует методы оценивания значимости отдельных признаков в модели
- Внутренняя оценка способности модели к обобщению (тест out-of-bag)
- Высокая параллелизуемость и масштабируемость

Недостатки:

- Алгоритм склонен к переобучению
- Требуется $O(NK)$ для хранения модели, где K – число деревьев

4. Оценка качества классификации

Пусть контрольная выборка состоит из M объектов. Из них m классифицируются правильно. Тогда доля правильных ответов A (от англ. accuracy) вычисляется по формуле:

$$A = \frac{m}{M} \quad (33)$$

Это одна из простейших оценок, и она не всегда отражает реальное качество. [26, 45]

Пусть множество документов разбито по категориям. Обозначим v – множество документов, принадлежащих классу, u – множество документов, приписанных классу алгоритмом классификации.

Полнота r (от англ. recall) классификации документов по классу вычисляется как отношение количества документов, принадлежащих к классу, к общему количеству документов, относящихся к данному классу:

$$r(u) = \frac{|u \cap v|}{|v|} \quad (34)$$

Точность p (от англ. precision) классификации документов по классу вычисляется как отношение количества документов, правильно приписанных к классу, к общему количеству документов, приписанных к данному классу:

$$p(u) = \frac{|u \cap v|}{|u|} \quad (35)$$

F-мера (F-measure) объединяет оценки точности и полноты в одну:

$$F(u) = \frac{2 * p(u) * r(u)}{p(u) + r(u)} \quad (36)$$

Если *p(u)=0* или *r(u)=0*, то тогда и *F(u) = 0*

Для получения сводных характеристик оценок качества классификации по всем классам вводится макро-усреднение характеристик по всем рубрикам.

$$Macro - p = \frac{1}{|C|}\sum_{i=1}^{|C|} p(u_i) \quad (37)$$

$$Macro - r = \frac{1}{|C|}\sum_{i=1}^{|C|} r(u_i) \quad (38)$$

$$Macro - F = \frac{1}{|C|}\sum_{i=1}^{|C|} F(u_i) \quad (39)$$

5. Существующие программные решения для автоматической классификации данных

5.1. Коммерческие проекты

MatLab Statistics Toolbox – пакет прикладных программ для решения задач технических вычислений. Расширение Statistics Toolbox включает реализации алгоритмов машинного обучения. Для работы в среде MatLab используется одноименный язык MATLAB. Хорошо документирован. В среду разработки интегрированы инструменты для визуализации данных. [27]

Решения компании SAS, которая предоставляет пакеты прикладных программ для интеллектуального анализа данных. Решения хорошо документированы, содержат примеры использования. Включает инструменты для визуализации данных. Основной язык программирования SAS BASE.

5.2. Некоммерческие проекты

NLTK (natural language toolkit) – пакет библиотек и программ для символьной и статистической обработки естественного языка. Написан на языке Python. Библиотека сопровождается обширной документацией. [29,30]

SciKit-Learn – пакет библиотек для решения задач машинного обучения. Написан преимущественно на языке Python, основные алгоритмы реализованы на Cython для достижения производительности. Библиотека хорошо документирована, содержит примеры использования и пошаговые инструкции анализа данных. SciKit – включает средства для визуализации анализируемых данных и их оценки основными популярными методами. [31]

Apache Mahout – проект Apache Software Foundation, основная цель - это создание масштабируемых алгоритмов машинного обучения, которые предлагаются для бесплатного использования по лицензии Apache. Представляет из себя библиотеку, написанную на языке java. Mahout включает реализации распределенных алгоритмов кластеризации и классификации. Есть возможность использования библиотеки в связке с Apache Hadoop, что позволяет эффективно масштабироваться в облаке. [32, 33]

Stanford NLP - набор библиотек для обработки и изучения естественного языка. Написан на языке java. Разрабатывается преимущественно Stanford NLP Group. Распространяется под лицензией GNU GPL v2. Библиотека сопровождается документацией. [34]

Все рассмотренные решения являются кроссплатформенными и имеют не плохую документацию. Поэтому выбор библиотеки будет так же учитывать:

- Наличие инструментов для визуализации и изучения данных

- Наличие библиотек для реализация полного стека инструментов, для реализации всех необходимых этапов классификации (от индексации данных до сравнения классификаторов)

В качестве основной библиотеки выбор сделан в пользу SciKit-Learn, так как:

- данное решение имеет удобный и хорошо документированный api
- включает в себя инструменты для реализации всех необходимых этапов классификатора данных
- поддерживает интерактивный режим через оболочку iPython для манипулирования данными и их визуализации, что очень важно при изучении данных и выборе методов классификации
- при необходимости данная библиотека предусматривает возможности масштабирования на несколько серверов.

5.3. Готовые решения автоматической классификации документов

Abbyy Flexy Capture - решение для потокового ввода данных и документов, которое автоматически извлекает информацию из бумажных документов и сохраняет ее в информационную систему предприятия. Стандартные этапы обработки документов включают импорт изображений из различных источников, распознавание, верификацию и экспорт полученных данных в различные информационные системы. [35]

Процесс работы с ABBYY FlexiCapture состоит из нескольких этапов. На первом этапе происходят установка системы и настройка гибких описаний для обработки документов. После того как создан проект со всеми настройками и шаблонами, все типы документов могут обрабатываться в одном потоке. Стандартные этапы обработки документов включают импорт изображений из различных источников, распознавание, верификацию и экспорт полученных данных в различные информационные системы. [35]

Рисунок 7: Автоматическая классификация документов

ABBYY FlexiCapture накладывает существующие гибкие описания на импортированные изображения и собирает отдельные неупорядоченные страницы в документы. Для идентификации различных типов документов в обрабатываемом потоке используется технология интеллектуального распознавания документов (IDR, Intelligent Document Recognition) и технология FlexiCapture. ABBYY FlexiCapture автоматически классифицирует документы с нежестко заданной структурой любой сложности, включая многостраничные документы с разным количеством страниц, многостраничные таблицы и документы, имеющие при себе приложения в виде картинок и текстов. [35]

OpenText Auto-Classification – Программное решение компании OpenText для автоматической классификации текстовых данных. Оно дает организациям мощные средства управления сроками хранения и решением судьбы больших объёмов такого контента, как: сообщения в социальных сетях, сообщения электронной почты, офисные документы и унаследованный контент, - позволяющие снизить правовые риски и расходы на поиск и выемку электронных документов и информации в ходе судебных разбирательств и расследований. [36]

Предлагаемое компанией OpenText приложение для автоматической классификации обеспечивает проведение согласованной, защитимой в случае споров, классификации контента без вмешательства конечных пользователей. Приложение использует технологию OpenText Content Analytics engine для просмотра всех документов, сообщений электронной почты, постов в социальных сетях и классификации контента в соответствии с корпоративной политикой и законодательно-нормативными требованиями. В отличие от аналитики, базирующейся на поиске или ключевых словах, приложение OpenText Content Analytics распознает языковые особенности, выявленные группами экспертов-лингвистов, что радикально улучшает точность классификации. [36, 37]

IBM FileNet

Комплексная корпоративная платформа для управления неструктурированной информацией FileNet — один из наиболее востребованных на мировом рынке программных продуктов, владельцем которого с 2006 года является IBM (приобретена у одноименной компании). FileNet P8 представляет собой программную платформу и готовый инструментарий для создания и развертывания систем управления неструктурированной информацией и интеграции приложений.

Решение обеспечивает надежное долговременное хранение, классификацию документов различных типов: текстовых файлов, сканов, сообщений электронной почты, голосовых сообщений, изображений, видеофрагментов и др. FileNet позволяет оперативно предоставлять документы для обеспечения бизнес-процессов, контролировать выполнение корпоративных требований к документам: обеспечивать точность и непротиворечивость информации, соблюдать срок ее хранения.

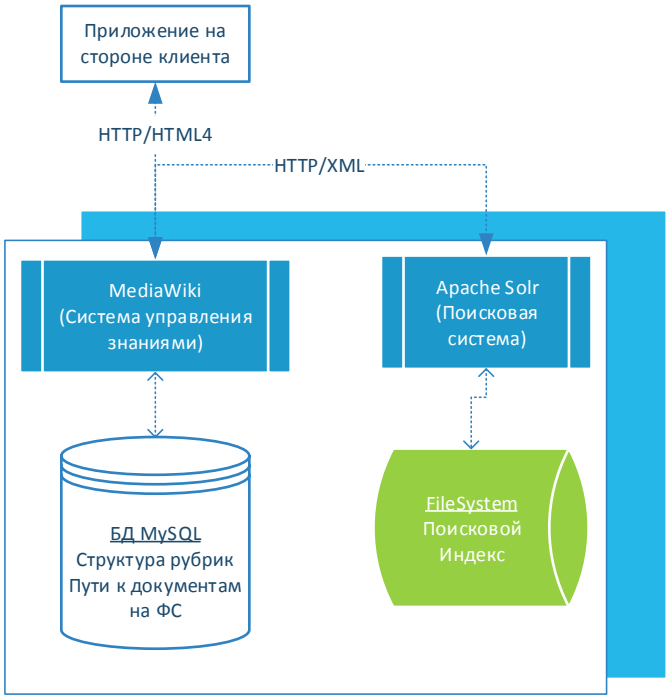
Платформа состоит из ряда компонент. За классификацию данных отвечает компонент FileNet Capture Advanced Document Recognition (ADR). ADR автоматизирует извлечение метаданных, классификацию и разделение изображений документов. [38, 39]

Описанные выше готовые решения являются платными и требуют установки своей платформы в рамках существующей инфраструктуры. У каждого решения существуют возможности по интеграции с популярными ECM системами и предназначены для предприятий с крупным и средним документооборотом. База знаний рассматриваемого предприятия является сравнительно небольшой по числу документов и так как это модернизация MediaWiki, то нет готового инструментария для интеграции, поэтому приведенные выше решения не оправдают своих вложений.

Глава 2 «Исследование и построение решения»

1. Описание инфраструктуры приложений и данных

Существующая база знаний предприятия представлена следующим набором компонент:



Уровень представления

Представлен клиентским приложением. Работает через браузер. Написан на HTML, Javascript и CSS.

Уровень бизнес логики

Представлен доработанным под требования предприятия веб-приложением MediaWiki. Основный язык приложения и его расширений – PHP.

Уровень хранения данных

Представлен базой данных MySQL и файловой системой сервера. База данных MySQL хранит операционные данные веб-приложения. Файловая система сервера хранит исходные документы и их распознанное текстовое содержимое.

Внешние веб-приложения

Платформа полнотекстового поиска Apache Solr, основная задача платформы, построение полнотекстового индекса для документов базы знаний.

Приложение расположено на одном виртуальном сервере с веб-приложением MediaWiki. Apache Solr запускается как отдельное веб-приложение внутри контейнера сервлетов Apache Tomcat. Веб-приложение использует Apache Lucene для реализации

полнотекстового поиска. Обмен данными с MediaWiki происходит по протоколу HTTP, через Web API данными в формате XML.

Сервер распознавания ABBYY Recognition Server, основная задача сервера - принять новые документы, распознать их содержимое, представить документы в виде текста и передать распознанные тексты обратно приложению MediaWiki.

1.1. Основные сценарии работы с Базой знаний

Для создания автоматической системы классификации текстов необходимо знать основные сценарии работы с текстами в рамках сложившейся инфраструктуры. Понимание сценариев работы позволит определить:

- Зависимости по данным и их назначение, т.е. какие данные, в каких приложениях хранятся, и для чего используются системой
- Зависимости по приложениям, т.е. какие приложения участвуют в создании и обработке данных

Необходимые сценарии:

- Загрузка нового документа
- Загрузка нового документа в поисковой индекс

Ниже приведен сценарий загрузки нового документа в Базу знаний MediaWiki:

Сценарий 1: Загрузка нового документа в базу знаний MediaWiki
Входные данные document ← новый документ document_id ← идентификатор нового документа category_id ← идентификатор категории document_text ← {} document_path ← полное имя документа на файловой системе **Результат работы** Запись в базе данных Распознанный текст на файловой системе Поисковой индекс в поисковой системе **Сценарий** 1: **отправить** document на сервер *AbbyyRecognitionServer* 2: document_text ← распознанный текст от *AbbyyRecognitionServer* 3: **сохранить** document_id, category_id, document_path в базе данных 4: **сохранить** document_text на файловой системе в файл document_path 5: **отправить** document_text, document_id на поисковой сервер *Apache Solr* a. Поисковой сервер *Apache Solr* добавляет документ в поисковой индекс

Из приведенного выше сценария следует, что входе загрузки документа задействованы три системы: MediaWiki, AbbyyRecognitionServer и Apache Solr. Важным данными для построения системы классификации являются:

- Идентификатор документа, при обучении системы, позволит идентифицировать документ. При классификации позволит найти документ в базе знаний и присвоить ему установленную категорию.

- Категория документа, нужна при обучении системы.
- Текст документа, нужен для индексации и извлечения термов.

У поисковой системы и системы классификации текстов есть общий этап – индексация данных. Поисковая система Apache Solr для построения поискового индекса использует текст документа на этапе индексации данных. Таким образом, можно воспользоваться уже существующим поисковым индексом для построения классификатора. У данного подхода есть ряд преимуществ:

- Повторное использование поискового индекса.
- Нет необходимости разрабатывать свой этап индексации с нуля.
- Экономия вычислительных ресурсов.
- Качественное разбиение термов.

Ниже приведен сценарий загрузки документа в поисковой индекс.

Сценарий 2: Загрузка нового документа в поисковой индекс
Входные данные document_id ← идентификатор нового документа document_text ← распознанный текст документа document_terms ← {} //вектор слов документа **Результат работы** Поисковой индекс в поисковой системе **Сценарий** 1: document_terms ← извлечение слов и их частот из document_text 2: document_weights ← взвешивание document_terms по внутреннему алгоритму 3: **добавить** document_id, document_terms, document_weights в поисковой индекс

Поисковой индекс документа представляет собой вектор из пар <ключ>:<значение>, где:

- ключ – значение терма
- значение – частота встречаемости данного терма в тексте.

Таким образом, логическое представление документа в поисковом индексе удовлетворяет требованиям системы классификации.

Существует ряд задач, которые необходимо дополнительно решить при использовании поискового индекса:

- Взвешивание термов, веса термов поискового индекса не подходят для получения качественного классификатора.
- Удаление лишних термов, поисковый индекс содержит стоп-слова, последовательности знаков препинания и числовые последовательности, которые ухудшают качество классификации.

Использование поискового индекса накладывает существенное ограничение на логическое представление данных для классификатора. Так как рассматриваемая поисковая система не хранит порядок слов в поисковом индексе, по нему невозможно восстановить граф термов документа.

В итоге, так как существуют статистические методы взвешивания термов, которые дают хороший результат при классификации для малых корпусов документов, в качестве логического представления документа решено использовать поисковой индекс.

2. Построение модели классификатора

Задача выбора модели классификатора документов сильно зависит от предметной области и характеристик рассматриваемых текстов. Реализации существующих методов машинного обучения являются универсальными и не учитывают специфики предметной области. Поэтому для выбора подходящего классификатора выбран экспериментальный подход. Классификаторы будут оцениваться по общепринятым методам оценки полноты, точности и f-меры, которые были описаны в пункте «Оценка качества классификации».

Для построения классификатора необходимо реализовать следующие компоненты:

- Индексатор документов – отвечает за предобработку данных.
- Классификатор документов – обучение и классификация документов.

2.1. Индексатор документа

Индексатора документов включает следующие компоненты:

- Компонент извлечения слов документа.
- Компонент нормализации слов, получение термов.
- .Компонент взвешивания термов.

Задача компонента извлечения слов – представить текст в виде массива последовательно идущих слов без знаков препинания и дополнительных символов разметки. Эту задачу решает индексатор поисковой системы.

Для нормализации слов возможны два основных подхода:

- Стемминг – отбрасывание окончаний и приставок слов.
- Лемматизация – приведение к нормальной форме слова.

Классифицируемые документы написаны преимущественно с использованием русского языка. Русский язык относится к группе флективных синтетических языков, то есть языков, в которых преобладает словообразование с использованием аффиксов, сочетающих сразу несколько грамматических значений. Такие языки допускают использование стемминга. [40]

Для использования лемматизация необходимо:

- уметь определять часть речи слова
- иметь доступ к хорошо размеченному, тематическому корпусу документов.

Ввиду научной специализации рассматриваемых статей и документов, нужно создавать размеченный корпус с нуля, т.к. существующие корпуса опираются на более бытовые по тематике и художественные тексты. Создание с нуля размеченного корпуса научных статей

не входит в планы рассматриваемой работы. Поэтому для нормализации предлагается использовать стемминг.

В качестве стеммера в модели применяется «Стеммер Портера» и его реализация на языке Python в NLTK. Данный стеммер адаптирован для работы с русским языком.

Компонент взвешивания может быть представлен следующими алгоритмами:

- Частота слова в документе
- TF-IDF
- TF-SLF
- Взвешивания термов текста с использованием графа

Реализация первых двух алгоритмов существует в SciKit.

Реализация методов взвешивания с использованием графа термов была написана как модуль на языке Python. Пример взвешивания термов данным методом можно посмотреть в пункте 2.3 данной диссертации. Для рассматриваемой базы знаний методы, основанные на графах, не участвовали в сравнительных тестах и не вошли в реализацию, так как по индексу существующей поисковой системы нельзя восстановить граф термов документа.

Реализация алгоритма TF-SLF написана на языке python. Для ускорения работы с большими матрицами термов используется библиотека numpy. Основные вычисления приведены к матричному виду.

Обозначения:

1. Число категорий: $|\boldsymbol{C}| = \boldsymbol{m}$.
2. Число термов в пространстве признаков: $|\boldsymbol{T}| = \boldsymbol{n}$.
3. ***DF*** – матрица, элементами которой являются $\boldsymbol{df_{ij}}$, строки – категории, столбцы – термы
4. ***NC*** – диагональная матрица, каждая строка соответствует категории, каждый элемент диагонали равен количеству документов в соответствующей категории в минус первой степени.
5. ***R*** – вектор столбец, каждый элемент равен сумме локальных частот терма

Для вычисления метрики *TF-SLF* необходимо:

1. Составить матрицу *DF* частот совместного вхождения в категорию терма и документа, и диагональную матрицу *NC*.

$$DF = \begin{pmatrix} df_{11} & \cdots & df_{1n} \\ \vdots & \ddots & \vdots \\ df_{m1} & \cdots & df_{mn} \end{pmatrix} (41) \qquad NC = \begin{pmatrix} 1/N_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1/N_m \end{pmatrix} (42)$$

2. Получить нормализованную матрицу локальных частот *NDF:*

$$NDF = NC * DF \quad (43)$$

3. Рассчитать вектор-столбец глобальных оценок частот по каждому терму:

$$R = \begin{Bmatrix} ndf_{11} + ndf_{21} + .. + ndf_{m1} \\ ... \\ ndf_{1n} + ndf_{2n} + .. + ndf_{mn} \end{Bmatrix} = \begin{pmatrix} R_1 \\ \vdots \\ R_n \end{pmatrix} \quad (44)$$

4. И получить вектор-столбец SLF:

$$SLF = \begin{Bmatrix} log\frac{|C|}{R_1} \\ ... \\ log\frac{|C|}{R_n} \end{Bmatrix} = \begin{pmatrix} SLF_1 \\ \vdots \\ SLF_n \end{pmatrix} \quad (45)$$

Для расчета матрицы TFSLF необходимо вектор-столбец транспонировать и умножить поэлементно на каждую строку матрицы TF:

$$TFSLF = \begin{pmatrix} TFSLF_1 \\ \vdots \\ TFSLF_n \end{pmatrix} = TF_i \circ SLF^T \quad (46)$$

Где '∘' - операция поэлементного умножения матриц.

2.2. Классификатор документа

Классификатор документа будет выбран с учетом результатов теста методов классификации на экспериментальных данных.

Для классификации будут использованы алгоритмы библиотеки SciKit-Learn, так как:

- Библиотека предоставляет реализации всех необходимых алгоритмов классификации.
- Библиотека предоставляет средства для визуализации и анализа полученных данных.

В эксперименте использованы следующие классификаторы:

1. **Классификация методом Байеса:**
   a. MultinominalNB – классическая реализация метода Байса для многоклассовой классификации
   b. BernoulliNB – в отличие от метода многоклассовой классификации Байеса не учитывает частоту вхождения терма, учитывается только факт вхождения в вектор – есть или нет
2. **Классификация методом опорных векторов:**
   a. LinearSVC – реализация метода опорных векторов с линейным ядром
3. **Классификатор Роше:**
   a. NearestCentroid – реализация метода Роше без дополнительных параметров
4. **Метод ближайших соседей**
   a. KneighborsClassifier – с числом соседей равным 10
5. **Случайный лес –** RandomForestClassifier с 75 деревьями решений

2.3. Полученные результаты

На рисунке 8 приведены результаты сравнения комбинаций методов взвешивания и классификации документов.

Аннотация:

- По вертикали: Оценка F-Macro (больше - лучше)
- По горизонтали сверху: Метод классификации документов
- По горизонтали снизу: Методы взвешивания документов

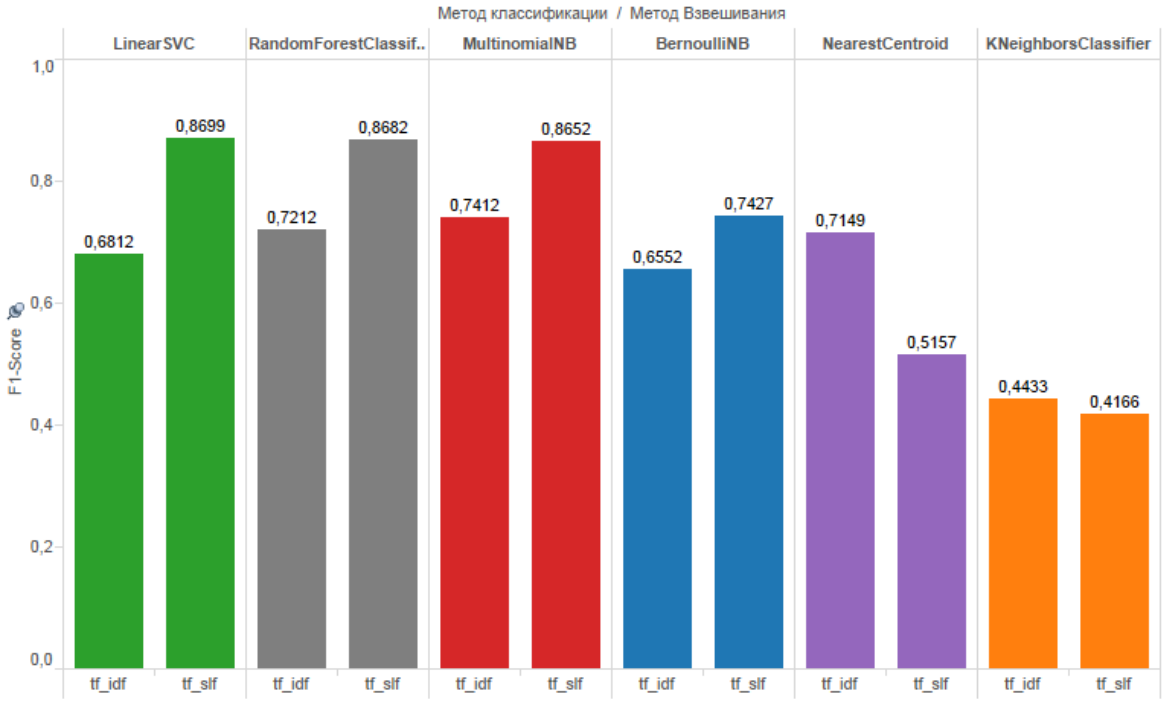


Рисунок 8: Сравнение методов взвешивания и классификации документов

Классификатор	F-macro			Лучший метод
	TF-IDF	TF-SLF	Лучшая оценка	
LinearSVC	0,681	0,869	0,869	TF-SLF
RandomForest	0,721	0,868	0,868	TF-SLF
MultinominalNB	0,741	0,865	0,865	TF-SLF
BernoulliNB	0,655	0,742	0,742	TF-SLF
NearestCentroid	0,714	0,515	0,714	TF-IDF
Knn	0,443	0,416	0,443	TF-IDF
Лучшая комбинация методов по оценке F-macro				
LinearSVC	0,681	0,869	0,869	TF-SLF

Таким образом, из полученных результатов видно, что использование метода взвешивания TF-IDF для классификации документов рассматриваемой базы знаний дало плохой результат. Самая высокая оценка F-macro для метода TF-IDF получена с использованием классификатора Роше (NearestCentroid).

Метод взвешивания TF-SLF позволил увеличить оценку F-macro для классификаторов: SVM (метод опорных векторов), RandomForest (случайный лес), Naïve Bayes (Байесовский классификатор)

На рисунке 9 представлены результаты оценки классификаторов в разрезах полноты и точности.

Аннотация:

- По вертикали: P-macro (тоночть)
- По горизонтали: R-macro (полнота)

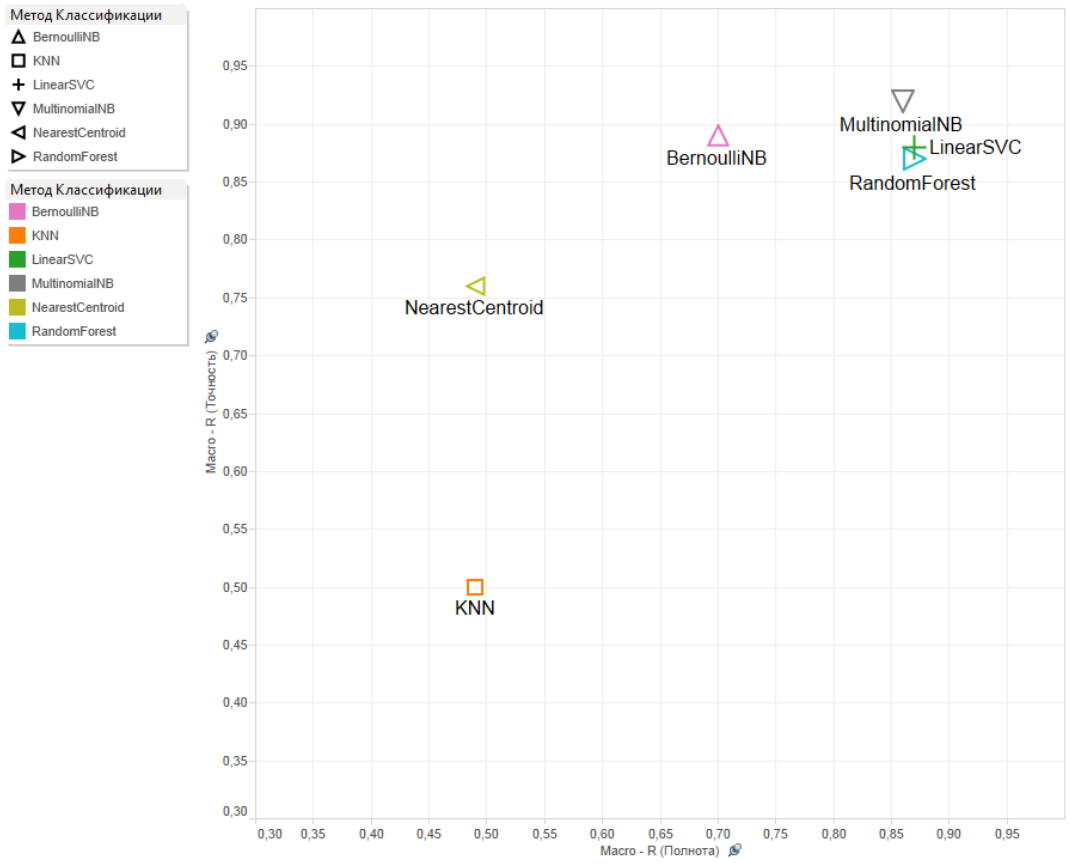


Рисунок 9: Полнота и Точность

Классификатор	P-Macro (Точность)	R-Macro (Полнота)
LinearSVC	0.88	0.87
RandomForest	0.87	0.87
MultinominalNB	0.92	0.86
BernoulliNB	0.89	0.70
NearestCentroid	0.50	0.43
Knn	0.76	0.49

Из результатов оценки полноты и точности алгоритмов следует, что самая большая точность - у метода классификации MultinominalNB, самый высокий результат полноты - у LinearSVC и RandomForest классификаторов.

В качестве метода классификации для базы знаний выбран метод MultinominalNB, так как он показал лучшую оценку точности 0.92 и сравнительно хорошую полноту 0.86.

Детальные оценки по рубрикам для MultinominalNB:

Идентификатор	Точность	Полнота	F1-score	Документов в тесте
136	1.00	0.71	0.83	17
140	1.00	0.68	0.81	25
149	0.62	1.00	0.77	5
152	1.00	1.00	1.00	2
158	0.56	1.00	0.71	10
162	1.00	1.00	1.00	14
164	0.71	1.00	0.83	5
178	1.00	1.00	1.00	15

Список значимых термов при классификации методом MultinominalNB:

Id	Название категории	Топ термов
136	Теория и моделирование компьютерных сетей	Маршрутизация, пакет, деть, коммутатор, спецификация, модель, пкс, политик, сет
140	Управление сетевой инфраструктурой	Mininet, and, управление, network, сет, the, сеть, цод, сетевой, quot
149	Обеспечение безопасности ПКС сетей	Сет, сеть, решение, solutions, платежей, mobile, платёж, мобильный, payment, платеж
152	Облачные технологии	Hardware, small, aws, techcrunch, techcrunchcom, opencloud, required, offers
158	Тестирование ПКС	Segment, средство, цод, сегмент, the, пкс, network, сет, сеть, сетевой
162	Семинары, школы, курсы	Admin, file, openstack, compute, server, the, keystone, sudo, nova, swift
164	Материалы конференций	Tom, update, portal, nfv, marshall, brinn, gec, слайд, geni, презентация
178	Другое	Цпикс, and, computer, specialists, the, onlab, lab, communications, журнал, acm

Из таблицы выше не сложно заметить, что для категорий 136, 149, 158 самые значимые термы пересекаются: сет, сеть, сетевой. Этим пересечением можно объяснить снижение точности классификации для данных категорий.

2.4. Итоговая модель

Итоговая модель классификации данных использует метод взвешивание TF-SLF и метод классификации MultinominalNB. Основным преимуществом данной модели является её высокая точность для рассматриваемой базы знаний.

С ростом базы и появлением новых терминов и документов классификация может ухудшиться. Поэтому по мере роста базы знаний следует переобучать классификатор. На данный момент в базе знаний очень мало документов.

Глава 3 «Описание практической части»

1. Используемые инструменты разработки

Система контроля версий Git

Git – распределенная система контроля версий, предоставляет каждому разработчику локальную копию всей истории разработки. В отличие от централизованных систем, распределенные позволяют вести разработку в отсутствии соединения с репозитарием.

Интерактивная оболочка iPython

IPython расширяет возможности интерпретатора python. Предоставляет инструментарий для интерактивного манипулирования с данными и их визуализации. В рамках работы оболочка IPython использовалась для изучения данных базы знаний.

Редакторы Vim и SublimeText

Основная разработка велась с использованием редактора SublimeText. При изменение и модификации конфигурационных файлов на удаленном сервере использовался Vim.

Tableau Public

Tableau public – инструмент для визуального анализа данных, создания интерактивных графиков и диаграмм. Использовался для визуализации графиков для данной работы.

2. Описание основных компонентов системы

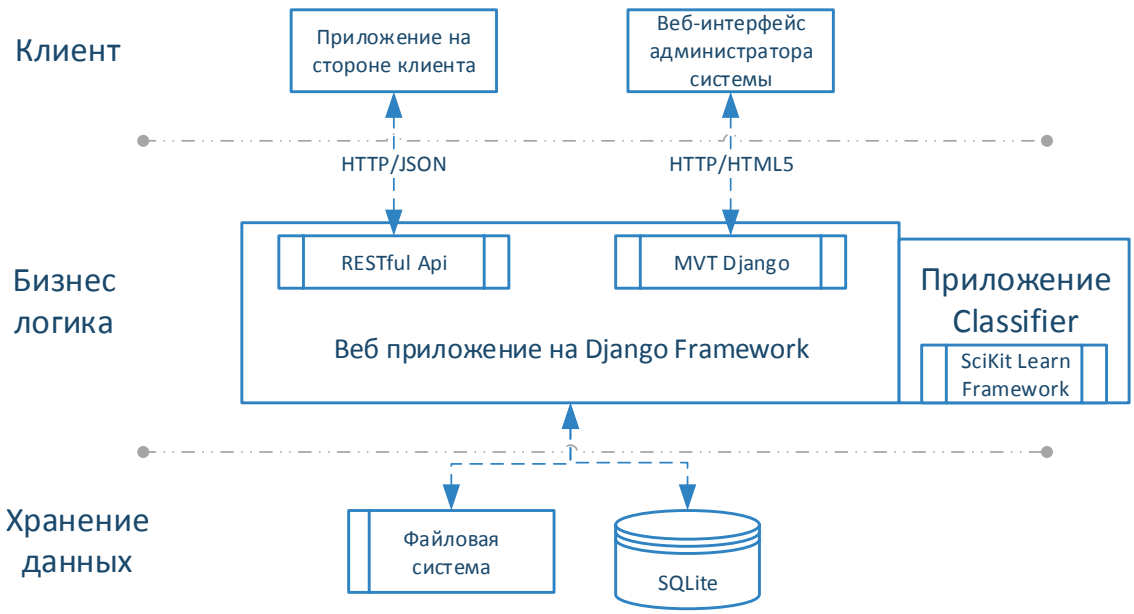


Рисунок 10 - Основные компоненты системы

Уровень представления

- Клиентское приложение для администрирования системы.
  - Написано на HTML, JavaScript и CSS.
- RestFull Api для загрузки, изменения и получения данных о документах.

Уровень бизнес-логики

Уровень бизнес логики обрабатывает поступающую от пользователей информацию.

Данный уровень построен в соответствии со стилем построения архитектуры REST.

REST (Representational State Transfer) – стиль построения архитектуры распределенного приложения. REST используют для построения приложений, в которых клиенты могут отправлять запросы службам, т.е. реализация подхода «клиент-сервер». Данные в REST должны передаваться по протоколу HTTP в виде небольшого количества данных в одном из форматов: HTML, Xml, JSON.

REST приложения должны удовлетворять следующим базовым принципам [41]:

- Агенты пользователей взаимодействуют с ресурсами, которыми может быть всё, что можно поименовать и представить.
- Каждый ресурс однозначно идентифицируется своим URI (Uniform Resource Identifier – универсальный код ресурса).
- Взаимодействие с ресурсами осуществляется с помощью единого интерфейса стандартных команд HTTP (GET, POST, PUT и DELETE).
- Форматы представлений – стандартные MIME-типы.
- Ресурсы описывают себя сами. Вся информация, необходимая для обработки запроса ресурса, содержится в самом запросе.
- В ресурсах могут содержатся ссылки на другие ресурсы.

Преимущества использования принципов REST:

- Единый интерфейс – возможности повторно использовать код.
- Расширение – REST дает возможность включать в каждый ресурс все состояния, необходимые для обработки конкретного запроса. Службы RESTful просто масштабировать при соблюдении данного ограничения.
- Функциональная совместимость – для написания клиентского приложения необходимо иметь http библиотеки доступные на всех современных операционных системах.

Уровень хранения данных

Уровень хранения данных представлен базой данных SQLite.

SQLite – компактная встраиваемая реляционная база данных. SQLite предоставляет библиотеку, с которой программа компонуется и движок SQLite становится частью программы. В результате для обмена используются вызовы api функций библиотеки SQLite. Такой подход уменьшает накладные расходы, время отклика и упрощает программу.

Несколько процессов или потоков могут одновременно читать данные из одной базы. Запись в базу можно осуществить только в том случае, если никаких других запросов в данный момент не обслуживается.

2.1. Реализация REST

Веб-приложение построено с использованием языка Python и веб-фреймворка Django.

Django – свободный фреймворк для веб-приложений. Использует шаблон проектирования MVC (model-view-controller)

Приложение на Django строится из одного или нескольких приложений, которые рекомендуется разрабатывать независимыми и подключаемыми. Основной принцип фреймворка – DRY (Don`t repeat yourself)

Для реализации RESTful служб используется подключаемый модуль rest-framework.

Использование фреймворков Django и rest-framework оправдано их свойствами. Django, в отличии от множества существующих веб-фреймворков, предоставляет удобный механизм явной конфигурации обработчиков URL при помощи регулярных выражений, этот механизм не зависит от структуры контроллеров приложения. Таким образом естественным образом возможно реализовать архитектуру RESTful приложения. Структурой URL управляет основное веб-приложение на Django, при обращении к URL ответственных за веб-службы управление передается контролерам служб, реализованных с использованием rest-framework.

2.2. Описание ресурсов приложения

Ресурс – уникальный объект, доступный по уникальному URL.

Для работы классификатора необходимо определить ресурсы для следующих сущностей:

- Классификатор – ресурс определяет используемый классификатор.
- Обучающий вектор – ресурс представляет обучающий вектор для классификатора.
- Классифицируемый вектор – ресурс представляет вектор для классификации классификатором.
- Категория – ресурс представляет категории, которые классификатор определяет для классифицируемых векторов.

Описание ресурса Классификатор – Classifier

Название поля	Назначение
Id	Уникальный идентификатор в базе данных
Name	Название классификатора в интерфейсе администратора
Description	Расширенное описание классификатора в интерфейсе администратора

Описание ресурса Категория – Label

Название поля	Назначение
Id	Уникальный идентификатор в базе данных
cls_id	Уникальный идентификатор классификатора, к которому относится категория
assigned_id	Ключ присвоенный категории на стороне клиента
Name	Название категории в интерфейсе администратора
Description	Расширенное описание категории в интерфейсе администратора

Описание ресурса Обучающий вектор – TrainVector

Название поля	Назначение
Id	Уникальный идентификатор в базе данных
assigned_id	Ключ присвоенный вектору на стороне клиента
cls_id	Уникальный идентификатор классификатора, к которому относится вектор
lbl_id	Уникальный идентификатор категории, к которой относится вектор
Data	Словарь значений типа <ключ>:<частота ключа> в формате json

Описание ресурса Классифицируемый вектор – TestVector

Название поля	Назначение
Id	Уникальный идентификатор в базе данных
assigned_id	Ключ присвоенный вектору на стороне клиента
cls_id	Уникальный идентификатор классификатора, к которому относится вектор
lbl_id	Уникальный идентификатор категории, к которой классификатор соотнес вектор
Data	Словарь значений типа <ключ>:<частота ключа> в формате json

2.3. Описание URL стркутуры RESTful api

Согласно основным принципам REST каждый ресурс однозначно определяется URL. Таким образом, для URL структуры получаем строго заданный формат.

- Путь ко всем сущностям рассматриваемого типа: *<path>/entity*
- Путь к определенной сущности по заданному id: *<path>/entity/{id}*

Корневой URL для доступа к службам: *http://<web-app name>/api/*

URL структура служб

Путь к ресурсам типа «*классификатор*»	Путь к ресурсам типа «*обучающий вектор*»
*api/**classifier*** *api/**classifier**/{**id**}*	*api/classifier/{id}/**train_vector*** *api/classifier/{id}/**train_vector**/{**id**}*
Путь к ресурсам типа «*категория*»	Путь к ресурсам типа «*классифицируемый вектор*»
*api/classifier/{id}/**label*** *api/classifier/{id}/**label**/{**id**}*	*api/classifier/{id}/**test_vector*** *api/classifier/{id}/**test_vector**/{**id**}*

2.4. Схема базы данных классификатора

Ресурсы, переданные со стороны клиента приложению, хранятся в базе данных приложения. Каждому типу ресурса соответствует своя таблица со схожим названием.

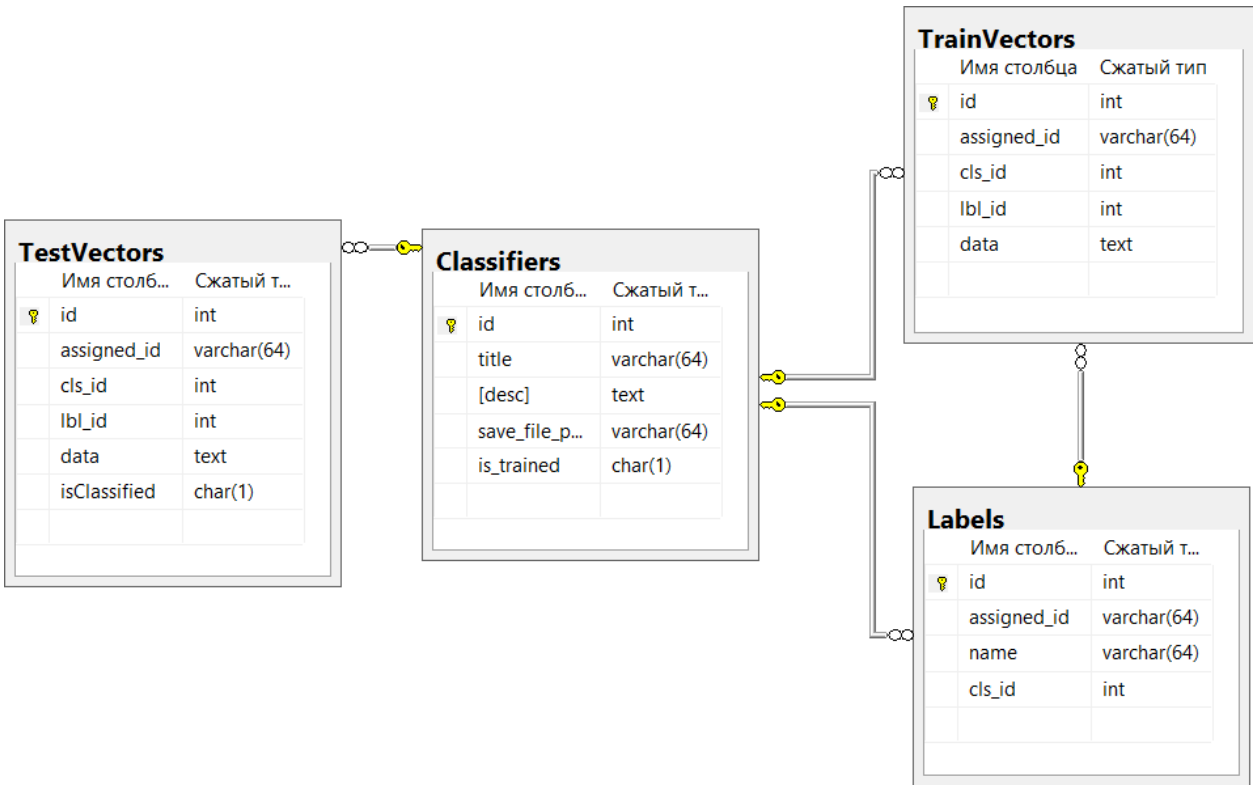


Рисунок 11 - Схема базы данных классификатора

Схема базы данных классификатора приведена на Рисунке 1. Поля таблиц базы данных почти полностью совпадают с определением полей соответствующих ресурсов.

Название ресурса	Название таблицы	Дополнительные поля
Classifier	Classifiers	○ **save_file_path** – хранит название классификатора на файловой системе ○ **is_trained** – хранит статус обучения классификатора (значения Да / нет)
Label	Labels	
Train Vector	Train Vectors	
Test Vector	Test Vectors	○ **is_classified** – хранит статус классификации вектора (классифицирован? Да / нет)

2.5. Сценарий работы клиентского приложения

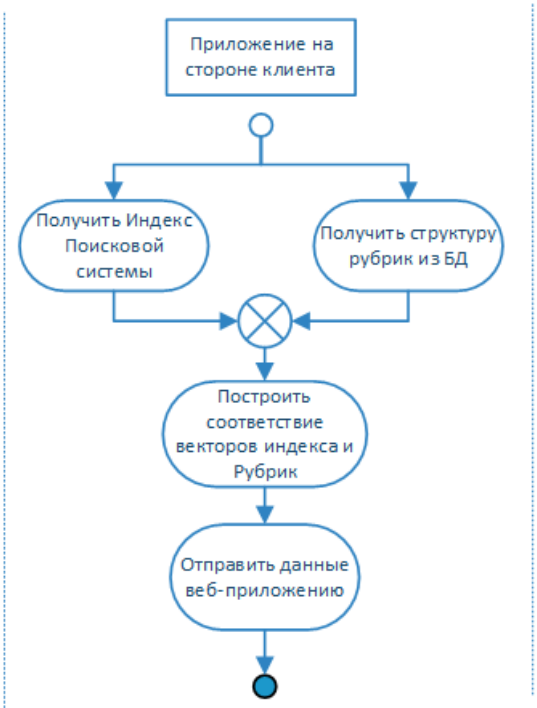
Для работы с веб-приложением необходимо реализовать клиентское приложение.

Основные функции клиентского приложения:

- Сбор данных из базы данных и поисковой системы.
- Подготовка данных к отправке на сервис веб-приложения.
- Запрос данных о решении классификатора и обновление категории для классифицируемого документа.

Ниже представлен основной сценарий загрузки данных для обучения веб-приложения:

Сценарий: Загрузка данных для обучения
1: index ← Поисковой индекс 2: structure ← Структура документов Базы знаний 3: data_transfer ← соответствие(index, structure) 4: отправить data_transfer на веб-приложение



Заключение

В рамках магистерской диссертации были исследованы и опробованы методы машинного обучения для решения задачи классификации текстов небольшой базы знаний. Получены следующие теоретические результаты для базы с числом документов меньше пятисот и сильно варьирующимся числом документов внутри каждой категории:

- Применение метода TF-SLF для извлечение термов позволяет улучшить качество классификации на многоязычных документах, в рамках корпуса с числом документов меньше 500.
- Применение многоклассовой классификации Байеса совместно с применением метода TF-SLF позволяет получить лучший результат по точности классификации текстов.

Предложено представление метода TF-SLF в матричном виде и реализован его алгоритм на numpy, расширении языка python для быстрых вычислений над матрицами.

По результатам исследований базы знаний и инфраструктуры предложена и реализована система автоматической классификации документов в виде веб-приложения с сервисом RESTful api. В рамках разработанной системы предложен сценарий использования существующего поискового индекса базы знаний для классификации данных.

Список литературы

[1] John Gantz, David Reinsel, «The Digital Universe: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East», December 2012

[2] Julie Lockner, «*Is The Data Explosion Impacting You? How Do You Compare To Your Peers?*», April 2013 (http://blogs.informatica.com/perspectives/2013/04/01/is-the-data-explosion-impacting-you-how-do-you-compare-to-your-peers/)

[3] «*Находимость корпоративных данных. Обзор опыта пользователей ECM-систем*»// Docflow, Abbyy, 2014

[4] «*Корпоративная база знаний*», Ноябрь 2010 (http://enterprisekb.ucoz.ru/news/korporativnoj_bazy_znanij/2010-11-25-1)

[5] «*Глоссарий*» (http://itmu.vsuet.ru/Posobija/Predstavlenie_znan/htm/glossary.htm)

[6] Юрий Лифшиц, «*Автоматическая классификация текстов*» // Лекция №6 из курса «Алгоритмы для интернета», Ноябрь 2006

[7] Агеев М. С. «*Методы автоматической рубрикации текстов, основанные на машинном обучении знаниях экспертов*» 2004

[8] «*Глоссарий*» // Веб-студия pawlov.info (http://www.pawlov.info/index.php/glossarij)

[9] Ellen Riloff, «*Little Words Can Make a Big Difference for Text Classification*» // Department of Computer Science, University of Utah

[10] Воронцов К. В. «Вероятностное тематическое моделирование» // Лекции по Машинному обучению, Октябрь 2013

[11] «*N-грамм*» // Wikipedia (http://ru.wikipedia.org/wiki/N-грамм)

[12] «Синонимы» // Wikipedia (http://ru.wikipedia.org/wiki/Синонимы)

[13] «Омонимы» // Wikipedia (http://ru.wikipedia.org/wiki/Омонимы)

[14] Губин М. В. «*Модели и методы представления текстового документа в системах информационного поиска*», 2005

[15] Abdur Rehman, Haroon A. Barbi, Mehreen Saeed, «*Feature Extraction for Classification of Text Documents*», 2012

[16] Алгоритм PageRank (http://en.wikipedia.org/wiki/PageRank)

[17] Brin, S.; Page, L. (1998). «*The anatomy of a large-scale hypertextual Web search engine*». Computer Networks and ISDN Systems

[18] Christopher D. Manning, Prabhakar Raghavan, Hinrich Schutze, «*Introduction to Information Retrieval»,* 2008

[19] Samer Hassan and Rada Mihalcea and Carmen Banea. «*Random-Walk TermWeighting for Improved Text Classification*» 2007

[20] «*Метод опорных векторов*» // Wikipedia (http://ru.wikipedia.org/wiki/Метод_опорных_векторов)

[21] Воронцов К. В. «*Лекции по методу опорных векторов*» // Курс лекций «*Машинное обучение*» Декабрь 2007

[22] «*Деревья решений – общие принципы работы*» // BaseGroup Labs (http://www.basegroup.ru/library/analysis/tree/description/)

[23] Николенко С. И. «*Деревья принятия решений*» // Конспект лекций

[24] «*Дерево принятия решений*» // Wikipedia (http://ru.wikipedia.org/wiki/Дерево_принятия_решений)
[25] «*Random Forest*» // Wikipedia (http://ru.wikipedia.org/wiki/Random_forest)
[26] Токарева Е. И. «*Иерархическая классификация текстов*», 2010
[27] «Matlab» // Wikipedia (http://ru.wikipedia.org/wiki/MATLAB)
[28] «About SAS» // www.sas.com (http://www.sas.com/en_us/company-information.html/)
[29] «Natural Language Toolkit» // Wikipedia (http://ru.wikipedia.org/wiki/Natural_Language_Toolkit)
[30] «Natural Language Toolkit» // NLTK (http://www.nltk.org/)
[31] «*Scikit-learn*» (http://scikit-learn.org/stable/#)
[32] «Введение в Apache Mahout» // IBM (http://www.ibm.com/developerworks/ru/library/j-mahout/)
[33] «What is Apache Mahout» // Mahout (https://mahout.apache.org/)
[34] «Stanford NLP» (http://www-nlp.stanford.edu/software/index.shtml)
[35] «Основные возможности ABBYY FlexiCapture» // Abbyy (http://www.abbyy.ru/flexicapture/opportunities/)
[36] Наталья Храмовская, «Автоматическая классификация документов – уже реальность?» (http://rusrim.blogspot.ru/2011/12/blog-post_2249.html)
[37] «Enterprise Content Management» // OpenText (http://www.opentext.com/what-we-do/products/enterprise-content-management)
[38] «Обзор IBM FileNet» // Tadviser (http://www.tadviser.ru/index.php/Продукт:IBM_FileNet)
[39] Shirley Braley, «Добавьте автоматическую классификацию содержимого в IBM FileNet P8» \\ IBM, Developer Works, 2008 (http://www.ibm.com/developerworks/ru/library/dm-0801wang/)
[40] «Стемминг» // Wikipedia (http://ru.wikipedia.org/wiki/стемминг)
[41] Jon Flanders «Введение в службы RESTful с использованием WCF», Январь 2009
    - http://msdn.microsoft.com/ru-ru/magazine/dd315413.aspx
[42] Chuntao Jiang, Frans Coenen, Robert Sanderson, Michele Zito. "*Text Classification using Graph Mining-based Feature Extraction"*
[43] Wei Wang, Diep Bich Do, Xuemin Lin, «*Term Graph Model for Text Classification*»
[44] Michael Granitzer «*Hierarchical Text Classification using Methods from Machine Learning*»
[45] Christopher D. Manning, Prabhakar Raghavan, Hinrich Schutze, «An Introduction to Information Retrival» // Cambridge UP, 2009
[46] R. Mihalcea and P. Tarau. «TextRank: Bringing Order into Texts», 2006

Приложение 1 – Детальное описание результатов оценки классификаторов

Детальные оценки по рубрикам для RidgeClassifier:

Идентификатор	Точность	Полнота	F1-score	Документов в тесте
136	0.83	0.88	0.86	17
140	0.66	0.92	0.77	25
149	1.00	0.40	0.57	5
152	1.00	0.50	0.67	2
158	0.67	0.40	0.50	10
162	1.00	0.93	0.96	14
164	1.00	0.80	0.89	5
178	1.00	0.93	0.97	15

Детальные оценки по рубрикам для Knn:

Идентификатор	Точность	Полнота	F1-score	Документов в тесте
136	0.50	0.65	0.56	17
140	0.41	0.96	0.96	25
149	0.00	0.00	0.00	5
152	0.00	0.00	0.00	2
158	0.75	0.30	0.43	10
162	0.00	0.00	0.00	14
164	1.00	0.40	0.57	5
178	1.00	0.40	0.57	15

Детальные оценки по рубрикам для RandomForest:

Идентификатор	Точность	Полнота	F1-score	Документов в тесте
136	0.83	0.88	0.86	17
140	0.79	0.92	0.85	25
149	1.00	0.40	0.57	5
152	1.00	1.00	1.00	2
158	0.62	0.50	0.56	10
162	1.00	1.00	1.00	14
164	1.00	1.00	1.00	5
178	1.00	1.00	1.00	15

Детальные оценки по рубрикам для LinearSVC:

Идентификатор	Точность	Полнота	F1-score	Документов в тесте
136	0.83	0.88	0.86	17
140	0.81	0.88	0.85	25

149	0.75	0.60	0.67	5
152	1.00	1.00	1.00	2
158	0.62	0.50	0.56	10
162	1.00	1.00	1.00	14
164	1.00	1.00	1.00	5
178	1.00	1.00	1.00	15

Детальные оценки по рубрикам для NearestCentroid:

Идентификатор	Точность	Полнота	F1-score	Документов в тесте
136	0.69	0.53	0.60	17
140	0.65	0.44	0.52	25
149	0.60	0.60	0.60	5
152	0.06	1.00	0.12	2
158	0.57	0.80	0.67	10
162	1.00	0.07	0.13	14
164	1.00	0.60	0.75	5
178	1.00	0.60	0.75	15

Детальные оценки по рубрикам для BernoulliNB:

Идентификатор	Точность	Полнота	F1-score	Документов в тесте
136	1.00	0.59	0.74	17
140	1.00	0.40	0.57	25
149	0.62	1.00	0.77	5
152	0.11	1.00	0.20	2
158	0.57	0.80	0.67	10
162	1.00	0.86	0.92	14
164	1.00	0.80	0.89	5
178	0.82	0.93	0.87	15

Приложение 2 – Листинг кода, Реализация TF-SLF

```
def get_dfs(category_matrix):
    """
    Return document frequency vector for matrix
    """
    #dfs = Counter(category_matrix.nonzero()[1])
    dfs = (category_matrix != 0).sum(0)
    return dfs.getA()[0]

def get_df_matrix(dfs_arrays):
    """
    Return dfs matrix for array
    """
    df_m = np.array([df for df in dfs_arrays])
    return df_m

def get_nc_for_category(category_matrix):
    """
    Return Nc: number documents in category
    """
    return category_matrix.shape[0]

def get_ndf_matrix(ncs_matrix, df_matrix):
    """
    Return Ndf Matrix: NCS * DF
    """
    return ncs_matrix.dot(df_matrix)

def get_ncs_matrix(ncs_array):
    """
    Return diagonal NCS matrix
    """
    length = len(ncs_array)
    indices  = range(length)
    ncs = np.zeros((length,length))
    for i in xrange(length):
        ncs[i][i] = 1.0/ncs_array[i]
    return ncs

def get_slf_vector(ndf_matrix):
    """
    Return slf vector
    """
    r_vector = ndf_matrix.sum(0)
    num_cats, num_terms = ndf_matrix.shape
    logargs = np.divide([num_cats]*num_terms, r_vector)
    slf_vector = np.log10(logargs)
    return slf_vector

def get_tf_slf(tf_sparse, slf_vector):
    n = tf_sparse.shape[1]
    slf_m = ssp.lil_matrix((n,n))
    slf_m.setdiag(slf_vector)
    tf_slf = tf_sparse * slf_m
    return tf_slf
```

Приложение 3 – Пользовательский интерфейс

Страницы Администрирования приложения.

Компоненты Django Framework

- django.contrib.admin
- grappelli

Возможности:

- Быстрый переход на страницы Добавить / Изменить
  - Пользователей
  - Категории, Классификаторы, Вектора
- Изменение настроек учетной записи

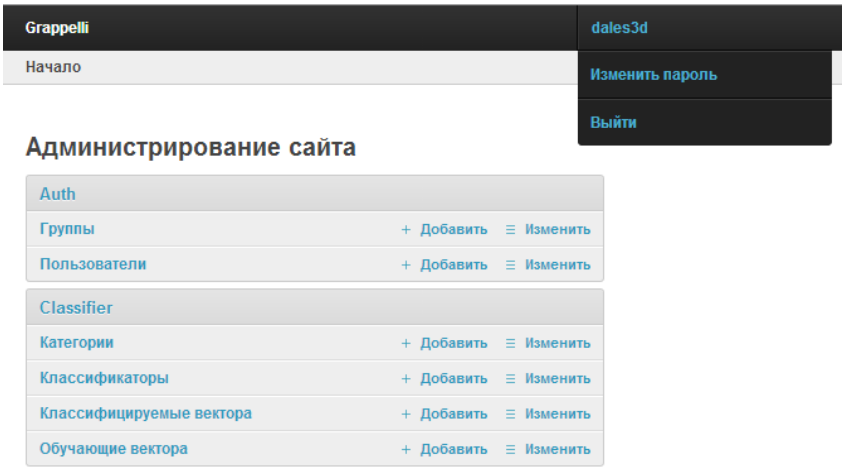


Рисунок 12: Основная страница администрирования

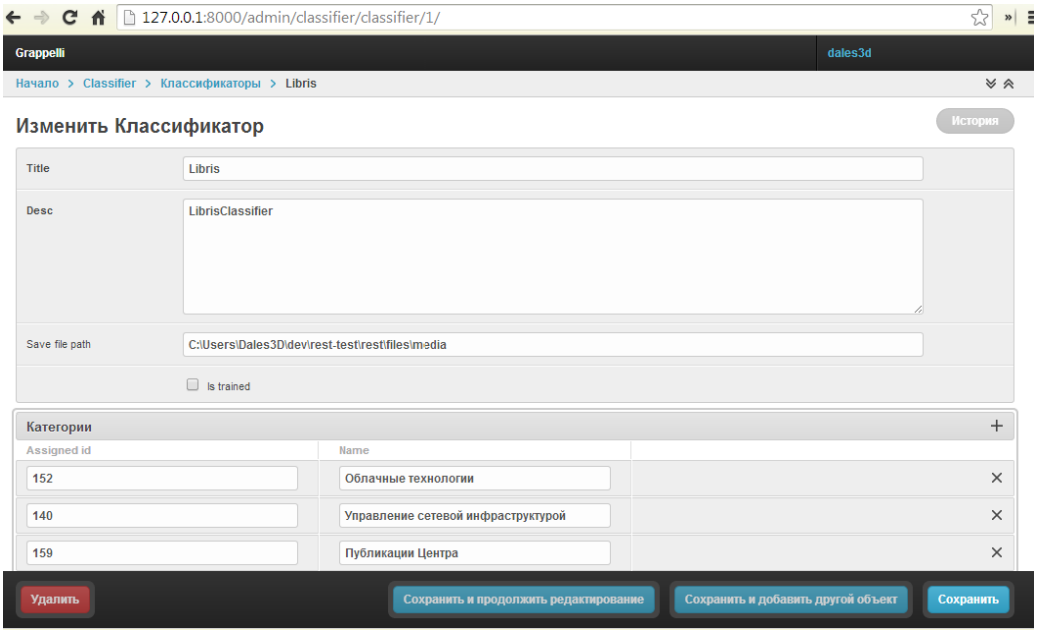


Рисунок 13: Пример страницы редактирования классификатора